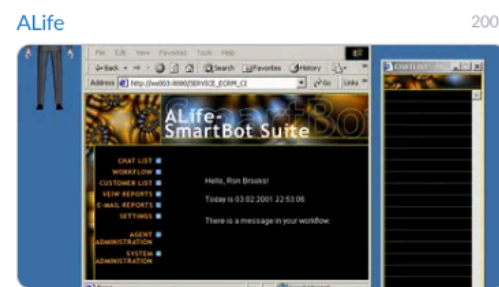
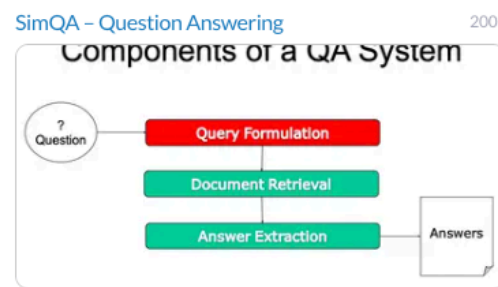
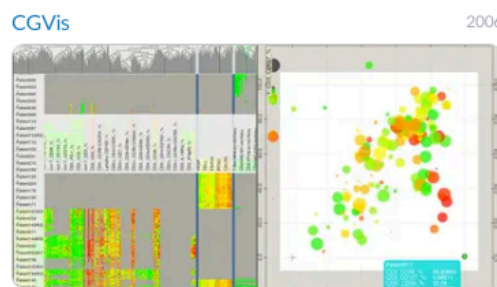
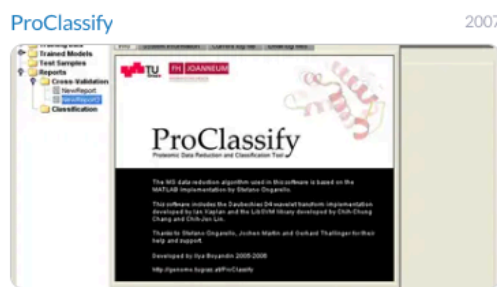
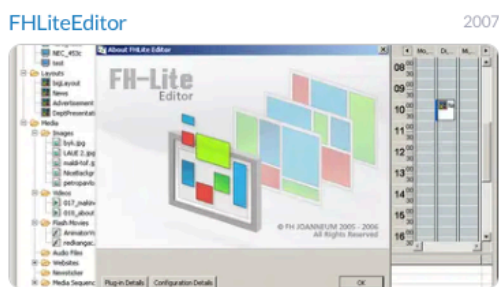
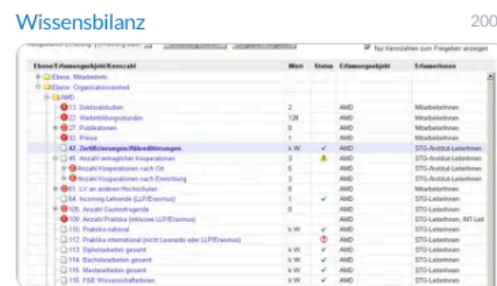
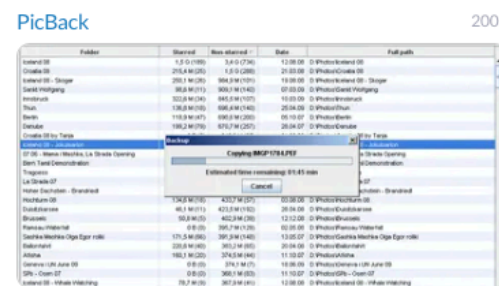
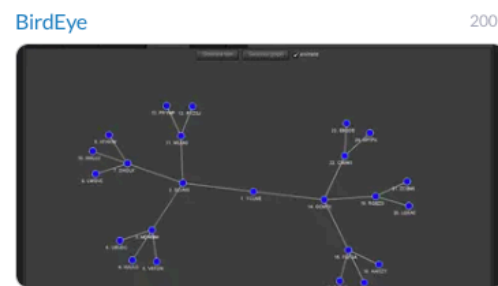
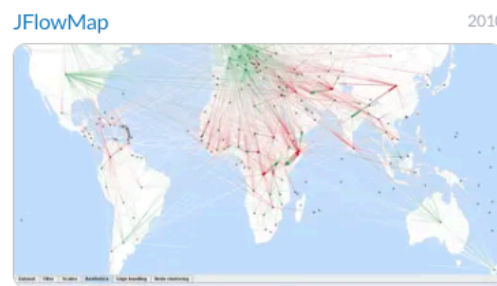
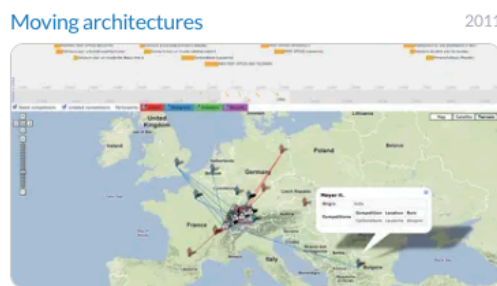
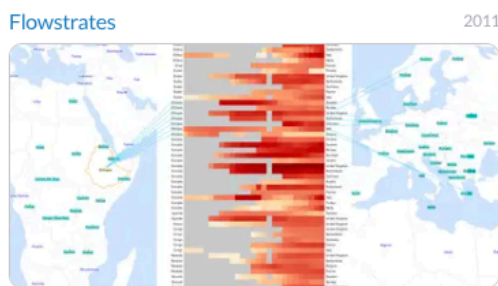
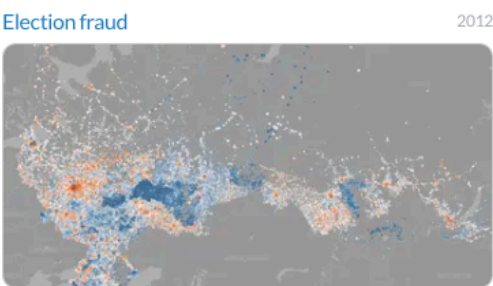
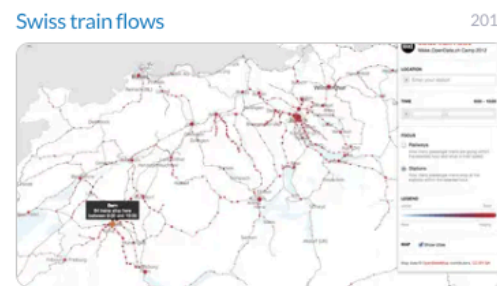
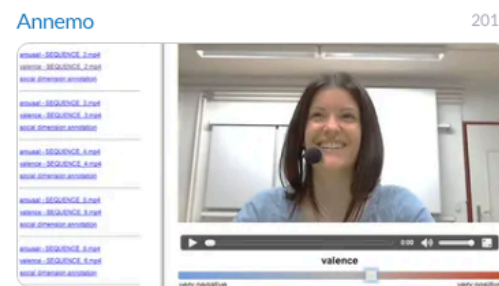
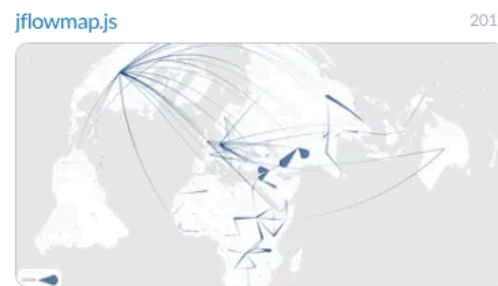
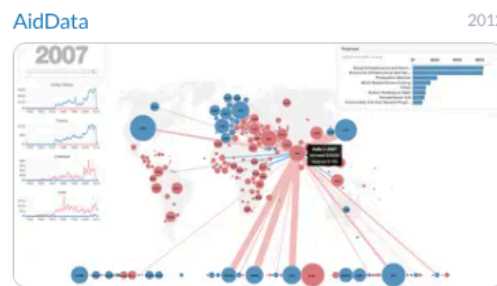
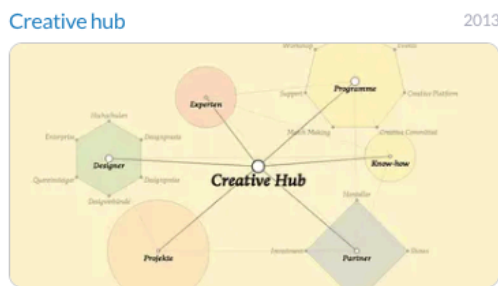
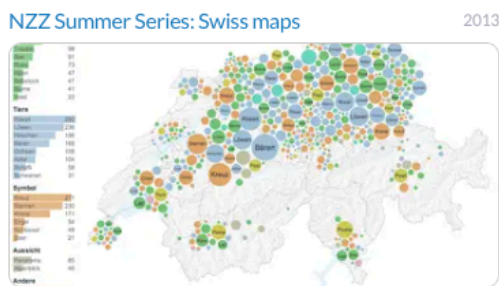
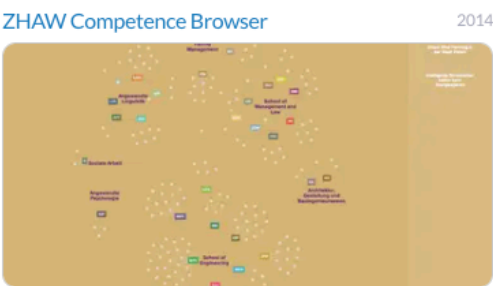
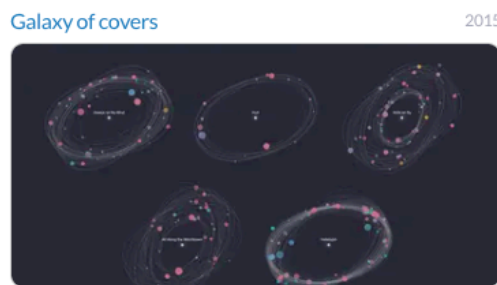
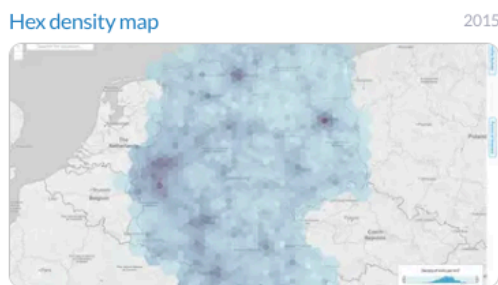
