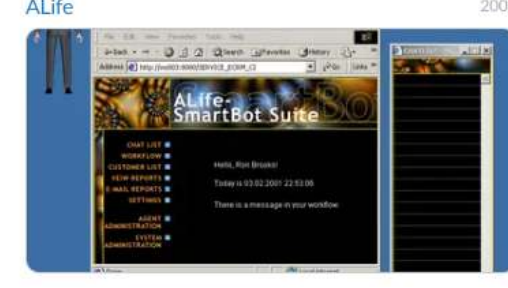
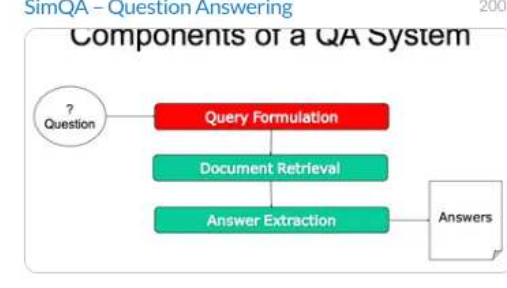
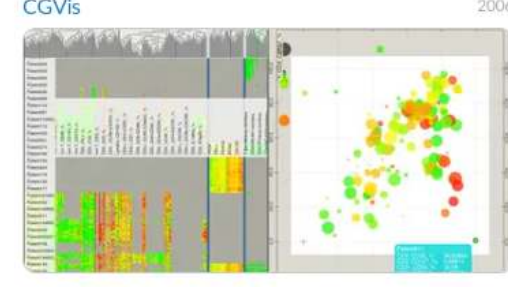
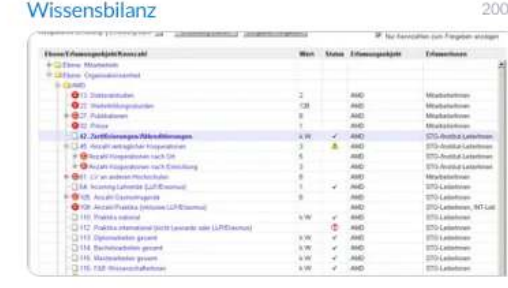
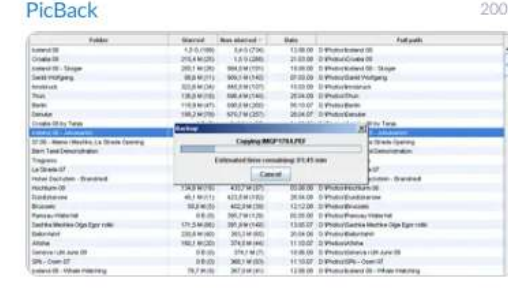
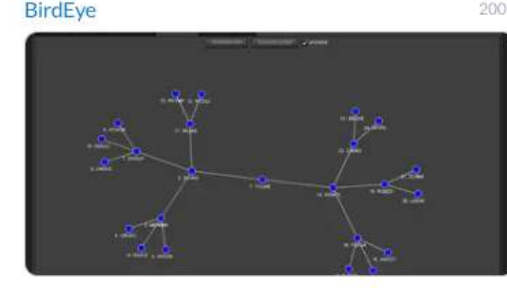
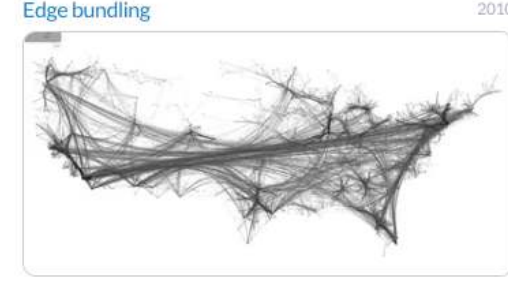
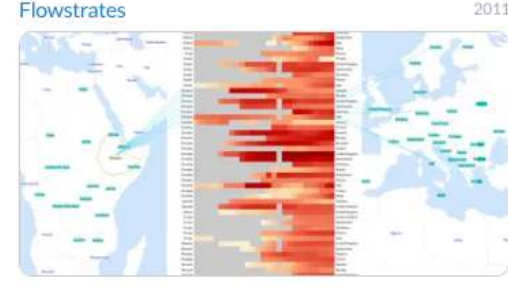
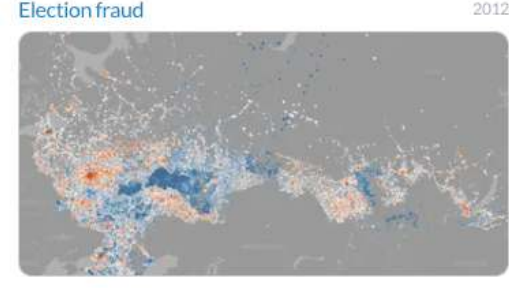
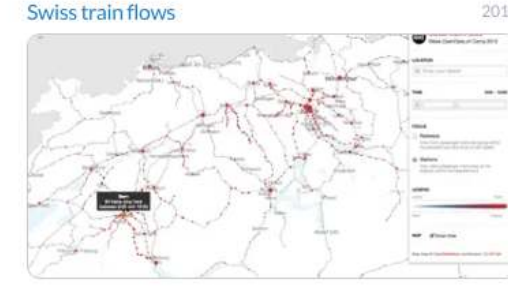
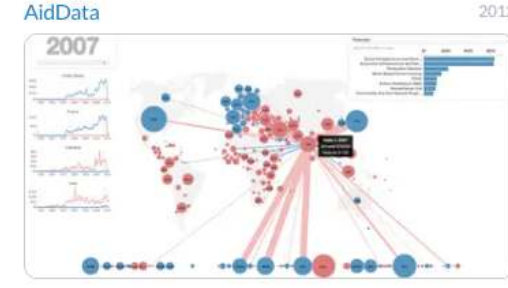
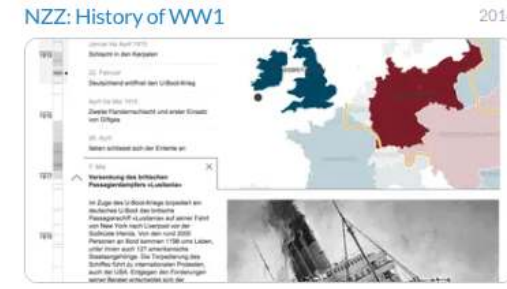
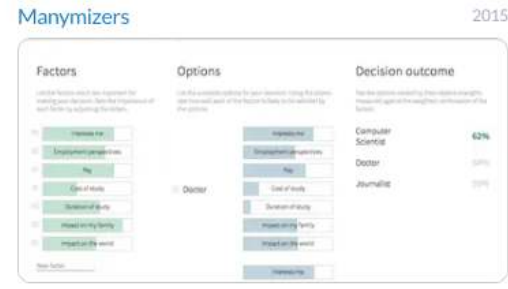
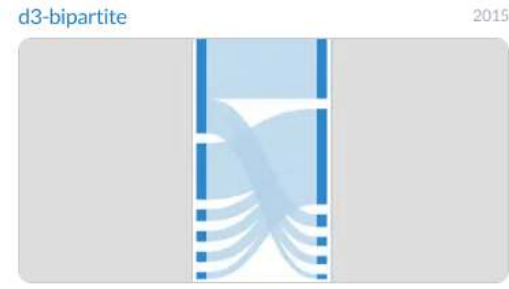
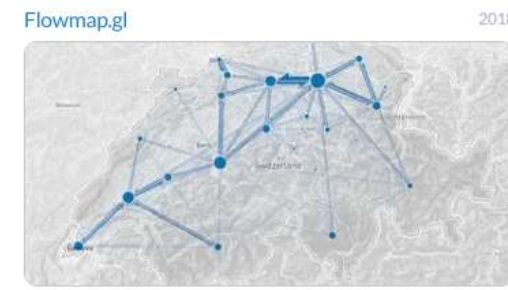
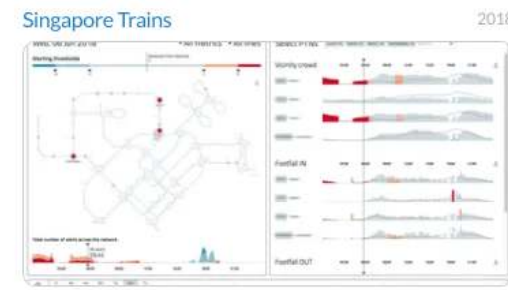
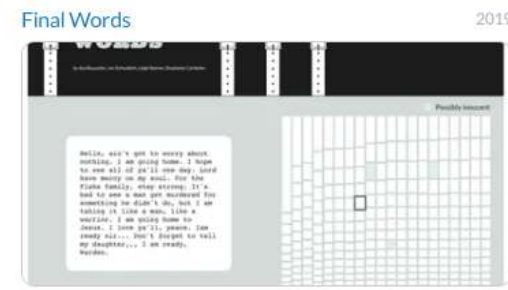
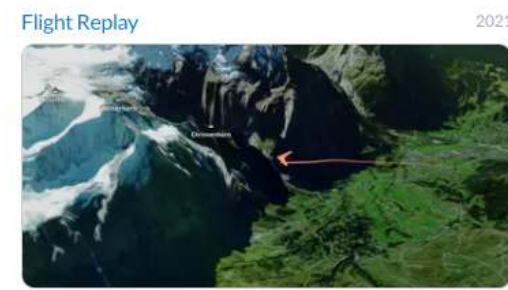
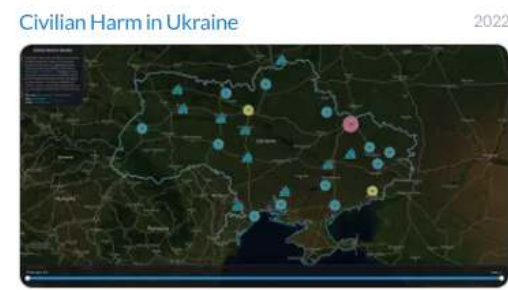
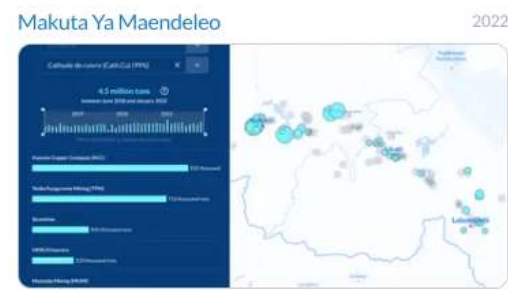
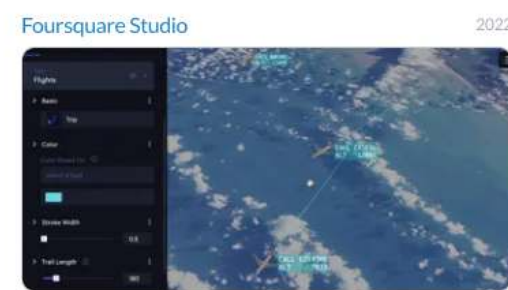


# SQLRooms

Toolkit for human + agent data analytics apps

Ilya Boyandin, Foursquare



**DATA SOURCES**

+ Add

**FILES**

- VBZ\_Fahrgastzahlen\_2023/REISENDE.parquet (28.25 MB)
- VBZ\_Fahrgastzahlen\_2023/HALTESTELLEN.csv (31.81 KB)
- VBZ\_Fahrgastzahlen\_2023/haltepunkt.csv (948.23 KB)
- VBZ\_Fahrgastzahlen\_2023/haltestelle.csv (34.76 KB)

**SQL QUERIES**

+ Add

- stops
- reisende\_1

**TABLES**

**haltepunkt**

- halt\_punkt\_id (BIGINT)
- halt\_punkt\_diva (BIGINT)
- halt\_id (BIGINT)
- GPS\_Latitude (DOUBLE)
- GPS\_Longitude (DOUBLE)
- GPS\_Bearing (BIGINT)
- halt\_punkt\_ist\_aktiv (BOOLEAN)

23,208 rows

**haltestelle**

- halt\_id (BIGINT)
- halt\_diva (BIGINT)
- halt\_kurz (VARCHAR)

**FILTER**

Total Number of Trips: **24,194,894**

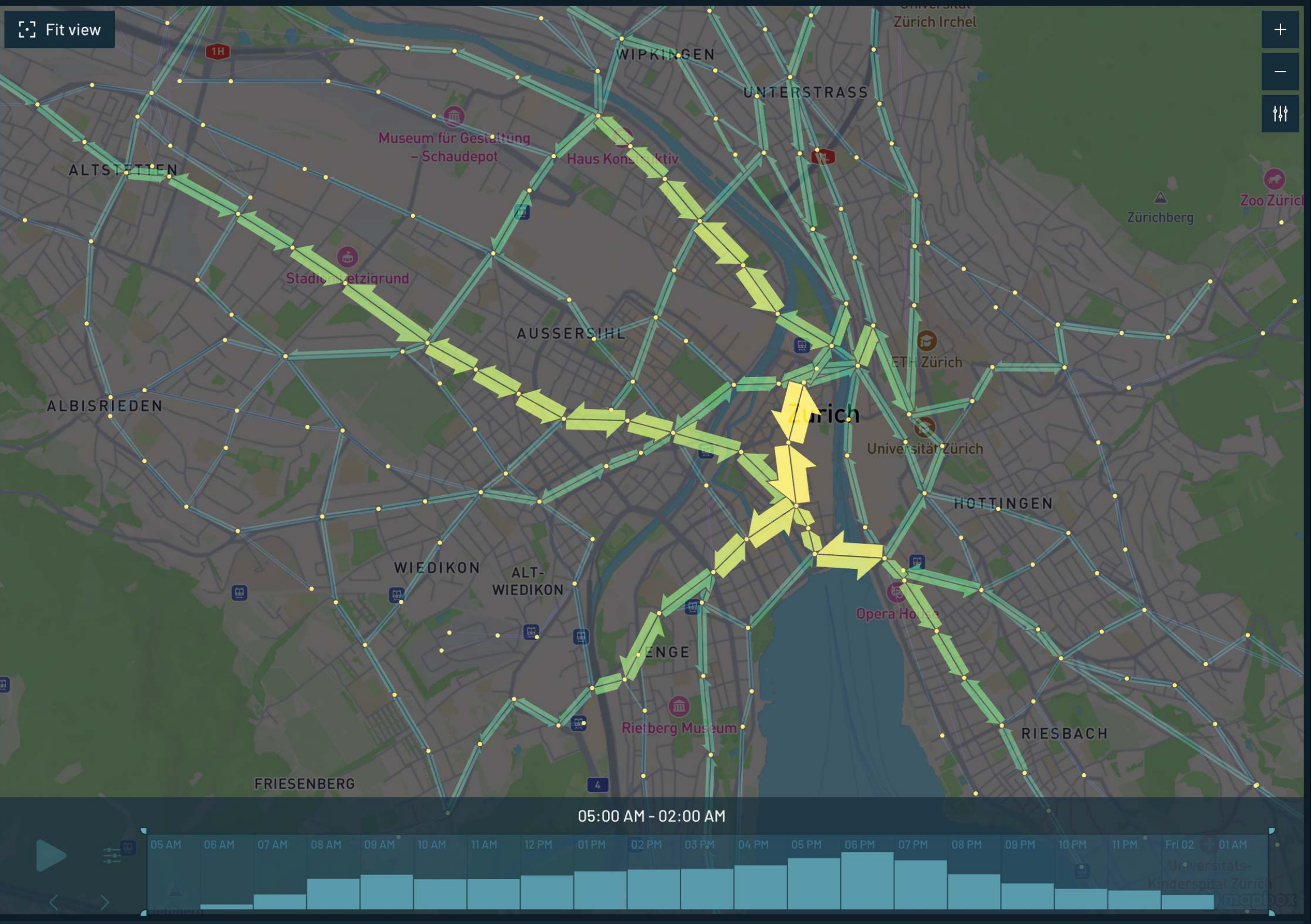
**LOCATION**

Search location...

