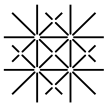


University of Basel

Multi-Model Geospatial Queries with Interactive Visualization in a Polystore System

Master Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems Research Group
https://dbis.dmi.unibas.ch/

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: David Lengweiler, MSc. and Yiming Wu, MSc.

Rafael Biehler
r.biehler@unibas.ch
22-059-091

31.01.2025

Acknowledgments

I would like to thank my supervisors, **David Lengweiler** and **Yiming Wu**, for their consistent support and guidance throughout this thesis. Their weekly meetings, assistance with debugging, and willingness to answer my questions were invaluable in helping me complete this work.
I am also grateful to **Heiko Schuldt** and **Marco Vogt** for providing me with this thesis topic and for their helpful advice.
I also want to thank **all participants** who helped me by being part of the user study.
Lastly, I want to thank my friends and family, and anyone who supported me during this thesis, directly or indirectly, for their encouragement and understanding.

Abstract

Managing and analyzing spatial data within large heterogeneous datasets remains a challenge in modern database systems. PolyDBMS systems provide a solution to manage these datasets by unifying different databases and their data models behind a single query interface, but they currently have limited support for storing and querying spatial data. This thesis examines how PolyDBMS can be extended with spatial capabilities by analyzing existing spatial database systems and proposing solutions to bridge this gap. This is achieved by enabling the storage of spatial data across models and integrating spatial functions to operate across different data models. These approaches are then implemented in the existing PolyDBMS system Polypheny. To improve the usability of working with spatial data, a framework is introduced, which assists the user with visualization and query-building capabilities that enable the user to build complex spatial queries using data in various underlying data models. An interactive map-based query mode is integrated into the user interface of a PolyDBMS system, allowing users to explore and validate spatial data by working with queries in a more interactive and visual way. The implementation of spatial capabilities in a PolyDBMS system was validated through a reference comparison with the integrated data stores for the document and graph model. The map-based query mode was evaluated through a user study, where participants completed tasks using a prototype. The results indicate that the introduced user interface enhances usability and provides valuable functionality for working with spatial data in the multi-model context. The reference comparison highlights that, despite performance challenges, the conceptual advancements in spatial support mark a significant step toward spatially-enabled PolyDBMS systems.

Table of Contents

1 Introduction

PolyDBMS databases provide a new method of managing the ever-increasing volume and variety of data that is generated today.[1] Devices are getting smaller and more powerful. The number of GPS tracking devices, IoT devices, smartphones, and wearables like smartwatches is steadily increasing, creating more location-based data. Geospatial[2] data comprises shapes like points, lines, polygons, and combinations. Handling this data requires specialized data formats and spatial functions to support features like searching, filtering, and aggregating data, which are typically only available in spatially-enabled databases. PolyDBMS, with its support for OLTP and OLAP queries, can handle large heterogeneous datasets; however, it has limitations in processing spatial data. We want to address this limitation of PolyDBMS systems by analyzing how they can be extended to better handle spatial data. This thesis addresses PolyDBMS's limitation by analyzing how spatial data can be handled in the multi-model context.

We will enhance the spatial capabilities of Polypheny, a leading PolyDBMS, by developing a unified spatial model that extends spatial support to both the document and graph models. This includes the interoperability between the data models, implementing spatial functions, integrating native data stores, and creating a new way of unit testing multiple systems.

To improve the user experience when working with spatial data, we will introduce a framework for visualizing spatial data that assists users with interactive features. The interactive features will be particularly useful for users with limited programming and database knowledge, making interaction with spatial data more intuitive and user-friendly. This framework analyzes how geometries can be visualized and how interactive features can be implemented in the multi-model context. The framework introduces the foundations for a new query mode in Polypheny, whose primary goal is improves the handling of spatial data in the database.

[1] M. Armstrong, "Global Data Creation is About to Explode [Digital image]," April 16, 2019, available at https://www.statista.com/chart/17727/global-data-creation-forecasts/.

[2] Geospatial data is spatial data that references a location on Earth. Spatial data is the more general term.

1.1 Motivation

The motivation for this thesis is an expert interview that was conducted by a member of the research group Angewandte und Umweltgeologie[3] (AUG) at the Department of Environmental Sciences at the University of Basel. This interview explored the challenges of working with large heterogeneous datasets and the complexities they introduce.
Writing queries in multiple query languages to different database systems can be a challenge, especially for researchers who are not very experienced in programming or database systems. Working with spatial data is becoming more prevalent for an increasing number of people, many of whom are not skilled in writing spatial queries. Polypheny and PolyDBMS, in general, provide solutions when working with a lot of data. However, they are so far lacking in spatial features. Having access to a spatially-enabled database makes working with spatial data possible but not necessarily straightforward. Using third-party tools to work with spatial data requires additional setup and integration.

To illustrate this problem further, we will provide the following use cases to illustrate how important working with spatial data has become:

Use Case 1: Environmental Sciences This use case is derived from an interview with an environmental scientist working at the AUG research group. Borehole data in Switzerland is collected from diverse sources and centrally managed by the research group. Each borehole is associated with multiple data types contributed by different drilling teams. The data contains relational data recorded by drilling instruments, metadata provided by the drilling team, and multimedia data such as documents, images, and specialized data formats. Depending on the canton, the team, and the goals of the data collection, different conventions and procedures are used, thereby also a different data schema. This poses problems, as the current solution consists of a relational database, that is not equipped to handle large amounts of heterogeneous data and requires time-intensive changes to the schema each time new data has to be inserted. Multimedia files that are difficult to work with inside the relational database are stored separately in a file system, which makes the overall system harder to use, as the data is stored in different systems, depending on the data type. The primary use case for this data is within specialized GIS applications, which typically require the following workflow:

1. Identifying relevant data through filtering spatially.
2. Data cleaning and preparation.
3. Exporting the data in a specific format for use in GIS software.

The existing workflow involves substantial manual effort, particularly in schema adjustments and data transformation. By integrating a spatially-enabled PolyDBMS system, the workflow of the environmental sciences could be greatly simplified. The scientists would greatly benefit from features that make working with databases and programming languages easier, simplifying the process further.

[3] Applied and Environmental Geology

Use Case 2: Genealogy Genealogy research focuses on reconstructing family histories and lineages based on primary sources. These sources often exist in analog formats, such as old books and archival records, which must first be digitized and converted into structured digital data. Processing these sources results in diverse data formats, including documents, images, and various genealogical formats data formats. Storing this data in a single system could simplify the workflow. Additionally, as genealogy interests academic researchers and amateur historians, data quality and structuring methodologies differ considerably. One active area of research is tracking human migration patterns by analyzing birthplaces, places of residence, and death places.
Genealogy research is conducted within the social sciences, where expertise in databases and programming languages is often limited. A spatially-enabled PolyDBMS would significantly enhance research workflows by addressing the following problems:

- Centralized management of heterogeneous data collected from diverse sources, including multiple file formats, documents, and images.
- Enhanced search capabilities using spatial filters.
- Storage and querying of family relationships using a graph-based model, which naturally represents genealogical connections.
- Storage of large files and documents within a document-oriented database.
- Organization of structured metadata and standardized records in a relational database.

By leveraging a spatially-enabled PolyDBMS, genealogists would gain a more flexible and efficient system for managing historical records, enabling advanced spatial analysis and improving the overall research process.

1.2 Requirements

Based on the use cases, as well as the motivation of the previous section, we will propose the following requirements:

- Many heterogeneous data sources contain spatial data. Working with this spatial data should be possible in multiple different data models.
- Many researchers, as illustrated in the use cases, are not very experienced in working with spatial data inside databases or lack experience in programming. Due to these reasons, we want to improve the workflow when working with spatial data to improve the following aspects:
  - Data Validation: One important aspect of working with data is finding the right data to work on. This often involves the visualization and filtering of spatial data.
  - Data Exploration: To make the data stored inside the different models of a PolyDBMS more accessible, it should be possible to be retrieved without having

to write spatial queries. Further, it should provide tools to refine the queries iteratively, for example, by editing the query or applying spatial filters with the assistance of the UI.

- Data Export: As the PolyDBMS is often only the first step of a larger workflow, allowing users to export data in a standardized format can bring much value.

1.3 Contributions

This thesis aims to make three distinct contributions:

1 Add support for multi-model GIS queries to Polypheny. This contribution extends the capabilities in working with spatial data in Polypheny by extending the document and graph model accordingly. As a result, GIS queries should be supported in all data models of Polypheny. Various changes inside Polypheny are necessary to achieve this, from the parsing of MQL and Cypher to spatial function implementations and the integration of native data stores. Furthermore, we want to ensure that the data models available in Polypheny provide a unified model for working with spatial data by handling the differences between the models accordingly. To achieve this, we will give a conceptual overview of spatial data in the multi-model context, covering how spatial data can be represented and how spatial functions are implemented. To evaluate this contribution, a reference comparison will be implemented that compares the implementation in Polypheny with the data stores as the reference implementation.

2 Visualization System with Interactive Query Building. This contribution aims to improve the user experience in working with spatial data by providing a visualization system with interactive features. It is analyzed how spatial data can be represented to the user and how data can be encoded visually. Interactive features improve usability, especially for individuals less experienced in working with programming languages and databases. Together, these features aim to supplement the representation of textual data and remove complexity when working with query languages, especially in combination with spatial queries. This conceptual contribution is the foundation of the implementation described in Contribution 3, which will also be used to evaluate this concept.

3 Map-Based Query Mode & Evaluation This contribution adds a new query mode to Polypheny that allows users to work with spatial data in a more visual and interactive way. Integrated into the Polypheny web interface, this query mode allows users to represent the query results on a map. Queries results can be interactively edited and modified by providing tools to change visualization modes and apply filters to the query. The usability and utility of this implementation will be evaluated by conducting a user study, where users are asked to accomplish tasks using the query mode.

1.4 Outline

Chapter 2 outlines the foundations of PolyDBMS and its implementation in Polypheny. It covers GIS fundamentals, Coordinate Reference Systems, spatial data relationships, common spatial data formats, usability principles, and the prior work this thesis builds upon. **Chapter 3** discusses other polystore systems and how they compare to Polypheny. **Chapter 4** analyzes how spatial data can be mapped in the multi-model concept and how spatial functions can be classified. It also introduces the conceptual contribution of the Interactive Visualization Framework, which describes how spatial data can be represented and how the user can interact with it. **Chapter 5** is split into two parts: First, it covers the modifications that were made in the database component of Polypheny and how the different models were implemented. Secondly, it introduces the Map-Based Query Mode. **Chapter 6** evaluates this thesis's contributions by describing how the evaluation was performed and discussin the results. **Chapter 7** concludes this thesis and provides ideas for how this work could be improved in the future.

2
Background

2.1 PolyDBMS

The first reference implementation for a polystore is the BigDAWG Polystore system introduced in [6]. A polystore aims to be a one-size-fits-all database that unifies all data models in a single database. It achieves this by utilizing several databases internally for the different data models, which can be accessed through a single interface.
This idea is also called *polyglot persistence*[4]. It is a derivative of the concept of *polyglot programming*, where the idea is to use different programming languages so that one can take advantage of the strength of each language. Polyglot persistence is the same concept but applied to databases. Instead of using one database for everything, one can use different databases and save the data in the database with the most fitting data model. This concept can also be applied to query languages: Instead of being forced to use a single query language for each data model, it should be possible to use the query language that is easiest to use, irrespective of the underlying data model.

PolyDBMS In [27], the term PolyDBMS is coined to describe polystore systems designed to function as fully-fledged database management systems (DBMS). This includes multiple data storage engines, which can be queried with different query languages. Queries can be, at least partially, pushed down to their native data store. Data modification through DML[5] and DDL[6] operations should be supported.

2.1.1 Polypheny

Polypheny is a polystore database first introduced in [25]. Polypheny is an implementation of the PolyDBMS concept.
Polypheny supports three main data models: Relational, document, and graph (more details provided in section 4.1). Data is organized into namespaces, each associated with a

[4] Martin Fowler, "Polyglot Persistence" 2011, available at https://martinfowler.com/bliki/PolyglotPersistence.html, accessed January 22, 2025.
[5] Data Manipulation Language
[6] Data Description Language

data model. For example, a relational namespace stores data as tables, and a document namespace stores data as a document collection. Additionally, namespaces allow the access of entities from other namespaces by utilizing views that automatically help in mapping the data to the correct data model.

Data Store Adapter Adapters provide the glue between the internal representation of data and queries. Adapters provide a mapping from the internal data model to the data model of the underlying store. They also provide the ability to convert operations from the internal query representation to the native query, allowing queries or sub-queries to be executed natively. Types that are sent from and to the native store are converted by the adapter.
The entry point for a query is a Query Interface, which supports accepting queries in one or multiple different query languages.

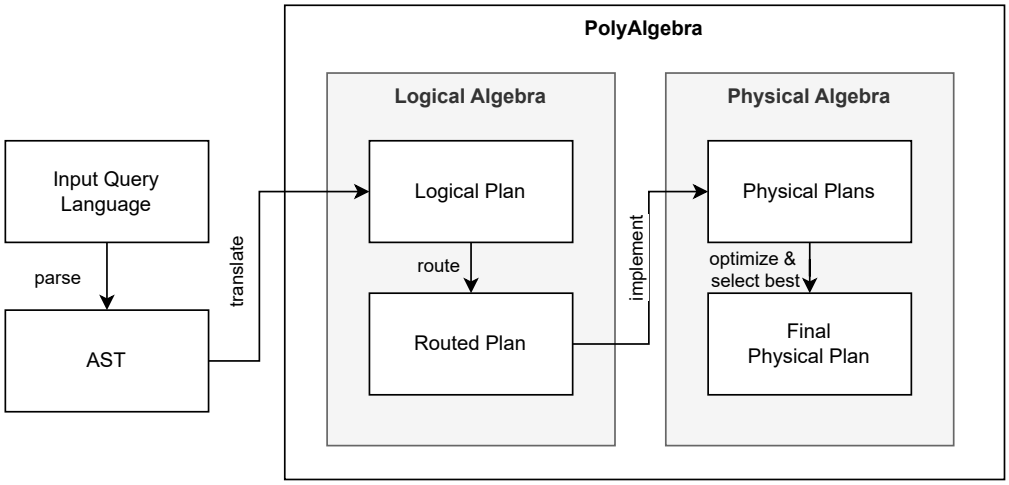


Figure 2.1: Query Processing Overview in Polypheny

Query Processing [26] Figure 2.1 provides an overview of how Polypheny processes a query. It starts with the query written in the Input Query Language (IQL), e.g. SQL, which is parsed into an abstract syntax tree (AST). Since Polypheny uses its own dialect of popular query languages, this parsing is implemented in JavaCC[7]. Next, the AST is transformed into a Logical Query Plan (LQP), which is based on the logical algebra. The logical algebra is an AST that is based on an extended relational algebra, that is able to express all query features. By routing the LQP, the Routed Plan (RP) is created, that determines which data store(s) will execute which part of the query. The RP is then implemented into multiple Physical Plans (PP), based on the physical algebra. The optimization process will select the cheapest valid physical plan according to the cost model, which is the final physical plan, which is the plan that actually gets executed.

[7] Java Compiler Compiler is a parser generator for Java. See: https://javacc.github.io/javacc/

PolyAlgebra The PolyAlgebra is a superset of the logical and physical algebra. The PolyAlgebra contains operators from all algebras (e.g. relational, document, etc.) which preserve the semantics from their original algebra. Not unifying similar operators from different data models into single operators provides several advantages:

1. The preservation of the data model-specific semantics of the operator. Correctly preserving the semantics becomes more difficult, once operators are merged.
2. Mapping operators to other operators is not difficult, which can be done for similar data models.
3. Many small & self-contained operators are much easier to extend, for example with operators from a new data model.

Query Optimization Converting the logical algebra to the physical algebra involves planning & optimization steps. Multiple candidates are created, for each possible way the logical algebra can be executed. Polypheny tries its best to *push down* operators, where it tries to do as much work as possible on the native data store, in order to reduce the amount of data that needs to be sent to the internal execution engine. However, pushing down operators can only be done, if the operation in the underlying data store supports the same semantics. Otherwise, the operation has to be executed inside Polypheny's internal execution engine, which is more expensive, due to the necessary I/O operations.

2.2 Geographic Information System

The basic idea about Geographic Information Systems (GIS) is succinctly described in [20]:

> The concept of geographic information systems (GIS) was introduced to cover all the essential needs of the scientific community to provide spatial information, analyze data, and create digital thematic maps through a computer.

GIS systems are most commonly powered by spatially-enabled databases. These databases support working with spatial types natively (e.g. querying, storing, modifying) by providing special types and functions. There are typically two types of data in GIS systems:

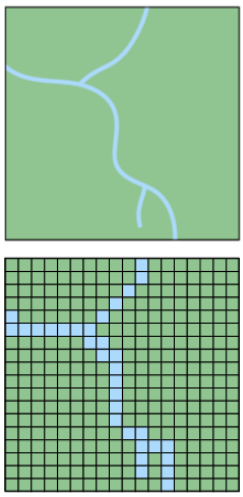


Figure 2.2: Vector data on top, raster data on bottom.[8]

Vector data This data type is comprised of nodes and edges, just like a graph. Nodes contain coordinates, for example, x and y or latitude and longitude. A single node is a point, by joining two points with an edge we can create a line, and by adding an additional point and two more edges we can form a region. Vector data typically represents real-world entities like objects (trees), places (rivers, forests), or events (birthplace). Data formats for storing vector data are discussed in subsection 2.2.4.

Raster data This data type is made up of pixels arranged in a grid, that contains additional metadata like coordinates and resolution, so it can be correctly overlayed on top of a map. Each pixel is comprised of one or several values, which can store just a single number (integer, floating point number) or multiple numbers, to represent a color, e.g. by using RGB, where one value is stored for red, green, and blue. Raster data can be discrete, where each value corresponds to a category, or continuous, where a range of values is used to allow gradual and smooth transitions.

In our work, the main focus will be on vector data, as this data format is more common and easier to work with in databases, while raster data is mostly stored as files and processed outside the database.

Another differentiation we have to make is spatial and geospatial. Spatial data talks about any data that can be spatially related to other data, such as points in a coordinate system. Geospatial data refers to data that is spatially related to the surface of the Earth. Spatial is the umbrella term, we will be using this in most places, from here on.

One important aspect of GIS is the visualization of spatial data, which typically is done by overlaying data on maps, e.g. projected 2D maps or directly on the sphere. There are several key concepts that have been established in the visualization of GIS data over the decades. Layering refers to the process of combining spatial data from different sources and displaying it on a map. This process helps us to better visually analyze the relations between the different spatial information, by putting it in relation to each other. The lowest layer is referred to as the base layer, which contains a map, like satellite imagery or a street map. The order in which layers are placed, as well as the interaction between the layers, influences how the final image looks. Symbols are used to represent land-

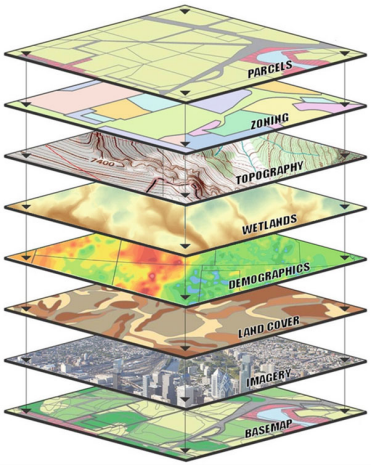


Figure 2.3: Layering example by [20]

[8] "How features are represented in a raster", available at https://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/how-features-are-represented-in-a-raster.htm, accessed January 27, 2025.

marks, like the summit of mountains or cities. Scales are used to indicate the zoom level. Legends give more information about the symbols and colors which are used.

Spatial Databases GIS workflows are supported by spatial databases (also called spatially-enabled databases). According to [13], a database needs to support the following requirements in order to be called a spatial database:

- It needs to support spatial data types in the data model and query language, by providing features for storing spatial data, and by providing functions for processing spatial data.
- It should provide spatial indexing and efficient algorithms for spatial joins.

2.2.1 Coordinate Reference Systems

When referring to a location on the Earth, we have to make sure to use a Coordinate Reference System (CRS), that contains the assumptions we have made, in order to refer to a place by using coordinates. The real shape of the earth is very irregular and complex, which is why we use a spheroid to approximate it. How this spheroid aligns with the earth is determined by the datum. A CRS describes the ellipsoid, the datum and various other parameters. Many different CRS have been created, because there is no one CRS, that can be used anywhere with a high enough accuracy.
Figure 2.4 illustrates how the different ellipsoids and their datum (relation of the elllipsoid to the real Earth) can vary drastically between different CRS, as shown by comparing WGS84 with GRS80 and Clarke 1866. The CRS WGS84, also known as World Geodetic System 1984, is the most popular CRS for encoding spatial data all over the world.