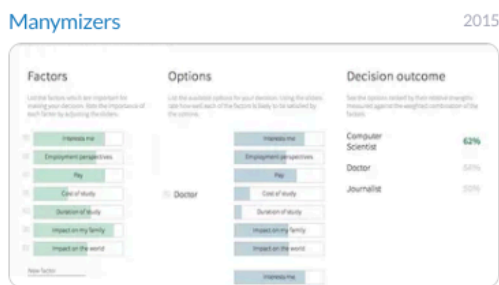
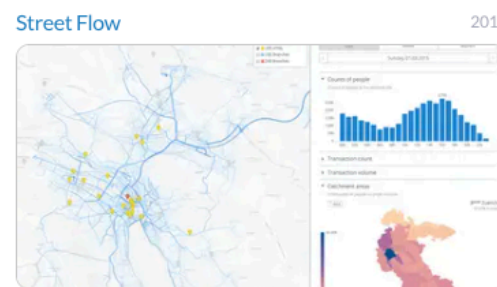
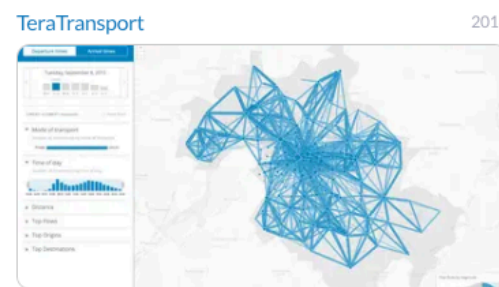
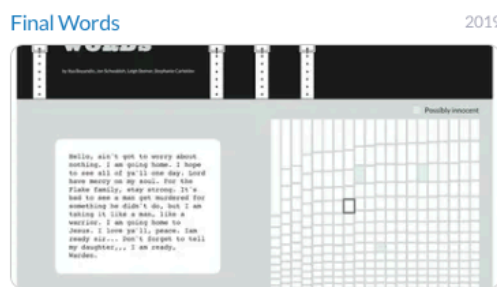
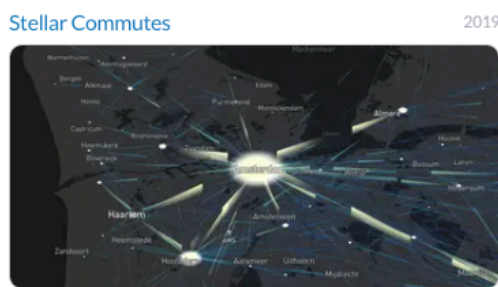
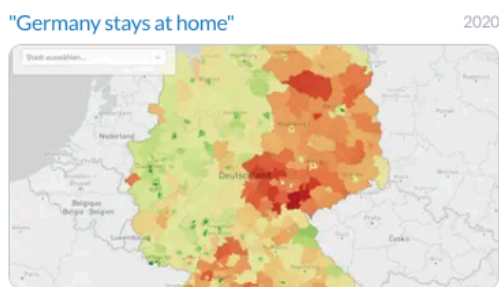
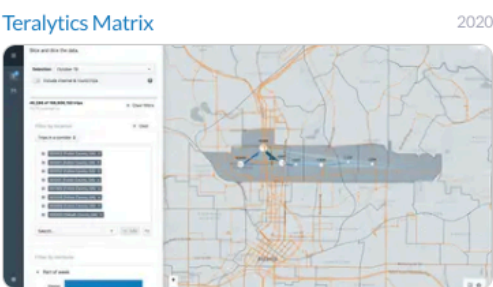
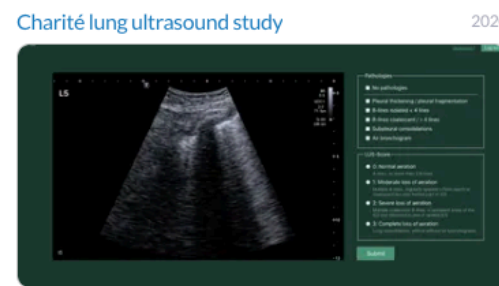
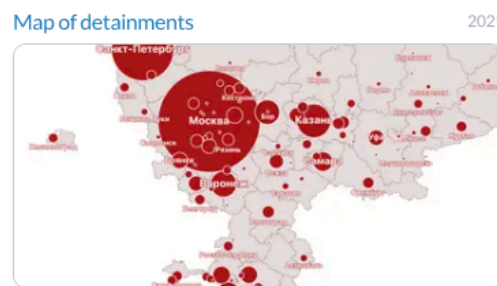
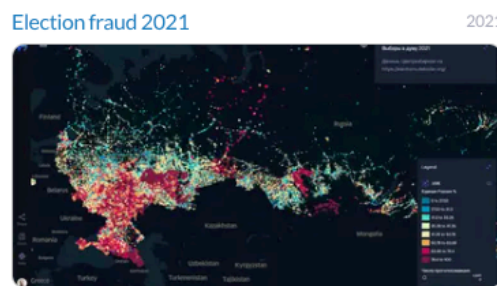
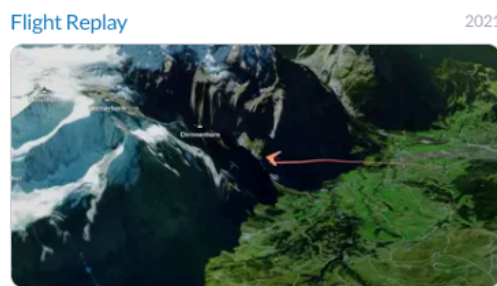
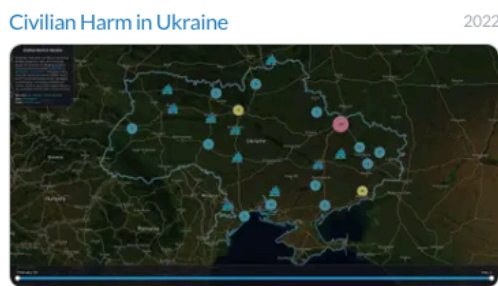
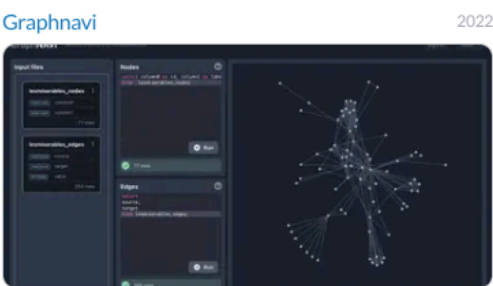
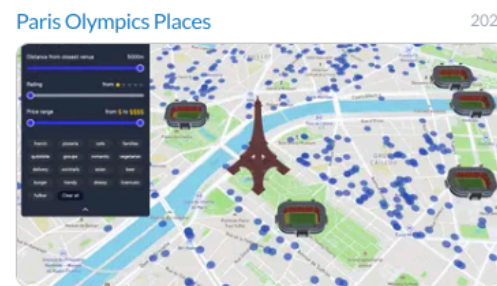
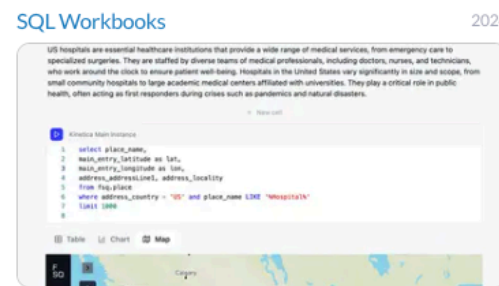
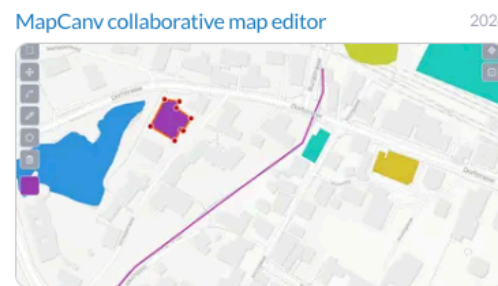
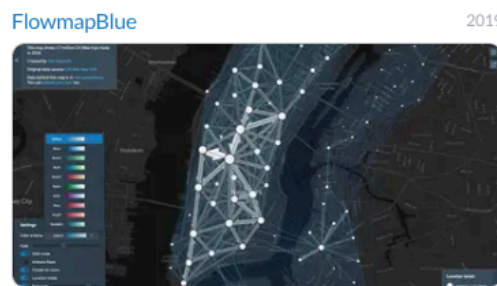
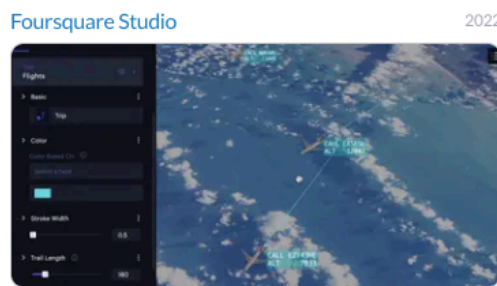
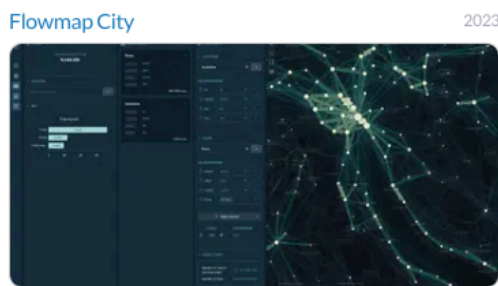
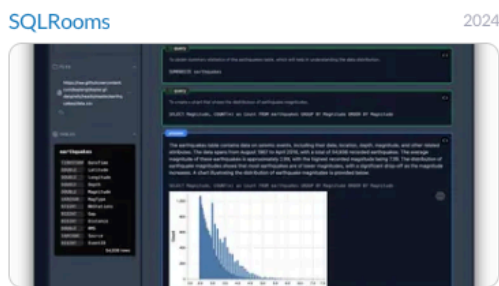