**LINIENNAME**

**Trips by Linienname**

Linienname	Trips
394	4.841k
344	3.523k
918	2.865k
79	2.428k
341	2.246k
99	2.115k
395	1.945k
396	1.568k
419	1.529k
315	
330	
333	
329	
39	
73	
349	
352	
753	
332	
29	



**LOCATIONS**

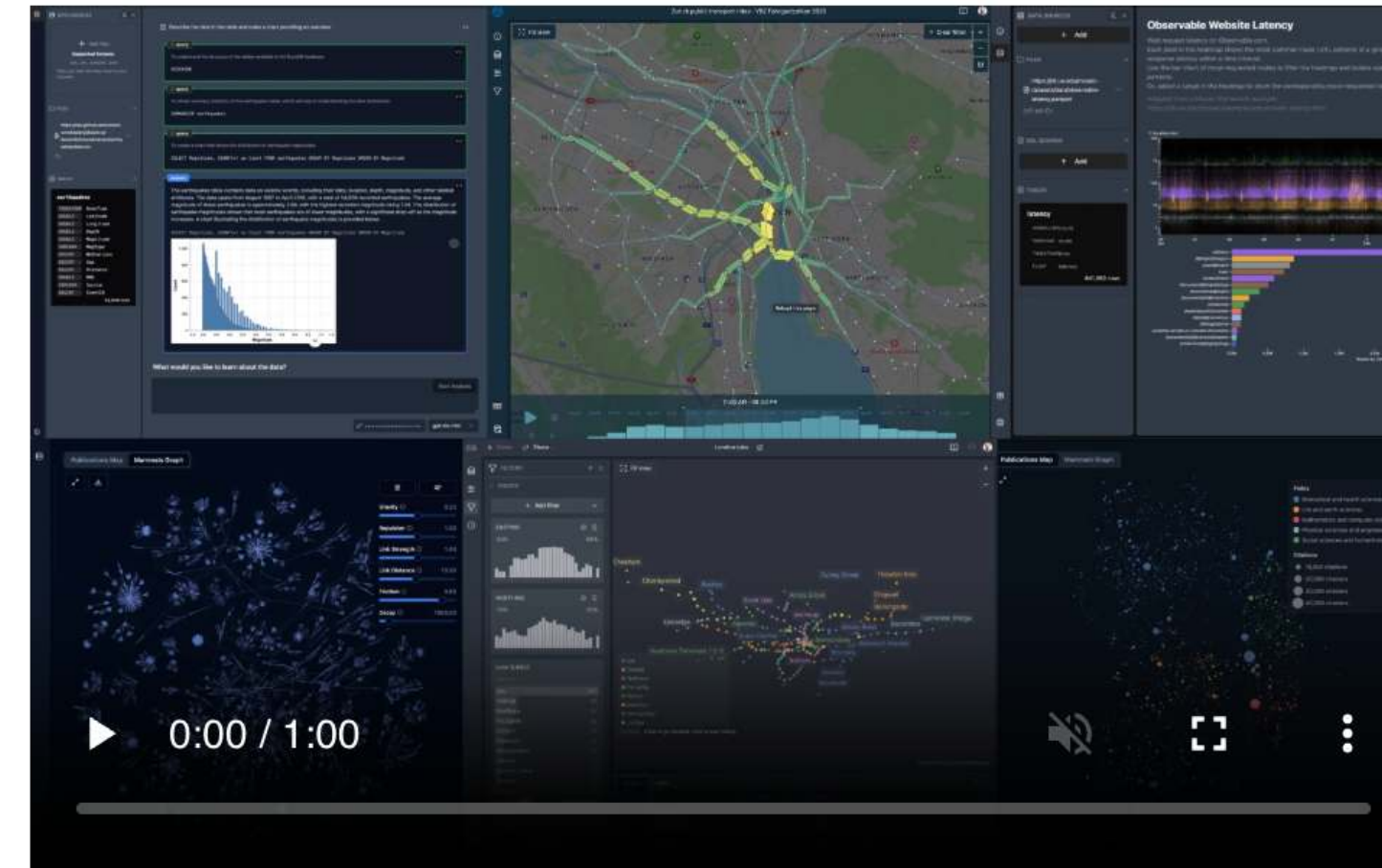
	origin_name	dest_name	origin	dest	count	time
1	Küsnacht ZH, Hesligenstrasse	Küsnacht ZH, Himmelistrasse	313	312	4.45455	19:47:42
2	Küsnacht ZH, Himmelistrasse	Küsnacht ZH, Hesligenstrasse	312	313	1.45455	06:38:42
3	Küsnacht ZH, Himmelistrasse	Küsnacht ZH, Hesligenstrasse	312	313	1.66667	07:08:42

Page 1 of 4,509 | 100 rows | 450,875 rows | Export CSV

# SQLRooms

## Build data-centric apps with DuckDB

An open source React toolkit for human + agent collaborative analytics apps

[What is SQLRooms?](#)[Example Apps](#)[View on GitHub](#)

### Local Analytics, No Backend Required

Leverage DuckDB's powerful SQL capabilities, enabling fast in browser data processing without a backend

### Own Your Data

Data remains on your local device for maximum privacy, sub-second analytics on large datasets, and offline functionality

### Privacy-Preserving AI Integration

Use agents that can write and execute SQL queries, and generate insights without sharing your data with model providers

### Modular Architecture

Pick and choose the functionality you need for composable, extensible applications, with integrations for popular data visualization libraries.

### Modern UI Components

Comprehensive set of React components including data tables, layouts, and visualization tools for building beautiful analytics interfaces

### Offline Use

Work with your data, run queries, and analyze results even without an internet connection. SQLRooms supports offline workflows using persistent storage via OPFS.

**DATABASE** ⌵ ✕

Status: Connected ⚡

+ Add data

+ Quick file drop

Supported formats:  
csv, tsv, parquet, json, arrow, shp, klm

**TABLES** ⌵

- California Earthquakes
  - main
    - earthquakes
      - timestamp DateTime
      - double Latitude
      - double Longitude
      - double Depth
      - double Magnitude
      - varchar MagType
      - integer NbStations
      - integer Gap
      - integer Distance
      - double RMS
      - varchar Source
      - integer EventID

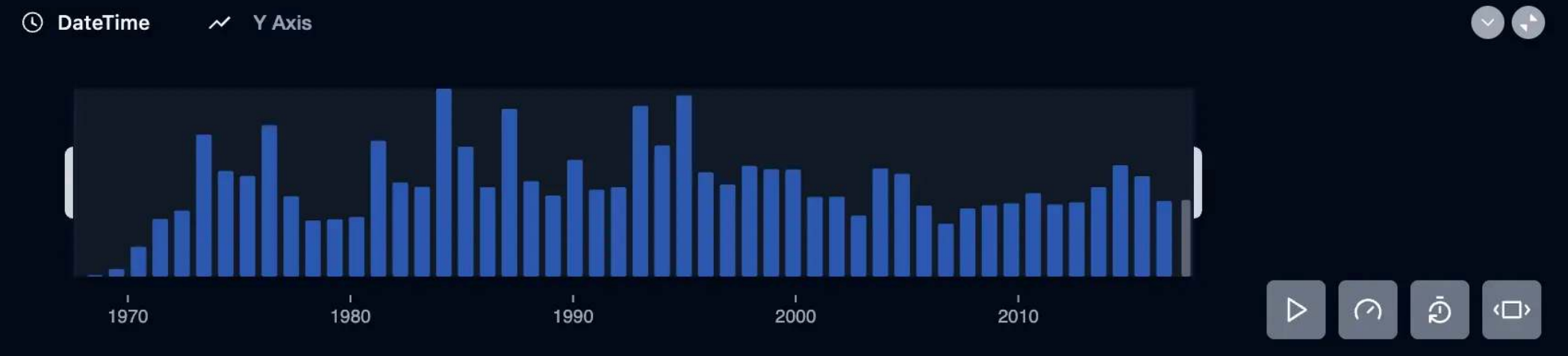
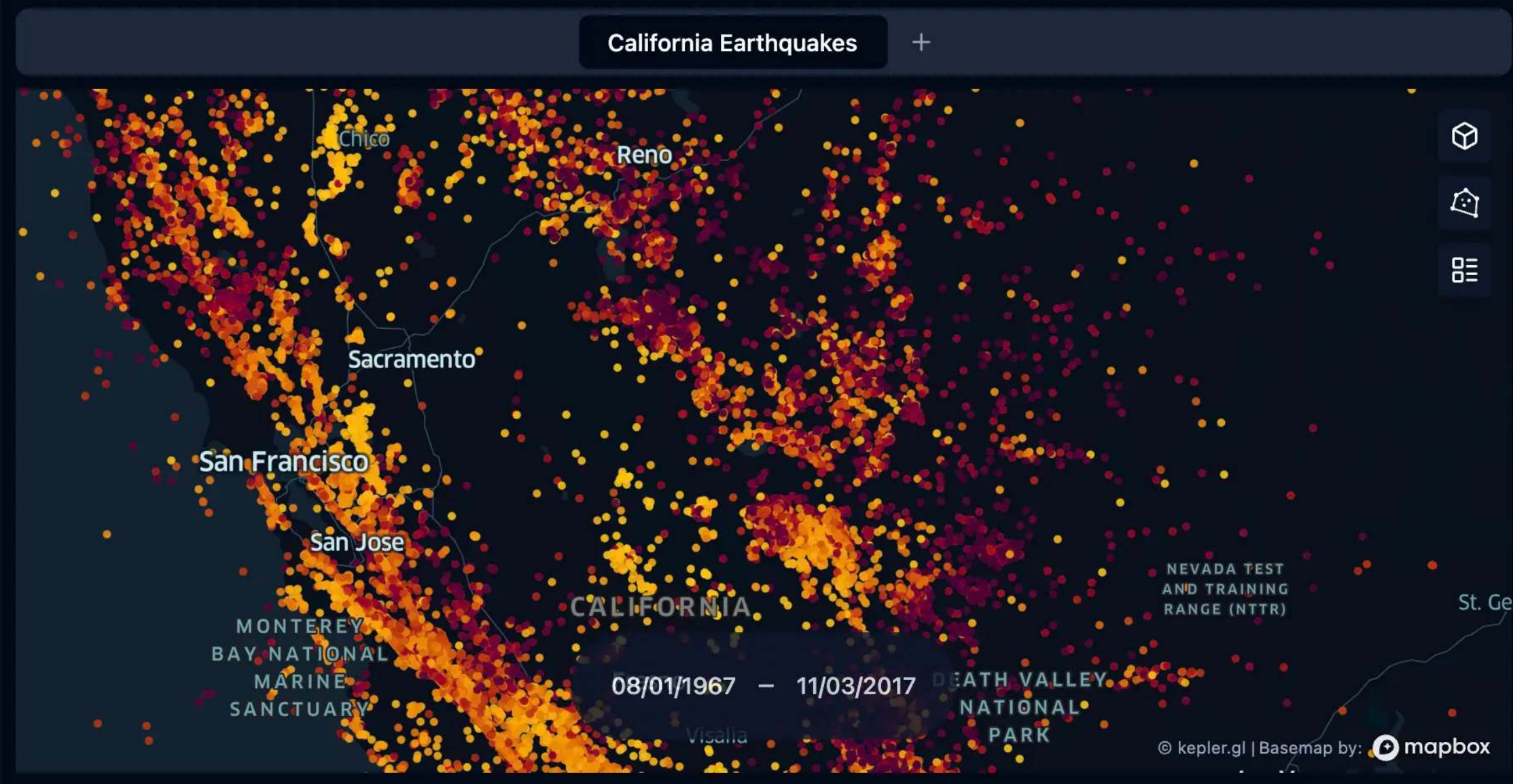
**TILESETS** ⌵

>

🗒

📍

🔄



▶ Run ⌘↵ Untitled ⋮ + 🗒

```
1 SELECT Magnitude, * FROM earthquakes WHERE Magnitude > 5
2 ORDER BY Magnitude DESC
```

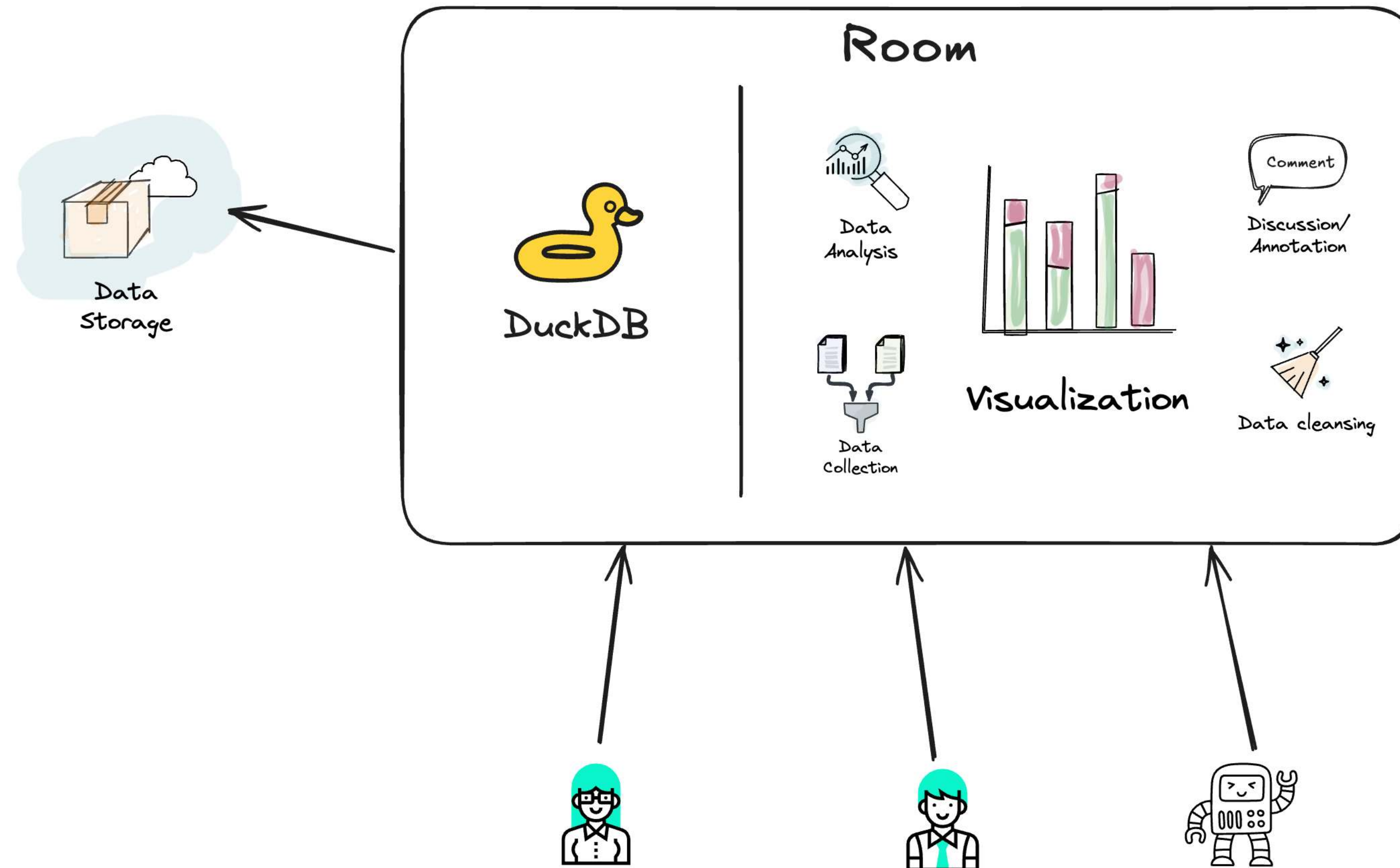
	Magnitude	Float64	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth
1		7.39	1992-06-28T11:57:35.390Z		34.186		-116.46117		1
2		7.2	2010-04-04T22:40:42.000Z		32.259		-115.287		1
3		7.2	1980-11-08T10:27:33.200Z		41.08417		-124.61567		1