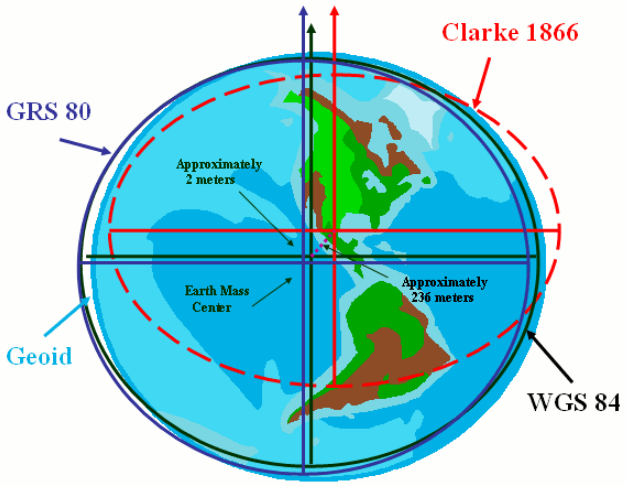


Figure 2.4: Relationship between Clarke 1866, WGS 84, GRS 80 and Geoid [9]

CRS are organized by assigning them with an EPSG[10] code, that contains all parameters

[10] European Petroleum Survey Group

for its use. By using the EPSG code, which is also known as a Spatial Reference ID, we can quickly define all assumptions we are making when specifying our coordinates. The Well-known Text representation of WGS84 is displayed in listing 2.1. In summary, it tells us which ellipsoid we use, and how it is placed (datum), that the prime meridian is Greenwich (0° longitude), that the units for the coordinates are degrees (latitude and longitude), and that it is identified by the code EPSG:4326. Independent of the specific mathematical model used, what matters is understanding which model is being applied, the level of precision it provides for a given region, and whether any conversion is required when integrating it with datasets using a different CRS.

```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
```

Listing 2.1: Well Known Text definition of WGS84

The user must always be aware of the CRS used when working with a dataset. Different datasets must be converted to use the same CRS, before making any kind of queries or comparisons.

2.2.2 Topological Relations

When working with spatial data, it is often useful to know what the topological relation between the data is. We can gain additional insights, by asking questions like: *How many boreholes have been drilled in the canton Aargau last year?*. Another example could be the overlapping of two shapes, a municipality and a protected nature reserve where the overlapping area has to be handled accordingly.

The relationship between two spatial regions can be described in nine different ways, as shown by [9]. The Dimensionally Extended 9-Intersection Model (DE-9IM), formally introduced in [15], builds on top of that idea by creating a framework that categories shapes into interior, exterior and boundary, and then allows the calculation of their relationship using an intersection matrix approach.

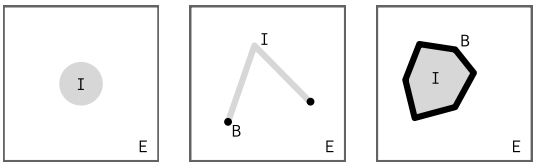


Figure 2.5: Interior `I`, boundary `B` and exterior `E` in DE-9IM.

Figure 2.5 shows how the interior, boundary, and exterior are defined for points, lines, and

regions (polygons). The interior is shown using a gray color, the boundary is shown in black, and the exterior is shown in white. For the point, the boundary is the empty set. For the line, the ends are the boundary, while the parts between are the interior.

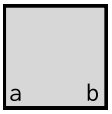
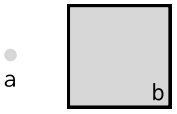
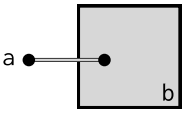
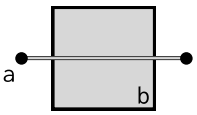
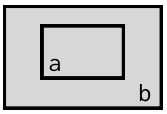
For each of the nine relation types (e.g., interior $\longleftrightarrow$ interior, interior $\longleftrightarrow$ boundary) we can calculate the dimensionality, which can be 0 (point), 1 (line), 2 (region) or F (no intersection). Table 2.1[11] shows the results of the intersection matrix for a line that intersects with a rectangle.

	I	**B**	**E**
I	1	0	1
B	0	F	0
E	2	1	2

Table 2.1: DE-9IM intersection matrix

The DE-9IM classifies nine common relationships based on the results of the intersection matrix, which are displayed in table 2.2.

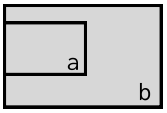
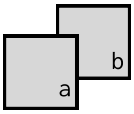
Table 2.2: DE-9IM Operators

Example	**Name**	**Description**
	`equals(a,b)`	`a` and `b` are equal.
	`disjoint(a,b)`	`a` and `b` do not have any points in common.
	`intersects(a,b)`	`a` and `b` have at least one point in common. Not disjoint.
	`touches(a,b)`	`a` and `b` have a common boundary, but no overlap.
	`crosses(a,b)`	`a` and `b` intersect, but are of different dimensions, like a line or polygon.
	`within(a,b)`	`a` is entirely inside `b`.

Continued on next page

[11] Example taken from: https://postgis.net/workshops/postgis-intro/de9im.html

Table 2.2 – continued

Example	Operation	Description
	`contains(a,b)`	`a` is inside `b`, but they share a boundary.
	`overlaps(a,b)`	Area of `a` overlaps area of `b`.

2.2.3 Metric Relations

Geometries can be put into relation by measuring the distance or direction between them. This helps us to answer questions like *What are the closest boreholes to my construction site?* We will further focus on measuring the distance because measuring the direction is not widely supported in most spatially-enabled databases. When calculating the distance, a trade-off between performance and accuracy has to be made by choosing how the distance should be calculated. According to [10], all methods provide reasonable accuracy below 10km.

Planar distance This type of distance calculation assumes a flat Cartesian plane. This calculation should only be used for short distances because it will get less accurate over longer distances because it ignores the curvature of the Earth. It can also be used when performance is more important than precision. The distance between two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is calculated using the Euclidean distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.1}$$

Spherical distance This type of distance calculation assumes that the earth is a perfect sphere. According to [10], the spherical distance can be off by about 21km for very long distances. The distance between two points $P_1(\phi_1, \lambda_1)$ and $P_2(\phi_2, \lambda_2)$, where ϕ is the latitude, and λ is the longitude, can be calculated using the Haversine formula [11]:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{2.2}$$

The radius of the sphere r is approximately the Earth's radius, at $6317km$.

Geodesic distance This is the most accurate and most complex way to calculate the distance between two points. It accurately tries to approximate the shape of the earth, which is an ellipsoid. The most common algorithms to do this are Vincenty's Formulae

and Karney's algorithms (implemented in GeographicLib [12]). According to [10], Karney's method is the slowest but the most accurate, with errors also as low as 7nm.

2.2.4 Data Formats

This section gives an overview of the two most-used data types for representing spatial data.

2.2.4.1 GeoJSON

GeoJSON is a data format for spatial data based on JSON[13], described in the GeoJSON specification [14]. It defines how to describe spatial data and its metadata. If not specified otherwise, all data is assumed to use the CRS WGS84 by default. There are seven supported geometry types: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `GeometryCollection`. A visualization of the GeoJSON types can be seen in Figure 2.6.

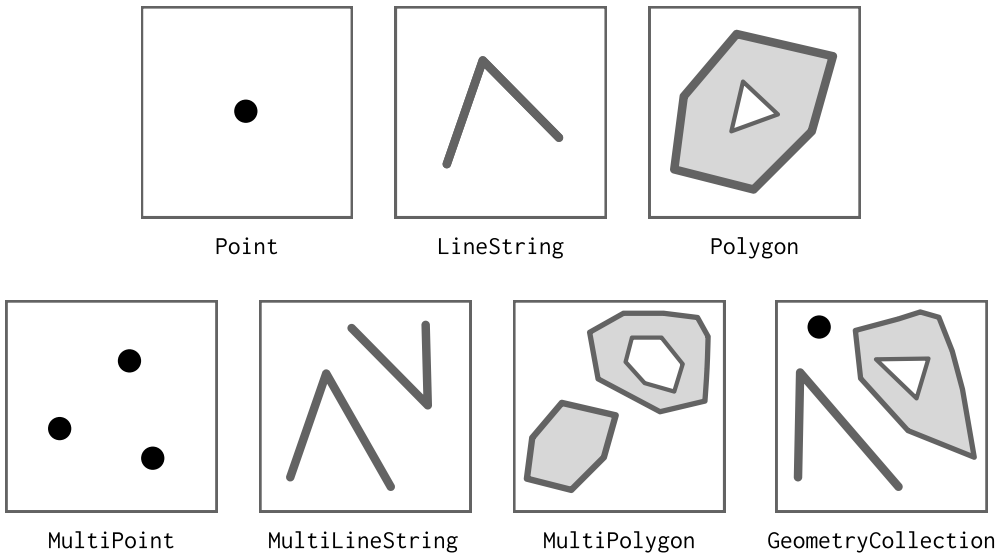


Figure 2.6: All seven GeoJSON types

Every geometry object needs two fields, `type` and `coordinates`. Depending on the type, the coordinates are interpreted differently. A single coordinate is called a Position and can have 2 or 3 values. The first two values are always longitude and latitude. The first position is undefined but can be used for height.
An example of a polygon with holes can be seen in listing 2.2. Here, the outer ring is defined first, followed by each hole. Each ring is a `LineString`.

[12] Charles Karney, "GeographicLib, Version 2.3,", available at https://geographiclib.sourceforge.io/, accessed January 22, 2025.
[13] The JavaScript Object Notation
[14] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, and Tim Schaub, "The GeoJSON Format," Internet Engineering Task Force (IETF), Standards Track, RFC 7946, available at http://www.rfc-editor.org/info/rfc7946, accessed January 22, 2025.

```
{
 "type": "Polygon",
 "coordinates": [
     [
         [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
         [100.0, 1.0], [100.0, 0.0]
     ],
     [
         [100.8, 0.8], [100.8, 0.2], [100.2, 0.2],
         [100.2, 0.8], [100.8, 0.8]
     ]
 ]
}
```

Listing 2.2: Polygon with holes in GeoJSON format

The specification describes two more types, `Feature` and `FeatureCollection`, that can be used to describe one or multiple spatially bounded things. A `Feature` has a member named `geometry`, whose value must be one of the seven geometry types defined earlier. A `FeatureCollection` is an object with a field `features` that contains a list of `Feature` objects. The original GeoJSON specification, published in 2008, included a `crs` field for setting a CRS identifier, but the field was removed in a newer version due to interoperability issues. In the most current specification, the CRS is always assumed to be WGS84. However, the usage of the `crs` field is not forbidden if all involved parties have a prior arrangement so that there is no risk of data being misinterpreted.
Other than the `crs` field, there is also the possibility of storing additional metadata under the `properties` field.
TopoJSON [15] is a popular extension of GeoJSON, which stores GeoJSON geometries with less redundancy by storing the arcs only once, and instead referring to them. This leads to decreased file sizes, especially for geometries with a lot of shared arcs, like countries or countries.

2.2.4.2 Well-known Text Representation for Geometry

The Well-known Text Representation for Geometry (WKT) format is specified in [14]. The format is able to represent the same geometry types as GeoJSON and even uses the same names. Unlike GeoJSON, it is not possible to attach a CRS or any metadata. Listing 2.3 shows what a polygon with holes would look like.

```
POLYGON (
    (100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0, 100.0 0.0),
    (100.8 0.8, 100.8 0.2, 100.2 0.2, 100.2 0.8, 100.8 0.8)
)
```

Listing 2.3: Polygon with holes in WKT format

[15] Mike Bostock and Calvin Metcalf, "The TopoJSON Format Specification," 2025, available at https://github.com/topojson/topojson-specification, accessed January 20, 2025.

The specification also allows a third dimension, which can either be a third coordinate `Z` or a measure variable `M`, used as additional value. A point with a third variable would be written as `POINT Z (30 10 5)`.
The same specification includes a binary variant known as the Well-known Binary Representation for Geometry (WKB). This is a more compact representation used to store and exchange spatial data that is not human-readable.
An extension of WKT and WKB, introduced by the PostGIS extension of PostgreSQL, introduces the ability to include a reference to a CRS [16]. In this extension, the WKT string is simply prefixed by the `SRID=#` string, where `#` corresponds to a 4-digit EPSG code, for example, `4326` to reference the WGS84 CRS.

2.2.5 Spatial Index

Spatial indexes are specialized data structures that enable efficient querying of spatial data. They are a core part of spatially-enabled databases that power the GIS workflow. Without spatial indexing, working with large datasets becomes infeasible, as every query for spatial data would require a sequential scan, which quickly becomes very costly when joining datasets.

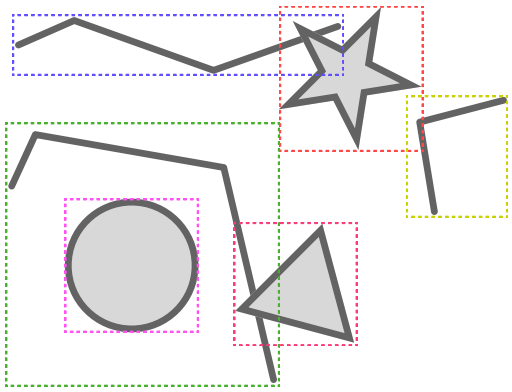


Figure 2.7: Geometries and their MBR

Just like regular indexes, spatial indexes reduce the number of comparisons we have to make. To do this, we can use several different approaches. Due to its simplicity, performance, and popularity, we will be using the R-tree data structure, as introduced by [12], as an example. In R-trees, each node represents the minimum bounding rectangle (MBR) of the contained geometries.
Leaf nodes contain the geometries that we want to index, while all other nodes in the tree group the geometries further down the tree by their MBR. An example of how geometries are grouped in the tree can be seen in 2.8. The further up we go in the tree, the more

[16] Sandro Santilli, "ZM values and SRID for Simple Features,", 2005, available at https://github.com/postgis/postgis/blob/2.1.0/doc/ZMSgeoms.txt, accessed January 20, 2025.
[17] PostGIS, "Spatial Indexing", available at http://postgis.net/workshops/postgis-intro/indexing.html, accessed January 25, 2025.

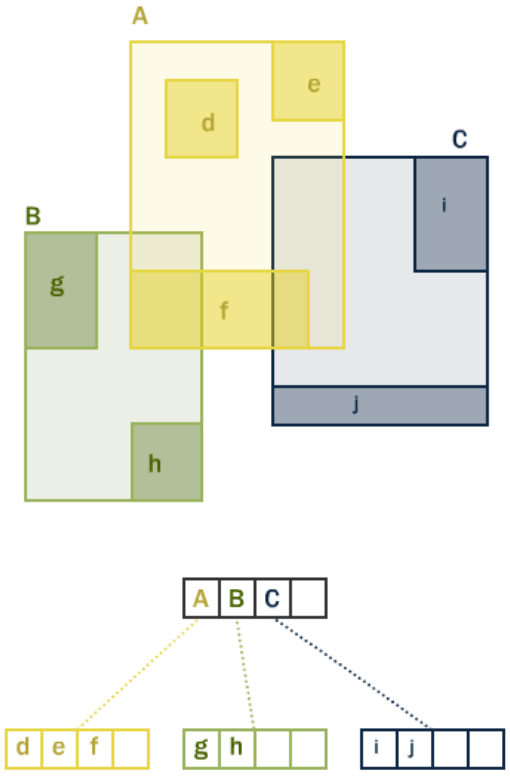


Figure 2.8: Example R-Tree hiearachy[17]

geometries are contained for each node. The tree is also balanced, which means that all leaf nodes are at the same depth. We can now search the tree by traversing its depth, which saves a lot of comparisons. Although the time complexity for searching is $O(n)$ in the worst case (for poorly distributed data in a not-properly balanced tree), this is usually not hit in practice. In practice, the performance is more similar to $O(log_M n)$, where M is the depth of the tree that scales logarithmically to the number of entries, and n is the number of geometries in the tree.

An example of various geometries and their MBR can be seen in Figure 2.7. If we want to figure out which shapes intersect with the star in the figure, we have to apply a two-step process. First, find all elements whose bounding boxes intersect with the bounding box of the star, significantly reducing the search space to a few potential candidates. In the second step, we can use the exact geometries of the shapes themselves to figure out which shape actually intersects with the star.

The data inside an R-tree is organized into pages and saved to disk. In this way, it is not necessary for the whole index to be loaded into memory to be able to use it. This is important as the R-tree itself grows with the number of geometries in the index, which is why it is beneficial to only load parts into memory when querying.

2.3 Usability Principles

Usability aims to ensure that systems, products, and services enable users to achieve their goals effectively, efficiently, and satisfactorily within their specific context of use. It is assessed through user performance and satisfaction, which depend on the conditions in which the system is used [17].

Usability is measured through quantitative and qualitative measures. Quantitative data consists of numerical values that can be measured and analyzed statistically. *Time on Task*, *Task Success Rate* or *Error Rate*, can be measured. In contrast, qualitative data is descriptive and relies on interpretation[18]. Qualitative data can be obtained by interviewing users or letting them use a system and talk about their thoughts.

2.3.1 System Usability Scale

The System Usability Scale [2] is the most widely used [23] scale for assessing the usability of systems. Since usability cannot be measured in absolute terms, the scale aims to make measuring and comparing the subjective assessment of usability across systems and contexts as easy as possible.

Table 2.3 lists the 10 questions that will be used to evaluate the final score. Questions with an odd index (1, 3, 5, 7, 9) are stated positively, while questions with an even index (2, 4, 6, 8, 10) are positive. Each question will be answered on a 5-point Likert scale, from 1 (Strongly disagree) to 5 (Strongly agree).

Index	**Question**
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

Table 2.3: System Usability Scale (SUS) Questions

The SUS score is calculated in the following way:

1. For all positive statements, subtract 1 from the response.
2. For all negative statements, subtract the user's response from 5.
3. Sum all adjusted scores and multiply the result by 2.5.

[18] National University, "What Is Qualitative vs. Quantitative Study?," 2025, available at https://www.nu.edu/blog/qualitative-vs-quantitative-study/.

The result will be a value in the range 0 (negative) - 100 (positive). As the value is meaningless on its own, we need to put it into perspective by comparing it with other systems. According to [1], each value range can be associated with a grade, which is shown in table 2.4. Extensive data collection [24] has shown that the average SUS score is approximately 68, which is why a score above this threshold is considered to be 'good'.

SUS Score	**Letter Grade**	**Interpretation**
0 - 60	F	Failing grade
60 - 70	D	Below average
70 - 80	C	Average
80 - 90	B	Above-average
90 - 100	A	Excellent

Table 2.4: SUS score interpretation

2.4 Previous Work

This implementation builds on top of the work by [22], which covers the foundation for integrating GIS capabilities into Polypheny. As part of their work, the `PolyGeometry` type was added that enables working with spatial data.
The data type supports spatial data in various data formats, such as GeoJSON or WKT. It supports different CRS, and the conversion between them, as well as various spatial functions by utilizing the Java Topological Suite (JTS)[19].
Additionally, GIS capabilities in the relational model were implemented, as well as the integration with PostgreSQL[20] with the PostGIS[21] extension.

[19] "locationtech/jts", available at https://github.com/locationtech/jts, accessed January 27, 2025.
[20] https://www.postgresql.org/
[21] https://postgis.net/

3
Related Work

Section 3.1 provides an overview of polystore systems. Section 3.2 discusses works about visualizing GIS data and querying.

3.1 Overview of Polystore Systems

BigDAWG [7] The BigDAWG polystore system is the first system to introduce the *polystore architecture* concept. Each query language and its associated data model is isolated in an *island*. Cross-island querying is supported by *shims*, that contain the logic to convert objects from one island to another, as well as a *scope*, that defines which semantics of which store should be used. The shim can also be used to map an island language to the native language of the store, thus allowing queries to be executed natively. To provide query optimization features across islands, an *equivalence dictionary* is used to map operations that share the same semantics, which allows queries to be executed on native data stores if the operator's semantics match. This is similar to Polypheny, where multiple operations exist in the PolyAlgebra that match the semantics of the underlying data store, where operations can only be pushed down if the semantics match. However, it is not clear if spatial functions are supported.

CloudMdSQL [19] The CloudMdsQL multistore system uses a SQL-like (CloudMdSQL)language that is able to query heterogeneous data stores (relational, NoSQL). It allows the execution of sub-queries in the native language of the store as function calls. Unlike Polypheny, operations are not automatically mapped from their own query language to the underlying language but must be written in the native query language, reducing query optimization possibilities.

AWESOME Polystore [5] The AWESOME polystore supports relational, semi-structured, graph and text data. It introduces the ingestion of data using the ADIL language, but does not provide further details on how query processing works.

3.2 GIS

Spatial-Query-By-Sketch [8] Introduces the concept of Spatial-Query-By-Sketch, which allows users to query a system spatially by drawing a sketch using a touch screen and a pen. The contribution mainly covers how the sketch is simplified and how spatial data is found with constraints similar to those of the sketch. Compared to ours, this approach is more abstract and covers the case of finding geometries by how they intersect rather than by where they are. As such, the presented system is more abstract and does not provide an implementation.

Map-Like Visualization [16] Provides a classification for techniques that use cartographic maps to represent abstract data. There are two main perspectives: imitation, which tries to make a visualization map-like, and schematization, which tries to make a map more visualization-like. Instead of just focusing on maps and how to layer data on them, it also shows other techniques that focus more on the data by making the map less readable, compared to our approach, where the visualization does not influence the map itself.

GIS on Mobile Devices [3] This work presents an approach to visualize and interact with GIS queries on mobile devices. Similar to our work, it examines how the result of GIS queries can be represented and how the user can interact with the data. However, unlike the Map-Based Query Mode introduced in Polypheny, this interface is very space-constrained, as it has to fit inside a phone screen, which is why it has to provide unique solutions for navigation and interactive querying. These solutions, however, could be interesting even for desktop interfaces, as they face similar problems, such as the problem of spatial data being outside the user's view.