SQLRooms

Toolkit for human + agent data analytics apps

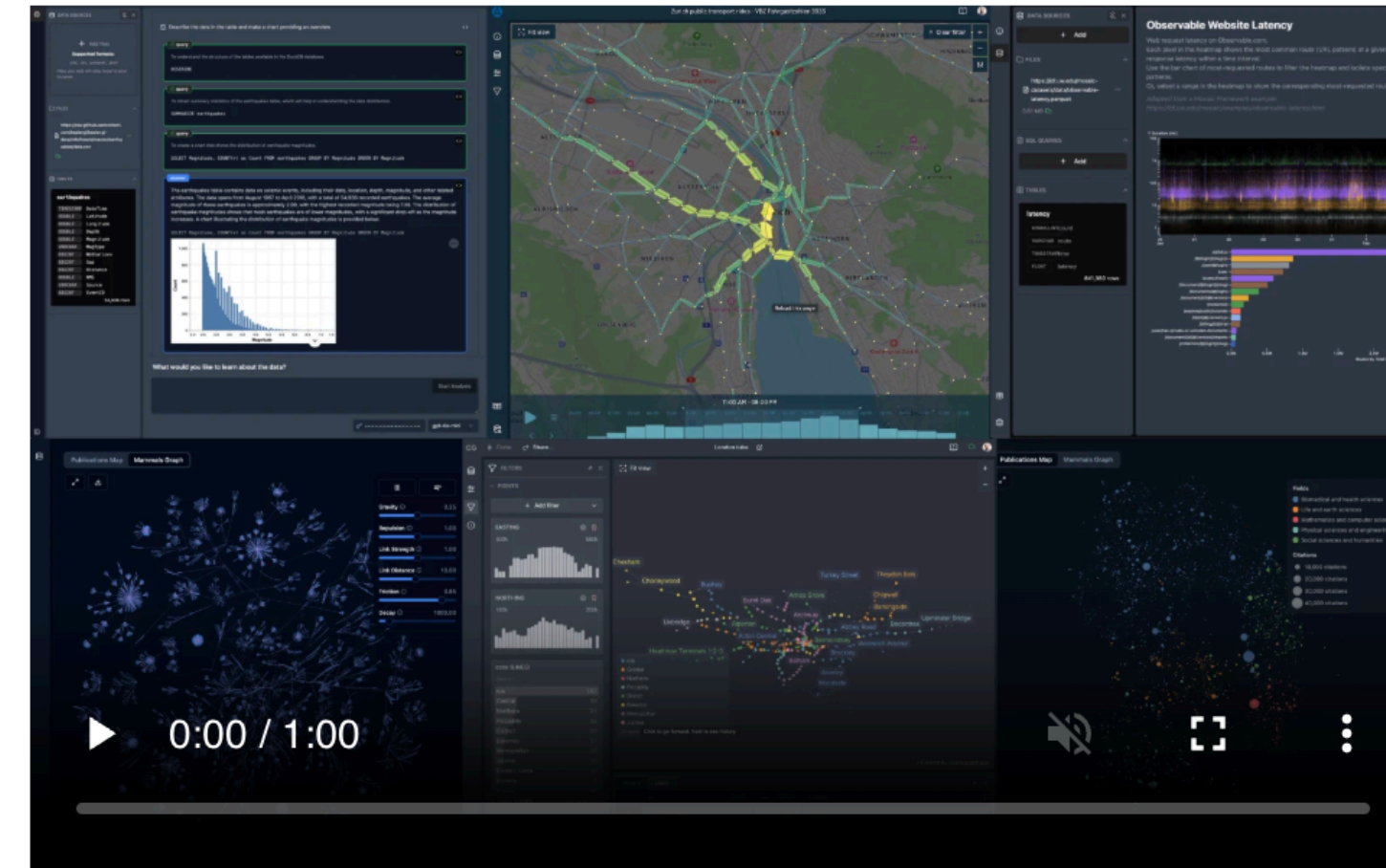
Ilya Boyandin, Foursquare



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.

FSQ Spatial Desktop - California Earthquakes.spatial

DATABASE Status: Connected

+ Add data

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

TABLES

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

TILESETS

California Earthquakes

Map showing earthquake locations in California, color-coded by magnitude. Major cities labeled: Chico, Reno, Sacramento, San Francisco, San Jose, Visalia, St. George. Geographical features: MONTEREY BAY NATIONAL MARINE SANCTUARY, DEATH VALLEY NATIONAL PARK, NEVADA TEST AND TRAINING RANGE (NTTR).

Time range: 08/01/1967 - 11/03/2017

© kepler.gl | Basemap by: mapbox

Y Axis

Bar chart showing the frequency of earthquakes over time (1970 to 2010).