100 rows Limit results to 100 rows + New table

**To simplify how developers build data analytics applications,  
using SQL as the common foundation for humans and AI**

# Room

a self-contained workspace where data lives, analysis happens, and (soon) collaborators meet



# Why SQL

## Universal Language for Data

Familiar to many data analysts, engineers

Rich Ecosystem & Standards

## Declarative, High-Level Abstractions

State *what* you want, not *how* to compute it

Optimizations in execution engine improve perf with no code changes

## Future-proof?

# SQL + LLMs

## LLMs Lower the Barrier to SQL

Power users can still write SQL directly

Less technical users can describe their intent in natural language

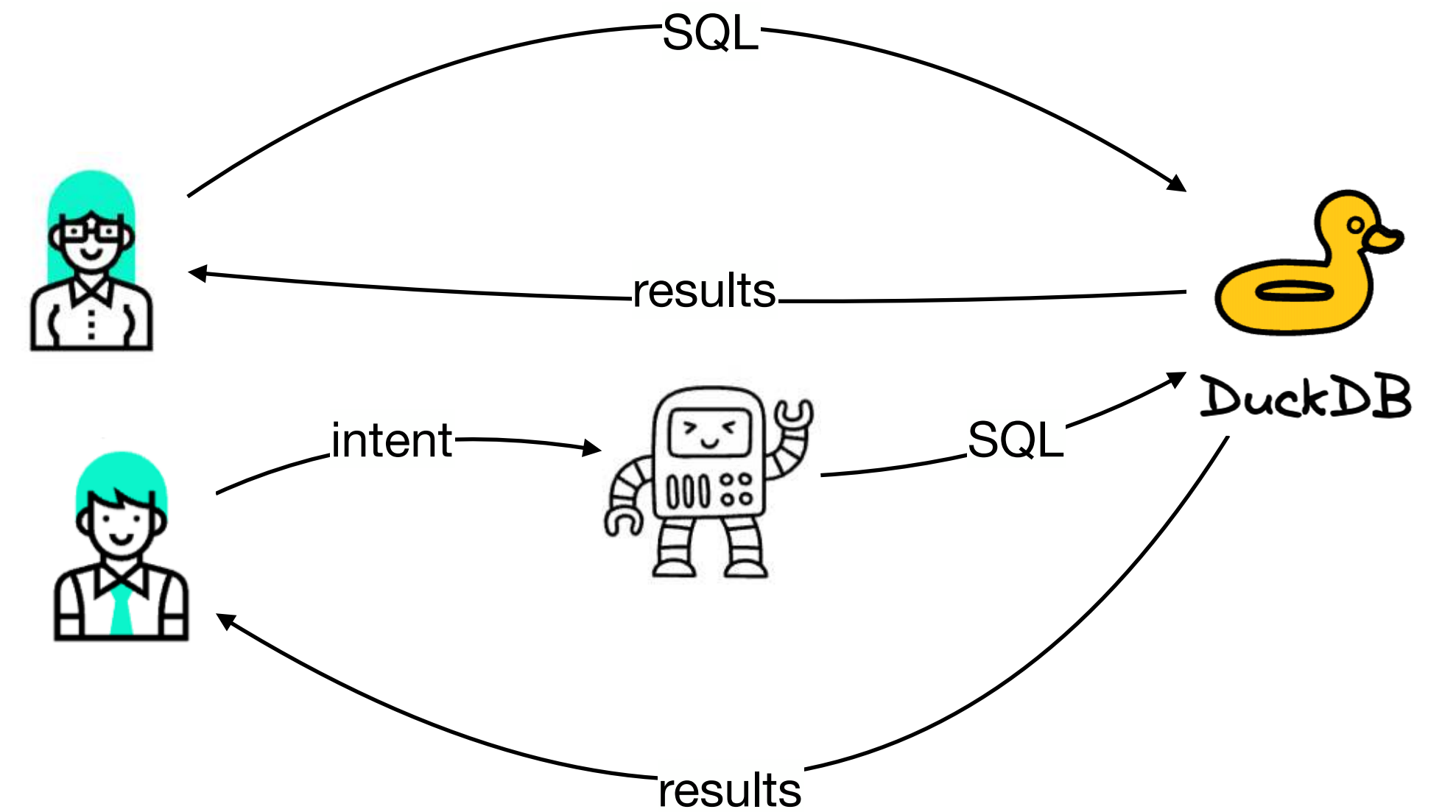
## SQL as a Stable Target Language

Standardized, well-documented, and structured

## Debuggable & Transparent

Still readable and inspectable by humans

SQL is both machine-friendly and human-friendly





# Why DuckDB

## Made for Analytics

columnar + vectorized → analytics-friendly, fast

## Embeddable Anywhere

In browser via WASM

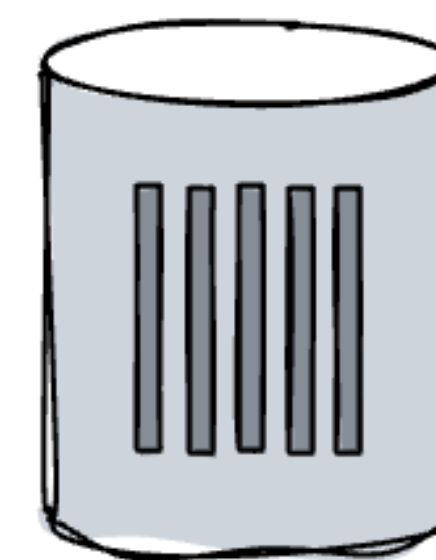
As a Python/R/C/C++ library

## Efficient Interop

Arrow-native query results → zero-copy data exchange

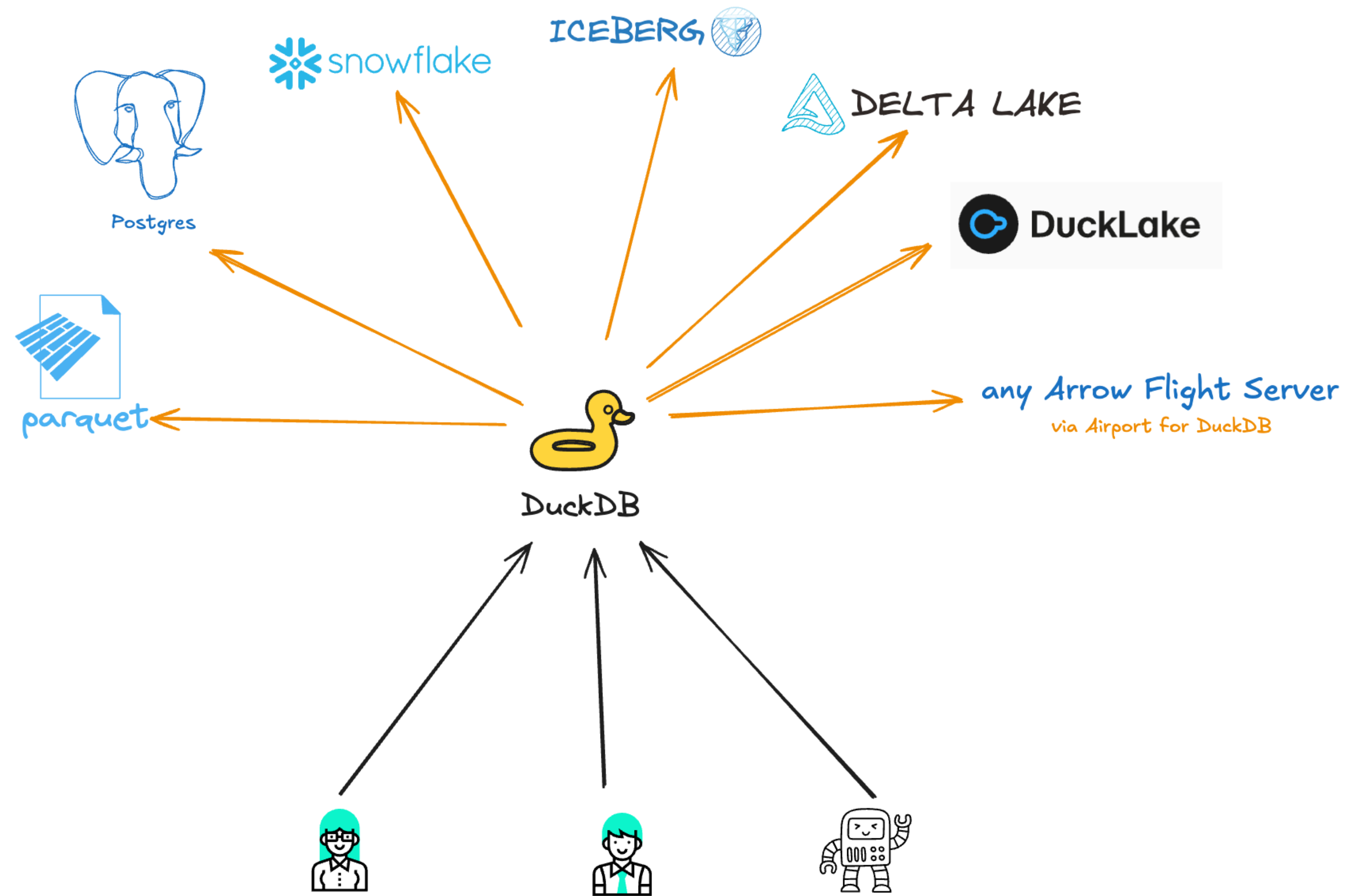
## Low Infra, Storage-First

Query data directly in S3/object storage (Parquet) → storage/compute separation



Columnar DB

# Multi-source queries



# Offline use example

## Progressive Web App (PWA)

Web Apps capable of working offline

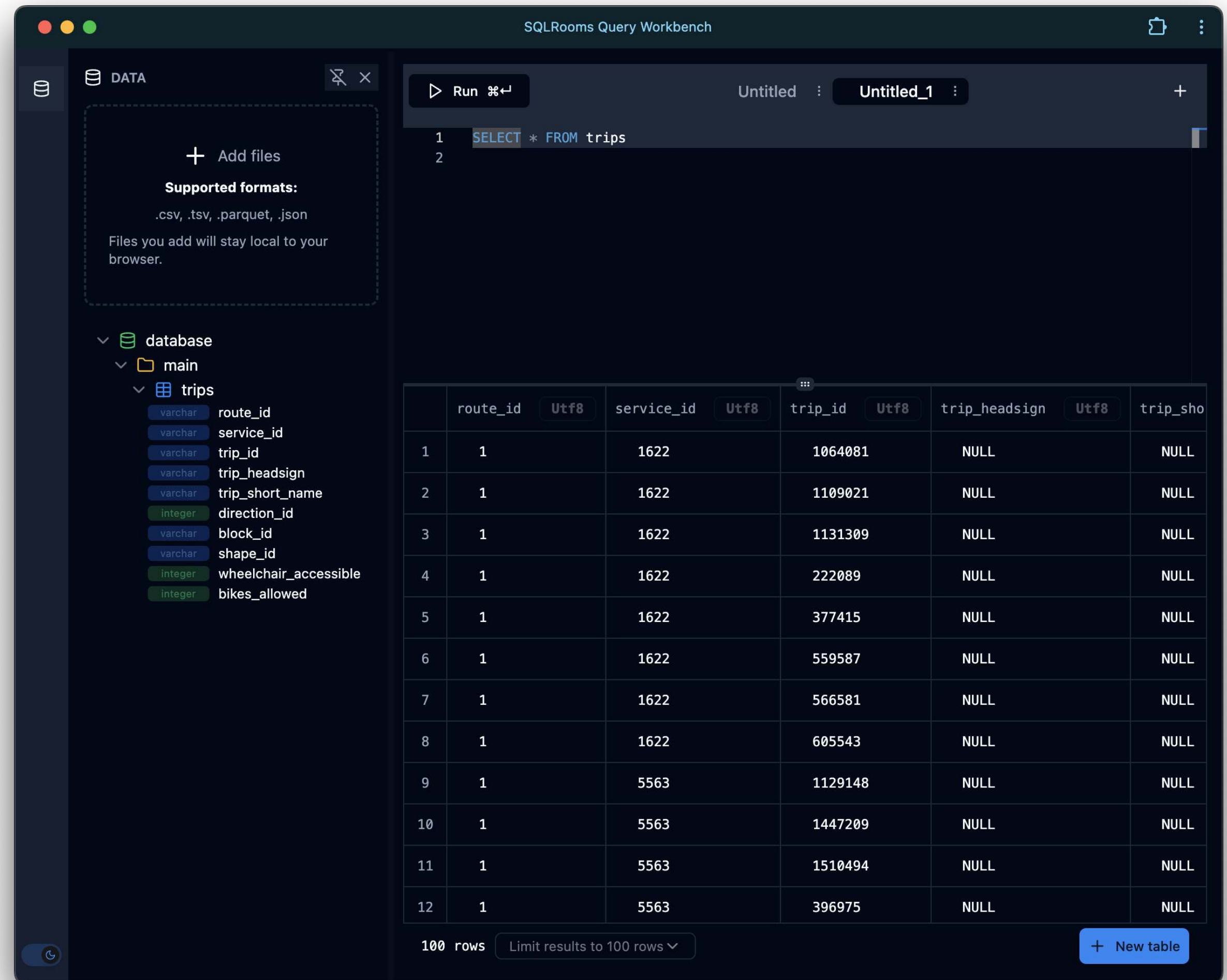
## OPFS for DuckDB Storage

Origin Private File System

## Local storage / IndexedDB

Persisting app state

## Local AI via Ollama



The screenshot shows the SQLRooms Query Workbench interface. On the left, a database schema is displayed for a 'trips' table with the following columns and data types:

- route\_id (varchar)
- service\_id (varchar)
- trip\_id (varchar)
- trip\_headsign (varchar)
- trip\_short\_name (varchar)
- direction\_id (integer)
- block\_id (varchar)
- shape\_id (varchar)
- wheelchair\_accessible (integer)
- bikes\_allowed (integer)

The main query editor shows the following SQL query:

```
SELECT * FROM trips
```

The query results are displayed in a table with 12 rows and 6 columns:

	route_id	service_id	trip_id	trip_headsign	trip_sho
1	1	1622	1064081	NULL	NULL
2	1	1622	1109021	NULL	NULL
3	1	1622	1131309	NULL	NULL
4	1	1622	222089	NULL	NULL
5	1	1622	377415	NULL	NULL
6	1	1622	559587	NULL	NULL
7	1	1622	566581	NULL	NULL
8	1	1622	605543	NULL	NULL
9	1	5563	1129148	NULL	NULL
10	1	5563	1447209	NULL	NULL
11	1	5563	1510494	NULL	NULL
12	1	5563	396975	NULL	NULL

At the bottom of the interface, it indicates '100 rows' and 'Limit results to 100 rows'. A '+ New table' button is visible in the bottom right corner.