4
Concepts

This chapter covers the conceptual contributions necessary for the implementation of spatial data representation and spatial functions in PolyDBMS. Section 4.1 defines the relational, document, and graph data models. Section 4.2 examines how spatial data can be mapped inside the different data models. Section 4.3 analyzes the characteristics of spatial functions and their variations, which is necessary for determining operator equivalence.

4.1 Multi-Model Data Representation

A data model structures data elements and defines standardized relationships between them[22]. Different data models allow us to choose the data model whose structure and standards fit our data best. This way, we do not have to try to transform the data to fit the model; instead, we can choose the model that best fits the data. There are many different data models, each with its own advantages and disadvantages, some with more popular implementations than others. According to the database management systems ranking by DB-Engines[23], the most popular data models are relational, key-value, wide-column and graph. In this section, we will limit our discussion to the relational, document, and graph models.

4.1.1 Relational Model

A relation, as initially defined by [4], is a set of tuples $d_1, ..., d_n$, where each element d_j is a member of a domain D_j, which is a set of possible atomic values the element can take on.
An attribute is the combination of a data domain and a name, which is typically referred to as a column in DBMS. For every relation, there exists a schema that defines the structure of the tuples, through an ordered list of attributes.
A relation has a degree n (the size of the tuples), where degree 1 called is unary, 2 binary, 3 ternary, and further n-ary. There exists a schema for every relation, Initially, a domain

[22] "What is a Data Model?," available at https://cedar.princeton.edu/understanding-data/what-data-model, accessed January 22, 2025.
[23] "DB-Engines Ranking," available at https://db-engines.com/en/ranking, accessed January 22, 2025.

D was restricted to atomic values; however, this requirement is relaxed in many modern databases, that allow unstructured or even semi-structured data as value.
When talking about databases, a relation is referred to as a table, where the tuples are the rows, and the attributes are the columns.

4.1.2 Document Model

The document model, as described in [18] is a data model best suited for storing semi-structured data. It consists of collections, which in turn are made up of documents. It follows a flexible schema model, which means that documents in a collection are not required to have the same fields. If two documents in a collection have the same field, they are not required to use the same data type. This allows data to be inserted without constraints on field names or data types, as no predefined schema is enforced by default.

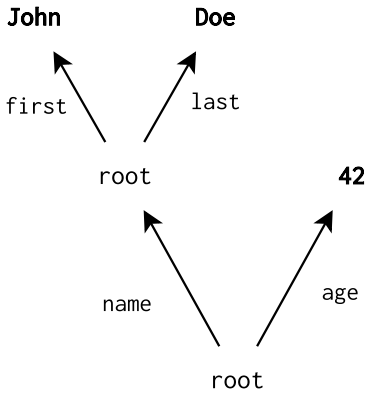


Figure 4.1: Example representation of a document

Document Documents in the document model can be represented by a tree, as shown in the example in Figure 4.1. In this example, the root of the tree is at the bottom. Values are stored in leaf nodes (depicted in bold in the figure). Nodes containing child nodes can be considered documents, which can be nested inside other documents. Values in a document are selected by specifying the edges that need to be followed inside the tree, which is also referred to as a path. For this purpose, the common notation is to use a dot. For example, the path `name.last` gives us the value `Doe`, while the path `name` gives us a nested document with two key-value pairs. The most common data storage format for storing documents is JSON, which supports several different data types, such as string, number, object, and array[24].

Required Fields Typically, the only field that is inherently required in a document is an identifier field, which is automatically generated upon creation. However, optional schema validation rules can be used to enforce constraints such as required fields or specific data

[24] "The JavaScript Object Notation (JSON) Data Interchange Format", Internet Engineering Task Force, 2014, available at https://datatracker.ietf.org/doc/html/rfc7159, abscessed January 27, 2025.

types for document collections. If a document does not conform to the defined schema, the insert or update operation fails with an error.

4.1.3 Graph Model

The graph model is useful for highly interconnected network-like data. The two most popular data models for graph databases are the Labeled Property Graph (LPG) and the Resource Description Framework (RDF). We will focus on LPG in this thesis, as it is the most commonly used and the most powerful graph model, supported by the most widely adopted graph databases.

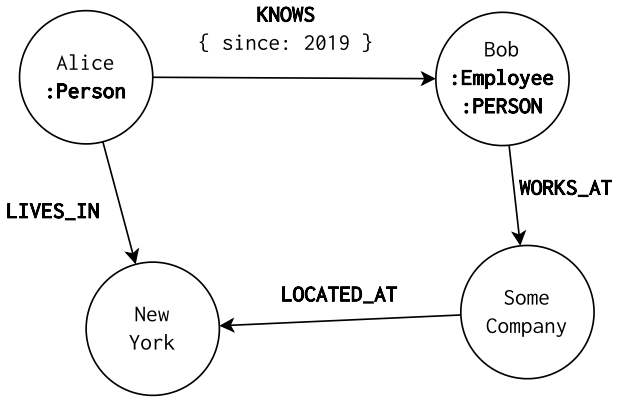


Figure 4.2: Example of an LPG

Figure 4.2 shows an example of an LPG graph. A graph consists of nodes (`Alice`, `Bob`, `NewYork`, `SomeCompany`) which can be connected to each other by edges (arrows). Nodes are generally used to describe entities, while edges are used to describe relationships between those entities. Edges in the LPG model are directed, meaning that for each edge, there exists a source and a target node (`Alice` knows `Bob`, but not the other way round). Nodes can have labels (`Person` and `Employee`), which are used to categorize them. Similarly, edges can have types (`KNOWNS`, `LIVES_IN`, `WORKS_AT`, `LOCATED_AT`), that describe the relationship. For both nodes and edges, it is possible to store additional information as key-value pairs (The relationship `KNOWN` contains the key `since` with the value `2019`). The LPG is schema-optional, which means that a schema is not required, but it is possible to enforce one.

4.2 Spatial Data Representation

In this section, we will analyze how the different shapes can be represented in the relational, document, and graph models. We look at shapes with increasing dimensionality, as these are increasingly difficult to map to the domains, especially considering the constraints.

4.2.1 0-Dimensional Data

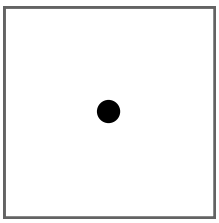


Figure 4.3: Point

The point is the simplest type of spatial data is the 0-dimensional point. As the point only consists of atomic values, it is easy to create a mapping between the point and the models. If we want to store a point on a plane, we need 2 or 3 values for x, y and z. If we want to store a point in relation to Earth, we need the additional value for the CRS, leading to a total of 3 or 4 values.

The mapping is straightforward:

- Relational $\longrightarrow$ 1 table with 3 or 4 columns.
- Document $\longrightarrow$ 1 document with 3 or 4 key-value pairs.
- Relational $\longrightarrow$ 1 node with 3 or 4 key-value pairs. As there are typically very few CRS, we could also store this information as a label.

Instead of representing the point by using native data types, we could also decide to use a WKT string. A GeoJSON object could be saved directly in the document model, while it would need to be serialized in the relational or graph model.

4.2.2 1-Dimensional Data

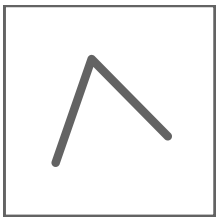


Figure 4.4: Line

Connecting multiple points together creates a line that does not have any area. The CRS can be represented by a single value, just like in the 1-dimensional case. However, in the 2-dimensional case, our line is made up of points, which can either have 2 or 3 coordinates. In a programming language, this would be represented by a nested array: one array per point with either two or three values, and one array that contains the point arrays.

Relational Model In the traditional relational model, we are only allowed to store atomic values, which excludes array and map types. To correctly represent a line with multiple points, we have to normalize it, by storing each point in the line in an extra table. Only in the case where the database has support for multi-dimensional arrays can we skip the normalization and store it as a single row.

Document Model Due to the hierarchical nature of the document, a nested array of points can easily be stored inside the document model. We can use the same structure as described in the GeoJSON specification.

Graph Model Figure 4.5 shows how we could represent a line in the graph model. Each line segment is a node that stores the coordinates as key-value pairs. The line segments are connected by edges.

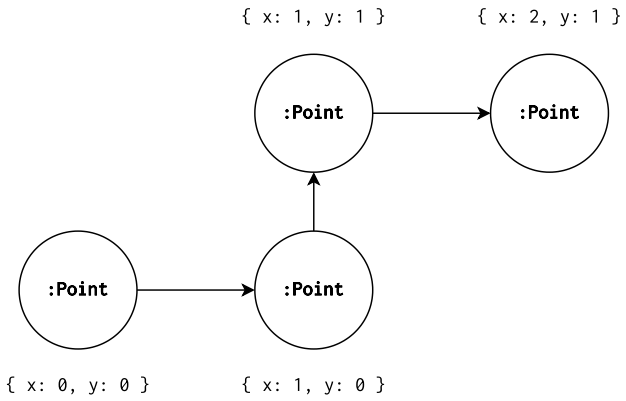


Figure 4.5: A line in the graph model

4.2.3 2-Dimensional Data

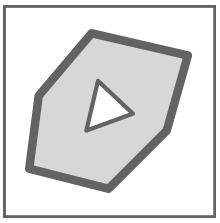


Figure 4.6: Polygon

To get from our 1-dimensional line to our 2-dimensional shape, we just need to connect the end of the line with the beginning. By doing this, we form a closed ring, where the inside and outside can be differentiated. We will further refer to this shape as a polygon. Rings that exclude an area inside the outer ring are called holes. This shape can be represented by an array, where the first element is a line that defines the outer ring, and every additional element is a line that defines a hole inside the outer ring.

Relational Model As polygons are an array of 2-dimensional shapes, we require another layer of normalization. Each polygon needs to be split into lines (each with its own table), which need to be split into points.

Document Model Polygons are natively supported as GeoJSON objects.

Graph Model We can extend our previous example, by connecting the last node in the example with the first node[25]. Holes can be represented by additional lines. The differentiation between inner and outer rings can be done using labels like `:Inner` and `:Outer`.

4.2.4 Spatial Data Formats

The previous chapter describes a theoretical mapping of spatial data into each data model. As shown, it is possible to represent shapes natively in this way; however, it is not very elegant, as single objects have to be deconstructed into multiple nodes or tables, depending on the data model. This approach would lead to performance issues, as the database would rapidly grow in size, resulting in a large number of tables or nodes. When reconstructing these shapes, joins would be necessary, which would hurt performance. Guarantees like

[25] The coordinates need to be slightly changed, in order to not generate a self-intersecting polygon, which is widely supported.

constraints and features like indexes, provided by the database, would not be necessary and would contribute to a worse performance overall. Lastly, having a different representation per data model for each shape complicates interoperability between databases. Due to these reasons, standards for storing spatial data have emerged that can be stored in the same way in all database models. GeoJSON and WKT can be serialized and stored in a single `string` / `varchar` value, making them easier to use and exchange between databases and applications. Specialized column types, such as `Geometry`, allow spatial data formats to be stored in a compressed binary format (e.g. as WKB) internally and also provide metadata to optimize spatial queries (e.g. with spatial indexes). In this sense, GeoJSON and WKT are only used as user-readable data formats intended for the user.

4.3 Spatial Functions

Now that we have defined how the representation of spatial data can be handled in different data models in the multi-model context, we can look at the spatial functions. Spatial functions give us the ability to process spatial data and gather insight from them. Processing data in the database is more efficient, as it happens very close to the storage of the data. By using aggregations and filters, the amount of data that has to be transmitted to the client is greatly reduced.
To facilitate the cross-model use of spatial functions, we will first define categories that group spatial functions based on their behavior. This way, we can better analyze the spatial function implementations to find out whether spatial functions of different data models are semantically equivalent. Determining whether two functions are semantically equivalent is crucial, as it allows the optimization process to replace one function with another in order to push the operator down to a different data store. One example is cross-model queries, where a spatial function from one data model can be substituted with its counterpart in another model to ensure execution within the data store. Equally important is the opposite case: if two spatial functions are not equivalent, the optimizer must not replace them, even if doing so would improve the execution plan, as this would compromise correctness.

Distance Measurement Type All spatial functions need to make an assumption for how distance is measured. This also applies to functions that determine topological relations, as shapes could be intersecting with one distance measurement but not with another. The possible values are: planar, spherical, and geodesic. Additionally, it is also possible that implementations determine the distance in a different, unique way.

3D-aware Most spatial functions take spatial data in a standardized format as input. For example, a WKT string supports storing a z coordinate, but not all spatial functions are implemented to use the information, even if it is provided.

Topological Relation Type Topological relations can be difficult, especially as there is no one universal standard that all databases use. Two spatial functions with the same name could have different outputs in different contexts, especially in edge cases. For example:

How are points on the boundary of a geometry treated when checking if one shape contains another shape? Does it use the `within` or the `contains` DE-9IM semantic?

Spatial Index Is a spatial index used if available, or is it even required?

Custom Functionality This category investigates other aspects of spatial functions which relate to custom behavior.

These categories offer starting points when comparing spatial functions in different data models. These categories are not exhaustive, and spatial functions could still behave differently, in some edge cases, due to bugs in the code, or an inexact definition. To assess real operator equivalence, it is important to create code to validate the behavior in different circumstances. There are some cases, where it could make sense if the user can specify if they prefer performance of absolute correctness. For example, when the user executes a cross-model query that could sped up by changing an operator for another operator, that has slightly different semantics, the user could be given a choice:

1. Use exactly the semantics as stated in the query. In this case, the operator cannot be changed with an equivalent operator, and the query has to be processed inside the PolyDBMS internally.

2. Allow the operator to be swapped with an operator that has very similar semantics. In this case, the other operator can be pushed down to be processed inside another native data store, which could potentially speed up the execution.

4.4 Interactive Visualization Framework

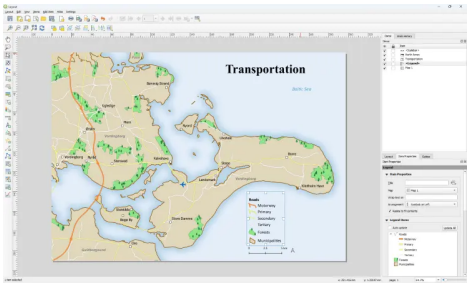
The visualization of data has become an integral part of the data analysis workflow. More data than ever is being created, as more and more devices have become *smart*, and thus contributing to the increase of heterogeneous data[26]. PolyDBMS provide a solution for managing ever-growing heterogeneous data by giving the user a single unified query interface. However, analyzing this data is still challenging, because integrating large amounts of data into specialized data visualization platforms is time-intensive and complex to set up. The Interactive Visualization Framework (IVF) presented in this section aims to solve that problem by making it easier to get started with data analysis tasks like spatial data validation and exploration of heterogeneous data stored in a PolyDBMS.
Subsection 4.4.1 lists the considerations that have been made when designing the IVF. Subsection 4.4.2 goes over the fundamentals of how data types can be represented visually. Subsection **??** goes over user interface design principles and shows a mockup that illustrates how a possible implementation could look. It also provides ideas about how to include interactive features.

[26] Transform Insights and Exploding Topics, "Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033 (in billions) [Graph]," June 7, 2024, available at https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

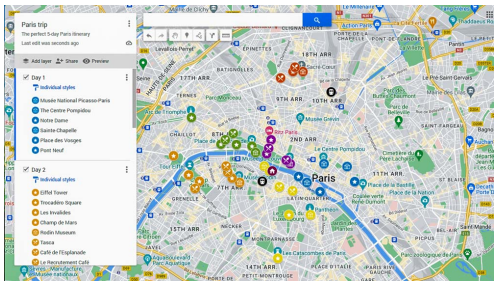
This section introduces the concept of the interactive visualization framework for geospatial data in the context of polystore database systems. The primary goal of this chapter is to create a way to better work with geospatial data when interacting with databases, which includes (a) the visualization of geospatial data, as in query results, or (b) interactively using geospatial data to improve working with queries. The next section discusses the considerations that have been considered when designing the system. After that, it describes how the different geospatial data types can be visualized. The section after that lists possible ways of interacting with the data, and the final section shows a theoretical implementation using a mockup.

4.4.1 Design Considerations

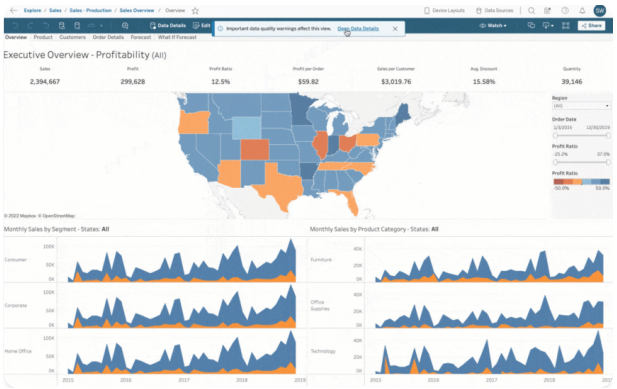
This section describes the considerations that have been considered for creating the framework.



(a) QGIS: An open-source GIS for creating, editing, analyzing, and visualizing spatial data.[27]



(b) Google Maps: An easy-to-use web-based map application providing different map types.[28]



(c) Tableau: An analytics platform for interactive data visualization and business intelligence.[29]

(d) D3: A JavaScript library for producing interactive data visualizations using SVG, HTML, and CSS.[30]

Figure 4.7: A selection of tools for visualizing (spatial) data.

[27] Image taken from https://qgis.org/
[28] Image taken from https://www.danae-explore.com/en/how-to-plan-a-trip-using-google-mymaps/
[29] Image taken from https://www.tableau.com/products/tableau
[30] Image taken from https://observablehq.com/@d3/zoomable-sunburst

Inspiration As we are not the first to tackle the problem of working with spatial data visually, we will be drawing inspirations from various sources, some displayed in Figure 4.7. The GIS field has been using cartographic maps for a long time, and as such, a wide array of tools exist for working with maps and data. As GIS applications are made for exports, they are often difficult to use without extensive training. As these tools are rather specialized, we will also be taking inspiration from more user-friendly tools, such as web-based map applications. These applications aim to provide a limited feature set that is easy to learn and use without requiring additional training. Because the fundamentals of data visualization work irrespective of being used in maps, we will also draw inspiration from data visualization and business intelligence platforms, as they often provide a middle-ground between specialized GIS software and end-user-friendly web apps.

Target Users The primary target audience for this framework comprises technical users who have some familiarity with databases and programming languages. However, they are not expected to have advanced GIS knowledge. While users of GIS software often have advanced requirements and are familiar with existing visualization solutions, the IVF can still offer some value by supporting their workflow in a different way.

Use Cases The frameworks aims to make these two primary use cases as easy to possible:

- **Data Validation:** Errors in spatial data are not easily spotted. A minuscule difference in a coordinate (e.g., latitude) may have a large impact on its location on Earth. Other issues like overlapping shapes, misaligned borders or missing data points are easy to spot visually. To enable this use case, it should not be too hard to go from dataset or query to visualization.
- **Data Exploration:** Examining individual datasets is particularly useful for validating data quality and identifying anomalies. A unique advantage of spatial data lies in its ability to be combined through layering. Unlike traditional database joins, which require predefined conditions to relate datasets, spatial layering allows users to visually overlay data from various heterogeneous sources without prior consideration of their interactions. This flexibility enables exploratory analysis, facilitating the discovery of patterns and relationships even in the absence of a predefined analytical goal.

Requirements The framework emphasizes usability and clarity as primary design principles, prioritizing intuitive interaction over extensive feature sets or advanced analytical functionalities:

- **Easy-to-Use:** One of the framework's main features is its ease of use. No extensive training should be necessary for users to be able to use the system. The user interface should ensure discoverability by implementing established UI conventions and utilizing a user-centric design.
- **Simplicity over Feature Completeness:** As this framework is designed primarily for data validation and initial data exploration, it does not require the comprehensive

feature set typically associated with GIS applications. Given its integration within a database web interface, the framework is not intended to produce finalized outputs, but rather provide users with tools to improve their experience in working with spatial data. Instead, the framework can support users in identifying relevant starting points, allowing them to export the necessary data for further analysis using more specialized and powerful software tools.

- **Optional:** If something can be expressed just as clearly or even more clearly in words, visualization in necessary [21]. The visualization should be used in a supporting way, so it should not replace the existing information, only enhance it. The visualization should never be the only way to see the information, and
- **Utilize the Multi-Model Context:** As the framework is conceptualized as part of a broader PolyDBMS web interface, it can utilize information about the data models. This kind of information is only available in the multi-model context, and as such, how it can be used should be explored.

4.4.2 Visualizing Geospatial Data Types

This subsection explores the different ways in which spatial data can be visualized. The GeoJSON specification lists three foundational[31]data types: point, line, and polygon (or, more general: area). [21] mentions a fourth type: three-dimensional symbols. To keep the complexity low and ease of use high, a true 3D map is out of scope. However, an axonometric[32]picture can still appear as 3D on a 2D map, which is often referred to as 2.5D.