Run

```
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

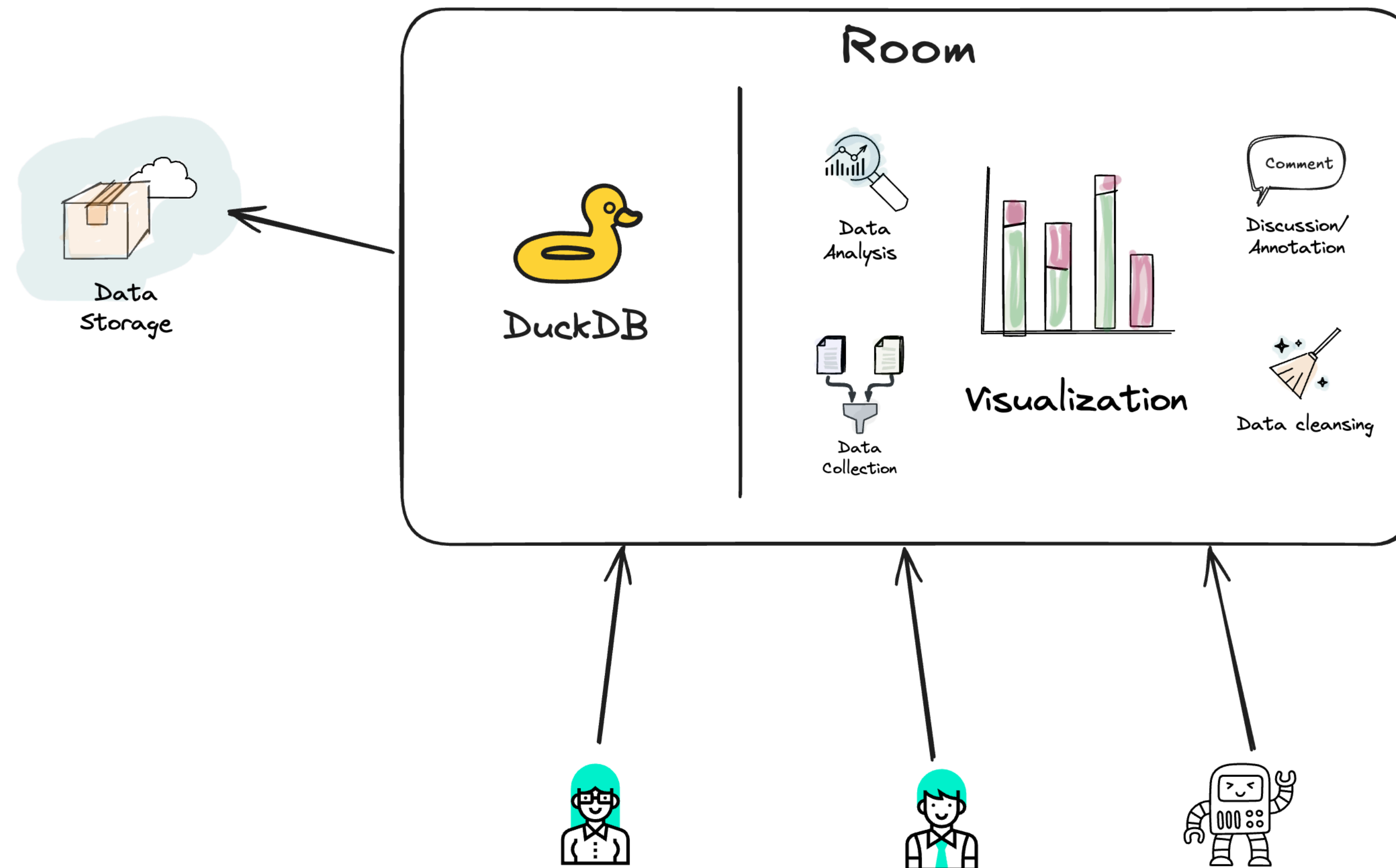
100 rows Limit results to 100 rows

+ 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

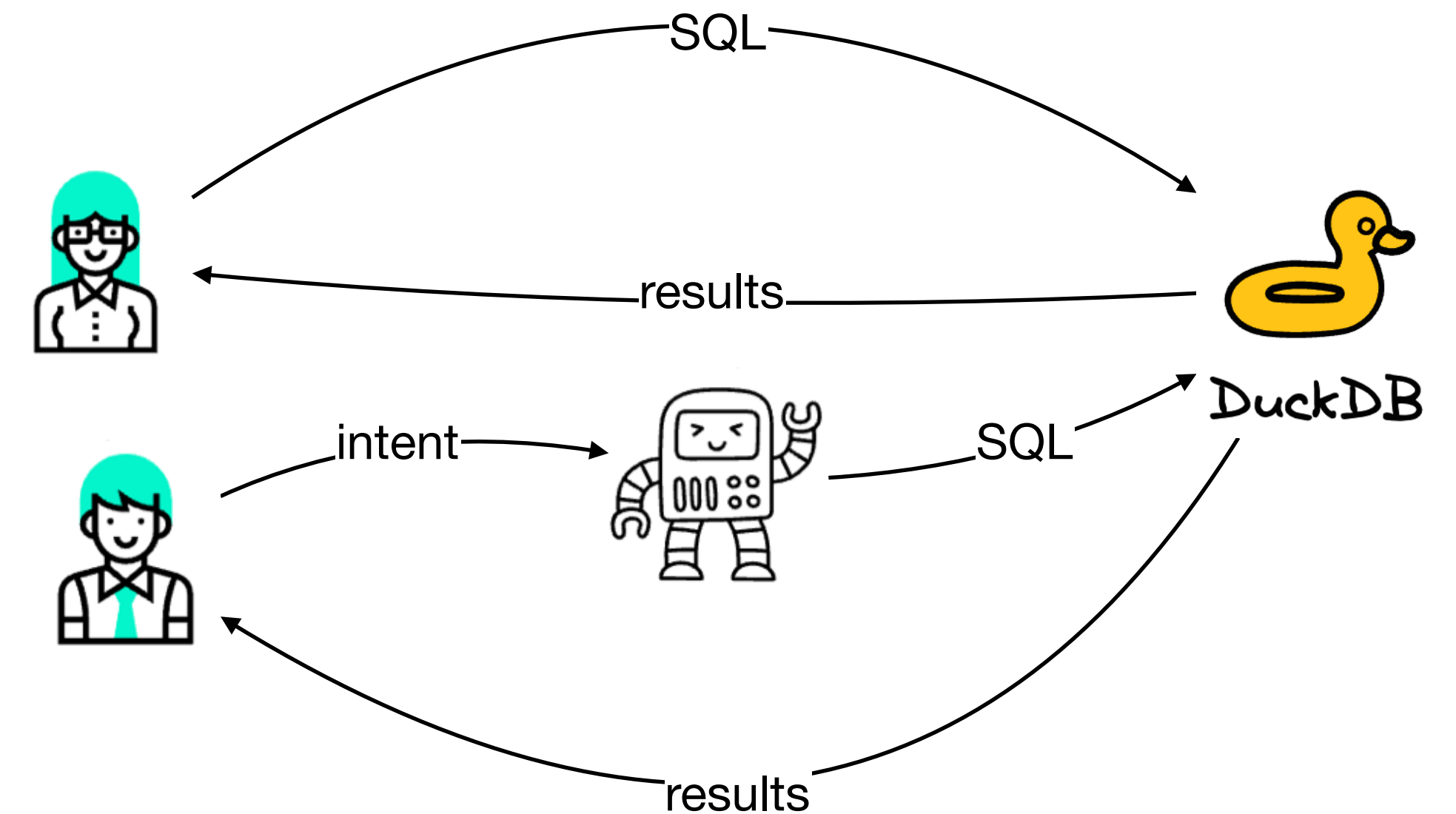
SQL as a Stable Target Language

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

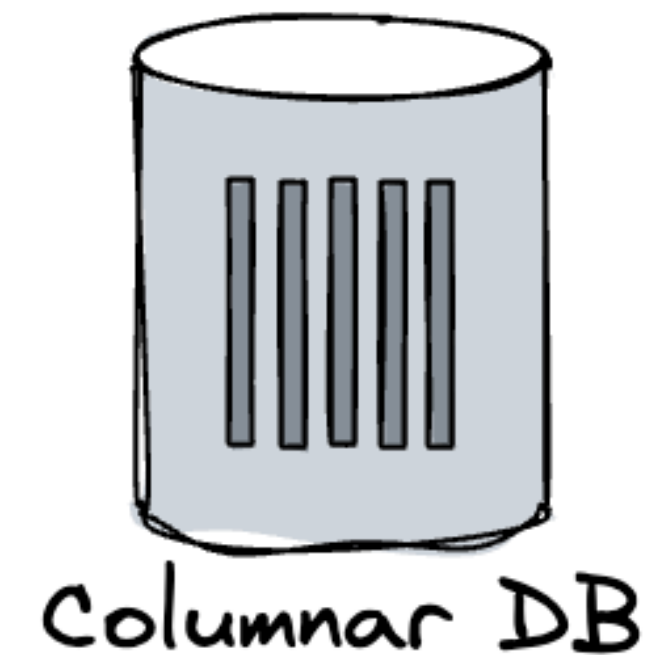
As a Python/R/C/C++ library

Efficient Interop

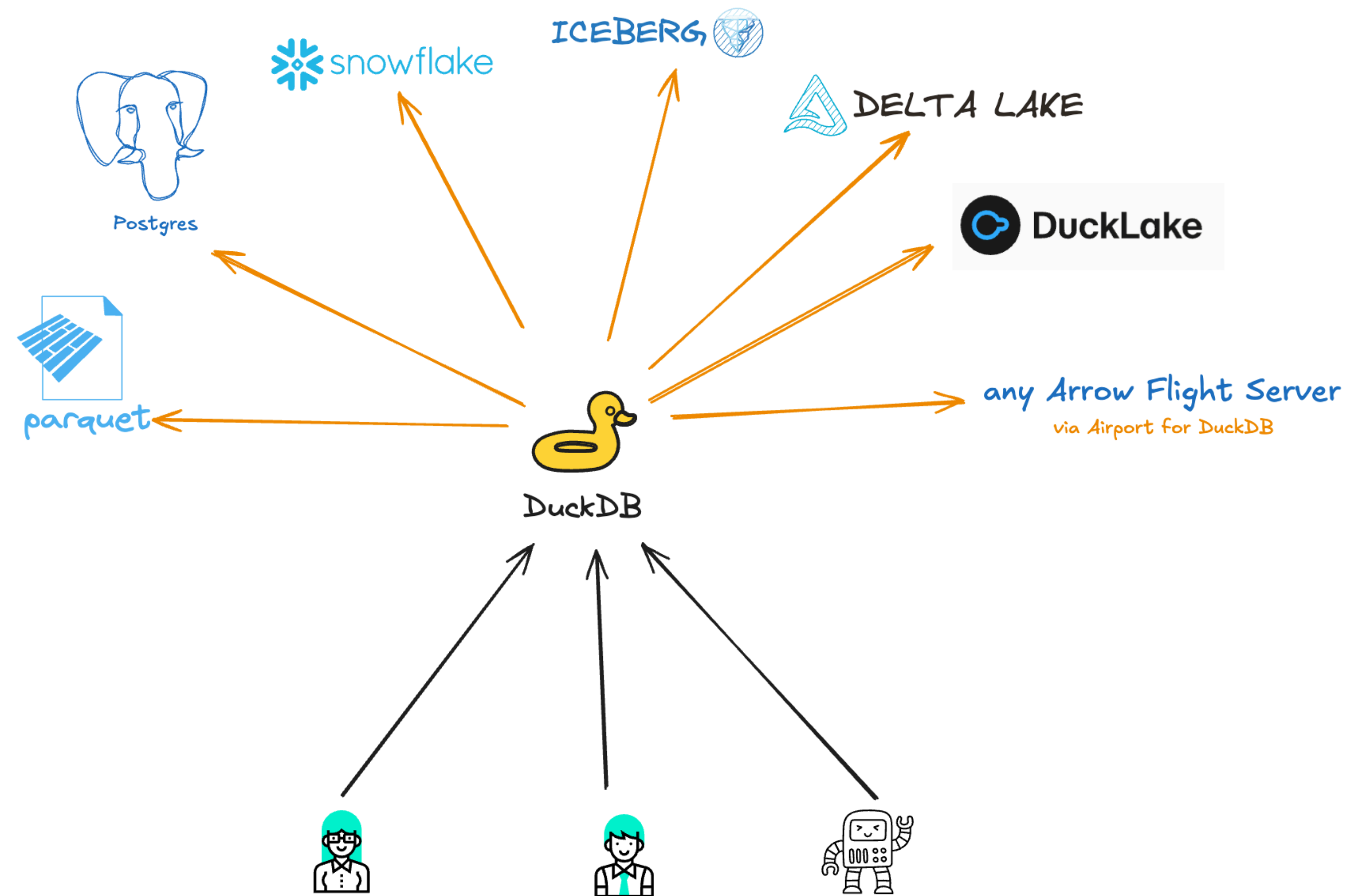
Arrow-native query results → zero-copy data exchange

Low Infra, Storage-First