<https://sqlrooms.org/offline-use.html>

# Single-node DB

## Own your data

No cloud necessary for storing and querying

## Fast Local Querying

No network round-trip or server lag

## Privacy

Nothing leaves your DuckDB node (e.g. browser) unless you decide so

AI: Query results do not have to be shared with external model providers

# Single-node scalability limits?

## Data size

Good for moderately sized data

No network overhead between distributed nodes

Not so suitable for massive datasets unless partitioned or federated

## High concurrency

WASM: DB Engine is in the end-user's browser => client-side horizontal scaling

Alternative: Session-backends

Spin up on demand, share among multiple users for collaboration

# sqlrooms-duckdb-server 0.2.2



```
pip install sqlrooms-duckdb-server
```



Released: Sep 23, 2025

A DuckDB server for SQLRooms

## Navigation

Project description

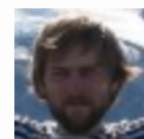
Release history

Download files

## Verified details

These details have been [verified by PyPI](#)

## Maintainers



ilyabo

## Unverified details

These details have **not** been verified by PyPI

## Project links

[homepage](#)

## Meta

- Requires: Python >=3.10

## Project description

### SQLRooms DuckDB Server

pypi v0.2.2

A Python-based server that runs a local DuckDB instance and supports queries over WebSockets, returning data in either [Apache Arrow](#) or JSON format. The server was originally developed for use with [SQLRooms](#), but can be readily used as a generic DuckDB server in other contexts.

*Note: This server was initially created as a fork of [Mosaic DuckDB Server](#), with additional features and improvements.*

*Note: This package provides a local DuckDB server. To instead use SQLRooms with DuckDB-WASM in the browser, stick to the default [WasmDuckDbConnector](#).*

## Features

- WebSocket endpoint (WS-only)
- Arrow IPC results (binary framed) and JSON responses
- Concurrent query execution using a shared thread pool (per-task cursors)
- Per-query cancellation via `duckdb.interrupt` (WebSocket `type:"cancel"`)
- WebSocket multiplexing with `queryId` correlation
- Subscribe/notify over WebSocket using Socketify publish/subscribe
- Disk-backed result cache with per-key locking (prevents duplicate compute)
- One-time retry on transaction conflicts (e.g., concurrent UPDATE vs ALTER)
- Graceful shutdown (SIGINT/SIGTERM): cancel queries, FORCE CHECKPOINT, close, stop executor
- Optional bearer authentication for HTTP and WebSocket endpoints

# Alternative data engines to support in future

## Apache DataFusion

query engine (SQL/Python Dataframe)

uses Apache Arrow as its in-memory format

## Apache SedonaDB

Single-node spatial DB (unlike Apache Sedona which is distributed)

similar to DuckDB but built from scratch for geospatial data

## Both work as Python libs, no WASM yet

Can be used via Python SQLRooms server

```
npm i @sqlrooms/*
```

# What SQLRooms provides

## Wiring up an analytical DB engine

via WASM, WebSockets or Motherduck

Functions for data loading, schema, querying, query management etc

## Composable UI components for data-centric apps

Tables, Schema-tree, Query editor, ...

## Composable state management

Room store composed from slices

Easy to add custom state and store functions

Easy to work with state across slices

## Integration with various libraries

e.g. Datavis: Vega, Mosaic, Recharts

# @sqlrooms/ui

```
<Button size="xs" onClick={newTable}>  
  <PlusIcon className="h-4 w-4" />  
  New table  
</Button>
```

```
<Slider  
  id={key}  
  min={min}  
  max={max}  
  step={step}  
  value={ [config[key]]}  
  onChange={(value) => handleParameterChange(key, value)}  
  className="w-full"  
>
```

# @sqlrooms/schema-tree

memory

main

earthquakes 54,936 rows

timestamp	DateTime
double	Latitude
double	Longitude
double	Depth
double	Magnitude
varchar	MagType
bigint	NbStations
bigint	Gap
bigint	Distance
double	RMS
varchar	Source
bigint	EventID
blob	geom

# @sqlrooms/data-table

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64	Ma
1		2011-06-15T12:29:08.000Z		31.46383		-115.4165		5.343	
2		2017-07-29T00:42:24.000Z		31.34		-115.379		9.21	
3		2013-12-20T05:27:12.000Z		31.431		-115.775		14.485	
4		2017-03-17T20:00:00.000Z		31.245		-115.692		12	
5		2010-06-25T23:23:15.000Z		31.296		-115.567		4.348	
6		2014-03-05T02:24:23.000Z		31.323		-119.409		6.735	
7		2012-12-14T11:09:55.000Z		31.48567		-119.51366		5.8	

« < Page 1 of 550 > » 100 rows ▾ 54,936 rows ⬇

# @sqlrooms/sql-editor

```
Run ⌘↵
```

Untitled

```
1 FROM earthquakes
2 SELECT date_part('year', DateTime) as year,
3 max(Magnitude)
4 GROUP BY year
```

**date\_part**

date\_part(ts: VARCHAR[], col1: DATE): STRUCT() ×  
date\_part(ts: VARCHAR[], col1: INTERVAL): STRUCT()  
date\_part(ts: VARCHAR[], col1: TIME): STRUCT()  
(+11 more overload)

Get subfield (equivalent to extract)

**Examples:**  
date\_part('minute', TIMESTAMP '1992-09-20 20:38:40')

	year	Int64	max(Magnitude)	Float64
1	1967			3.6
2	1968			4.3
3	1969			5.7
4	1970			4.7
5	1971			4.73
6	1972			5.1
7	1973			4.7
8	1974			5.2
9	1975			5.7


52 rows Limit results to 100 rows

+ New table

# @sqlrooms/ai

## Assistant



 show a time series of max quake magnitude by year

To create a time series of maximum earthquake magnitudes by year, I will extract the year from the DateTime column, calculate the maximum magnitude for each year, and then order the results by year.

```
SELECT strftime('%Y', DateTime) AS Year,  
MAX(Magnitude) AS MaxMagnitude  
FROM earthquakes  
GROUP BY Year  
ORDER BY Year;
```



 Show Query Result

I will create a line chart to visualize the maximum earthquake magnitudes by year, which will help in understanding trends over time.

What would you like to do?



# @sqlrooms/vega

The screenshot displays the @sqlrooms/vega web application interface. The left sidebar contains a navigation menu with sections for DATA SOURCES, FILES, SQL QUERIES, and TABLES. The 'flows' table is selected, showing columns for origin, dest, and count, with 676 rows. The main workspace is titled 'What would you like to learn about the data?' and contains a query editor with the following SQL query:

```
SELECT origin, SUM(count) as total_count FROM flows GROUP BY origin ORDER BY total_count DESC LIMIT 10
```

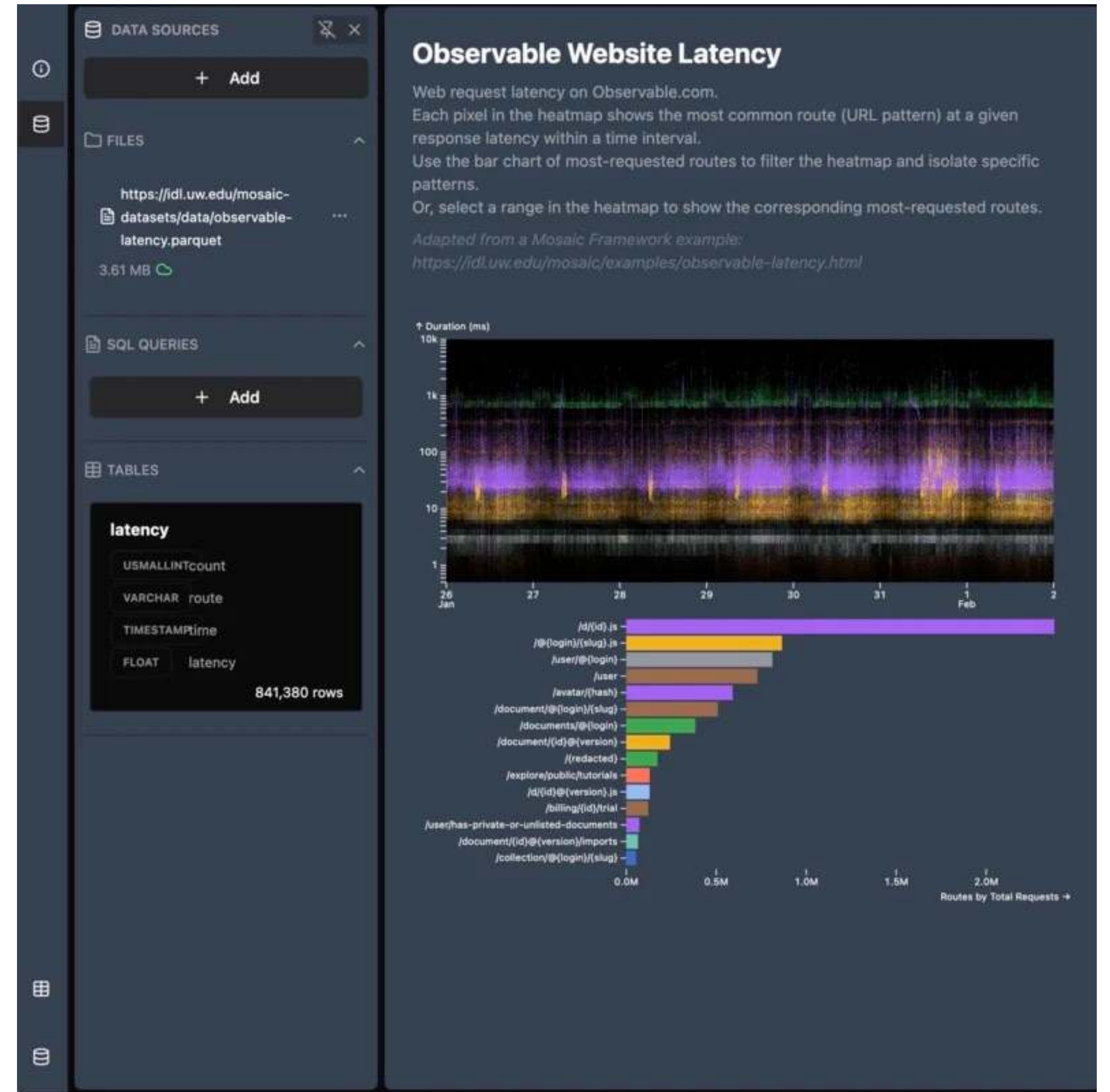
The query results are displayed in an 'answer' section, showing the top origins based on total count:

The top origins based on the total count are: 1. ZH with 89,370 2. BE with 64,344 3. VD with 58,276 4. AG with 44,662 5. GE with 34,616 6. SG with 28,851 7. LU with 24,849 8. FR with 20,605 9. TI with 19,402 10. VS with 19,180

A bar chart titled 'Top Origins by Total Count' visualizes this data. The Y-axis represents 'Total Count' (0 to 80k) and the X-axis represents 'Origin'. The bars show the total count for each origin, with ZH having the highest count at approximately 89,370.

Origin	Total Count
ZH	89,370
BE	64,344
VD	58,276
AG	44,662
GE	34,616
SG	28,851
LU	24,849
FR	20,605
TI	19,402
VS	19,180

# @sqlrooms/mosaic



# @sqlrooms/kepler

The screenshot displays the FSQ Spatial Desktop interface for 'California Earthquakes.spatial'. The interface is divided into several sections:

- Database Panel (Left):** Shows the 'California Earthquakes' database with a 'main' schema containing an 'earthquakes' table. The table schema includes: DateTime (timestamp), Latitude (double), Longitude (double), Depth (double), Magnitude (double), MagType (varchar), NbStations (integer), Gap (integer), Distance (integer), RMS (double), Source (varchar), and EventID (integer).
- Map View (Top Right):** A map of California showing earthquake locations as colored dots. The map is titled 'California Earthquakes' and includes a date range filter from '08/01/1967' to '11/03/2017'. Major cities like San Francisco, San Jose, Sacramento, Reno, and Chico are labeled. National parks like Death Valley National Park and the Nevada Test and Training Range (NTTR) are also visible.
- Bar Chart (Middle Right):** A bar chart showing the frequency of earthquakes over time, with the x-axis representing years from 1970 to 2010. The y-axis is labeled 'Y Axis'.
- SQL Query Editor (Bottom):** A query editor window titled 'Untitled' containing the following SQL query:

```
1 SELECT Magnitude, * FROM earthquakes WHERE Magnitude > 5
2 ORDER BY Magnitude DESC
```
- Table View (Bottom):** A table displaying the results of the SQL query, showing the top 3 earthquakes by magnitude. The table has columns for Magnitude, Float64, DateTime, Timestamp<MICROSECOND>, Latitude, Float64, Longitude, Float64, and Depth. The first three rows are:

	Magnitude	Float64	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth
1	7.39		1992-06-28T11:57:35.390Z		34.186		-116.46117		1
2	7.2		2010-04-04T22:40:42.000Z		32.259		-115.287		1
3	7.2		1980-11-08T10:27:33.200Z		41.08417		-124.61567		1