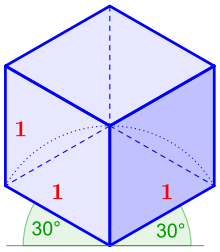


Figure 4.8: Isometric Cube.[33]

Categorization of Data [21] Before we define what kind of attributes we can use for each shape, we will look at how we can use that attribute to convey information. Not every visual data attribute has the same ability to convey information. The geometries that will be shown on the map have characteristics called *variables*. The possible values of variables form a value series, which is called a *scale*. Table 4.1 lists the four kinds of scales for a variable.
We will now list the dimensions through which the appearance of the point, line, and area geometries can be changed:

Point As the point is the only geometry that does not have a predefined *shape*, it only refers to a specific location on a map / in space. We can use whatever shape we want,

[31] The other four types mentioned in the specification are derived from these fundamental components through combinations.
[32] An axonometric projection is a type of projection in which an object is rotated around one or more axes to show multiple sides. An isometric projection is a specific type of axonometric projection where the three axes appear equal.
[33] https://commons.wikimedia.org/wiki/File:01-W%C3%BCrfel_isometrisch.svg

Type	Scale	Description
Quantitative	**Nominal**	Divides items into separate classes that are in no relation to each other, except that they are separate. Example: Nationality, boolean (broken ⇔ not broken)
	Ordinal	Like the nominal scale, but the classes have a rank order. The distance between the classes is undefined. Example: Likert scale
Qualitative	**Interval**	Consists of individual items that have an order and a distance between them. Can use a linear, logarithmic, or other scale. Example: Temperature, year
	Ratio	Like the interval scale, but there is a non-arbitrary zero value. Values can be shown in percentages. Example: Physical quantities (length, weight), GDP

Table 4.1: Four kinds of scales

like a circle, rectangle, triangle, cross, star, etc. Some shapes, for example, triangles, can be *rotated* to create new shapes; other shapes cannot or should not because the difference would be hard to spot. The more shapes we use, the harder it is to differentiate between them, which is why shapes should only be used for encoding nominal data. We can also adjust the *size* of the point freely. Changing the point size should only be done if the exact location of the point is not important, which is usually based on context. The size attribute can be used to represent interval data,

Line With lines, we can adjust their *thickness* and their *style* (solid, dashed, dotted, curly, ...). Too many different styles can be difficult to distinguish, which is why this attribute should only be used for nominal data. The thickness can only be increased up to a certain point before the accuracy of the shape suffers, which is why we should not rely on this attribute for encoding interval data.

Area Areas can be thought of as lines that are connected in a way to create an area. As such, line thickness and line style can be applied here as well. The area can be styled by its *texture*, which can take on many different values, for example solid, striped, dotted, pattern, and image. Like most attributes that change details, this should be reserved for nominal data.

Color and *Shading* can be used for all geometry types. Color is arguably the most useful attribute for conveying interval data, as human vision is very sensible to incremental color changes. As not all humans see color in the same way or are even able to see the same colors, special care has to be used when designing what colors should be used. Varying a single color only by its brightness can be perceived even for color-blind individuals. *Labels* can be used to show additional information, like a ranking or a
All of the previous attributes can be freely combined in order to see patterns in the data. The more data and layers that will be visible, the more attributes are required. For example, using different color scales for multiple layers can quickly be confusing. This section listed

the visual tools that can be used to visualize geometries on a map and what kind of data each attribute is able to represent.
The next section builds on these tools to allow user interaction.

4.4.3 Interactivity in the UI

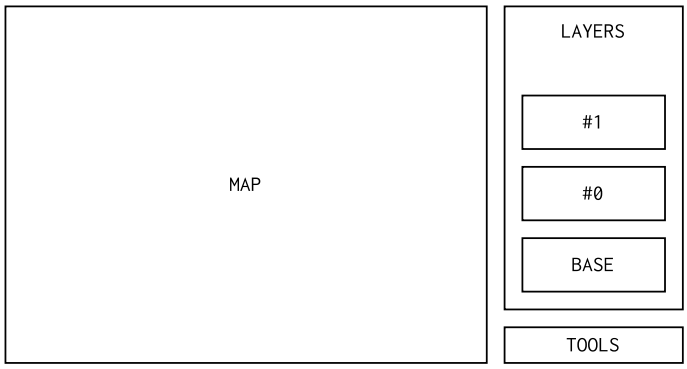


Figure 4.9: Mockup

As the interaction and configuration should not get in the way of actually seeing and analyzing the spatial data, a master-detail interface makes sense. As seen in figure 4.9, the left side makes up the master section, the actual data, while the right side makes up the configuration section, which shows details for each layer and other tools for managing the data.
To start an interaction, the user will run a query[34] which will return results. Each query and its results should be attached to a layer through which the visualization can be controlled. Each layer should be controlled individually, and the user should be given tools to manage the configuration between the layers, such as layer ordering or layer blending. The direct integration of the IVF into the PolyDBMS gives access to contextual information and additional metadata for each query that would otherwise not be accessible. The query context can be analyzed to give information that can be used in the layer visualization:

- Are the results sorted?
- Are the results grouped?
- Which data model is associated with the query language?
- Does the query contain spatial features, such as a polygon defined in a spatial feature?

User interaction should continue after a query is executed and its results are added to the layer, allowing for further interaction. As the whole process of working with map data is interactive and exploratory, the user should have access to tools that allow them to interact

[34] Either the query was typed in by the user, or the query was generated in the UI, for example, by selecting an entity (e.g., a table) in the database.

with the UI. For example, it should be possible to edit the query and rerun it. This allows the user to iteratively narrow down which information is important and which should be filtered, sorted, or grouped, necessary for data exploration work.

In PolyDBMS, all spatial functions are accessed through the query languages. Since manually writing spatial data in formats like GeoJSON or WKT can be cumbersome, users should be provided with tools to simplify this process. Enabling direct shape drawing on the map reduces reliance on third-party tools and minimizes manual effort.

5 Implementation

This chapter is the result of the background and concepts work. It extends Polypheny with various features that, together, improve the way users can work with spatial data in a multi-model context. Section 5.1 describes the work involved in extending Polypheny's support to the document model by integrating MongoDB, as well as to the graph model by integrating Neo4j. It also discusses the changes necessary to unify the different models and how they can be extended to improve the user experience further. Section 5.2 focuses on extending the web interface of Polypheny, providing the user with a query mode that focuses on working with spatial data.

5.1 Unified Model for Spatial Data in Polypheny

Section 5.1.1 describes the spatial functionalities of the MongoDB and Neo4j databases. Section 5.1.2 states how these models are integrated into Polypheny to create a unified model. Section 5.1.3 is about extending the capabilities of the models beyond the capabilities of the native data stores.

5.1.1 Native Store Functionality

In order to create a unified model for working with spatial data, we first need to create an overview of the supported data representations and spatial function support. This is necessary to integrate the internal spatial data representation with the representation used by the data stores. To do this, we must first understand how spatial data representation and spatial functions are implemented in MongoDB and Neo4j. Once we are familiar with the implementation details, we can integrate them into Polypheny and handle the different choices made by both models accordingly. Section 5.1.1.1 covers the spatial features of Neo4j, and Section 5.1.1.2 covers the spatial features of MongoDB.

5.1.1.1 Neo4j

Compared to other spatially-enabled databases, the spatial features provided by Neo4j are fairly limited. The only spatial data type that can be natively represented is the point type

with two or three coordinates. Points can be created by using the `point({ ... })` function that takes in a map that can contain the fields `x`, `y`, `z`, `latitude`, `longitude` and `height`. Depending on which fields are supplied a `Point` object will be created with a different CRS value, stored in the `crs` field. If the supplied fields are invalid, for example, by mixing `x` and `latitude`, the returned point will be `NULL`.
Points can be created in four ways. Depending on how a point is created, it will store a different CRS value, which will influence the behavior of spatial functions. Table 5.1 gives an overview.

Code	**Assigned CRS**
`point({ x: 1, y: 2 })`	7203
`point({ x: 1, y: 2, z: 3 })`	9157
`point({ latitude: 47.5, longitude: 7.5 })`	4326
`point({ latitude: 47.5, longitude: 7.5, height: 100 })`	4979

Table 5.1: Creating a point using the the Cypher `point` function.

While the assigned CRS for 4326 and 4979 reference their real-world counterparts (as assigned by the EPSG), 7203 as well as 9157 are used as placeholders and have nothing to do with the CRS known under the names EPSG:7203[35] or EPSG:9157[36].
Neo4j provides two spatial functions that can be used with these `Point` objects. The `point.distance(p1, p2)` function takes the points `p1` and `p2` as arguments, calculates the distance between them, and returns the result as a float. Depending on the CRS of the points, a different distance measurement will be used. If the CRS of the points do not match, `NULL` will be returned. For 7203 and 9157, the planar distance will be calculated with two or three dimensions, respectively. For 4326, the spherical distance is calculated. For 4979, a spherical distance is calculated, which considers the height, by calculating the Euclidean distance for the height difference. One important detail that differentiates Neo4j from other spatially-enabled databases is a different approximation for the circumference of the Earth: in Neo4j, this is chosen to be 6378km, instead of the more common 6371km, as used in MongoDB, PostgreSQL, as well as internally in Polypheny.
The other spatial function is `point.withinBBox(p1, lowerLeft, upperRight)`. This function takes three `Point` objects as input and returns a boolean as a result that indicates whether `p1` is inside the bounding box specified by `lowerLeft` and `upperRight`. Unlike `point.distance`, there are only minimal differences between the different CRS. A simple number comparison is made for 7023, 9157, and 4326 to check if the coordinates are inside the bounds. This does not consider the Earth's shape, which is why we will classify it as planar. The function does not contain any handling for the International Date Line, where the longitude flips from +180° to -180°, nor any handling for the converging longitudes as the poles.
Neo4j also supports spatial indexes [37], which improve the performance for queries using the

[35] The EPSG:7203 is used in the Jackson county in Indiana, USA. See https://epsg.io/7203
[36] The ESSG:9157 covers the country Angola and parts of the shore. See https://epsg.io/9157
[37] Neo4j, Inc., "Spatial values and indexes," 2025, available at https://neo4j.com/docs/cypher-manual/current/values-and-types/spatial/#spatial-values-point-index, accessed January 21, 2025.

`point.distance` and `point.withinBBox` functions. Multidimensional points are first mapped to a single dimension by using a space-filling curve approach. The resulting values are then stored in a generalized B+Tree. This approach works for 2D and 3D points.
Neo4j allows users to create their own functions and procedures that can be called from Cypher. The library Neo4j Spatial [38] adds many spatial features, such as support for more spatial data types by storing WKT and WKB, extended CRS support, more spatial functions for computing topological relations like within, cover, or overlap, and file imports. While the library provides a lot of functionality that is otherwise not present in Neo4j, it also has a unique way of interacting with the database. Instead of directly adding spatial data to the database, they must be added to layers. Each layer has its own index for querying and supports different features. This makes the library harder to use in a multi-model context, as the layers concept is not trivially translatable to other spatially-enabled database systems.

5.1.1.2 MongoDB

MongoDB can represent spatial data in two different ways. Just like in Neo4j, the semantics of the spatial functions are dependent on the format the data is stored in, which will be covered in more detail later in this section. The first one is called *Legacy Coordinates*, and as the name suggest, is the legacy approach. Here, two coordinates are stored in an array, where the longitude comes first, the latitude second. GeoJSON support for `Point`, `LineString`, and `Polygon` types was added in version 2.4 in 2013 [39]. Support for the rest of the spatial types defined in GeoJSON was added in later versions. Table 5.2 gives an overview of the functions with are available in MQL.

Function	**Return Type**	**Description**
`$geoWithin`	boolean	Checks if a geometry is covered by the geometry filter.
`$geoIntersects`	boolean	Checks if a geometry intersects with the geometry filter.
`$near`	ordered documents	Like `$geoWithin`, but returns a collection of documents, ordered from nearest to farthest.
`$nearSphere`	ordered documents	Like `$near`, but uses a spherical distance measurement with Legacy Coordinates.

Table 5.2: Overview of spatial functions available in MQL

The `$geoWithin` function can be used in two different ways: Either by specifying a GeoJSON `Polygon` or `MultiPolygon` under a `$geometry`, or by using one of the following shape operators: `$box`, `$polygon`, `$center` or `$centerSphere`. When using the shape operators, the function will use a planar distance measure, when using a GeoJSON object as the filter, a spherical distance measurement will be used. An example for can be seen in listing 5.1.

```
1 // GeoJSON approach, will use spherical geometry
```

[38] Craig Taverner and Andreas Berger, "Neo4j Spatial," 2025, available at https://github.com/neo4j-contrib/spatial, accessed January 21, 2025.
[39] MongoDB, Inc., "Release Notes for MongoDB 2.4," 2013, available at https://www.mongodb.com/docs/v6.2/release-notes/2.4/, accessed January 21, 2025.

```
// Note: The $geometry operator cannot be used on Legacy Coordinates.
db.boreholes.find({
   location: {
      $geoWithin: {
         $geometry: {
            type: "Polygon",
            coordinates: [
               [
                  [6.0, 47.2], [6.3, 47.3], [6.5, 47.0], [6.3, 46.8],
                      [6.0, 47.2]
               ]
            ]
         }
      }
   }
});

// Shape operator will use planar geometry
db.boreholes.find({
   location: {
      $geoWithin: {
         $center: [
            [6.2, 47.1],
            0.05  // ~ 5km
         ]
      }
   }
});
```

Listing 5.1: Example of `$geoWithin` function used in a query

The `$geoIntersects` works in a similar way to the `$geoWithin` function, but with the topological intersects semantics and without the ability to use shape operators. As only GeoJSON objects can be queried, it is only possible to define a GeoJSON filter using `$geometry`, and the spherical distance measurement will always be used.

The `$near` and `$nearSphere` functions have the same cover semantics as `$geoWithin`, but documents will be returned sorted from nearest to farthest. The calculated distance is not included in the results. Both functions require a spatial index. `$nearSphere` will use a spherical distance measurement, even if the data it is called on is using the Legacy Coordinates format. The point for comparing the distance against can be defined as a GeoJSON object under the `$geometry` field or using Legacy Coordinates with the `$near` field. When using `$geometry`, we can also specify the options `$minDistance` and `$maxDistance` in meters. When using Legacy Coordinates, we can only use `$maxDistance`, which has to be specified in radians.

Another way to query with MQL is by aggregation pipeline. Here, the query is split into several stages, where an operation is performed on the input documents in each stage. The

operation is performed, and the result is sent to the next stage as input documents. The `$geoNear` is a spatial function that can only be used as a stage inside an aggregation pipeline. The extensive options are listed in Table 5.3.

Field	Description
distanceField	Field, where the calculated distance is stored in the output document.
distanceMultiplier	Optional. Applied to the calculated distance, which can be used to convert radians to meters.
includeLocs	Optional. Specify the location field if the document contains multiple location fields.
key	Optional. Specify which index should be used if the are `2d` as well as `2dsphere` indexes available.
maxDistance	Optional. If used with GeoJSON data, the distance is specified in meters; otherwise, it is specified in radians.
minDistance	Optional. If used with GeoJSON data, the distance is specified in meters; otherwise, it is specified in radians.
near	A GeoJSON object or Legacy Coordinates, depending on which distance measurement should be used.
query	Optional. Limits the results to the documents that match the query.
spherical	Optional. Use `$nearSphere` semantics if true, otherwise, use `$near` semantics.

Table 5.3: Possible options of the `$geoNear` stage.

All spatial functions can utilize a spatial index, which is either a `2d` index for planar distance measurements or a `2dsphere` index for spherical distance measurements. Which index is used will depend on the data format, as well as how a function is called. For example, using a query with a `$geometry` argument always uses the spherical distance measurement with the `2dsphere` index, but using a `$box` or other shape operator will use planar distance measurements with a `2d` index. Only the functions `$near`, `$nearSphere` and the aggregation pipeline stage `$geoNear` require a spatial index.

Just like in Neo4j, there is no real support for CRS. GeoJSON has the implicit CRS 4326, for which spherical distance measurements will be used. Legacy Coordinates do not have an associated CRS and will use planar distance measurements.

The only CRS that can be used inside MongoDB that has an effect on the query is the custom CRS `urn:x-mongodb:crs:strictwinding:EPSG:4326`, which is used to solve an edge case where polygons greater than a single hemisphere are used[40]. When specifying a polygon as a search filter for `$geoWithin` or `$geoIntesects`, the application needs to figure out if the left or the right side of the area should be included in the results. Depending on the winding order[41] this could be the inside or the outside of the polygon. To make this process simpler, MongoDB always uses the ”smallest of the two” polygons. In the case of a polygon the size of a hemisphere, this logic selects the wrong polygon. By specifying the custom

[40] Derick Rethans, ”MongoDB 3.0 features: Big Polygon,” 2015, available at https://derickrethans.nl/big-polygon.html, accessed January 20, 2025.

[41] A polygon is created by following the coordinates. If the winding order is anti-clockwise, it is assumed that the inside of the shape is to the left of the specified coordinates. However, this constraint is not enforced

CRS, MongoDB is told to use the area on the left side in the direction of the points, which is called strict winding. By selecting this option, we remove any ambiguities and can correctly use even the biggest polygons for querying.

5.1.2 Unified Data Model

Each data model implementation supports different features, represents the same data in different ways, and uses different approximations. This section is about the changes that were implemented in order to make the data models work together.

Type Conversions through Adapters MongoDB expects spatial data to be in the GeoJSON format if it describes shapes other than points or specifies a CRS. Points without a CRS can also be stored as Legacy Coordinates. Using the wrong type can result in an error depending on the context. For example, a spatial function has different behaviors depending on whether the argument is supplied as a GeoJSON object or as Legacy Coordinates. This is why, depending on the context and the CRS, the `PolyGeometry` object needs to be either mapped to a GeoJSON document or to an array. Neo4j only supports its own proprietary `Point` type. Depending on the fields used, Polypheny will create a different `PolyGeometry` object, with a different CRS.

Different CRS Usage Both Neo4j and MongoDB use different ways of specifying the CRS, which is why a mapping is needed. Every time data is loaded from the native store, it has to go through the adapter that handles the conversion. Neo4j uses the values 7203 and 9157 not to reference the CRS but to reference the planar case with two and 3 dimensions, respectively. During the mapping to Polypheny, both values will be mapped to the SRID 0, which is used inside Polypheny. In MongoDB, Legacy Coordinates are used for point coordinates on a plane, and a GeoJSON object is used for geospatial with the SRID 4326. Legacy Coordinates will be mapped to 0, while all GeoJSON objects will be mapped to 4326 unless they store another valid SRID reference to another CRS. The internal CRS `urn:x-mongodb:crs:strictwinding:EPSG:4326` is ignored, as this is an implementation detail specific to MongoDB.

MongoDB `$near` Operator Conversion The MongoDB operators for supporting the near, `$near`, `$nearSphere`, `$geoNear`, logic are represented using a single operator in Polypheny. Only when the operator is executed internally will the operator be converted into the following four operators:

1. Projection: Add a computed distance field with a temporary variable name.
2. Filter: A filter operation will be applied if arguments specify any filters, like a minimum or maximum distance.
3. Sort: The results will be sorted by the added temporary distance field.
4. Projection: Remove the added computed distance field.

The near operator is only replaced during optimization using a converter rule. To force the optimization process to apply this rule, another rule is created that makes all plans invalid, that try to use the near operator internally. This is done to ensure that the operation can be pushed down to MongoDB, if possible. If the near operation is implemented during translation to the PolyAlgebra, it would be very difficult to figure out that these four operations should be converted to a near operation, especially considering that certain operations could be pushed down but not others.
When converting queries for execution in MongoDB, they will be converted into aggregation pipeline format, which poses additional requirements. `$near` and `$nearSphere` have to be converted to `$geoNear`, as their usage is not allowed inside a `$match` stage of the aggregation pipeline.

Distance Approximation Both the relational and the document model use the same Earth radius for their calculations. In Neo4j, a different value is used, which leads to slightly different results. In order to create a cohesive, unified data model, the decision was made to use the same distance approximation as the rest of Polypheny. If the distance approximation of Neo4j is needed, in order to be backward compatible with other processes, a second function can be specified, which correctly gets mapped to the Neo4j distance operator and uses the same calculation as Neo4j. Using a second operator allows executing the function in both systems and to compare their results.

Operator Equivalence Based on the conceptual classification of spatial functions, there are many ways in which two spatial functions from different models may differ. To make pushing down operations possible, we need to ensure that certain functions are actually equivalent, as replacing a function with another function that behaves differently could lead to unexpected results. To verify this, reference comparison tests are necessary. The same calculation must be executed internally and on all supported data stores, and their results must be compared.

Spatial Index Creation In MongoDB, the `$near` family of functions require spatial indexes. As Polypheny does not support spatial indexes yet, the user may not be used to creating them. To make the experience as user-friendly as possible, spatial indexes will be automatically created when pushing down a near operation to the MongoDB store, if required. This will have a performance impact on the first query, but otherwise, it will give the user a big performance improvement for additional queries, as well as not having to specify the error manually. The index type and field are automatically derived from the function call.