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



Multi-source queries



Offline use example

Progressive Web App (PWA)

Web Apps capable of working offline

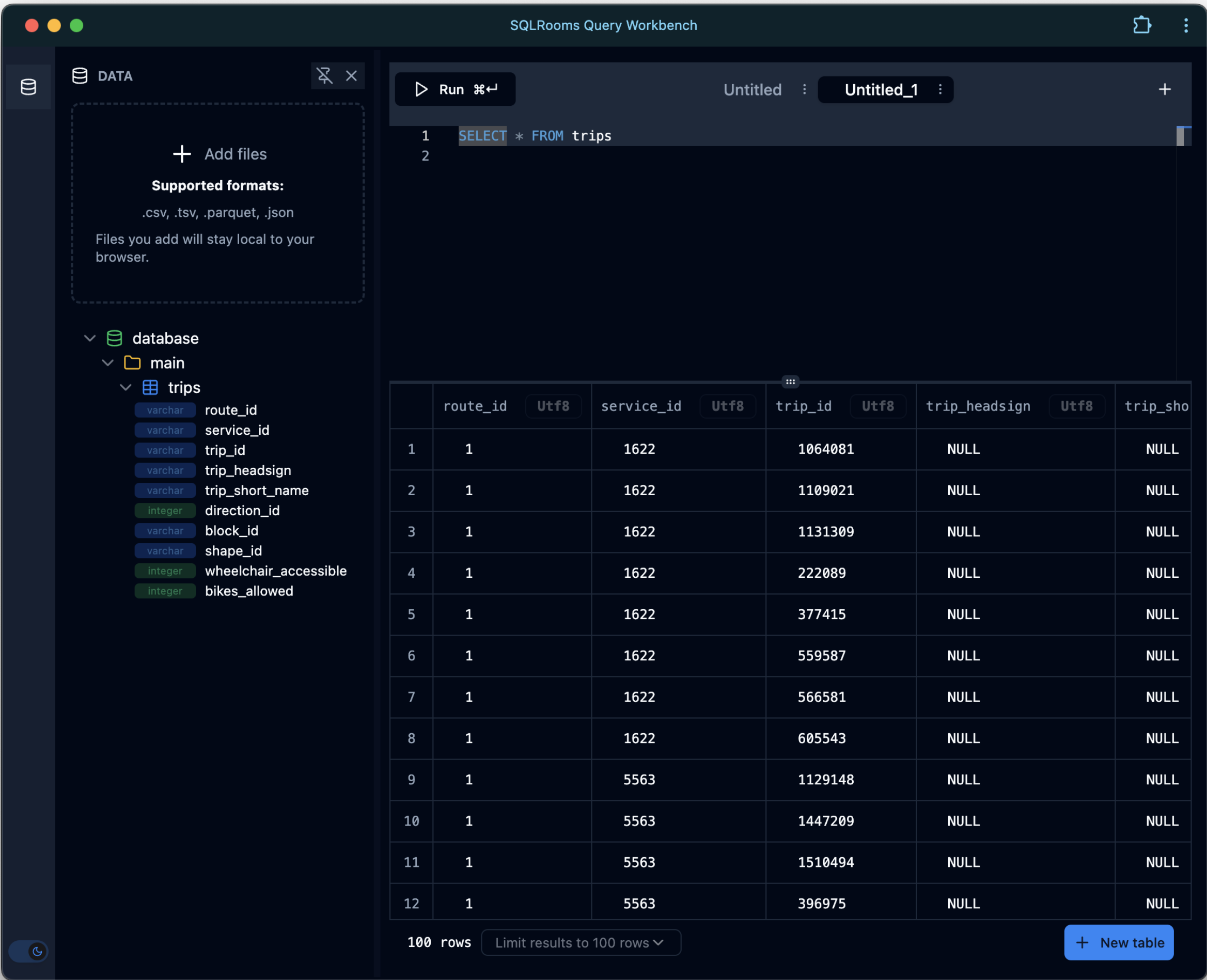
OPFS for DuckDB Storage

Origin Private File System

Local storage / IndexedDB

Persisting app state

Local AI via Ollama



<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

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

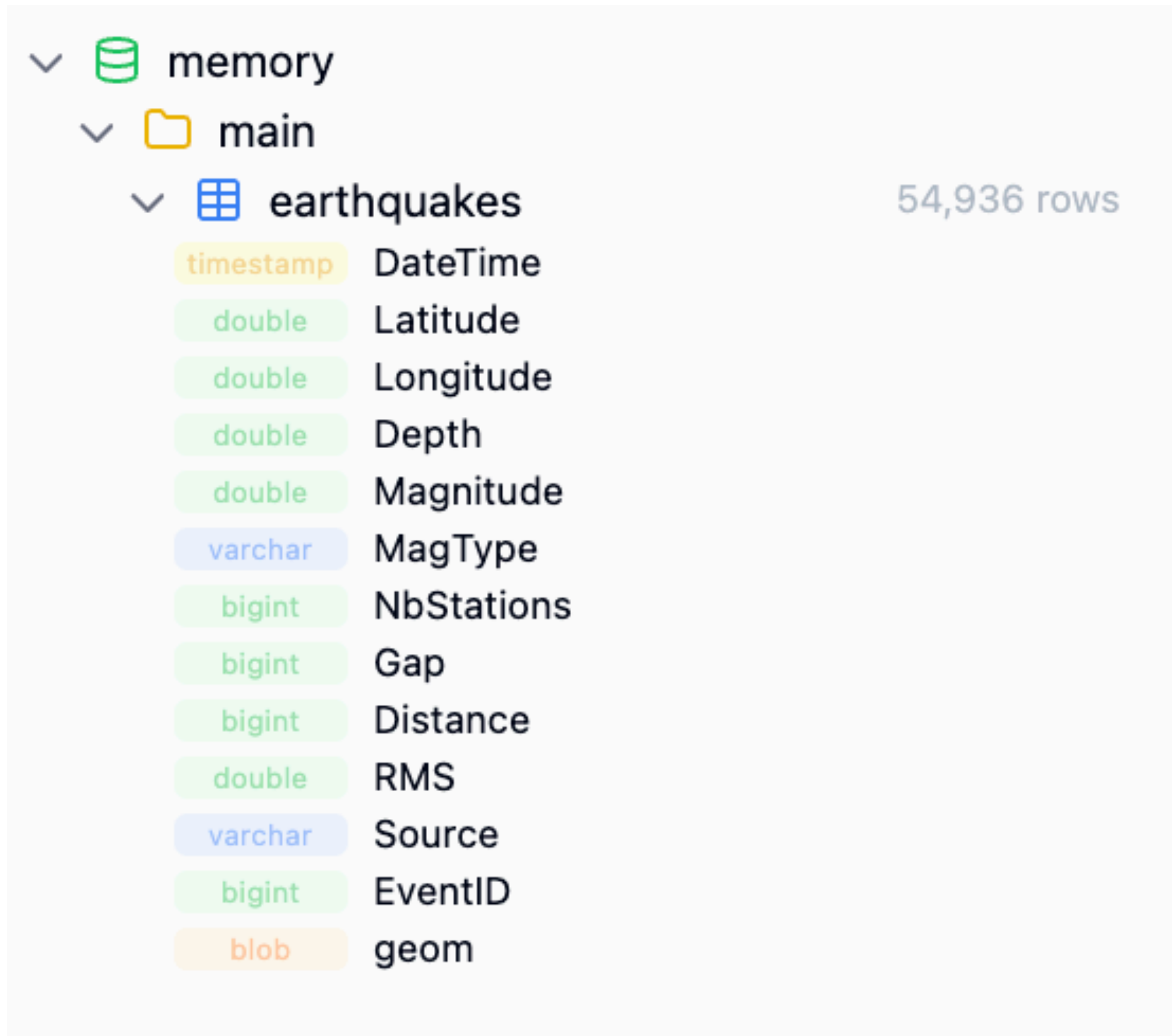
e.g. Datavis: Vega, Mosaic, Recharts

@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;
```