# @sqlrooms/cosmos

DATA SOURCES

FILES

publications-sample-1p...  
397.37 KB

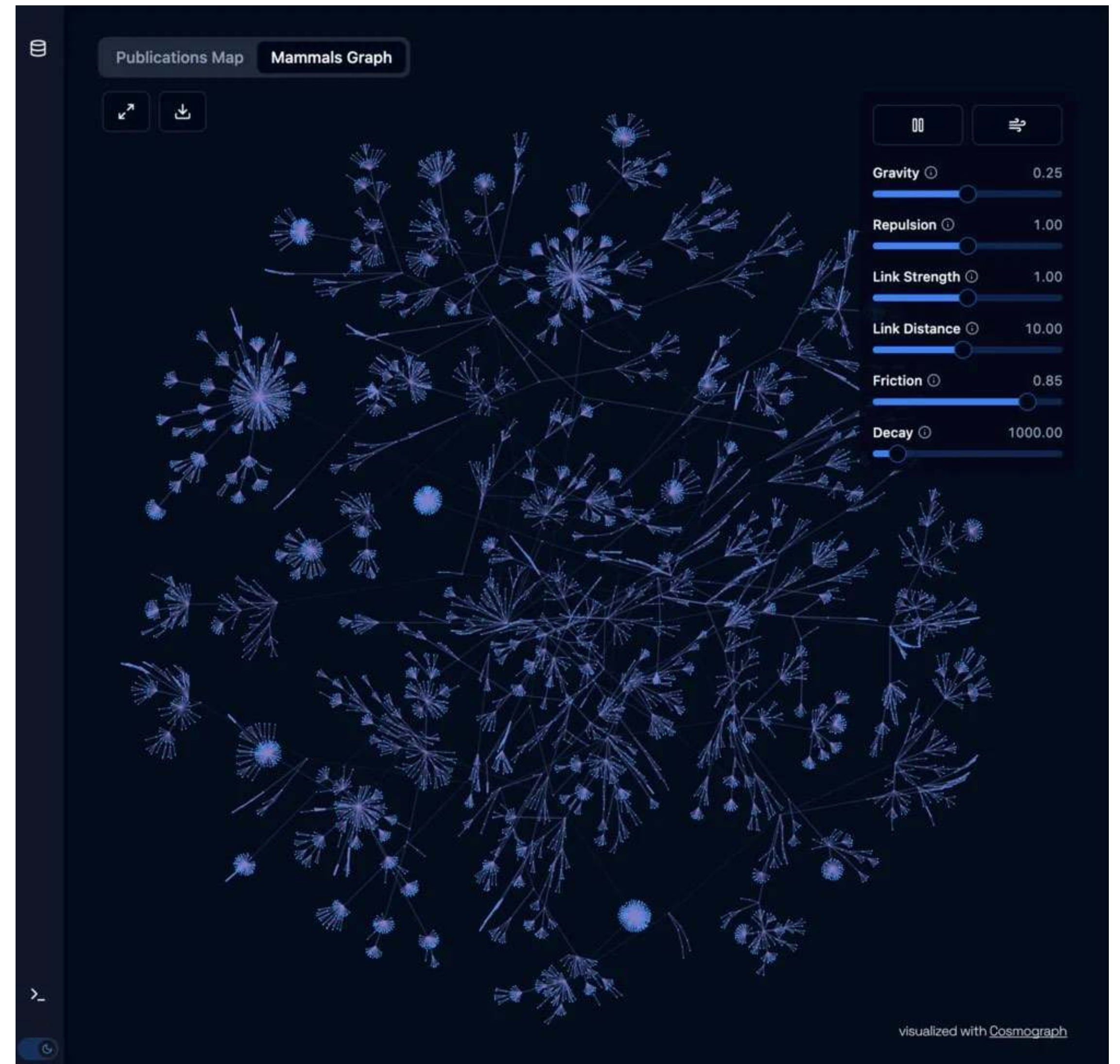
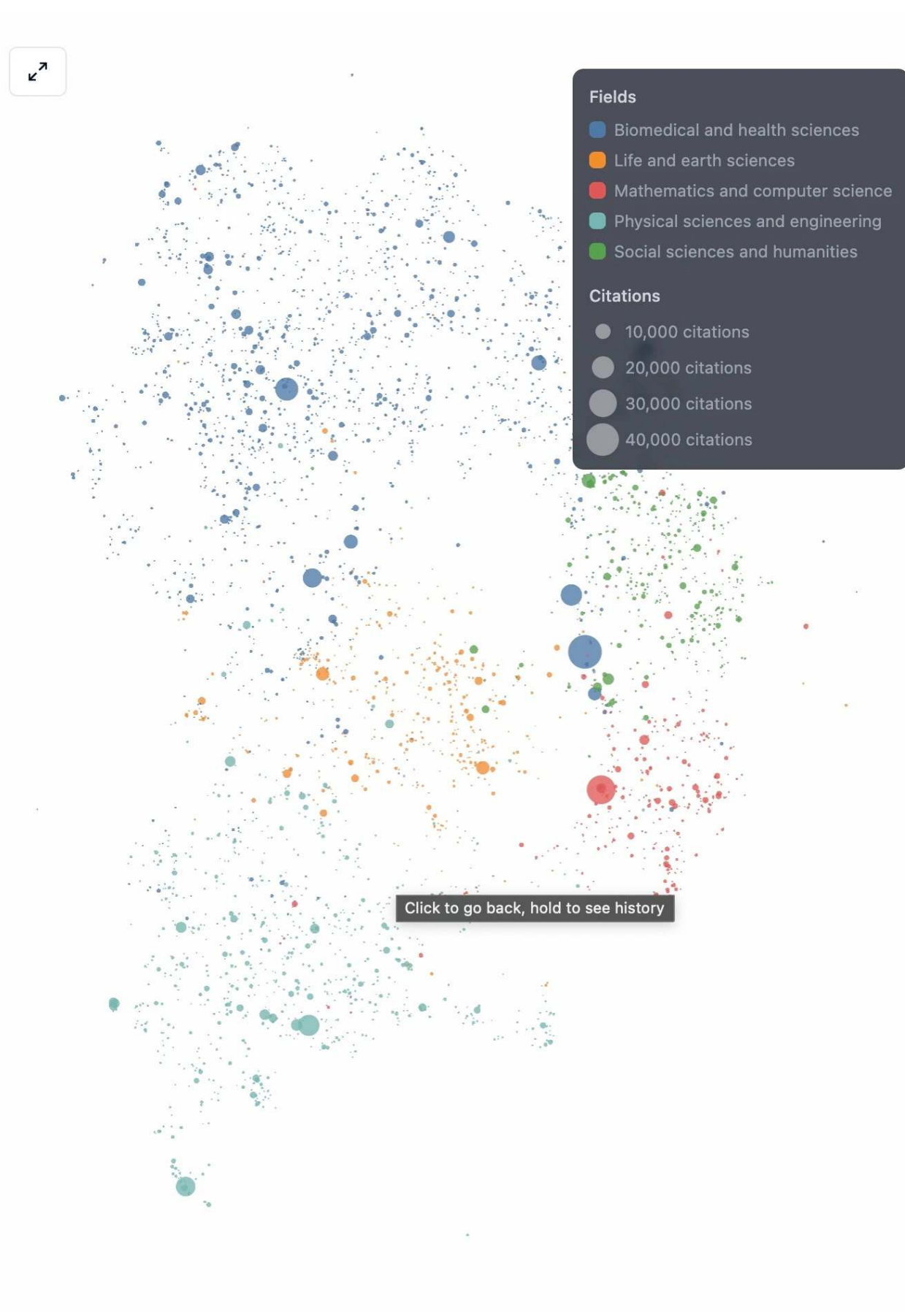
SQL QUERIES

+ Add

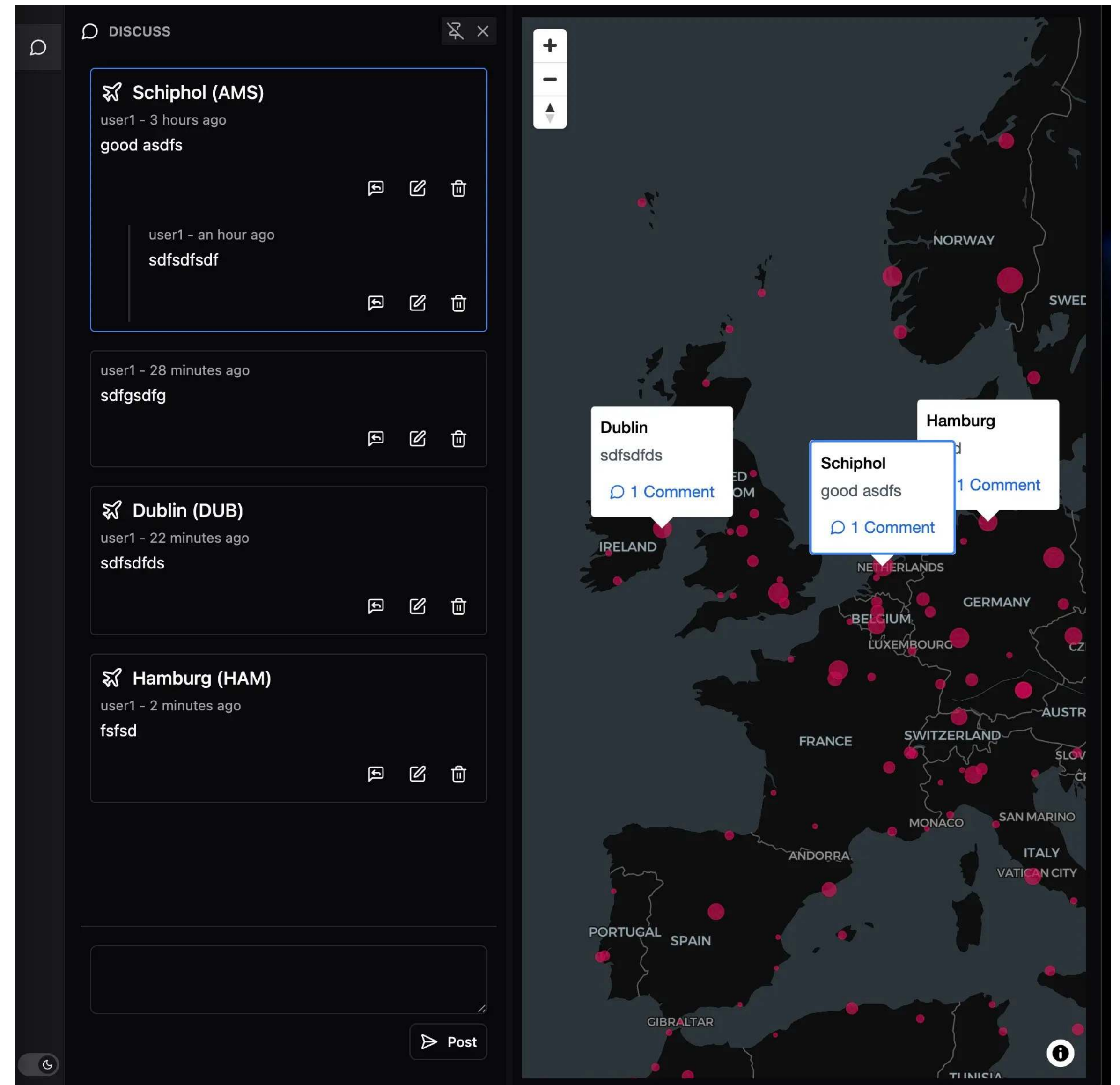
TABLES

publications	
BIGINT	id
VARCHAR	title
VARCHAR	source
DATE	pub_date
BIGINT	n_cits
BIGINT	micro_cluster
VARCHAR	main_field
DOUBLE	x
DOUBLE	y

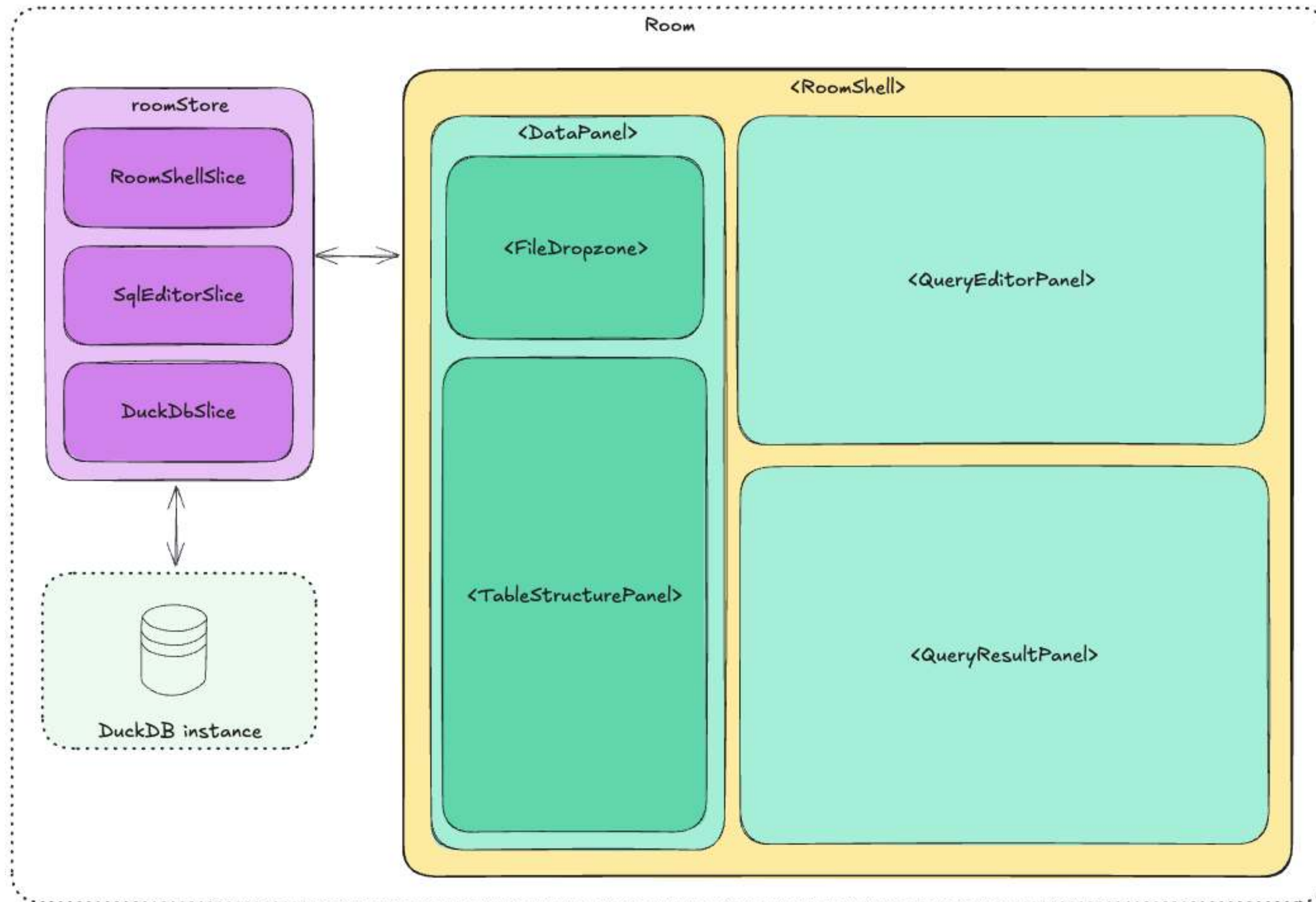
4,096 rows



# @sqlrooms/discuss



# Modular toolkit



The screenshot shows a web-based data application interface. On the left, a file explorer shows a folder named "gtfs\_de\_full" containing a "main" folder with various table names like "agency", "areas", "calendar", etc. The main area is split into a query editor and a query result panel. The query editor contains the following SQL code:

```
1 ATTACH 'https://pub-334685c2155547fab4287d84cae47083.r2.dev/gtfs-de/gtfs-de.duckdb' as gtfs_de_full (READ_ONLY);
2
3 SELECT * FROM gtfs_de_full.calendar;
4
5
6
```

The query result panel displays a table with 10 rows and 8 columns. The columns are: service\_id, Utf8, monday, Int32, tuesday, Int32, wednesday, Int32. The rows contain service IDs and their corresponding values for each day of the week.

	service_id	Utf8	monday	Int32	tuesday	Int32	wednesday	Int32
1	1854			0		0		0
2	1999			0		0		0
3	2004			0		0		0
4	2687			0		0		0
5	2892			0		0		0
6	2961			0		0		0
7	4062			0		0		0
8	4258			0		0		0
9	4317			0		0		0
10	4708			0		0		0

At the bottom, it shows "100 rows" and a "Limit results to 100 rows" dropdown menu.