5.1.3 Extensions

The capabilities of each data model are different. The previous section is about unifying these different models to allow them to work together. This model is about extending the models beyond the scope of the native data store from which it is derived. Our work shows that Neo4j has the fewest abilities, compared to the relational and graph models,

when working with spatial data, as it provides only support for point types and two spatial functions for working these points. The storage of spatial data can simply be extended through the use of serializable data formats like WKT or GeoJSON, which could be used to store spatial data inside nodes and edges. Extending Neo4j with spatial functions is possible through extensions, as discussed in 5.1.1.1. As there is no spatial index support for these types, querying may become a bottleneck for increasing node and edge counts.
MongoDB, on the other hand, has extensive support for spatial data through GeoJSON and also supports more spatial functions. Proper support for CRS and geodesic distance measurement is only available through the relational model by using the PostGIS extension, which is beyond the scope of most database systems, including Polypheny.

5.2 Map-Based Query Mode

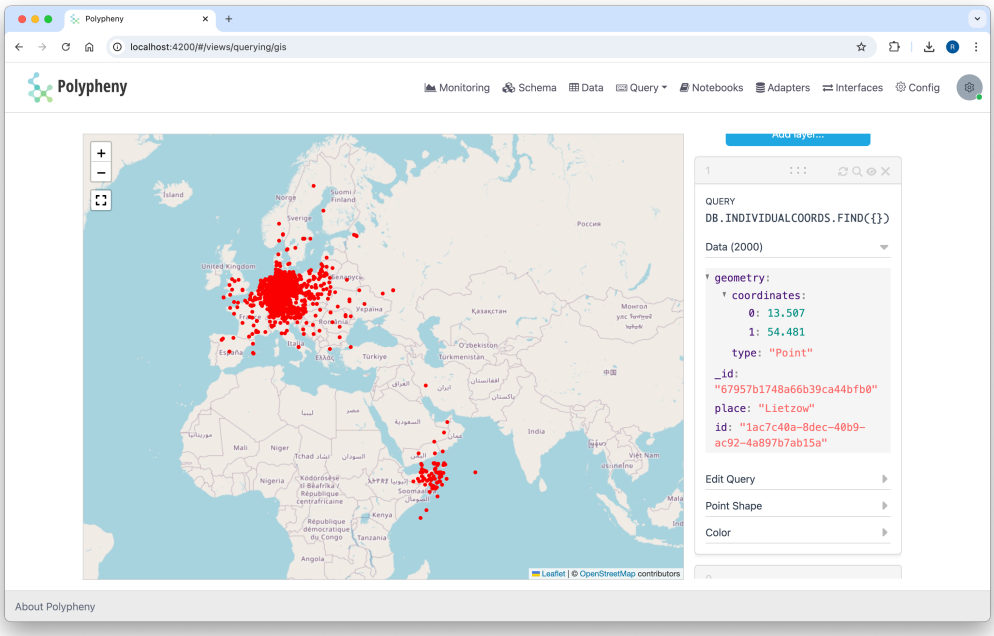


Figure 5.1: Polypheny web interface showing the map-based query mode

This section describes the map-based query mode (MBQM) implemented into the web interface, as shown in Figure 5.1. It builds upon the work of the concepts section, which explores how spatial data can be represented in the multi-model context and how the models differ from one another. It is also the result of the interactive visualization framework, which explores how to represent data visually and how to interact with it.
More importantly, the newly added query mode aims to provide useful features for working with interactive data. It aims to be easy to use, even for users who are not very familiar with query languages, by providing interactive features to refine queries iteratively.
Lastly, the query mode also serves as a tool for evaluation. Since the theoretical results are difficult to evaluate directly, they can be assessed by building a system that can be tested.

5.2.1 Overview

The web interface is an Angular 17 application that connects to the database through a WebSocket connection. Navigation is provided by the header, which contains links to the main views of the application, for example:

- **Monitoring** Shows general information, running transactions, background tasks, and various statistics.
- **Schema** Shows schema information for relational data like column types, constraints, and indexes.
- **Data** Allows the user to preview data stored in tables, document collections, or graphs.

- **Query** Shows the query console for running queries and other experimental query modes like graphical querying, a plan builder for editing the PolyAlgebra directly, and the MBQM.
- **Notebooks** Jypyter Notebook management
- **Adapters** Add adapters to connect to instances of databases like PostgreSQL, MongoDB, and Neo4j either manually or through Docker.
- **Interfaces** Adding, removing, and configuring interfaces for applications to connect to the database.
- **Config** Manage settings

The main places where the user interacts with data are the Query Console and the Data page. The user can execute queries in the Query Console, which sends them to the server. Because the query execution is implemented as a stream, the web interface has no way of knowing how many results are returned or how much data is contained in those results. This is why, by default, only the first 10 results are returned without a limit clause. The data page lets users view tables, collections, and graphs. Like in the Query Console, the data view is limited to 10 results at a time. The user can paginate to view more results.

5.2.2 Integration

When working with traditional queries, where the results are only textual, it is sufficient to see the first few results for most queries. This approach usually works because most queries use sort or group operations to make the most important data appear at the top of the results.

There is a difference when working with data visually. Here, it is important to get a total view of the data by displaying the full query results. This is especially true for the data exploration use case. Limiting the results makes sense to avoid overloading the web interface with data that could possibly be too large, either because there are too many results or the data contained in the results is too large. If the results are too large to load into the web interface, we must apply strategies to minimize the total size, such as removing large objects such as multimedia (videos, images) and binary data. The Query Console is supposed to be a pure view in the database, where data should not be discarded, which is why it is incompatible with MBQM. To work around this limitation, the MBQM needs its own view, where the data lifecylce can be better controlled. As a compromise, a static version (as shown in Figure 5.2) that just supports the result visualization is implemented into the Query Console and Data view section, which allows the user to navigate to the full MBQM, where interactive features are supported.

5.2.3 Architecture

The previous section discussed why the implementation is split into two modes: one static, one interactive. To support both use cases, the feature was implemented as two separate components:

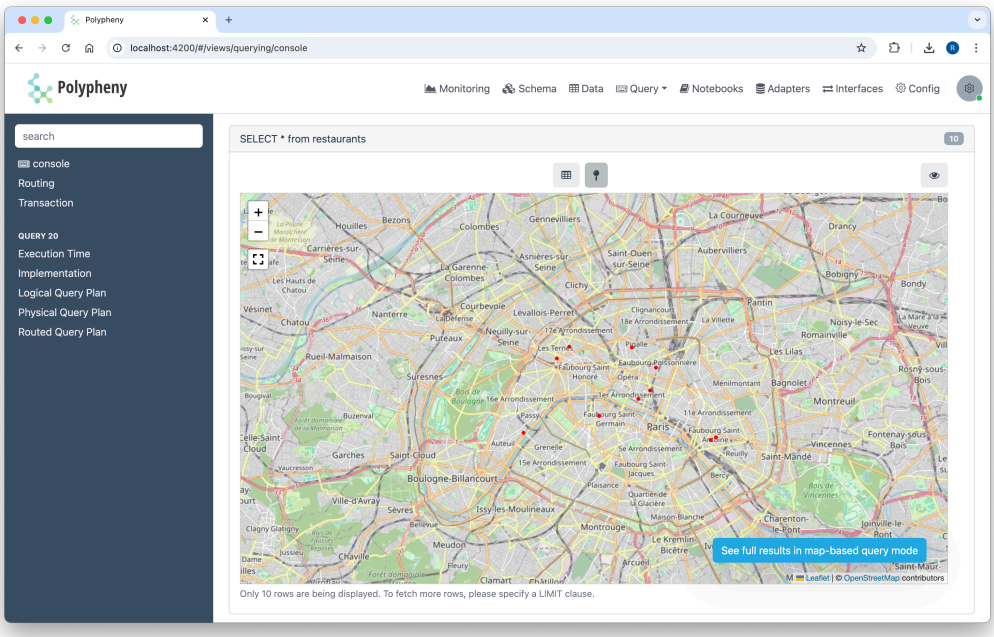


Figure 5.2: Map preview of the paginated results inside the query console

- The `data-map` component takes care of handling the representation of the results. Except for navigational features like zooming, panning or the enabling screen mode, the map is fully static.
- The `map-layers` component supports the interactive features. Here, layers can be added, modified and removed.

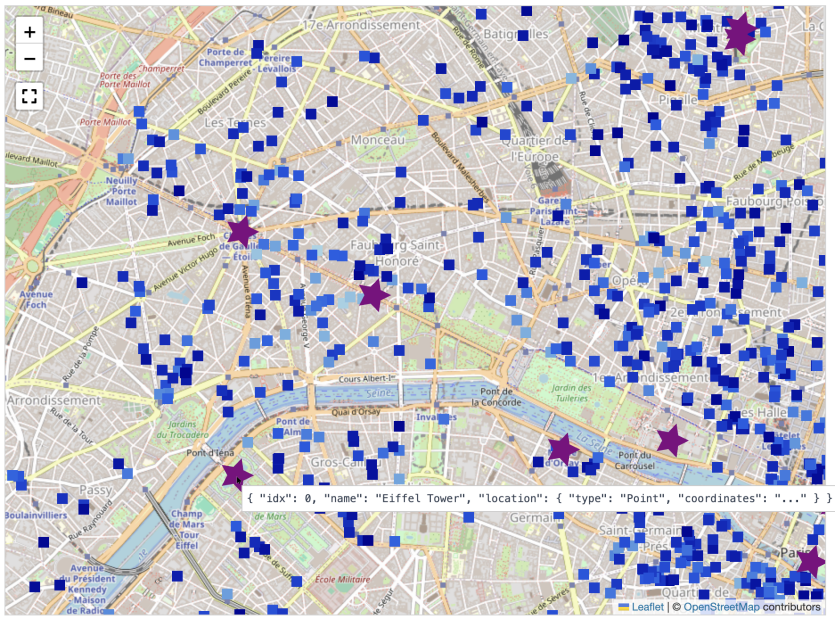


Figure 5.3: The `data-map` component

Figure 5.3 shows the `data-map` component. The mapping features are provided by Leaflet[42]. Leaflet is chosen over other open-source libraries due to its simplicity, permissive license, and popularity. It also handles loading tiles from different sources, like the free-to-use OpenStreetMap[43] tiles. It handles navigation like panning and zooming. Implementing additional features through plugins, like a full-screen mode[44] is possible.
Shapes are drawn using an SVG element that is positioned over the map. SVG elements are created by the d3[45] library. d3 is preferred over the integrated leaflet abilities for adding vector shapes due to its powerful feature set and easy-to-use API. For example, creating a tooltip (as seen in the Figure) for an SVG element or even adding other interactive behavior is easy. While SVG has an easy-to-use API and offers more interactivity, its performance is the biggest problem, especially when the number of items increases. Just a few thousand shapes can already drastically slow down the map interaction and increase render times. The alternative is the canvas element, in which the shapes are first rasterized before being displayed. This is many times faster, as the pixel canvas limits the resolution (compared to the infinite resolution of the vectors inside the SVG) to certain dimensions, and also because there is no DOM manipulation necessary.

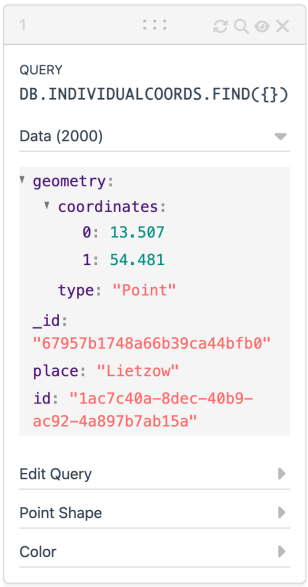


Figure 5.4: Layer Card

Layer Card Interactive features are provided by the `map-layers` component, where the user can configure the layers for display on the map. Each layer is associated with a layer card that gives the user information, control over how the data is displayed, and even interactive features for editing the query. Just like the layers are drawn on top of each other, the layer cards are displayed in a vertically-scrollable container next to the map, and can be rearranged by drag & drop. The layer card sections that will be shown only if applicable:

- **Data:** Shows the data attached to the first geometry of the results.
- **Edit Query**: Allows the user to interactively filter the query by drawing a polygon. The section also features an edit button that allows the user to edit and rerun the query.
- **Point Shape:** Edit point size and shape. Selectable shapes are: circle, square, triangle, star, and cross.
- **Area Shape:** Edit area stroke thickness and texture. In the prototype, only the solid option is implemented.
- **Color:** Allows the user to choose their own color with the Static mode or choose a

[42] https://leafletjs.com/
[43] https://www.openstreetmap.org/#map=5/-1.12/6.83
[44] https://github.com/Leaflet/Leaflet.fullscreen
[45] https://d3js.org/

color based on a variable with the Gradient mode. The user can choose if a linear or sequential scale should be used.

The layer card also provides a toolbar that allows the user to perform the following actions (in order): rerun query, zoom map to fit all shapes from layer, toggle layer visibility, and remove layer.

State Management State management is implemented by using a singleton service, which keeps the state between the `map-layers` and the `data-map` in sync by using reactive types from RxJS[46]. Each section in each layer card corresponds to a configuration object that can be dynamically attached to the layer. For each configuration object, the correct Angular component will be resolved. If any configurations change, an event is emitted, which hash all layer configurations and compares them to the configurations of the active layers. If they differ, the user will be shown a *Refresh Layers* button. This makes it possible to apply multiple changes without causing a map re-render, which could take up to a few seconds, depending on the number of data points.
As Polypheny does not have active sessions, navigating away from the query mode resets all state.

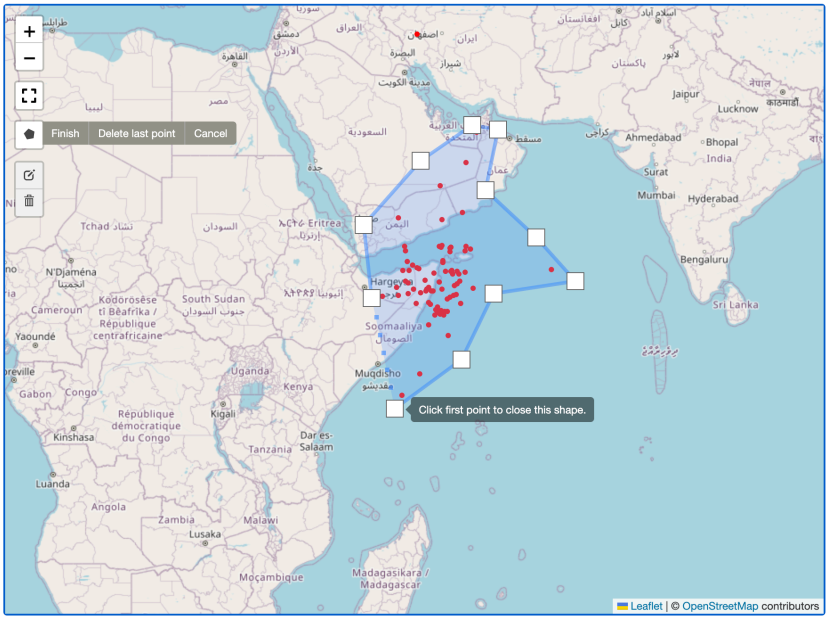


Figure 5.5: Creation of the polygon shape using the Query Filter feature.

Query Filter Figure 5.5 shows the interactive query feature, where the user can apply a spatial filter to the data by drawing a polygon shape. The UI for creating the polygon is created using the Leaflet.draw[47] library. The created polygon is converted into

[46] https://rxjs.dev/
[47] https://github.com/Leaflet/Leaflet.draw

the WKT format. The query for the layer is converted into its PolyAlgebra representation by using the Plan Builder query mode internally. This PolyAlgebra is then edited to include a spatial filter, as seen in listing 5.2[48]. In this example, the original query was `DOC_SCAN[doc.individualcoords]`.

```
1 DOC_FILTER[MQL_GEO_WITHIN(
2   geometry,
3   'SRID=4326;POLYGON (( ... ))':DOCUMENT,
4   -1:FLOAT)
5   ](
6     DOC_SCAN[doc.individualcoords]
7 )
```

Listing 5.2: Example: Apply a spatial filter using the PolyAlgebra

[48] To improve the readability, polygon coordinates were replaced by an ellipsis.

6 Evaluation

We use two types of evaluation to evaluate the contributions made in this thesis. The section 6.1 describes how we use unit tests and validate results with the reference implementation. The section 6.2 describes how we conducted the user study to evaluate the Map-Based Query Mode, which is the implementation of our concepts. Finally, we show the results in section 6.3.

6.1 Reference Comparison

To validate the correctness and performance of the implementation of the spatial features described in section 5.1, we will be using two approaches.

Unit Tests Correctness will be tested by creating unit tests. There are two types of tests. The first one is an internal test, where we store data internally[49], and then execute the spatial operation in the execution engine of Polypheny. We then compare the values with the expected output. The second type of test is a unit test, where the same queries are performed through the execution engine of Polypheny, as well as on the native data store. This query confirms that the implementation is correct by comparing the results with the reference implementation.

Performance Comparison To measure the performance of our implementation, we will be running a benchmark. The benchmark will be run on the execution engine of Polypheny and on the native store, which is deployed and integrated into Polypheny by using a docker image. A benchmarking framework will be created that can be configured to insert randomly generated spatial data and then run a query that will be measured. The benchmark function will be run 15 times to ensure accuracy, and the first five warm-up runs will be discarded. To ensure that the database cannot cache the query results, the same randomized queries will be used for both systems.

[49] The data will be stored in the in-memory database HSQLDB

6.2 User Study

This section describes how the user study was conducted. Subsection 6.2.1 outlines the research questions we aim to answer. Subsection 6.2.2 discusses the details of the study's design. Subsection 6.2.3 lists the tasks participants were expected to complete. Finally, subsection 6.2.4 provides an overview of the survey questions.

6.2.1 Goal

Conducting this user study mainly aims to evaluate two contributions, namely the Interactive Visualization Framework described in Chapter 4, as well as the implementation of the Map-Based Query Mode, as described in Chapter 5. The main objective is to answer two questions:

(1) Does the Map-Based Query Mode provide useful features when working with spatial data?

(2) Does the implementation of the Map-Based Query Mode provide a good user experience?

For Question (1), we want to find out whether the features we implemented are actually an improvement over the previous state, where the database interface did not support any features like visualizing spatial data interactive spatial queries.
Question (2) aims to find out whether the way the user interface is implemented provides a good user experience. We want to know whether users had problems when interacting with the UI and using the newly added features. A good user experience is critical for the adoption of features, which will only be used if the interaction is positive. Otherwise, the user will likely try to find another, more pleasant way to solve the same problem.

6.2.2 Study Design

The user study was conducted remotely and in an unmoderated way. The participants were sent a instruction document [50] which contained the following information:

- Background information to give context unfamiliar to participants unfamiliar with Polypheny
- A description of the contribution of this thesis, as well as the goal of the user study, so that participants know what to pay attention to.
- Usage instructions like how to access Polypheny, how to use Polypheny, and contact details to reach us if there are any problems.
- Descriptions of Tasks the participants are expected to accomplish.

In combination, the document contained all the necessary information so that the user study could be carried out remotely and unmoderated. This method was chosen to increase the

[50] The full document can be seen in appendix B.

number of participants and make the user study more flexible overall. Live support was available if users had issues that were not addressed by the document.

6.2.3 Tasks

The participants were asked to work on three tasks. Tasks 1 and 2 contain step-by-step instructions to allow participants to get to know the system and get familiar with the user interface. Task 3 did not contain step-by-step instructions but asked the participants to try to answer a question, where the participants only had a textual description of how it could be achieved. It was expected that users would be familiar with the user interface by then.

6.2.3.1 Task 1: Data Validation

The first task tests the data validation use case, where the participants had to visualize spatial data to get more information that is difficult to obtain from the raw coordinate values.

Scenario A historian collects genealogy data from various sources, which differ greatly in quality. As part of the data validation, the coordinates for birth and death places need to be analyzed further, as errors cannot be identified by just looking at the numbers. We want to get a list of all points that may be incorrect so we can analyze them further.

Step-by-Step Instructions The participant was asked to add a new layer from a query by executing a given MQL command. Then, the participant has to visually analyze the data. The suspicious data points can be checked by hovering over the point, revealing a tooltip that shows additional information, sometimes even a label for a place. After identifying the suspicious points, the participant was asked to apply a filter to a layer by drawing a polygon. This causes the data to be filtered spatially. Now that only the suspicious points are left, the participant was asked to export the layers, which downloads a GeoJSON file to the system. Done!

Used Functionality Add a layer from a query, navigate the map, use the tooltip, spatially filter by drawing a polygon, export the layers

6.2.3.2 Task 2: Pattern Finding

The second task tests the pattern finding use case, where we visualize data alongside the spatial data, to find irregularities in the data, which is otherwise hard to find.

Scenario The city of Bern has placed temperature sensors all around the city, to gather climate-related data. Because the sensors have a tendency to drift, we regularly want to check their readings. To do this, we want to compare sensors with other sensors in their vicinity. If the the sensor does not align with its neighbors, then it will probably need to be recalibrated.

Step-by-Step Instructions The participant was asked to navigate to the Data section, find the collection with the temperature data, and then click a button to get to the full Map-

Based Query Mode. Then, the participant was asked to apply the color mode Gradient and set the variable to `temp`. After refreshing the layers, the temperature can be seen by looking at the color of the points. The participant is asked to increase the point size to see the color better. By looking at the visualization, the participant should be able to identify the malfunctioning sensors and edit the query so that only these points are included. Then, the points need to be exported. Done!