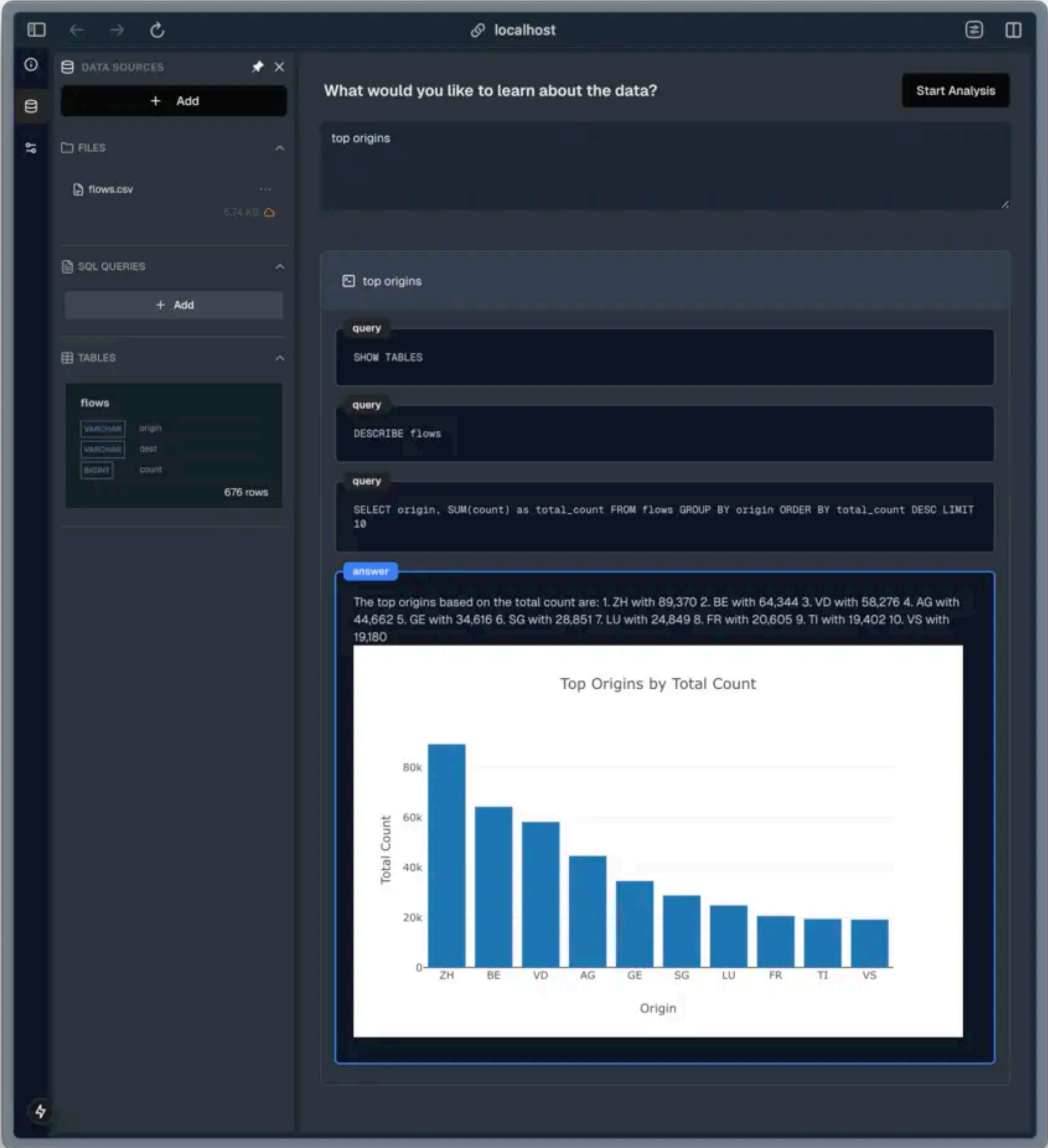
Show Query Result

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

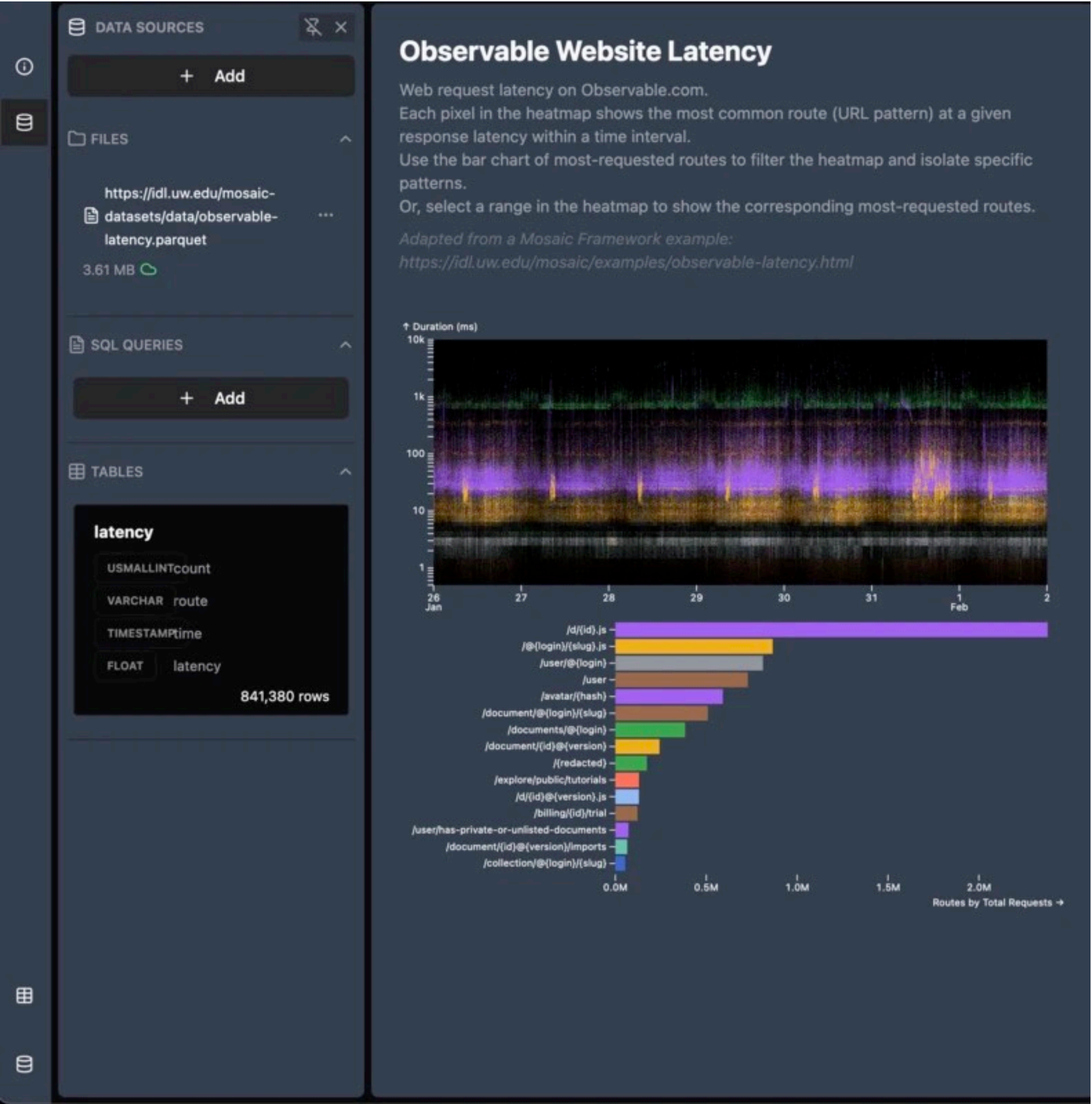
What would you like to do?