# Room Store

```
const {roomStore, useRoomStore} = createRoomStore((set, get, store) => ({
  ...createRoomShellSlice({
    config: {
      dataSources: [
        {
          type: 'url',
          tableName: 'earthquakes',
          url: 'https://.../earthquakes.parquet',
        },
      ],
    },
  }))(set, get, store),
}));
```

```
export const MyRoom = () => (
  <RoomShell roomStore={roomStore}>
    <MyComponent />
  </RoomShell>
);
```

```
function MyComponent() {
  const isTableReady = useRoomStore((state) =>
    Boolean(state.db.findTableByName('earthquakes')));
  const queryResult = useSql<{maxMagnitude: number}>({
    query: `SELECT max(Magnitude) AS maxMagnitude FROM earthquakes`,
    enabled: isTableReady,
  });
  if (!isTableReady) return 'Loading...';
  const row = queryResult.data?.toArray()[0];
  return `Max earthquake magnitude: ${row?.maxMagnitude}`;
}
```

# Composing Store from Slices

```
const {roomStore, useRoomStore} = createRoomStore((set, get, store) => ({  
  // Default slice  
  ...createRoomShellSlice()(set, get, store),  
  
  // Mix in sql editor slice  
  ...createSqlEditorSlice()(set, get, store),  
});
```

```
const runQuery = useRoomStore((state) => state.sqlEditor.parseAndRunQuery);
```

# Adding Custom Store Function

```
const {roomStore, useRoomStore} = createRoomStore((set, get, store) => ({  
  
  ...createRoomShellSlice()(set, get, store),  
  ...createSqlEditorSlice()(set, get, store),  
  
  celebrateSuccess() {  
    const {queryResult} = get().sqlEditor;  
    if (queryResult?.status === 'success') {  
      set({hurray: true});  
    }  
  },  
  
});
```

# Adding Custom Store Function

```
const {roomStore, useRoomStore} = createRoomStore((set, get, store) => ({  
  
  ...createRoomShellSlice()(set, get, store),  
  ...createSqlEditorSlice()(set, get, store),  
  
  async celebrateSuccess() {  
    const {queryResult} = get().sqlEditor;  
    if (queryResult?.status === 'success') {  
      set({hurray: true});  
    }  
  },  
  
}));
```

# Room layout

```
...createRoomShellSlice({
  config: {
    layout: {
      type: LayoutTypes.enum.mosaic,
      nodes: {
        direction: 'row',
        first: 'data-sources',
        second: 'main',
        splitPercentage: 30,
      },
    },
  },
  room: {
    panels: {
      'data-sources': {
        title: 'Data Sources',
        icon: DatabaseIcon,
        component: DataSourcePanel,
        placement: 'sidebar',
      },
      'main': {
        title: 'Main view',
        icon: () => null,
        component: MainView,
        placement: 'main',
      },
    },
  },
})(set, get, store),
```

The screenshot shows a data exploration tool interface. On the left, there is a file browser for a database named 'gtfs\_de\_full'. The browser shows a 'main' folder containing various tables such as 'agency', 'areas', 'attribution', 'booking\_rules', 'calendar', 'calendar\_dates', 'fare\_attributes', 'fare\_leg\_join\_rules', 'fare\_leg\_rules', 'fare\_media', 'fare\_products', 'fare\_rules', 'fare\_transfer\_rules', 'feed\_info', 'frequencies', 'levels', 'location\_group\_stops', 'location\_groups', 'networks', 'pathways', 'rider\_categories', 'route\_networks', 'routes', 'shapes', 'stop\_areas', 'stop\_times', 'stops', 'timeframes', and 'transfers'. On the right, there is a SQL query editor with the following code:

```
1 ATTACH 'https://pub-334685c2155547fab4287d84cae47083.r2.dev/gtfs-de/
2 gtfs-de.duckdb' as gtfs_de_full (READ_ONLY);
3 SELECT * FROM gtfs_de_full.calendar;
4
5
6
```

Below the query editor, a table of results is displayed. The table has columns for 'service\_id', 'Utf8', 'monday', 'Int32', 'tuesday', 'Int32', 'wednesday', and 'Int32'. The first 10 rows are shown, with all 'Int32' values being 0.

	service_id	Utf8	monday	Int32	tuesday	Int32	wednesday	Int32
1	1054			0		0		0
2	1999			0		0		0
3	2004			0		0		0
4	2687			0		0		0
5	2892			0		0		0
6	2961			0		0		0
7	4062			0		0		0
8	4258			0		0		0
9	4317			0		0		0
10	4708			0		0		0

At the bottom of the table, it indicates '100 rows' and 'Limit results to 100 rows'.

# Room layout

```
export const Room = () => {
  return (
    <RoomShell className="h-screen" roomStore={roomStore}>
      <RoomShell.Sidebar>
        <ThemeSwitch />
      </RoomShell.Sidebar>
      <RoomShell.LayoutComposer />
      <RoomShell.LoadingProgress />
    </RoomShell>
  );
};
```

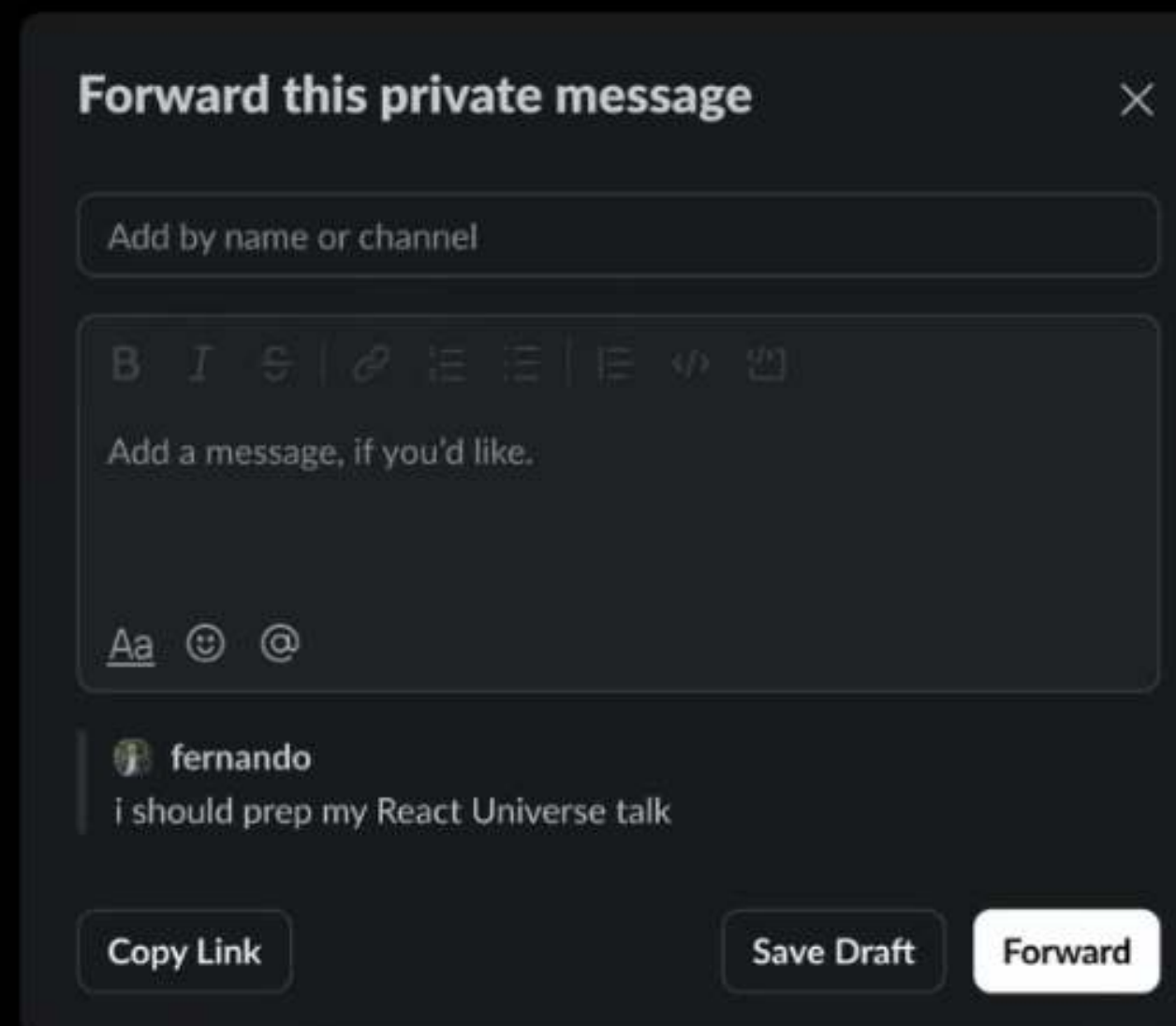
The screenshot shows a data browser interface with the following components:

- File Upload Section:** A box with a plus icon and the text "Add files". Below it, it lists "Supported formats: .csv, .tsv, .parquet, .json" and a note: "Files you add will stay local to your browser."
- Table of Contents:** A tree view showing a folder named "gtfs\_de\_full" containing a sub-folder "main" with various tables like "agency", "areas", "attribution", etc.
- SQL Query Editor:** A text area with a "Run" button. The query is:

```
1 ATTACH 'https://pub-334685c2155547fab4287d84cae47083.r2.dev/gtfs-de/gtfs-de.duckdb' as gtfs_de_full (READ_ONLY);
2
3 SELECT * FROM gtfs_de_full.calendar;
4
5
6
```
- Data Table:** A table with columns: "service\_id", "Utf8", "monday", "Int32", "tuesday", "Int32", "wednesday", "Int32". It contains 10 rows of data.
- Footer:** A status bar showing "100 rows" and a dropdown menu "Limit results to 100 rows".

# Composition is all you need

```
function ForwardMessageDialog() {  
  return (  
    <ForwardMessageProvider>  
      <ForwardMessageComposer />  
  
      <MessagePreview />  
  
      <Actions>  
        <CopyMessageLink />  
        <SaveDraftButton />  
        <ForwardButton />  
      </Actions>  
    </ForwardMessageProvider>  
  )  
}
```



FERNANDO ROJO  
VERCEL


COMPOSITION  
IS ALL YOU NEED

# AI slice + tools

```
...createAiSlice({
  tools: {
    ...createDefaultAiTools(store, {query: {}}),
    chart: createVegaChartTool(),
    echo: {
      description: 'A simple echo tool that returns the input text',
      parameters: z.object({
        text: z.string().describe('The text to echo back'),
      }),
      execute: async ({text}: {text: string}) => {
        return {
          llmResult: {
            success: true,
            details: `Echo: ${text}`,
          },
        };
      },
      component: EchoToolResult,
    },
  },
})(set, get, store),
```

## Assistant



 show a time series of max quake magnitude by year

To create a time series of maximum earthquake magnitudes by year, I will extract the year from the DateTime column, calculate the maximum magnitude for each year, and then order the results by year.

```
SELECT strftime('%Y', DateTime) AS Year,
MAX(Magnitude) AS MaxMagnitude
FROM earthquakes
GROUP BY Year
ORDER BY Year;
```



 Show Query Result

I will create a line chart to visualize the maximum earthquake magnitudes by year, which will help in understanding trends over time.

What would you like to do?



FSQ Spatial Desktop - Untitled.spatial
AI ASSISTANT

**DATABASE** Status: Connected

+ Add data

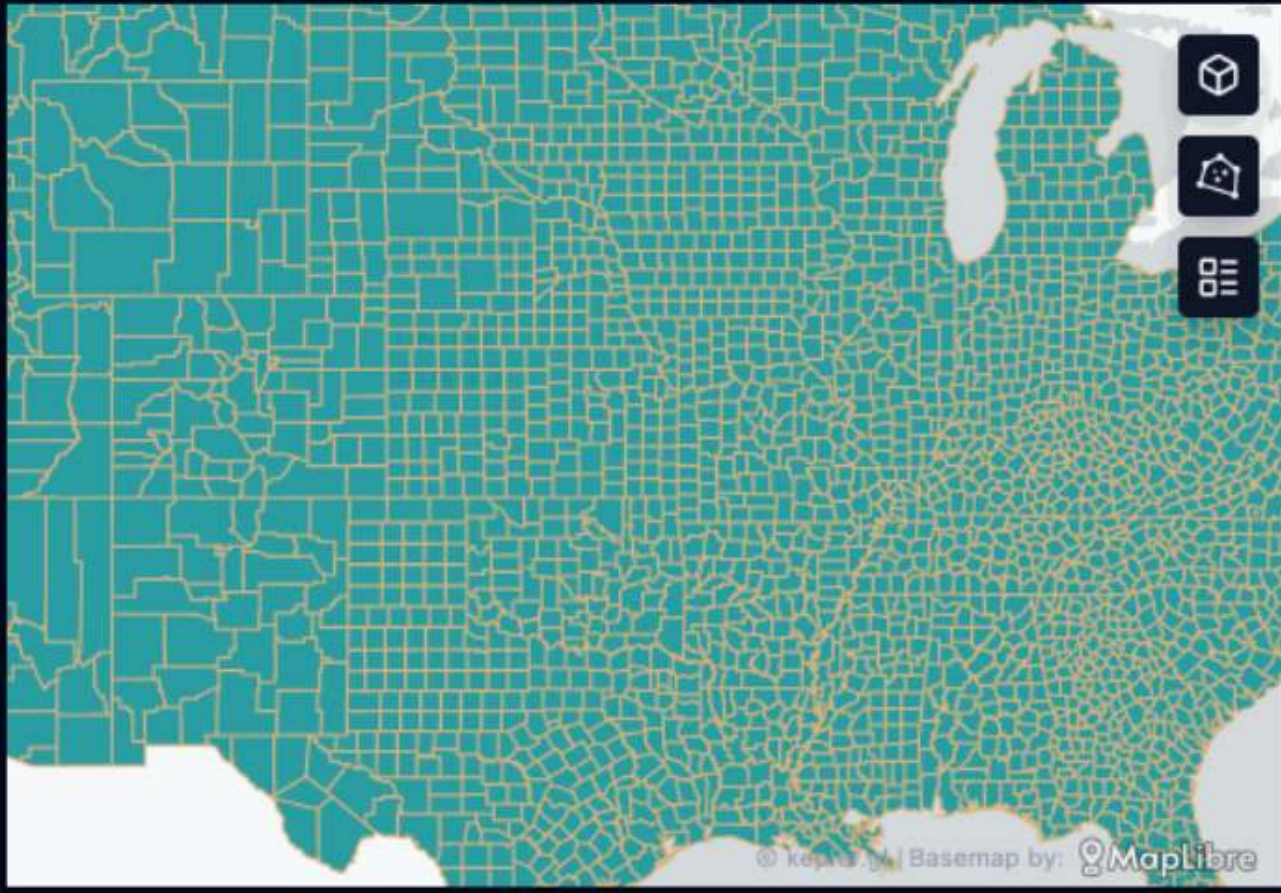
+ Quick file drop  
Supported formats: csv, tsv, parquet, json, arrow, shp, kml