Used Functionality Add a layer from the data section, navigate the map, use the tooltip, use the color mode gradient, edit a query and rerun, edit the point size, export the layers

6.2.3.3 Task 3: Data Exploration

The third task tests the data exploration use case, where we investigate the data without a specific goal. This task gives the user two questions they can try to answer.

Scenario You are a researcher and are preparing a trip to Paris to attend a conference. Because food is very important to you, you decide to analyze the TripAdvisor[51] data about restaurant reviews. You also have access to a dataset containing the polygons for the city districts and another dataset containing locations of popular tourist attractions in the city.

Questions Unlike the first two tasks, this task does not contain step-by-step instructions. It is assumed the participant is now familiar with the necessary functionality. Instead, it poses two questions:

1. Which district should we stay in?
2. Find out if expensive restaurants are also the restaurants with the best average ratings.

For Question 1, the user was given a SQL query to use inside the database that aggregates restaurants spatially by using the provided dataset about city districts. For Question 2, a description was given that described how the question could be answered.

Used Functionality Add multiple layers, navigate the map, use the full-screen mode, use the tooltip, use the color mode gradient, edit the point shape and size, visually compare two layers

6.2.4 Survey

The full form can be seen in appendix C. The first section collects information about technology proficiency, especially regarding programming and databases. The second section uses the SUS scale to ask 10 questions to gauge the general usability. The third section aims to find out whether the Map-Based Query Mode is an improvement over the previous

[51] The TripAdvisor European restaurants dataset can be accessed here: https://www.kaggle.com/datasets/stefanoleone992/tripadvisor-european-restaurants (The dataset was spatially limited to the city of Paris, for performance reasons)

state. Together, these questions aim to answer the two questions posed in the subsection 6.2.1.
Due to the low number of expected participants, the questions are mainly subjective and rely extensively on the Likert scale.
While objective measurements would technically be possible, the low number of participants makes the results hard to analyze, as individual differences between participants have a big influence on the metrics. As users were not recorded, gathering additional information like completion time would not provide any more value, as we do not know whether the user was having problems or if the user decided to spend more time on a task by choice. A fully moderated approach, where the participants would be recorded and talked through solving the task, would provide valuable information, especially for improving the Map-Based Query Mode, but this was out of the scope of this study. Even though a moderated approach would lead to higher-quality results, an unmoderated approach suffices for the purposes of this thesis. Additional studies can be conducted if the goal is to improve the Map-Based Query Mode further.

6.3 Results

This section gives an overview of the results. Section 6.3.1 presents the benchmark numbers that were gathered. Section 6.3.2 describes the outcome of the user study.

6.3.1 Reference Comparison

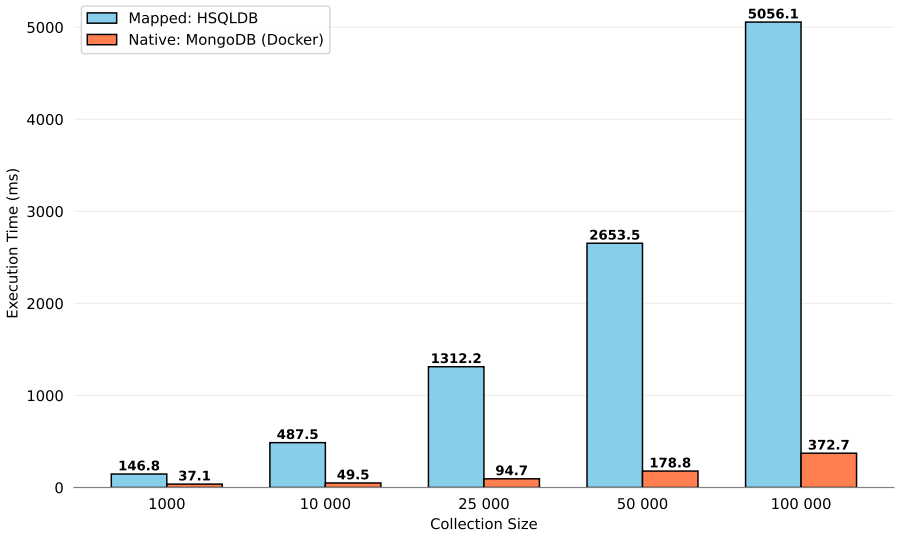


Figure 6.1: Measured execution time in milliseconds for `$geoWithin` inside HSQLDB and MongoDB for different collection sizes.

Figure 6.1 shows execution times for running `$geoWithin` with collection sizes from 1000

to 100 000 documents. For the blue bar, `$geoWithin` was executed inside the internal execution engine in Polypheny. The orange bar shows the execution for documents stored inside MongoDB, where the query is also executed inside MongoDB natively. We can see that for all collection sizes, MongoDB is multiple times than Polypheny. The increase in execution times seems linear for MongoDB, while it seems exponential for the internal data storage. The performance difference shown is here misleading, as the data stored inside the internal storage engine in Polypheny is stored in the relational model, and must first be converted to the document model.

6.3.2 User Study

During the 7 days the evaluation was accessible, a total of 10 students participated in the user study. Of those, seven have a background in Computer Science, and 3 did not. The average participant had 6.5 years of programming experience and was familiar in working with databases. Most participants, except 2, had no experience working with GIS.
Figure 6.2 shows the calculated SUS score for each individual. The average is 83, which is considered very user-friendly. The lowest given score is 67.5, which is still considered average. This could indicate that some problems still exist that not all participants have encountered.

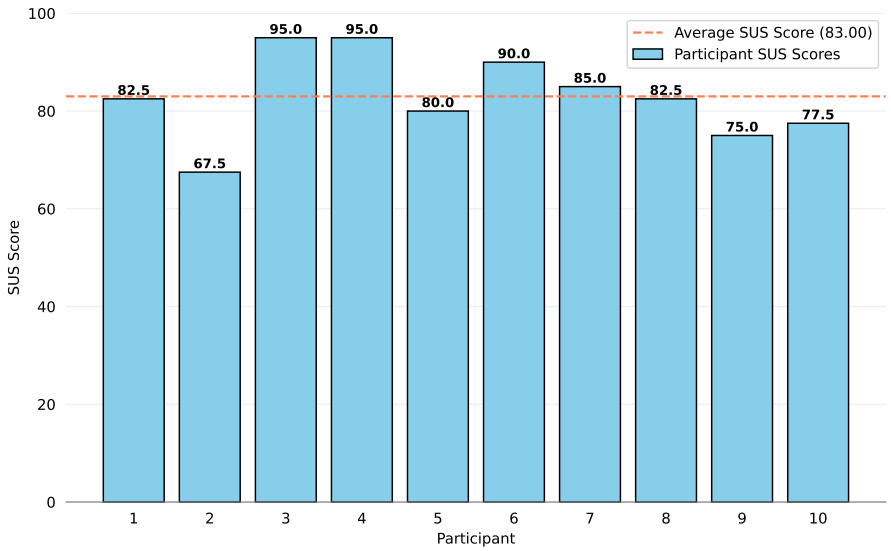


Figure 6.2: SUS score calculated for all participants

Participants were asked to rate their confidence using a Likert scale when completing each task with and without the prototype. Figure 6.3 shows the average confidence as two lines: the blue line shows the confidence with using the prototype, the orange line shows the confidence without using the prototype. On average, participants had a higher confidence with the prototype (4.16) than without (2.70), which we can see for each task.
Participants had higher confidence in solving task 2 (Pattern finding for temperature sensors)

without using the prototype compared to the other tasks. This is probably due to two reasons: (1) task 2 has the lowest amount of data points, and (2) the answer was more straightforward to find compared to the other tasks because all the points that were identified as suspicious were at the extreme end of the value distribution, which makes the aspect of comparing them to nearby sensors superfluous, and thus allows the task to be easily solvable by creating a simple query with a condition to filter out all values below a certain threshold. This could have been prevented using a dataset where the suspicious sensors were along a line, where the faulty sensors cannot be found on the extreme ends but between the other values.
We can also see that participants had the lowest confidence in solving Task 3 with the prototype, which is probably because the task does not contain any step-by-step instructions. However, the confidence rating is still higher than without the prototype.

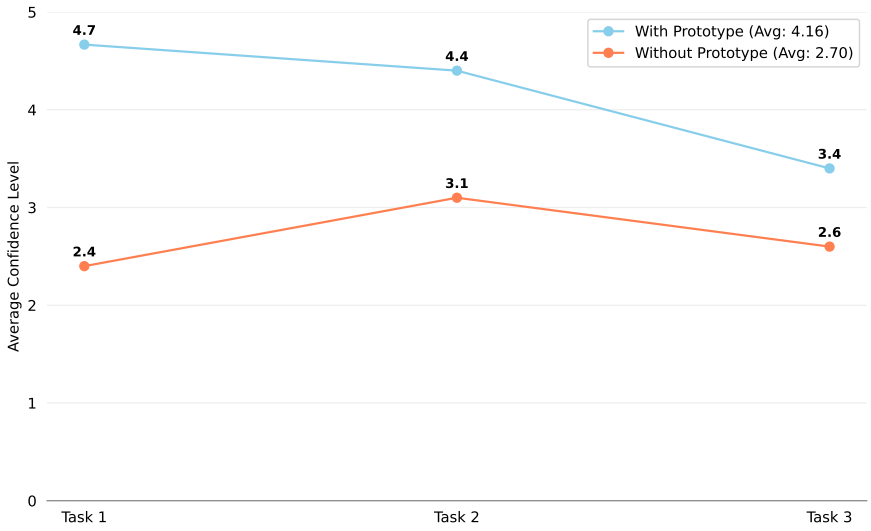


Figure 6.3: Average confidence ratings for all three tasks

The most common response to the question *Which problems did you encounter when working with the prototype?* was related to bugs when using the prototype, like the map being empty after a query is executed and the zoom button not working properly. Two problems were usability-related; one participant reported it being difficult to find the correct buttons, and another user reported slow performance when adding multiple layers to the map.
For the question *Were there any features you felt were missing?*, the most common answer was related to the gradient color visualization mode, where participants expected a legend that shows which value corresponds to which color and the option to change the colors. Other answers included no data validation for wrong inputs (e.g., nonexistent color), a dropdown to show which variables are available, the possibility of adding shapefiles, and a way to interactively add conditions to the query that are not spatial, like a filter condition.

7

Conclusions and Future Work

7.1 Conclusions

This thesis significantly advanced the spatial capabilities of the PolyDBMS model by extending it and defining multi-model spatial support. We showed how spatial data support can be integrated into the multi-model context on a conceptual level. The concept was validated by extending the document and graph model in Polypheny with spatial data and function support. The models were extended to create a unified spatial model that integrates the different models by implementing features to consolidate differences automatically. The implementation makes it possible to use Polypheny on heterogeneous datasets and brings it closer to being a spatially-enabled PolyDBMS.

The second contribution introduced the groundwork for the interactive visualization framework. The framework explores how visualizations can improve working with data, especially spatial data that can be visualized by utilizing cartographic maps.

Based on the conceptual work, the map-based query mode was implemented. It provides a simple-to-use user interface and the ability to work with spatial data in multiple data models in a more interactive and visual way. This was made possible by the previous contribution, which unified the handling of spatial data in the multi-model context. By providing interactive features, the implementation is especially helpful for users with limited technical experience, for example for users who are not very experienced in creating queries that utilize spatial functions.

The contributions of this thesis were evaluated in multiple ways. Unit tests were used to validate the correctness of the results. The correctness of the implementation was further validated by comparing the results of spatial functions with the same spatial functions from the native data stores. Benchmarks were executed, to compare the performance between the spatial functions run inside the internal execution engine and on the native data stores. Results were mixed, as the performance of the native store greatly outperformed Polypheny. A user study was conducted to evaluate the MBQM. Over one week, ten participants evaluated the implementation by performing predefined tasks and providing qualitative feedback. The responses were predominantly positive. The system received a high score on the SUS scale, indicating strong overall usability. Participants reported greater confidence in problem-solving when using the MBQM compared to the traditional query-language interface.

7.2 Future Work

While improvements could be made to bring Polypheny closer to a spatially-enabled database, there is still work to be done.

A big improvement would be the addition of spatial filters, which would significantly improve the performance and enable the user to work with large amounts of data. This performance hit is especially noticeable when working with operations that cannot be pushed down to the native store and are then executed in the internal execution engine of Polypheny.

Another improvement would be to extend the capabilities provided by the data models to provide additional spatial functions that can be used inside Polypheny. This is currently only possible by executing cross-model queries that utilize operators of another model.

The MBQM showed good results in testing, which helped users accomplish tasks with greater confidence. The MBQM lacks some quality-of-life improvements; for example, a legend that shows what color corresponds to what value and the ability to see minimum and maximum values of the scale could be added. Another addition would be configurable labels attached to the spatial data that are always visible, not only on hover. Lastly, the integration with the database could still be deepened, for example, by visualizing entire graphs with their nodes and edges.

The implementation performed well for data validation tasks. However, the data exploration aspects could still be improved. For example, by providing users with the ability to add any filter, not just spatial ones. Giving the user advanced visualization tools, like support for charts embedded on top of the map, would make the visualization more informative. Switching from SVG to Canvas-based rendering would allow the prototype to handle large amounts of data points, which is necessary when exploring large datasets.

Bibliography

[1] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[2] J Brooke. Sus: A quick and dirty usability scale. *Usability Evaluation in Industry*, 1996.

[3] Stefano Burigat and Luca Chittaro. Visualizing the results of interactive queries for geographic data on mobile devices. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems*, GIS '05, page 277–284, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931465. doi: 10.1145/1097064.1097103. URL https://doi.org/10.1145/1097064.1097103.

[4] Edgar F Codd. Further normalization of the data base relational model. *Data base systems*, 6:33–64, 1972.

[5] Subhasis Dasgupta, Kevin Coakley, and Amarnath Gupta. Analytics-driven data ingestion and derivation in the awesome polystore. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2555–2564, 2016. doi: 10.1109/BigData.2016.7840897.

[6] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The bigdawg polystore system. *SIGMOD Rec.*, 44(2):11–16, aug 2015. ISSN 0163-5808. doi: 10.1145/2814710.2814713. URL https://doi.org/10.1145/2814710.2814713.

[7] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. The bigdawg polystore system. *ACM Sigmod Record*, 44(2):11–16, 2015.

[8] MAX J. EGENHOFER. Query processing in spatial-query-by-sketch. *Journal of Visual Languages Computing*, 8(4):403–424, 1997. ISSN 1045-926X. doi: https://doi.org/10.1006/jvlc.1997.0054. URL https://www.sciencedirect.com/science/article/pii/S1045926X97900549.

[9] MAX J. EGENHOFER and ROBERT D. FRANZOSA. Point-set topological spatial relations. *International journal of geographical information systems*, 5(2):161–174, 1991. doi: 10.1080/02693799108927841. URL https://doi.org/10.1080/02693799108927841.

[10] Vissarion Fisikopoulos. Geodesic algorithms: An experimental study. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42: 45–47, 2019.

[11] Kenneth Gade. A non-singular horizontal position representation. *Journal of Navigation*, 63(3):395–417, 2010. doi: 10.1017/S0373463309990415.

[12] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984. ISSN 0163-5808. doi: 10.1145/971697.602266. URL https://doi.org/10.1145/971697.602266.

[13] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994. ISSN 0949-877X. doi: 10.1007/BF01231602. URL https://doi.org/10.1007/BF01231602.

[14] John Herring et al. Opengis® implementation standard for geographic information-simple feature access-part 1: Common architecture [corrigendum]. 2011.

[15] John R. Herring. OpenGIS® Implementation Standard for Geographic Information - Simple Feature Access - Part 1: Common Architecture, May 2011. URL http://www.opengeospatial.org/legal/. Corrigendum, Reference number: OGC 06-103r4.

[16] Marius Hogräfer, Magnus Heitzler, and Hans-Jörg Schulz. The state of the art in map-like visualization. *Computer Graphics Forum*, 39(3):647–674, 2020. doi: https://doi.org/10.1111/cgf.14031. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14031.

[17] International Organization for Standardization. Ergonomics of Human-System Interaction – Part 11: Usability: Definitions and Concepts, 2018. Confirmed in 2023.

[18] Martin Kleppmann. Designing data-intensive applications, 2019.

[19] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jimenez-Peris, Raquel Pau, and José Pereira. The cloudmdsql multistore system. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 2113–2116, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450335317. doi: 10.1145/2882903.2899400. URL https://doi.org/10.1145/2882903.2899400.

[20] Stavros Kolios, Andrei V. Vorobev, Gulnara R. Vorobeva, and Chrysostomos Stylios. *Geographic Information Systems*. Springer International Publishing, Cham, 2017. ISBN 978-3-319-53086-4. doi: 10.1007/978-3-319-53086-4_1. URL https://doi.org/10.1007/978-3-319-53086-4_1.

[21] Juuso Koponen and Jonatan Hildén. *Data visualization handbook*. Aalto korkeakoulusäätiö, 2019.

[22] Danylo Kravchenko. Integrating gis capabilities into polypheny. Master's thesis, University of Basel, February 2024. Available at https://dbis.dmi.unibas.ch/teaching/studentprojects/integrating-gis-capabilities-into-polypheny/.

[23] James R. Lewis. The system usability scale: Past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018. doi: 10.1080/10447318.2018.1455307. URL https://doi.org/10.1080/10447318.2018.1455307.

[24] James R Lewis and Jeff Sauro. Item benchmarks for the system usability scale. *Journal of Usability Studies*, 13(3), 2018.

[25] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. Polypheny-db: Towards a distributed and self-adaptive polystore. pages 3364–3373, 12 2018. doi: 10.1109/BigData.2018.8622353.

[26] Marco Vogt, Nils Hansen, Jan Schönholz, David Lengweiler, Isabel Geissmann, Sebastian Philipp, Alexander Stiemer, and Heiko Schuldt. Polypheny-db: Towards bridging the gap between polystores and htap systems. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB Workshops, Poly 2020 and DMAH 2020, Virtual Event, August 31 and September 4, 2020, Revised Selected Papers*, volume 12665 of *Lecture Notes in Computer Science*, pages 25–36. Springer International Publishing, 2021. doi: 10.1007/978-3-030-87669-1_3.

[27] Marco Vogt, David Lengweiler, Isabel Geissmann, Nils Hansen, Marc Hennemann, Cédric Mendelin, Sebastian Philipp, and Heiko Schuldt. Polystore systems and dbmss: Love marriage or marriage of convenience? In El Kindi Rezig, Vijay Gadepally, Timothy Mattson, Michael Stonebraker, Tim Kraska, Fusheng Wang, Gang Luo, Jun Kong, and Alevtina Dubovitskaya, editors, *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 65–69, Cham, 2021. Springer International Publishing. ISBN 978-3-030-93663-1.

A

Appendix

B

User Study Document

This document was shared with users to guide them through the user study.

User Testing: Prototype for Map-Based Query Mode in Polypheny-DB

The user testing is online between Monday the 6th and Tuesday the 14th of January.

Estimated time to complete: 15 - 25 minutes

1. Read this document
2. Complete tasks
3. Fill out survey

Background

Polypheny is a polystore database created by the **Database Information Systems Research Group (DBIS)** at the University of Basel. Polypheny can handle data in three data models: (a) relational, (b) document collection and (c) graph. Each data format brings its own advantages and disatvantages:

- **(a) Relational:** Best for structured data, that rarely changes. Data must adhere to a schema (columns and column types must be determined before data is inserted). This data format is usually queried using SQL.
- **(b) Document:** Best for semi-structured data, where the format is structured (e.g. JSON), but the fields are dynamic, so it is difficult to create a schema. This data format is usually queried using Mongo Query Language (MQL).
- **(c) Graph:** Best for highly interconnected data, like a network or social media data. This data format is usually queried using Cypher.

Each name space is associated with one of the three data models - this helps logically seperating the data. Despite this logical separation, cross-namespace queries are still allowed, for example to join data from entities in different namespaces. The output of a query is independent on the namespace and just relies on a query language and its model.

My Contribution

In my Master Thesis, I added geospatial support to the document and graph model by following existing geospatial capabilities and extending it when needed. This means that I added the ability to work with data that represents geometries, like points, polygons, lines, and so on. To make working with geospatial data easier, I added a new **map-based query mode** to the Polypheny-DB web UI. This new feature can be accessed in three ways:

1. After running a query in the Query Console, the data representation can be switched between table view, document view, graph view and map view. Here, the first 10 results of the query are shown. By clicking a button, the user can navigate to the map-based query mode and look at the full results.
2. The user can directly navigate to the Map-Based query mode, and start with an empty map. The user has then the ability to add layers by running queries or loading data from external files.
3. The Data section allows looking at the data in different namespaces. This data view allow supports the map visualization like in the results section, where the user can navigate to the full Map-Based query mode.

The Goal of this User Study

This user study aims to evaluate the ease-of-use and the usability of this Map-Based query mode. To do this, I would like you to interact with the prototype of the map-based query mode, and use it to solve three tasks. In these tasks, you will take on the role of a data scientist, and will work on different problems, that require the visualization of the mapping results. After you had time to interact with the prototype, you can fill out a feedback form and submit it. Thank you!

- Each task starts with a small description of what we want to achieve, as well as a description of the data.
- Task 1 and 2 contain step-by-step instructions, Task 3 just contains descriptions.