@sqlrooms/vega



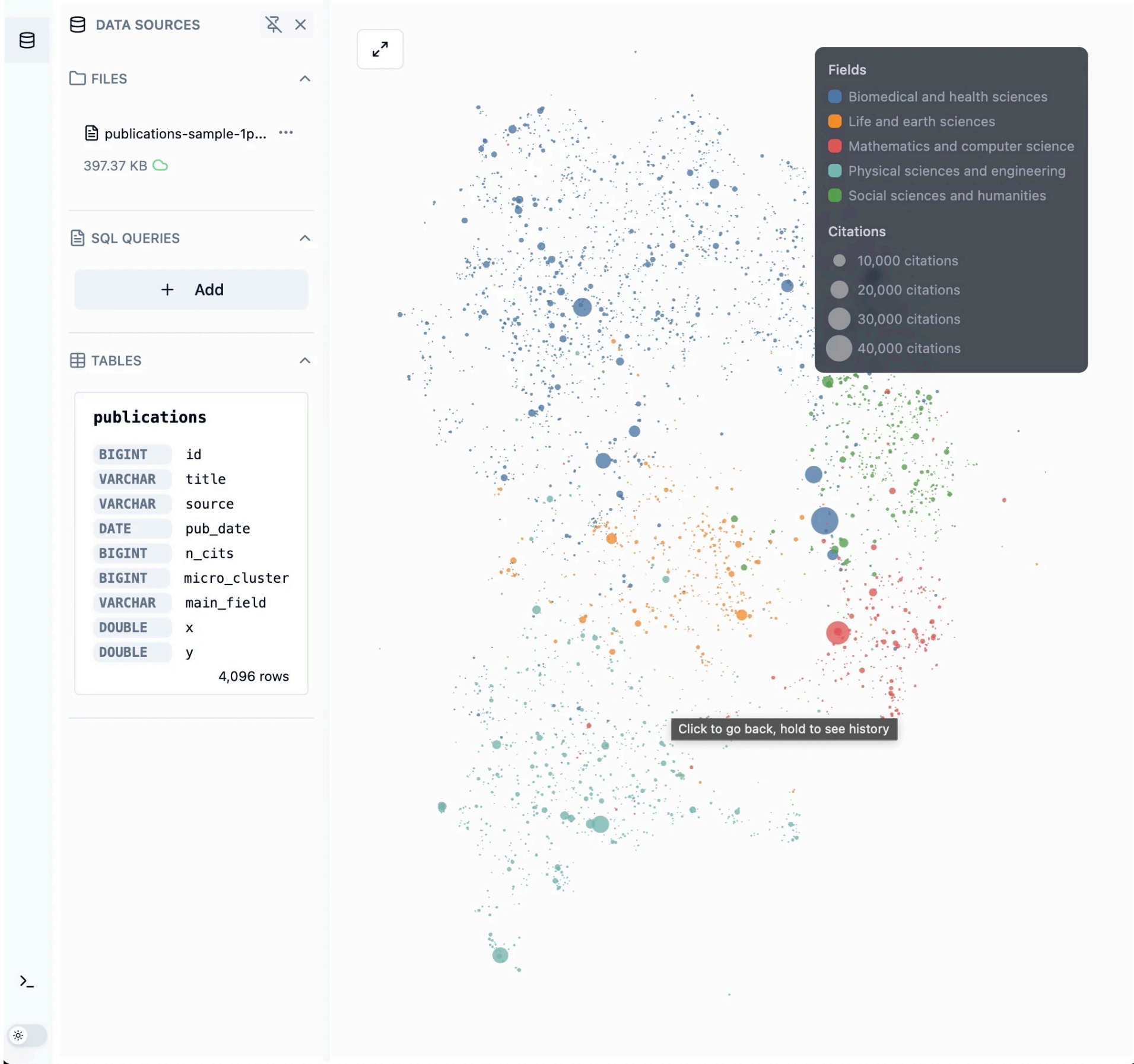
@sqlrooms/mosaic



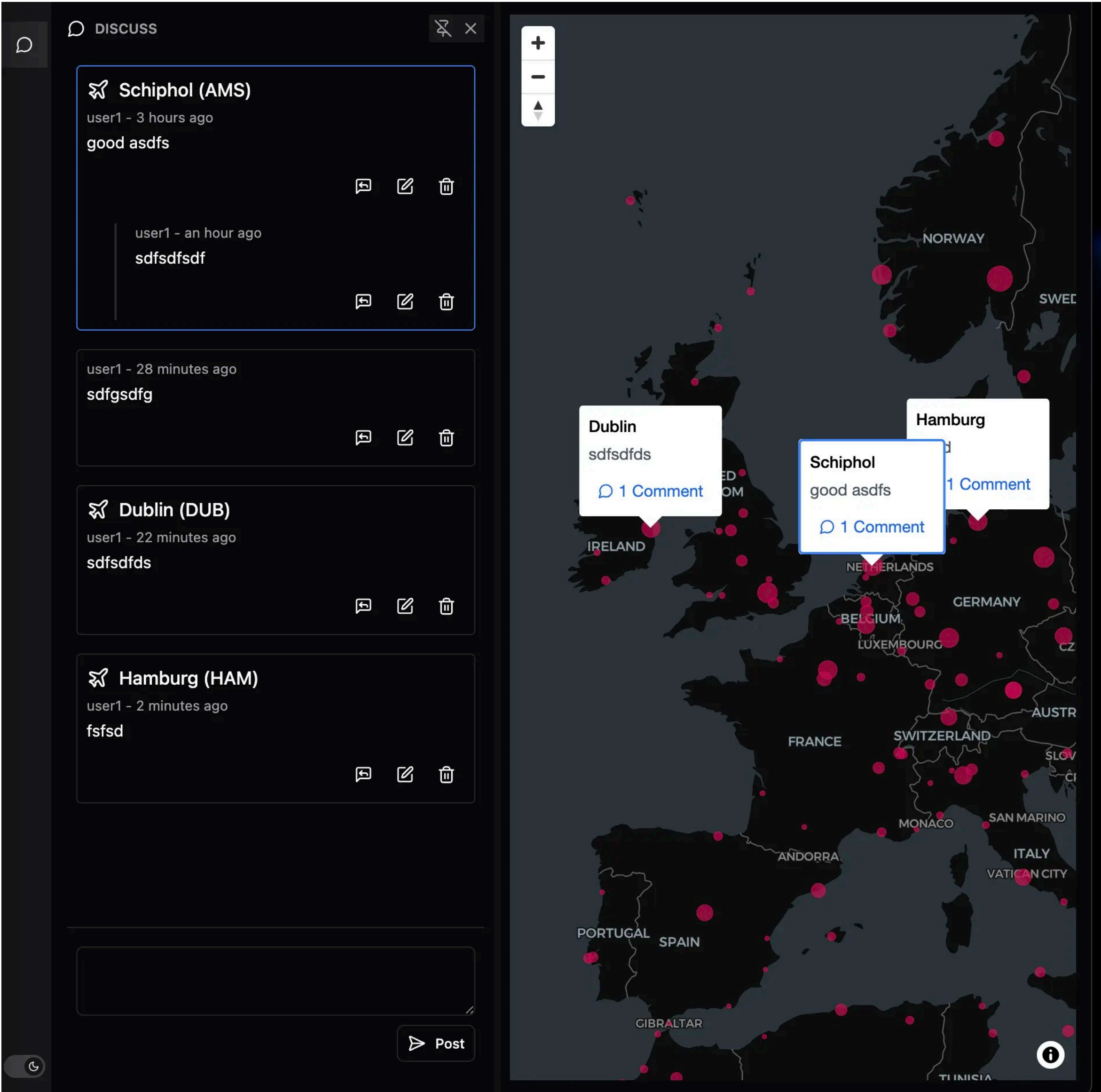
@sqlrooms/kepler



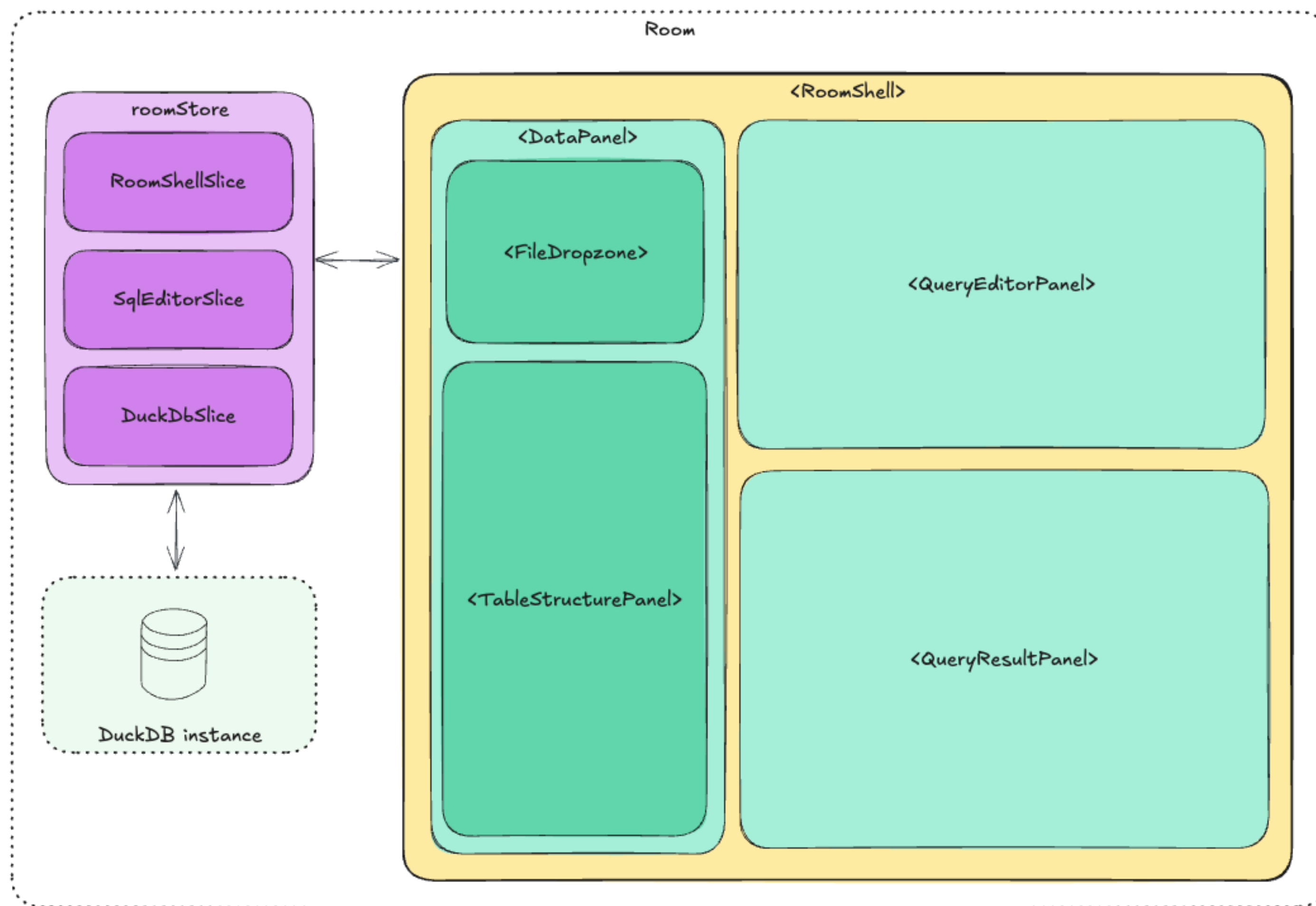
@sqlrooms/cosmos



@sqlrooms/discuss



Modular toolkit



The screenshot shows the Room application interface. On the left, a sidebar lists the loaded data source: **gtfs_de_full** with a sub-folder **main** containing various tables like **agency**, **areas**, **calendar**, etc. The main area is divided into three panels: **<FileDropzone>** (top left), **<QueryEditorPanel>** (top right), and **<QueryResultPanel>** (bottom right). The **<QueryEditorPanel>** 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 **<QueryResultPanel>** displays the results of the query in a table format:

	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, it indicates "100 rows" and "Limit results to 100 rows".

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: DataSourcesPanel,
        placement: 'sidebar',
      },
      'main': {
        title: 'Main view',
        icon: () => null,
        component: MainView,
        placement: 'main',
      },
    },
  },
})(set, get, store),
```

DATA

+ Add files

Supported formats:
.csv, .tsv, .parquet, .json

Files you add will stay local to your browser.

gtfs_de_full

main

agency

areas

attributions

booking_rules

calendar

calendar_dates

fare_attributes

fare_log_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

transfers

Run

Untitled

+

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

	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	

100 rows 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>
  );
};
```

+ Add files

Supported formats:

.csv, .tsv, .parquet, .json

Files you add will stay local to your browser.

gtfs_de_full

main

agency

areas

attributions

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

transfers

Run

Untitled

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

	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

100 rows Limit results to 100 rows

Composition is all you need

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

Forward this private message

Add by name or channel

B I S | 🔗 ↺ ↻ | ⌨️ ↵

Add a message, if you'd like.

Aa 😊 @

fernando
i should prep my React Universe talk

Copy Link Save Draft Forward



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

Untitled Map +

AI ASSISTANT

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

Comparison of Homicide vs. Unemployment Rate (1960): California vs. Arizona

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

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

Run

Untitled : orrelation_HR60_UE60

1_Line_Parameters_HR

AZ_HR60_UE60_Correlation

```
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	Float64
1	California		0.3201930656099818		0.5279757225818	
2	Arizona		0.5126066287349511		0.7082142262046	

2 rows Limit results to 100 rows

+ New table

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

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

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 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

CHART

Type Line

General Styles

X-AxisYear

Y-AxisMaxMagnitude Mean

Mean of MaxMagnitude

Year

@sqlrooms/canvas

DATA

+ Add files

Supported formats: .csv, .tsv, .parquet, .json

Files you add will stay local to your browser.

memory

canvas

Query

Query 1

Query 2

main

earthquakes

54,936 rows

54,936 rows

54,936 rows

54,936 rows

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T00:42:24.000Z		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	2017-12-14T11:00:55.000Z		31.48567		-119.51366		5.8	

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T00:42:24.000Z		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	2017-12-14T11:00:55.000Z		31.48567		-119.51366		5.8	

	DateTime	Timestamp<MICROSECOND>	Latitude	Float64	Longitude	Float64	Depth	Float64
1	2011-06-15T12:29:00.000Z		31.46383		-115.4165		5.343	
2	2017-07-29T00:42:24.000Z		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	2017-12-14T11:00:55.000Z		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.

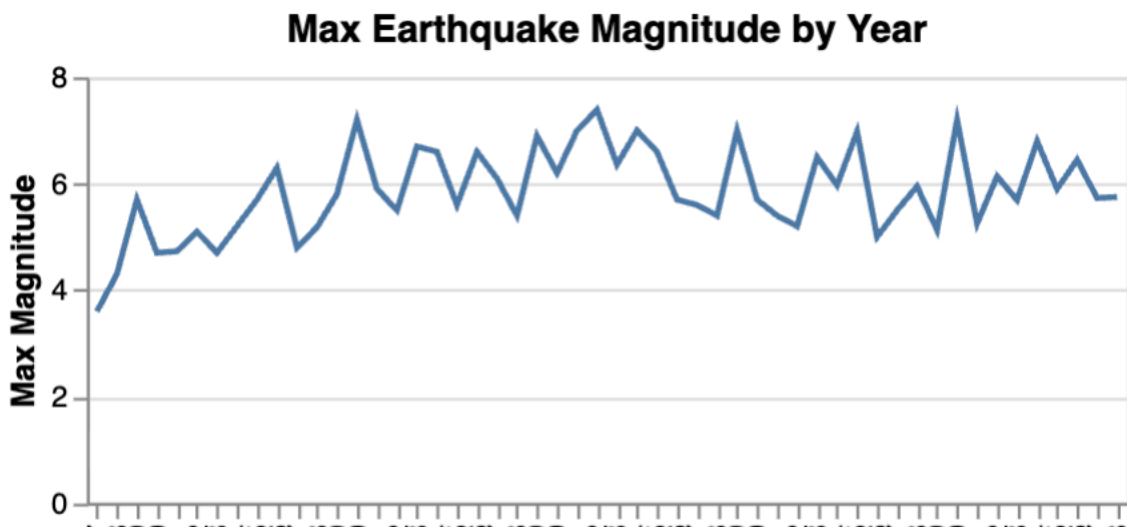
```
SELECT strftime('%Y', DateTime) AS Year,
MAX(Magnitude) AS MaxMagnitude
FROM earthquakes
GROUP BY Year
ORDER BY Year;
```

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;
```