**TABLES**

- Untitled
- main
  - natregimes 3,085 rows

**TILESETS**

Untitled Map



Untitled : orrelation\_HR60\_UE60

Run

```

1 SELECT STATE_NAME, corr(HR60, UE60) AS correlation,
2   regr_slope(HR60, UE60) AS slope, regr_intercept(HR60,
3   UE60) AS intercept
4 FROM natregimes
5 WHERE STATE_NAME IN ('California', 'Arizona') AND HR60 IS
6   NOT NULL AND UE60 IS NOT NULL
7 GROUP BY STATE_NAME;

```

	STATE_NAME	Utf8	correlation	Float64	slope	Flo
1	California		0.3201930656099818		0.5279757225818	
2	Arizona		0.5126066287349511		0.7082142262046	

2 rows Limit results to 100 rows

+ New table

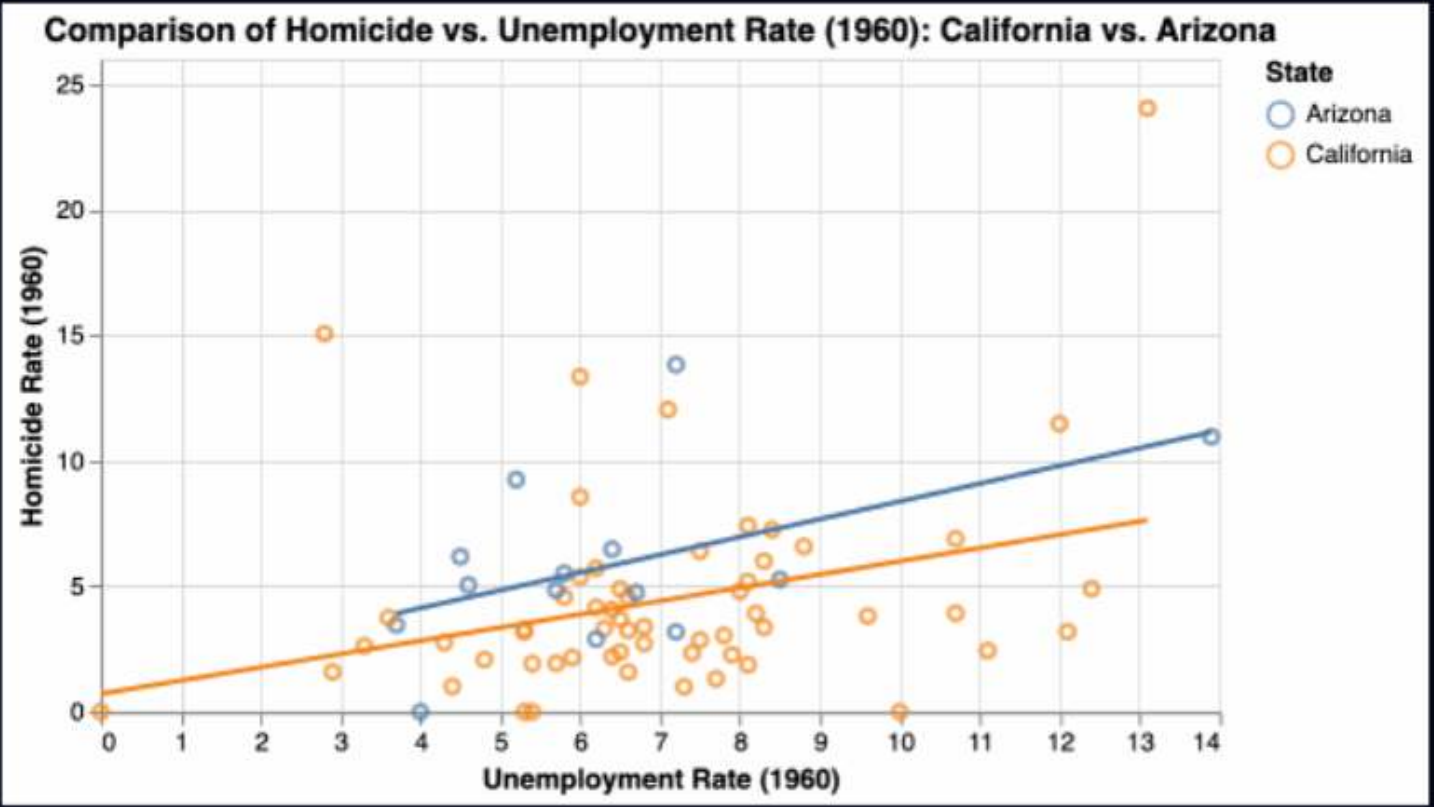
Default Session gpt-4.1 + New Session

```

FROM natregimes
WHERE STATE_NAME IN ('California', 'Arizona') AND UE60 IS NOT NULL
AND HR60 IS NOT NULL;

```

Show Query Result: Query



The comparison plot is ready! It shows:

- Each county in California and Arizona as a point, colored by state.
- A separate regression line for each state, reflecting their unique trends.

This visualization makes it easy to see that:

- Arizona's points and regression line show a stronger positive relationship between unemployment and homicide rates compared to California.
- California's trend is positive but less steep.

You can view this plot in the chart component to explore the differences visually. If you'd like to compare more states or add more details, just let me know!

Type here what would you like to learn about the data? Something like 'What is the max magnitude of the earthquakes by year?'

gpt-4.1

FSQ Spatial Desktop - Untitled.spatial

**DATABASE** ✕

Status: Connected

+ Add data

+ Quick file drop

Supported formats:  
csv, tsv, parquet, json, arrow, shp, kml

**TABLES** ^

- ▼ Untitled
- ▼ main
- > natregimes 3,085 rows
- > natregimes\_hr60pop60 3,085 rows

**TILESETS** ^

Untitled Map +

Untitled : orrelation\_HR60\_UE60

\_Line\_Parameters\_HR  
AZ\_HR60\_UE60\_Correc  
\_HR60\_per\_PO60\_Co

▶ Run ⌘↵

```

1 SELECT *, CASE WHEN P060 > 0 THEN HR60 * 100000.0 / P060
ELSE NULL END AS HR60_per_PO60 FROM natregimes;

```

	REGIONS	Float64	NOSOUTH	Float64	POLY_ID	Int32
1		1		1		1
2		2		1		2
3		2		1		3
4		2		1		4
5		2		1		5

100 rows Limit results to 100 rows ▼

+ New table

**AI ASSISTANT** ✕

Default Session gpt-4.1 + New Session ⚙️

I here was an error: the data classification tool could not find the HR60\_per\_PO60 column because it only exists in the result of the previous query, not in the original table.

To fix this, I need to save the query result (with the new HR60\_per\_PO60 column) as a temporary table, and then run the quantile classification on that table. Would you like me to proceed with this approach?

yes please

saveQueryToTable { "tableName": "natregimes\_hr60pop60", "sql..."

dataClassifyTool { "datasetName": "natregimes\_hr60pop60", "v..."

addMapLayer { "tableName": "natregimes\_hr60pop60", "lay..."

Your quantile map for HR60/PO60 (homicide rate per 100,000 population in 1960) is ready, using a color palette inspired by Van Gogh's Starry Night! Here's how it works:

- Counties are colored by their homicide rate per population, divided into 5 quantile bins.
- The colors range from deep blue (#233d7b), through lighter blues (#355ca1, #4e8fc0), to bright yellow (#f7d842) and gold (#f7a900), echoing the swirling night sky and stars in Starry Night.

What this map shows:

- Counties with the lowest rates are deep blue, while those with the highest rates shine in yellow/gold.
- This makes it easy to spot geographic patterns and outliers in homicide rates across the US in 1960.

You can view and interact with this map layer in the map component. If you want to adjust the color scheme or number of bins, just let me know!

Type here what would you like to learn about the data? Something like 'What is the max magnitude of the earthquakes by year?'

gpt-4.1 ⬆️

**Coming next**

# @sqlrooms/notebook

DATA

+ Add files

Supported formats:  
.csv, .tsv, .parquet, .json  
Files you add will stay local to your browser.

memory

main

earthquakes 5

- DateTime
- Latitude
- Longitude
- Depth
- Magnitude
- MagType
- NbStations
- Gap
- Distance
- RMS
- Source
- EventID
- geom

SQL

```
SELECT strftime('%Y', DateTime) AS Year, MAX(Magnitude) AS MaxMagnitude  
FROM earthquakes GROUP BY Year ORDER BY Year
```

	Year	Utf8	MaxMagnitude	Float64
1	1967			3.6
2	1968			4.3
3	1969			5.7
4	1970			4.7
5	1971			4.73
6	1972			5.1
7	1973			4.7
8	1974			5.2
9	1975			5.7
10	1976			6.3

SQL

```
SELECT strftime('%Y', DateTime) AS Year, MAX(Magnitude) AS MaxMagnitude  
FROM earthquakes GROUP BY Year ORDER BY Year
```

Page 1 of 6 | 10 rows | 52 rows | Refreshed 11 minutes ago | + New table

Notebook 1 Notebook 2 +

Add new Run all Show input bar

SQL

```
SELECT strftime('%Y', DateTime) AS Year, MAX(Magnitude) AS MaxMagnitude  
FROM earthquakes GROUP BY Year ORDER BY Year
```

	Year	Utf8	MaxMagnitude	Float64
1	1967			3.6
2	1968			4.3
3	1969			5.7
4	1970			4.7
5	1971			4.73
6	1972			5.1
7	1973			4.7
8	1974			5.2
9	1975			5.7
10	1976			6.3

Page 1 of 6 | 10 rows | 52 rows | Refreshed 14 minutes ago | + New table

Chart

SQL

Type Line

General Styles

X-Axis Year

Y-Axis MaxMagnitude Mean

Year	Mean of MaxMagnitude
1967	3.6
1968	4.3
1969	5.7
1970	4.7
1971	4.73
1972	5.1
1973	4.7
1974	5.2
1975	5.7
1976	6.3
1977	5.1
1978	5.2
1979	5.5
1980	6.3
1981	5.5
1982	5.5
1983	6.7
1984	6.7
1985	5.7
1986	6.7
1987	6.0
1988	5.5
1989	6.9
1990	6.3
1991	7.0
1992	7.4
1993	6.4
1994	7.0
1995	6.7
1996	5.7
1997	5.5
1998	7.0
1999	5.7
2000	5.7
2001	5.3
2002	5.3
2003	6.5
2004	6.0
2005	7.0
2006	5.0
2007	5.7
2008	6.0
2009	5.2
2010	7.2
2011	5.3
2012	6.2
2013	5.7
2014	6.8
2015	6.5
2016	6.5
2017	5.7
2018	5.7

+ Add files

Supported formats: .csv, .tsv, .parquet, .json  
Files you add will stay local to your browser.

- memory
- canvas
  - Query 1 54,936 rows
  - Query 2 54,936 rows
  - Query 1 54,936 rows
  - Query 2 54,936 rows
- main
  - earthquakes 54,936 rows

Query

```
SELECT * FROM earthquakes
```

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T08:42:24.000Z		31.34		-115.379		9.21	
3	2013-12-20T05:27:12.000Z		31.431		-115.775		14.485	
4	2017-03-17T20:00:00.000Z		31.245		-115.692		12	
5	2010-06-25T23:23:15.000Z		31.296		-115.567		4.348	
6	2014-03-05T02:24:23.000Z		31.323		-119.409		6.735	
7	2012-12-14T11:00:55.000Z		31.48567		-119.51366		5.8	

Query 1

```
SELECT * FROM canvas."Query"
```

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T08:42:24.000Z		31.34		-115.379		9.21	
3	2013-12-20T05:27:12.000Z		31.431		-115.775		14.485	
4	2017-03-17T20:00:00.000Z		31.245		-115.692		12	
5	2010-06-25T23:23:15.000Z		31.296		-115.567		4.348	
6	2014-03-05T02:24:23.000Z		31.323		-119.409		6.735	
7	2012-12-14T11:00:55.000Z		31.48567		-119.51366		5.8	

Query 2

```
SELECT * FROM canvas."Query 1"
```

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T08:42:24.000Z		31.34		-115.379		9.21	
3	2013-12-20T05:27:12.000Z		31.431		-115.775		14.485	
4	2017-03-17T20:00:00.000Z		31.245		-115.692		12	
5	2010-06-25T23:23:15.000Z		31.296		-115.567		4.348	
6	2014-03-05T02:24:23.000Z		31.323		-119.409		6.735	
7	2012-12-14T11:00:55.000Z		31.48567		-119.51366		5.8	

### Assistant

show a time series of max quake magnitude by year

To create a time series of maximum earthquake magnitudes by year, I will extract the year from the DateTime column, calculate the maximum magnitude for each year, and then order the results by year.