Tasks

1. Connect to the University VPN

VPN: Secure connection to the network of the University of Basel

2. Go to Polypheny DB Web UI

The Prototype is hosted here: blue04.maas.dmi.unibas.ch

The website is served in HTTP. If the website cannot be shown correctly, you need to disable the HTTPS-only mode. To do this, follow the instructions in the Disable HTTPS-Only Mode in Firefox section.

3. Complete tasks

Follow step-by-step instructions to complete the tasks. You are free to explore the UI on your own, if you prefer. If you need any help, refer to the How to use section, if you have any questions, you can contact me, see details in the Help section.

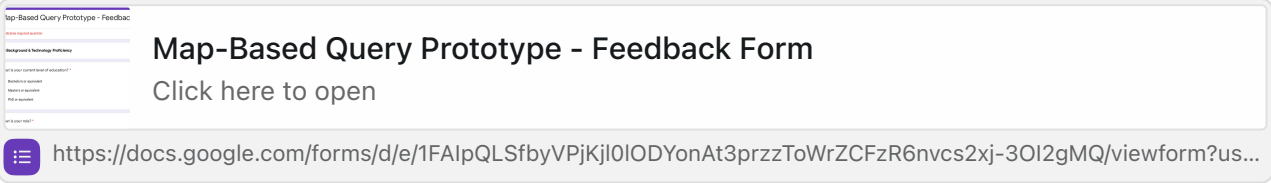
Tasks:

Task 1: Data Validation for Family Ancestry
You are conducting family ancestry research. One part of your data collection efforts includes co...

Task 2: Find Pattern for Temperature Sensors
You are in charge of the maintance of the temperature sensors that are placed throughout the ci...

Task 3: TripAdvisor Restaurant Ratings in Paris
Unlike Task 1 and 2, this task does not include step-by-step instructions, but a description for so...

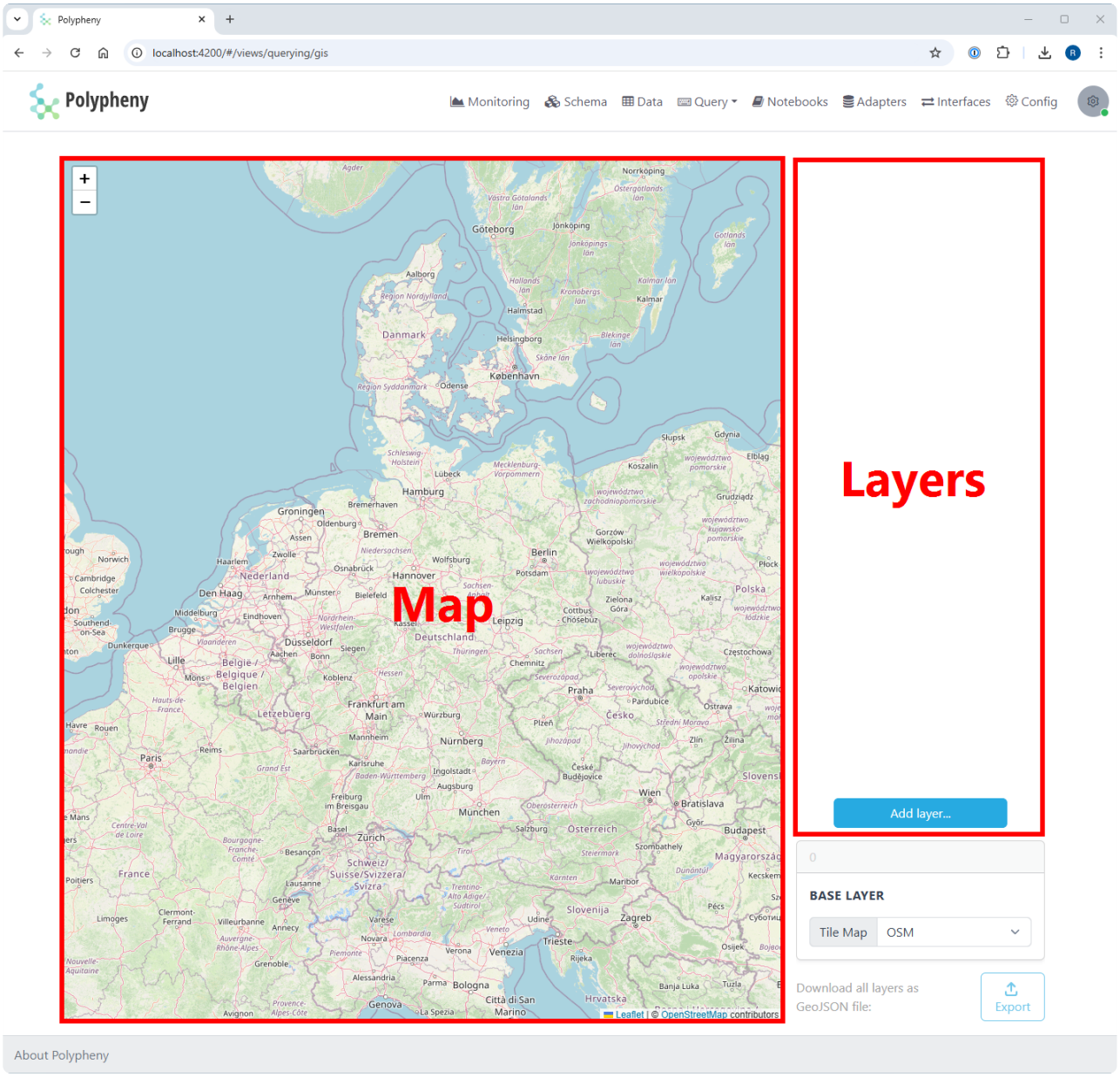
4. Fill out Feedback Form



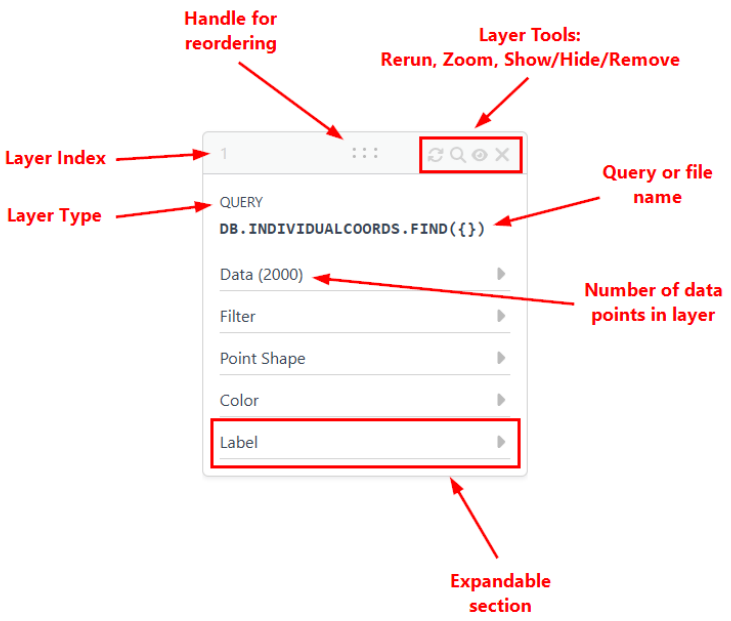
How to use

The UI is made up of about three sections:

1. The **Map** on the left: It contains all the data points
2. The **layer panel** on the right: Data from queries or files is stored inside layers. Each layer can be configured separately, and will be edited with a **Layer Card.**
3. **Other settings** on the bottom right. Such as the **Base Layer** or the **Export** functionality.


Polypheny
localhost:4200/#/views/querying/gis
Polypheny
Monitoring
Schema
Data
Query
Notebooks
Adapters
Interfaces
Config
Map
Layers
Add layer...
0
BASE LAYER
Tile Map
OSM
Download all layers as GeoJSON file:
Export
Leaflet | © OpenStreetMap contributors
About Polypheny

Layer Card



Each layer can be configured through their layer card.

Help

If you need any help, you can reach me in the following **Zoom** meeting room on

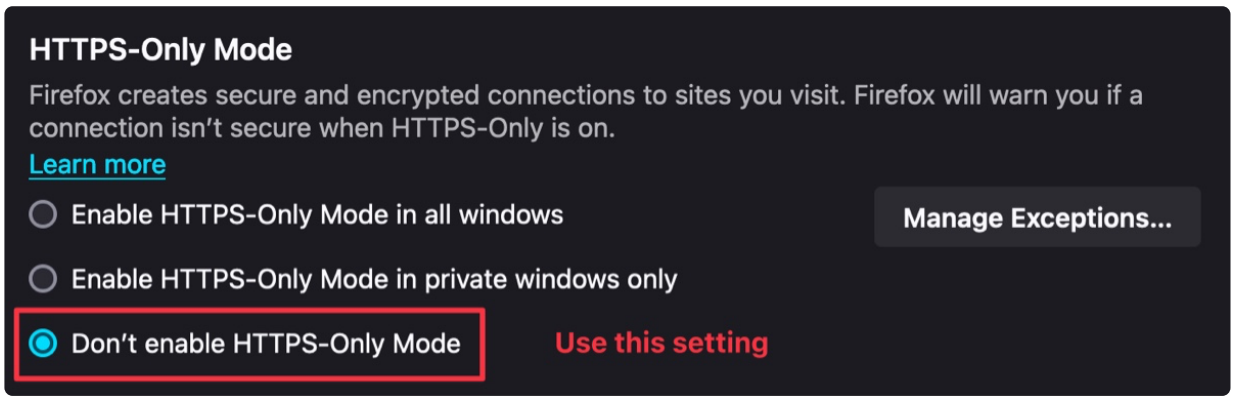
- ~~Tuesday, the 7th between 09:00 and 17:00:~~
- ~~Friday, the 10th between 09:00 09:30 - 12:00 and 13:00 - 17:00 16:00~~

Join our Cloud HD Video Meeting
Zoom is the leader in modern enterprise video communications, with an easy, reliable cloud...
https://unibas.zoom.us/j/9022120763?pwd=MThxRVBFcDFPUDVrNXEzVmk0ckxOQT09

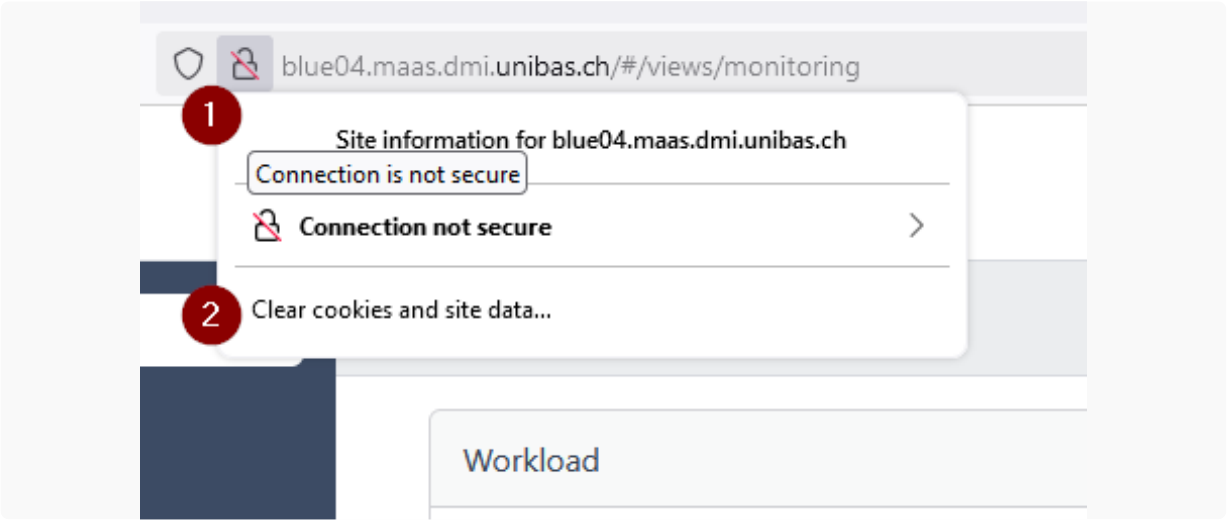
Or you can also write me a message anytime on **Telegram** or **WhatsApp** at +49 15234081748 or via email to r.biehler@unibas.ch.

Disable HTTPS-Only Mode in Firefox

1. Go to settings, then Privacy & Security, and then scroll down
2. Disable for all websites



3. Clear Site Cache



↑ User Testing: Prototype for Map-Based Query Mode in Polypheny-DB

Task 1: Data Validation for Family Ancestry

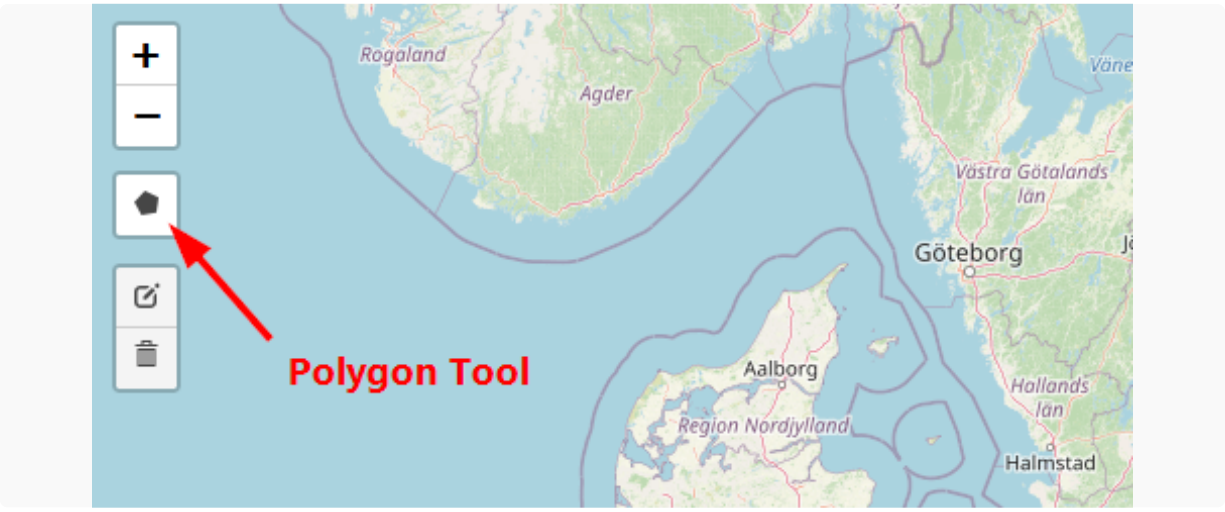
You are conducting family ancestry research. One part of your data collection efforts includes collecting data from various different resources and of varying quality. One part of your cleaning procedures condists of checkin the validity of coordinates. As this is difficult only by looking at the latitude and longitude values, you decide to plot them on a map.

To do this, you complete the following steps:

1. Open the Polypheny-DB Web UI by clicking here.
2. Click on **Query**, then on **Map-Based Query.**
3. In the Layers Panel on the right, click on the **Add Layer...** button.
4. In the popup that opens

1. set the **Query Language** to `MQL`.
2. set the **Namespace** (above the Query Editor) to `doc`.
3. paste the following code inside the **Query editor:**
   `db.individualCoords.find({})`
4. Hit the **Execute & Add** button

5. Now the data should appear on the map. Click the **magnifier icon** on the layer card, to adjust the visible area of the map so that all data points are visible.
6. Based on the sources where you gathered the data, some data points are in places you did not expect them. It looks like the latitude and longitude of some coordinates is swapped! To remember those data points for manual review, you want to export them.
7. You can verify your suspicion by hovering over the data points and looking at the tooltip text. It should contain the key `place`. Do this for some data points.
8. **Pan** and **zoom** the map to ensure all potentially incorrect data points are visible.
9. Expand the **Edit Query** section on the layer card.
10. Click on the button **Enable Drawing on Map**. This adds a toolbar to the map that contains drawing tools, such as the **Polygon Tool**, shown in the image below:



11. Click the **Polygon Tool** button to start drawing a polygon.
    - Each left click adds a point to the polygon. Once the last and the first point is connected, the polygon is finished. If you add a point by mistake, you can remove it clicking **Delete last point** in the grey toolbar next to the **Polygon Tool** button.
12. Finish the polygon by connecting the first and the last point. The polygon you just drew, as well as all data points outside of it should now be gone. The data was filtered successfully.
13. Export the data by clicking the **Export** button in the lower right. A GeoJSON file will be downloaded to your device, that contains all the points from all layers.

14. You are done! The exported points can now be analyzed further. Thanks!

Extra:

Without the polygon tool, you have to create the query by hand. Constructing the query by hand looks like this, and is not very easy to do, without any additional tools:

```
db.individualcoords.find({
  geometry: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [
          [
            [44.97802734375, 7.710991655433217],
            [48.05419921875, 7.27529233637217],
            [49.63623046875001, 3.5134210456400448],
            [46.47216796875001, -1.0106897682409002],
            [48.14208984375001, -2.2406396093827206],
            [56.35986328125001, 9.362352822055605],
            [64.48974609375001, 10.876464994816295],
            [62.16064453125001, 12.254127737657381],
            [55.08544921875, 12.55456352859367],
            [55.78857421875001, 25.64152637306577],
            [46.47216796875001, 16.256867330623454],
            [47.35107421875001, 10.444597722834875],
            [45.37353515625001, 9.232248799418674],
            [44.97802734375, 7.710991655433217]
          ]
        ]
      }
    }
  }
});
```

← You can now go to Task #2

Task 2: Find Pattern for Temperature Sensors

You are in charge of the maintance of the temperature sensors that are placed throughout the city of Bern. These sensors measure the temperature in the city at various location throughout the day. Because the sensors have a tendency to drift, you need to ensure that the temperatures values seem plausibe. To do that, you want to compare sensors with other sensors that are in their vicinity. If one sensor does not agree with its neighbors, then the sensor probably needs to be recalibrated.

To achieve your task, you complete the following steps:

1. Open the Polypheny-DB Web UI by clicking here.
2. In the navigation bar at the top, click on **Data.**
3. In the left sidebar, expand the node **public**.
4. Under this node, click on **berntempsensors**.
5. The data preview in table format should be visible in the main area of the UI.
6. Above the table, there are two buttons visible. One with a **table icon**, and one with a **map marker icon**. Click on the **map marker icon button** to switch to the map preview.
7. This loads the first 10 entries of the dataset and displays it on the map. To see the full results, click on **See full results in map-based query mode.**
5. Now all the data should appear on the map. Click the **magnifier icon** on the layer card, to adjust the visible area of the map so that all data points are visible.
6. Click on the **Data section** inside the layer card to get an idea what kind of data each data point contains.
7. To visualize the temperature variable as a color, expand the **Color** section and set the **Mode** input from `Static` to `Gradient`. In the same section, set the **Variable** input to the value `temp`.
8. You can now press the **Refresh layers** button at the bottom right of the map. The points on the map should now appear in different shades of blue, depending on the value of the variable temperature.
9. Because the color is hard to see, we want to increase the point size. To do that, open the **Point Shape** section and set the **Size** input to 10.
10. You can also try out different icon modes, like a star or a rectangle. To do this, set the **Mode** combobox in the **Point Shape** section to the desired shape.
11. Again, click on the **Refresh layers** button to apply the changes to the map.

12. We can see that some data points are noticeably darker than others. To remember them for later, we want to export them. To do this, hover over the data points, to check what their `temp` value is.
13. Expand the **Edit Query** section, then click on **Edit Query**. We can now update the query to include a treshold: `SELECT * FROM "public"."berntempsensors" WHERE temp > INSERT_TRESHOLD`.
14. Export the data by clicking the **Export** buton at the bottom of the layers section (you may need to scroll down).
15. Done

← You can now go to Task #2

Task 3: TripAdvisor Restaurant Ratings in Paris

Unlike Task 1 and 2, this task does not include step-by-step instructions, but a description for solving it instead. If you cannot complete the task, you can also look at results.

You are a researcher, and are preparing your trip to Paris, where you will be attending a conference. Because good food is very important to you, you decide to plan ahead, by using a TripAdvisor dataset about restaurants and their reviews.

Table `restaurants`

This table was created from this dataset: TripAdvisor European restaurants dataste from Kaggle. Due to performance considerations, the restaurants have been limited to the city of Paris, and then limited to 1000 restaurants using a random sample.

Column Name	Column Type	Note
restaurant_link	`VARCHAR(255) NOT NULL`	ID in TripAdvisor Dataset.
NAME	`VARCHAR(255) NOT NULL`	
address	`VARCHAR(255) NOT NULL`	Point geometry
location	`GEOMETRY NOT NULL`	
top_tags	`VARCHAR(255) NOT NULL`	
average_price	`INT NOT NULL`	average price spent in €
cuisines	`VARCHAR(255) NOT NULL`	
special_diets	`VARCHAR(255) NOT NULL`	
excellent	`INT NOT NULL`	5/5⭐
very_good	`INT NOT NULL`	4/5⭐
average	`INT NOT NULL`	3/5⭐
poor	`INT NOT NULL`	2/5⭐
terrible	`INT NOT NULL`	1/5⭐
total_ratings	`INT NOT NULL`	Sum of the columns excellent, very_good, average, poor and terrible.
rating	`DOUBLE NOT NULL`	Weighted average rating based on columns excellent, very_good, average, poor and terrible.

Table `arrondissement`

This table lists the districts of Paris. For each district, there is a number and a name, as well as the geometry as a poylgon.