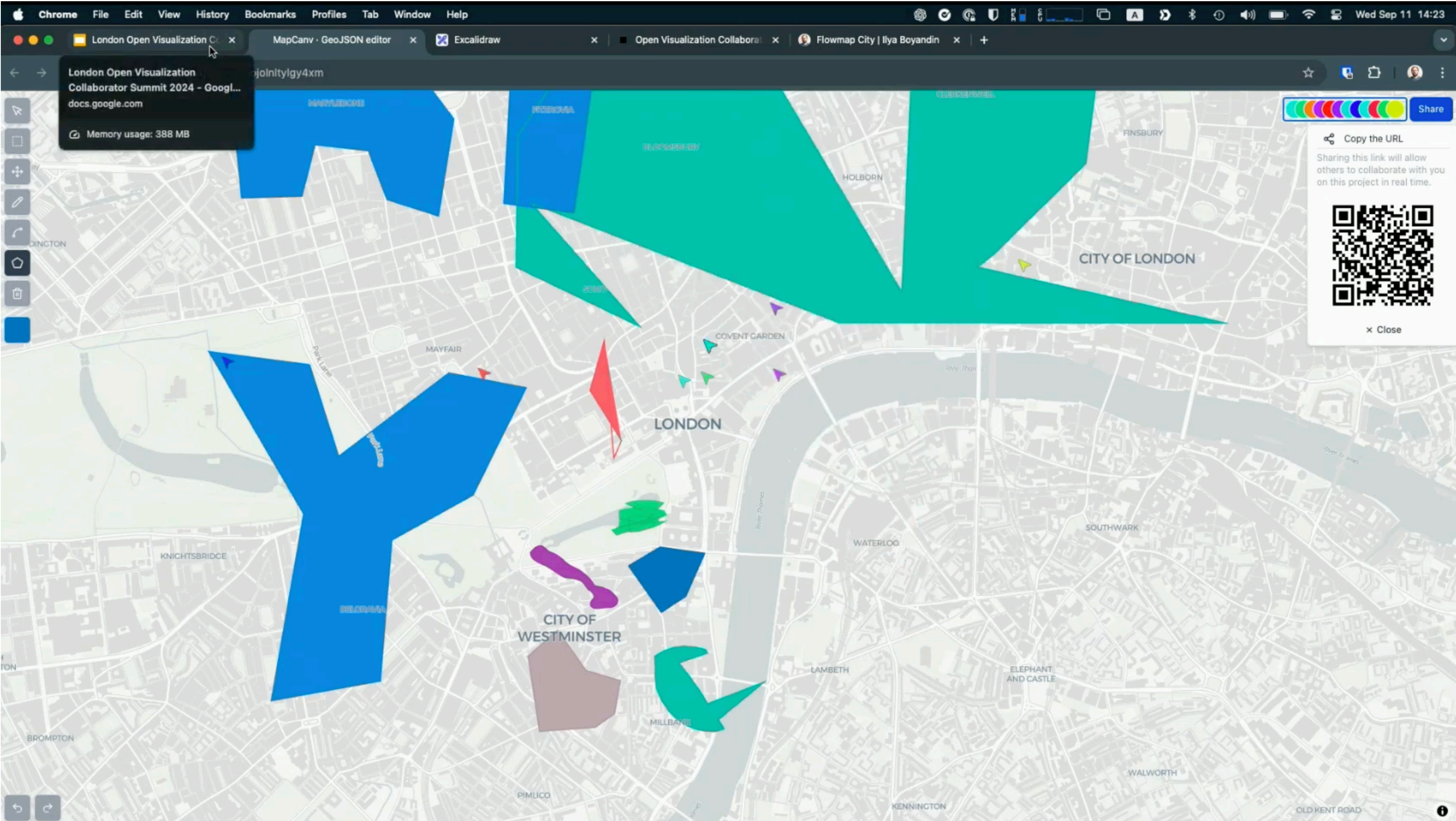
Show Query Result



What would you like to do?

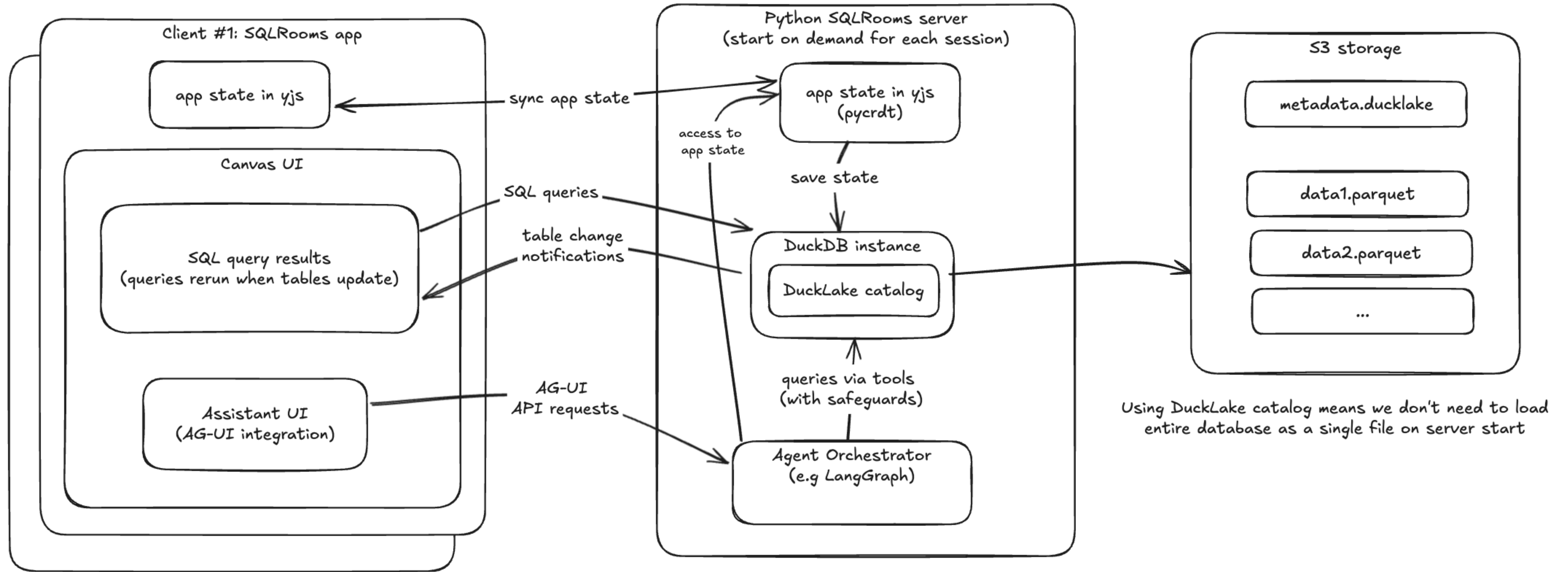


Realtime Collaboration



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}` : 'Loadin
19 }
20
21 export default function App() {
22   return (
23     <div style={{fontFamily: 'system-ui', padding:
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!

What would you like to change in the app?'

gpt-4.1



Terminal

ready

added 107 packages in 23s

12 packages are looking for funding
run `npm fund` for details

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](#)

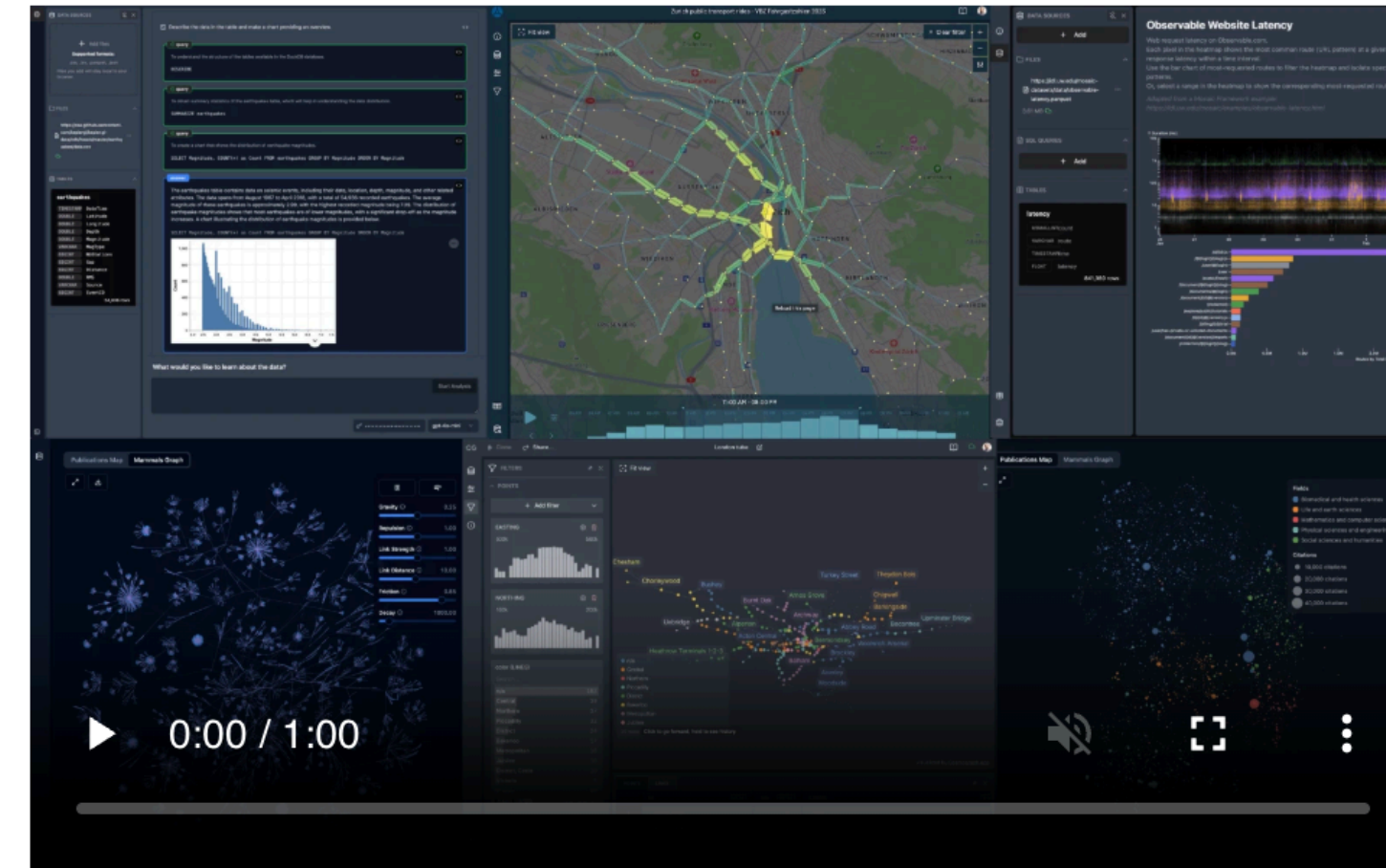
[@sqlrooms/sql-editor](#)

[@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.