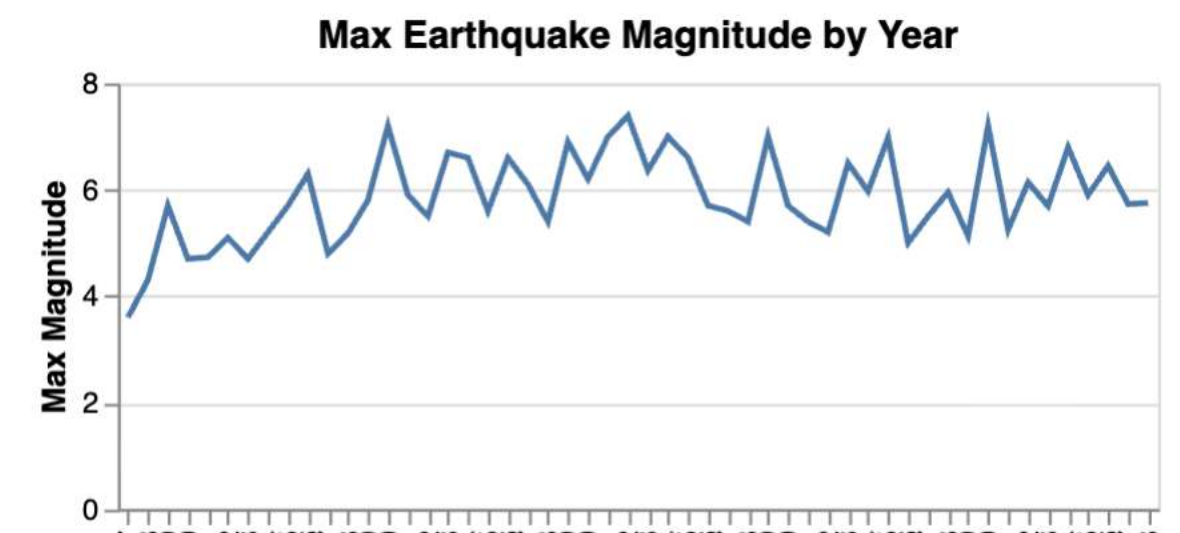
```
SELECT strftime('%Y', DateTime) AS Year,
MAX(Magnitude) AS MaxMagnitude
FROM earthquakes
GROUP BY Year
ORDER BY Year;
```

Show Query Result

I will create a line chart to visualize the maximum earthquake magnitudes by year, which will help in understanding trends over time.

```
SELECT strftime('%Y', DateTime) AS Year,
MAX(Magnitude) AS MaxMagnitude
FROM earthquakes
GROUP BY Year
ORDER BY Year;
```

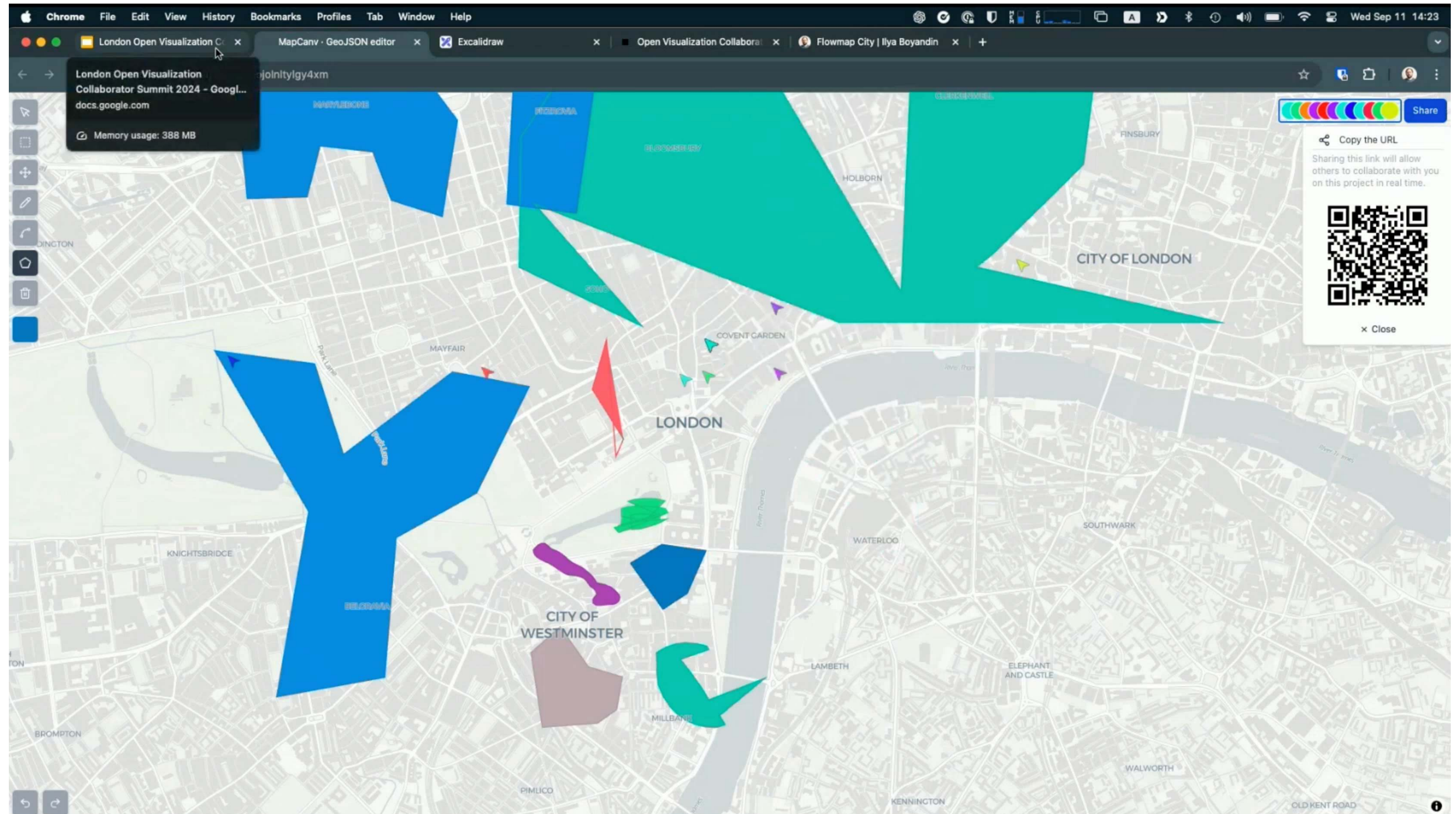
Show Query Result



What would you like to do?

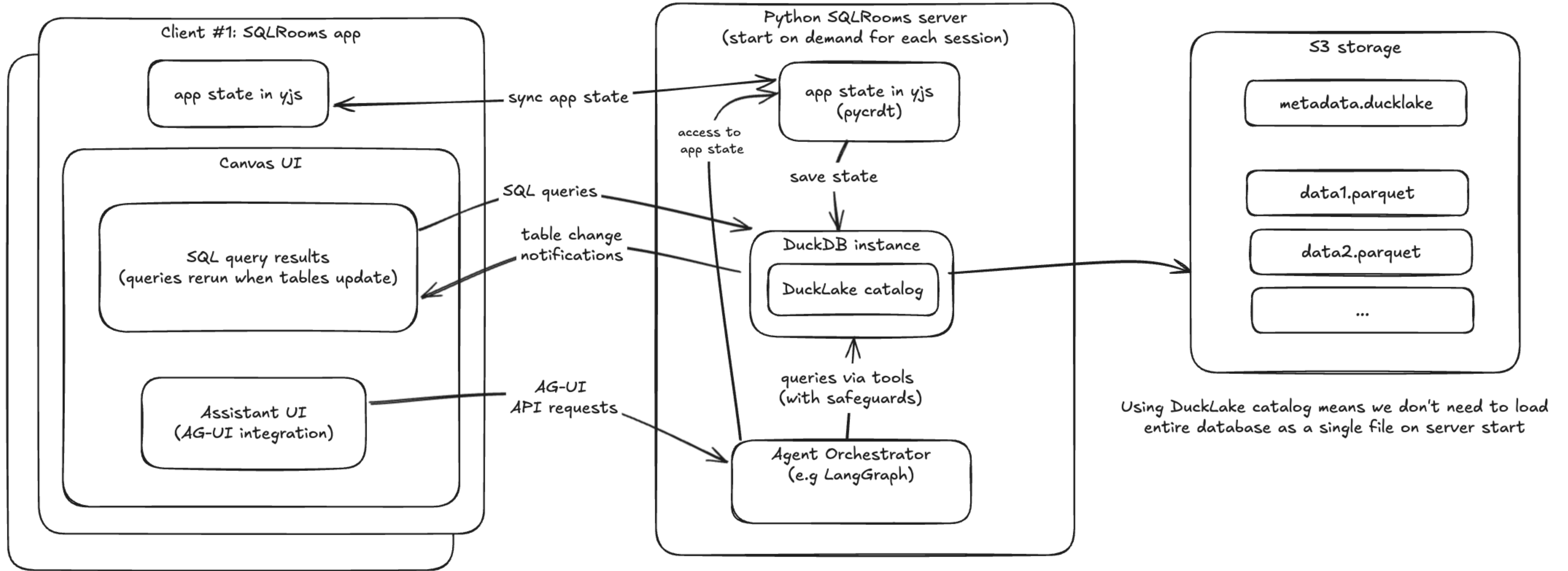
# @sqlrooms/canvas

# Realtime Collaboration



# Humans + agents collaborative analytics with SQLRooms

- Shared data canvas (via DuckDB + DuckLake + Yjs sync)
- Multi-user collaboration (humans)
- Multi-agent collaboration (LLM agents with tool access + shared state)
- Seamless UI control with Human-in-the-loop (AG-UI front-end layer)



# SQLRooms AI App Builder

Some of the building blocks

AI assistant, Code editor

WebContainer

WASM micro-OS (by StackBlitz)

Sandboxed Node.js environment

Not open-source, but free for non-commercial use

Default Session gpt-4.1

+ New Session

- Files
  - src
    - App.js
    - main.j
    - index.ht
    - package
    - vite.conf

App.jsx main.jsx index.html package.json

```
1 import {createDuckDbSlice, useSql} from '@sqlrooms
2 import {
3   createRoomSlice,
4   createRoomStore,
5   RoomStateProvider,
6 } from '@sqlrooms/room-store';
7
8 const {roomStore, useRoomStore} = createRoomStore(
9   ...createRoomSlice()(set, get, store),
10  ...createDuckDbSlice({})(set, get, store),
11  ));
12
13 function MyRoom() {
14   const queryResult = useSql({
15     query: `SELECT 'Hello, world!' as message`,
16   });
17   const row = queryResult.data?.toArray()[0];
18   return row ? `Message: ${row.message}` : 'Loading...';
19 }
20
21 export default function App() {
22   return (
23     <div style={{fontFamily: 'system-ui', padding: '10px'}}>
24       <h1>SQLRooms in WebContainer</h1>
25       <RoomStateProvider roomStore={roomStore}>
26         <MyRoom />
27       </RoomStateProvider>
28     </div>
29   );
30 }
31
```

# SQLRooms in WebContainer

Message: Hello, world!

Terminal ready

```
added 107 packages in 23s
12 packages are looking for funding
run `npm fund` for details
```

What would you like to change in the app?'

gpt-4.1



## Introduction

[Overview](#)[Key Concepts](#)[Modular Architecture](#)

## Developer Guide

[Getting Started](#)[State Management](#)[Query Cancellation](#)[Theming](#)[Offline Use](#)[Upgrade Guide](#)

## Examples

[Example Apps](#)[Case Studies](#)

## Reference

[Docs for LLMs](#)[Core Packages](#)[ai](#)[duckdb](#)

# Docs for LLMs

This page provides access to machine-readable documentation files for Large Language Models and AI assistants supporting the [llms.txt convention](#).

## [/llms.txt](#)

A concise version of the SQLRooms documentation that includes links to additional resources.

```
https://sqlrooms.org/llms.txt
```

## [/llms-full.txt](#)

A comprehensive version containing the complete SQLRooms documentation and API reference concatenated into a single file.

```
https://sqlrooms.org/llms-full.txt
```

### On this page

[/llms.txt](#)[/llms-full.txt](#)

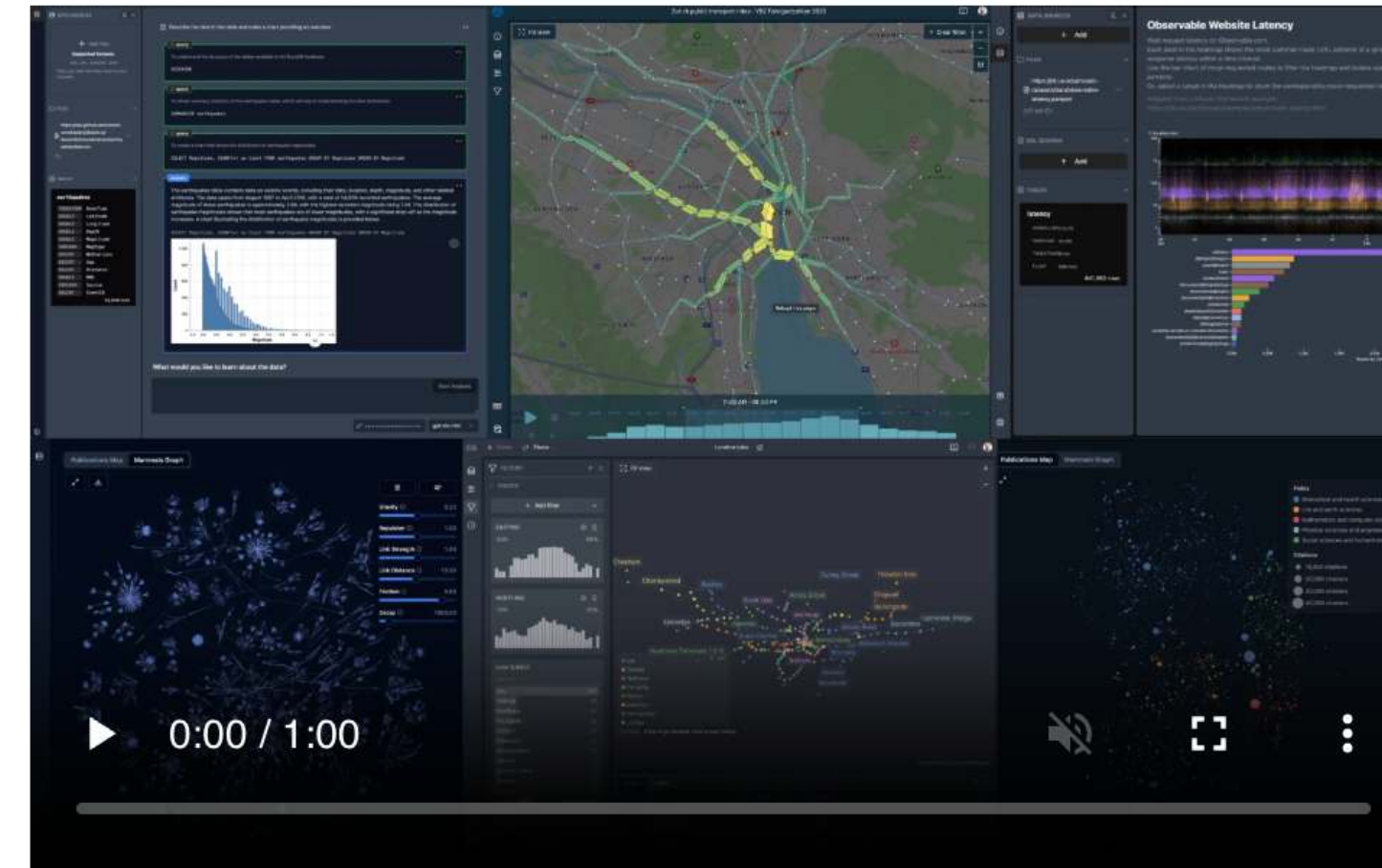
Package-level documentation

[@sqlrooms/ai](#)[@sqlrooms/duckdb](#)[@sqlrooms/room-shell](#)[@sqlrooms/ui](#)[@sqlrooms/sql-editor](#)[@sqlrooms/schema-tree](#)

# SQLRooms

## Build data-centric apps with DuckDB

An open source React toolkit for human + agent collaborative analytics apps

[What is SQLRooms?](#)[Example Apps](#)[View on GitHub](#)

### Local Analytics, No Backend Required

Leverage DuckDB's powerful SQL capabilities, enabling fast in browser data processing without a backend

### Own Your Data

Data remains on your local device for maximum privacy, sub-second analytics on large datasets, and offline functionality

### Privacy-Preserving AI Integration

Use agents that can write and execute SQL queries, and generate insights without sharing your data with model providers

### Modular Architecture

Pick and choose the functionality you need for composable, extensible applications, with integrations for popular data visualization libraries.

### Modern UI Components

Comprehensive set of React components including data tables, layouts, and visualization tools for building beautiful analytics interfaces

### Offline Use

Work with your data, run queries, and analyze results even without an internet connection. SQLRooms supports offline workflows using persistent storage via OPFS.