Column Name	Column Type	Note
num	`INT NOT NULL`	
NAME	`VARCHAR(100) NOT NULL`	
location	`GEOMETRY NOT NULL`	Polygon geometry

Table `sightseeing`

The table contains the names and coordiantes of well-visited sightseeing locations in Paris.

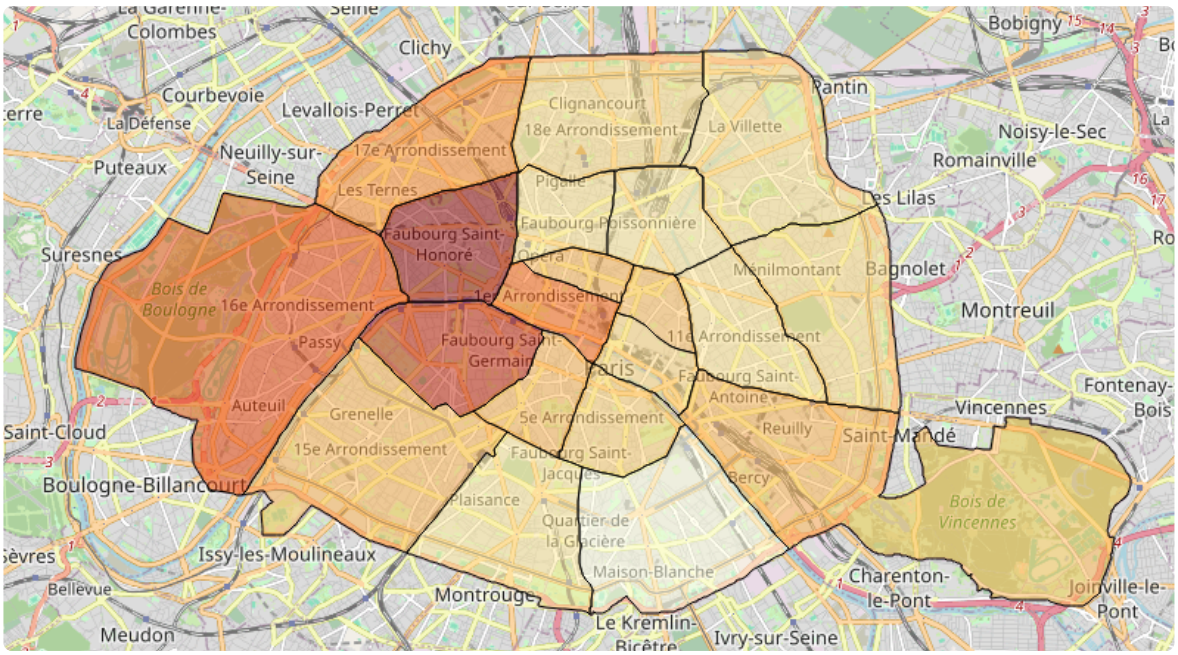
Column Name	Column Type	Note
name	`VARCHAR(255) NOT NULL`	ID in TripAdvisor Dataset.
location	`GEOMETRY NOT NULL`	Point geometry

Question 1: Which district should we stay in?

We can decide based on number of restaurants, average rating or price level. To do this, you can use the query below, which aggregates the restuarants based on the districts of the city. Then, use different color visualizations to highlight the different aggregated values.

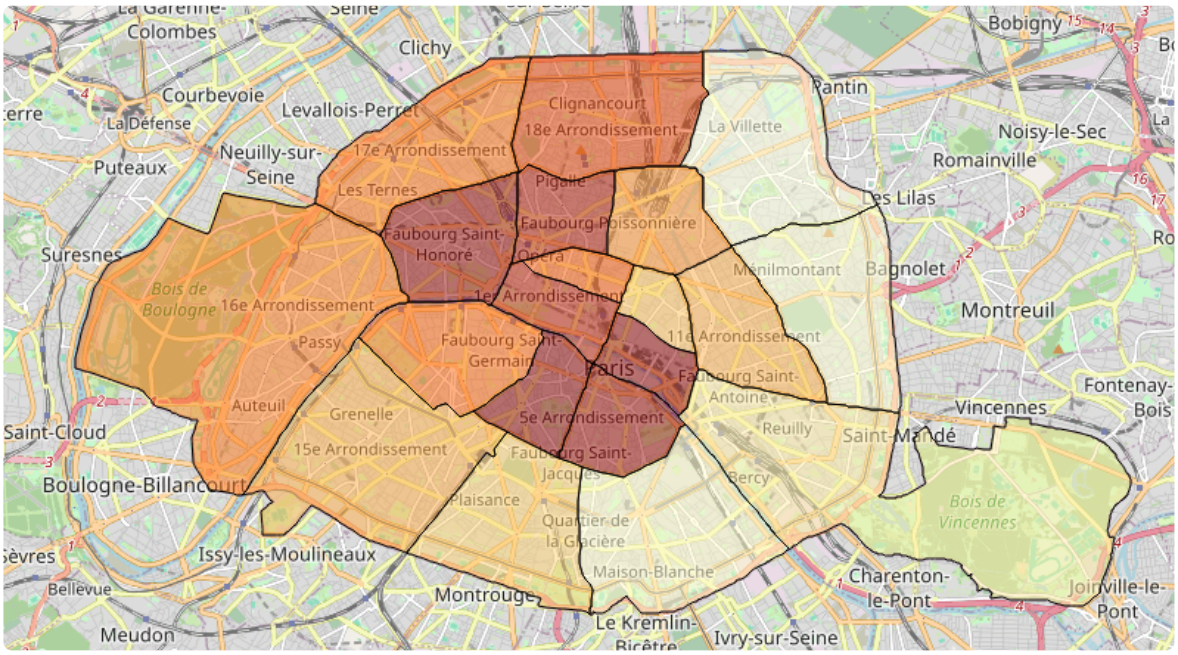
```
SELECT
    a.num,
    a.name,
    a.location,
    COUNT(r.restaurant_link) AS total_restaurants,
    AVG(r.rating) AS average_rating,
    AVG(r.average_price) AS average_price
FROM
    arrondissement a
LEFT JOIN
    restaurants r
ON
    ST_CONTAINS(a.location, r.location)
GROUP BY
    a.num, a.name, a.location
ORDER BY
    a.num
```

- Results (click to expand)
  - Based on average price, we should not stay in the western districts, namely 7, 8, 16 and 17.



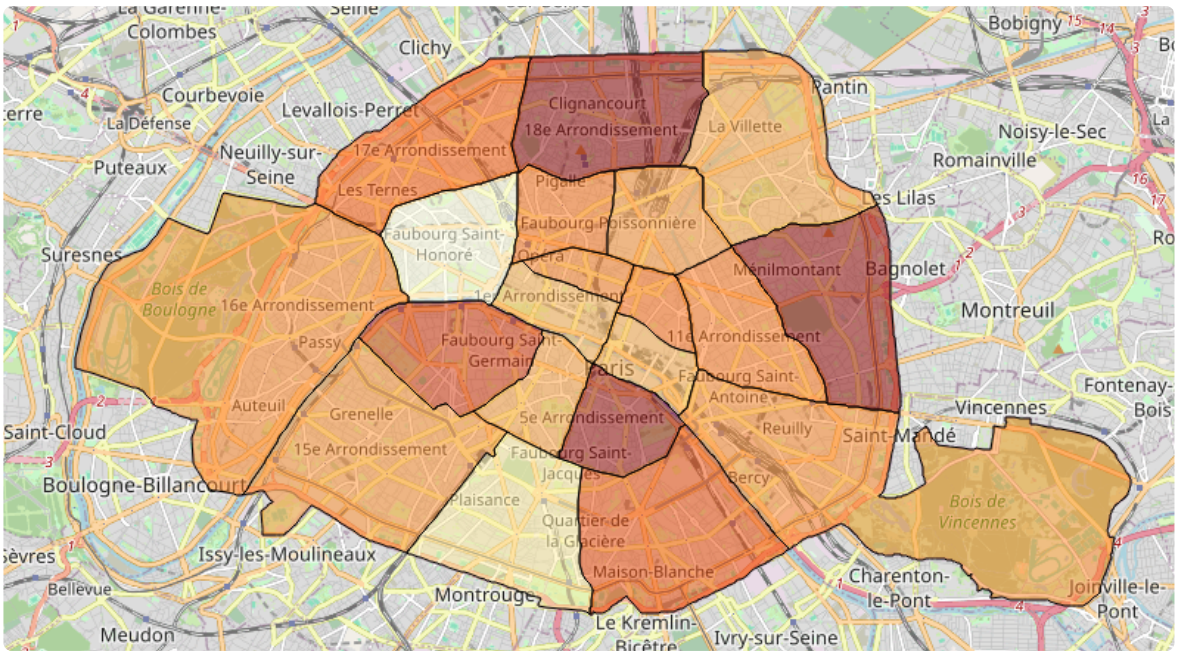
- Variable set to `total_restaurants`
- Opacity (Area) set to `0.5`
- Scale set to `Linear`

▼ Based on number of restaurants, we could stay in 1, 4, 5, 6, 8, 9 or 18.



- Variable set to `average_rating`
- Opacity (Area) set to `0.5`
- Scale set to `Linear`

▼ Based on average rating, 5, 18 or 20 is higher rated than the rest. We should avoid the 8th.



- Variable set to `average_rating`
- Opacity (Area) set to `0.5`
- Scale set to `Linear`

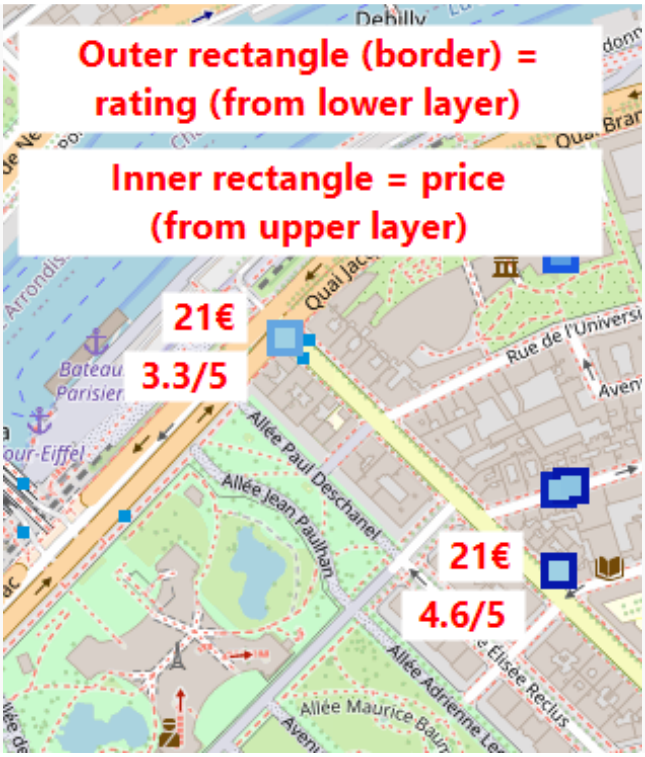
There is no clear answer. The 18th seems to have a lot of restaurants with high ratings on average, but it is not very central. The 5th is very central, and could be a good choice.

Question 2: Find out, if expensive restaurants are also the restaurants with the best average ratings.

Idea: Create two layers from the query `SELECT * FROM restaurants`. For the bottom layer, use slightly larger point size, so that it can still be seen, even though the points from the upper layer are drawn on top. Use the color visualization on different variables with the mode Gradient to check how the colors for each data point interact with each other. Hover over the tooltips to get an idea of the actual values represented by the colors.

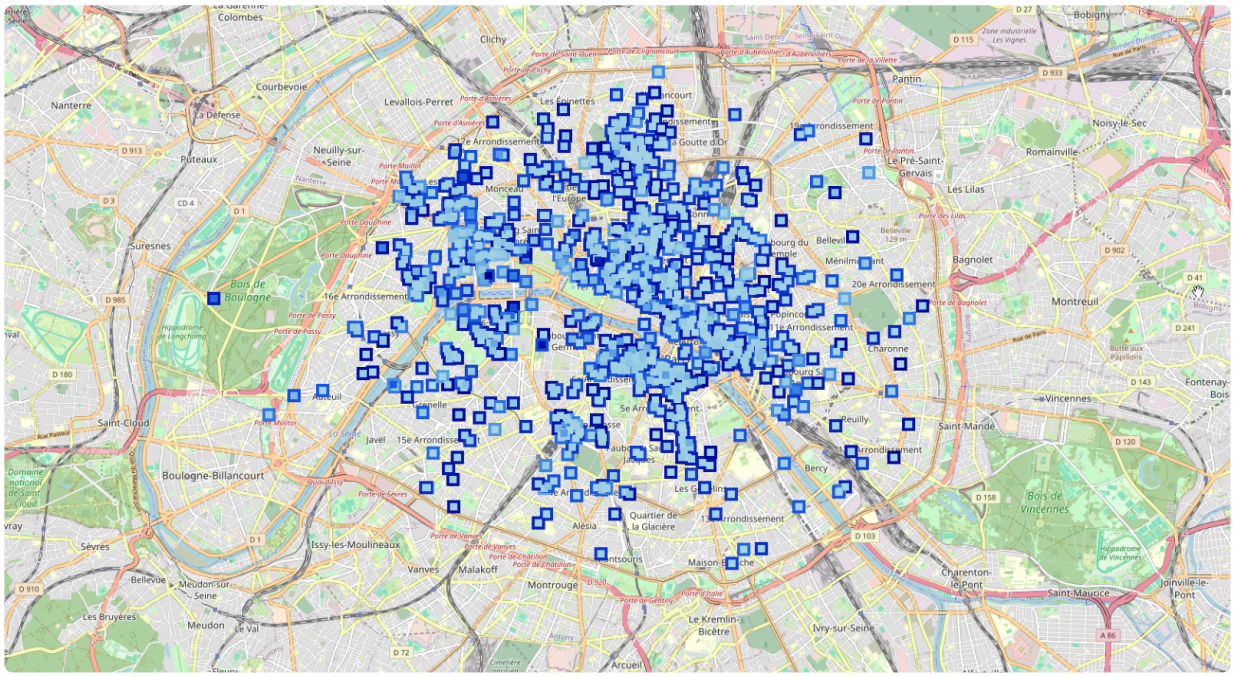
▼ Results (click to expand)

When the visualization is configured like this:

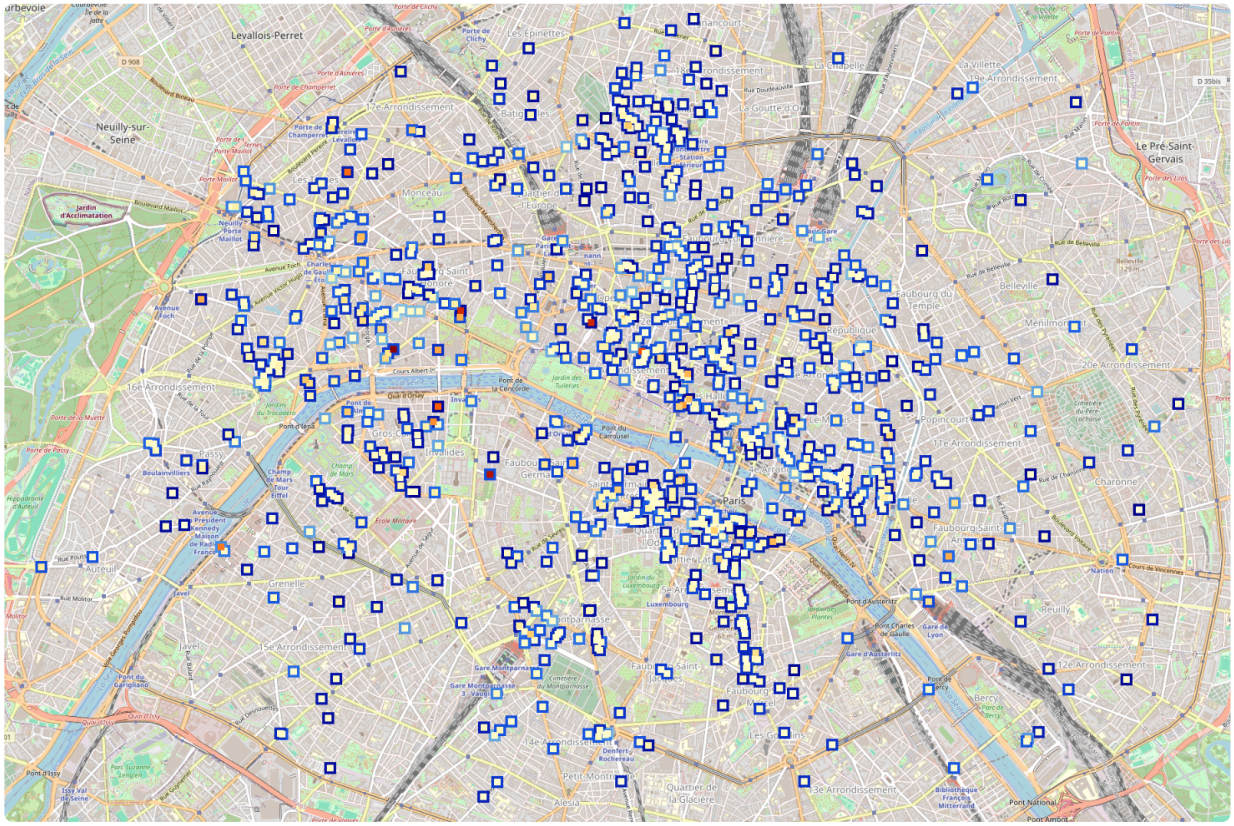


We can see that most restaurants are pretty similar, where the border color (rating) is darker than the inner color (price). There are however, some exceptions, for example the two restaurants in the screenshot above, that are just right next to the Eiffel Tower. Here, they both have the same price, but very different ratings.

In general, we are looking for restaurants with a light inner color and a dark border, so high rating, low price.



Changing the price visualization to use a linear scale, we can more easily see the outliers. The restaurants with a red to deep red color have prices upwards of 250€. Using the sequential scale makes these outliers less impactful overall.



← All done! Please fill out the feedback form.

C

User Study Evaluation Form

The participants were asked to fill out this form after using the implementation. The form was created and hosted on Google Forms[52].

The left label was lost for all questions using the 1-5 scale during the export process:

- Question 5: *Not* should be *Not at all*
- Question 8 to 17: *Stro* should be *Strongly disagree*
- Question 18 to 23: *Not* should be *Not at all*

[52] forms.google.com

Map-Based Query Prototype - Feedback

* Indicates required question

1/3 Background & Technology Proficiency

1. What is your current level of education? *

   Mark only one oval.

   - ◯ Bachelor's or equivalent
   - ◯ Master's or equivalent
   - ◯ PhD or equivalent

2. What is your role? *

   Mark only one oval.

   - ◯ Student
   - ◯ Academic Staff (e.g. Professor, Lecturer, Researcher)
   - ◯ Other: _____

3. Which department do you belong to? *

Mark only one oval.

- ◯ Mathematics and Computer Science
- ◯ Environmental Sciences
- ◯ Arts, Media, Philosophy
- ◯ Biomedicine
- ◯ Biozentrum
- ◯ Clinical Research
- ◯ Public Health (DPH)
- ◯ Ancient Civilizations
- ◯ Biomedical Engineering
- ◯ Chemistry
- ◯ History
- ◯ Languages and Literatures
- ◯ Pharmaceutical Sciences
- ◯ Physics
- ◯ Social Sciences
- ◯ Sport, Exercise and Health
- ◯ Other: _____

4. How many years of experience do you have with programming? *

_____

5. How experienced are you in working with databases? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very experienced

6. Which of the following query languages have you worked with?

Tick all that apply.

- [ ] SQL
- [ ] Mongo Query Language
- [ ] Cypher

7. How experienced are you in working with Geographic Information Systems (GIS)?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very experienced

2/3 Usability Feedback

8. I think that I would like to use this system frequently. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

9. I found the system unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

10. I thought the system was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

11. I think that I would need the support of a technical person to be able to use this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

12. I found the various functions in this system were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

13. I thought there was too much inconsistency in this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

14. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

15. I found the system very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

16. I felt very confident using the system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

17. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly agree

3/3 Feedback

18. How confident are you in completing Task 1 using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

19. How confident are you in completing Task 1 without using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

20. How confident are you in completing Task 2 using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

21. How confident are you in completing Task 2 without using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

22. How confident are you in completing Task 3 using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

23. How confident are you in completing Task 3 without using the Prototype?

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very confident

24. Which problems did you encounter when working with the prototype?

25. Were there any features you felt were missing?

Thank you
for testing and filling out the form!

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis: Multi-Model Geospatial Queries with Interactive Visualization in

Name Assessor: Prof. Dr. Heiko Schuldt

Name Student: Rafael Biehler

Matriculation No.: 22-059-091

I attest with my signature that I have written this work independently and without outside help. I also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.

Additionally, I affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used. This paper may be checked for plagiarism and use of AI-supported technology using the appropriate software. I understand that unethical conduct may lead to a grade of 1 or "fail" or expulsion from the study program.

Place, Date: Bad Säckingen, 31.01.25 Student: Rafael Biehler

Will this work, or parts of it, be published?

☐ No

☑ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: 31.01.2025

Place, Date: Bad Säckingen, 31.01.25 Student: Rafael Biehler

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis.