

Spatial Explorer: Spatial SQL for Maps, in the Browser and in VS Code

DuckDB-WASM, MapLibre GL and Svelte — one codebase, a web app and a VS Code extension, and your files never leave your machine

Pukar Bhandari

pukar.bhandari@outlook.com

2026-02-25

Table of contents

1 Try it	2
2 What it does	2
2.1 Look	2
2.2 Query	2
2.3 Style	3
2.4 Analyze	3
2.5 Export and print	3
3 One codebase, two shells	3
4 Running DuckDB in a webview	4
5 Big data without re-querying on every pan	5
6 The query <i>is</i> the product	5
7 Formats that fight back	5
8 Styles that travel	6
9 What I'd say about scope	6
10 Privacy and network use	6
11 Wrapping up	6

Most of my day-to-day GIS work happens in R or Python: load a shapefile, write a few spatial queries, make a quick map to sanity-check the result. That is fast when you are already in a scripting environment, and annoying when you just want to *look* at a file, or hand someone a tool without asking them to install anything.

I wanted something simpler: open a file, write a SQL query, see the answer on a map. No server, no accounts, no desktop GIS. That itch became **Spatial Explorer**. It started in February as a single-page app inspired by Kyle Walker's Spatial SQL Explorer, and it has since been rebuilt into a web

app **and** a VS Code extension that share one engine. This post is about what it does now and why it is built the way it is.

💡 Get it

- **Web app:** ar-puuk.github.io/spatial-sql-explorer (embedded below)
- **VS Code extension:** *Spatial Explorer: Spatial SQL & Map Viewer*, on the VS Code Marketplace or Open VSX (Cursor, VSCodium and friends)

1 Try it

Use **Try sample data** for US counties, or drop in your own file. Everything runs in your tab.

Open app GitHub

2 What it does

The scope is deliberately small: **look** at a file, **query** it, **style** the result, **export** it.

2.1 Look

- **Formats:** GeoJSON, GeoPackage (several layers), SQLite/Spatialite (Cube 7 networks, for example), FlatGeobuf, Shapefile, GeoParquet, Parquet, GML, CSV, PMTiles and Excel. File Geodatabases (experimental) and GTFS transit feeds open too, and a zip opens like the file inside it.
- A map and an attribute table kept in sync, any source CRS, and **129 basemaps from 26 providers** with real previews, plus a globe view.
- One search box for layer values, coordinates and places.
- Map tools: measure distances and areas, draw shapes to select or filter rows (or to use as a SQL table called `drawn`), and turn on terrain with hillshade and elevation. These are scratch geometry only; your data is never edited.

2.2 Query

DuckDB spatial SQL runs against the file by its own name, with autocomplete and inline errors. Results replace the map and the table.

```
SELECT * FROM us_counties
SELECT * FROM layer
SELECT * FROM us_boundaries.counties
SELECT * FROM ST_Read('data/links.shp')
```

Line 1

The open file `us-counties.parquet` is available under its cleaned-up name.

Line 2

...and always as `layer`, whatever it is called.

Line 3

A multi-layer file such as a GeoPackage exposes its layers as views in a schema named after the file.

Line 4

Other files can be read in the same query: by workspace-relative path in VS Code, or files you have added in the web app.

There are also **24 ready-made snippets** that fill in the open layer's own table and column names, and a word search (ranked, with stemming) beside the plain text search.

2.3 Style

Single, categorized, graduated (by a column or a SQL expression), continuous, as-is, bivariate, rule-based, heatmap and cluster styles, with ramps previewed on the map. Under **Vary by attribute**, several properties can each follow their own column (size, width, outline, shape, pattern, opacity, rotation), so one map can show roads colored by level of service *and* widened by traffic. Labels, layer opacity and 3D polygon extrusion are there too, and styles are saved as a QGIS `.qml` sidecar.

2.4 Analyze

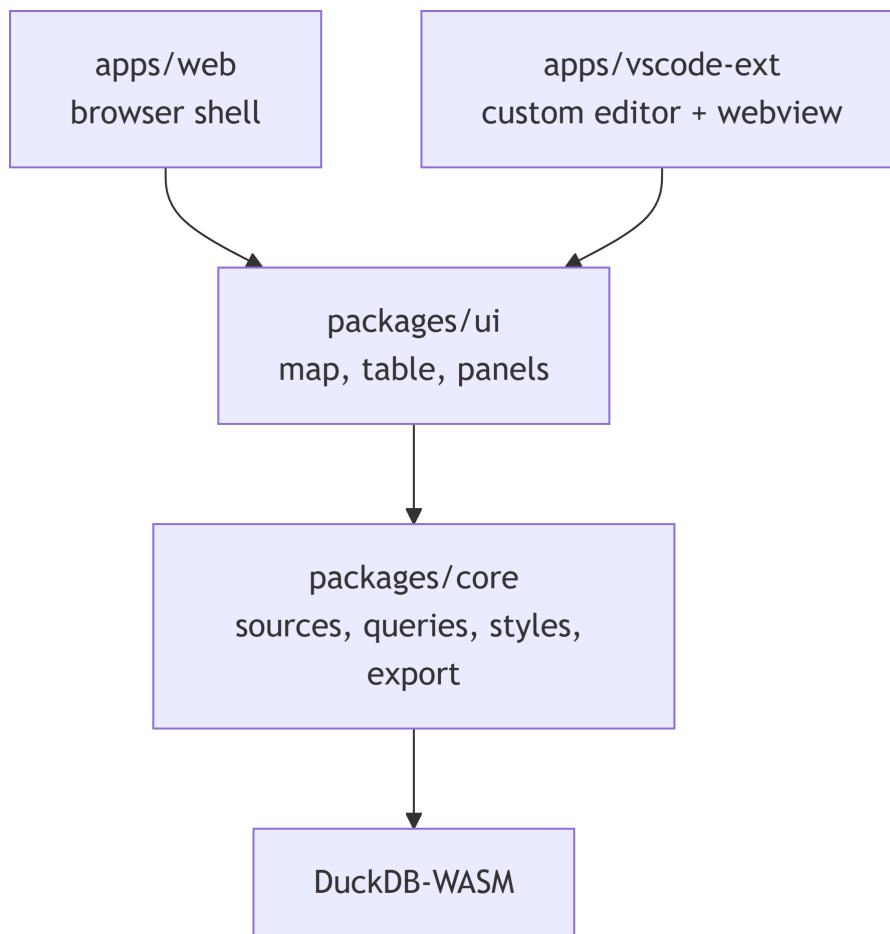
- **Grid cells**: bin any layer into H3 hexagons or A5 equal-area pentagons, counted or averaged per cell, with a live preview and the SQL it writes.
- **Dot density**: one dot stands for an amount you choose, placed at random inside its polygons, with up to six columns in their own colors.
- A **time slider** that filters the map and table by a date column and plays across time, and arrows that show the direction of lines.

2.5 Export and print

Export all, filtered or selected rows to GeoParquet, GeoPackage, GeoJSON, newline-delimited GeoJSON, FlatGeobuf, Shapefile, KML, CSV, Parquet, Excel or JSON, in any EPSG CRS. **Print** lays the map out as a page (paper size, title, legend, scale bar, north arrow, a table and chart of the rows, even an atlas with a page per feature) and saves PNG, PDF or SVG with editable text.

3 One codebase, two shells

The first version was a single page of JavaScript with module-level state, and it showed. It worked, but sharing a link or restoring a session meant threading state through everything. When I wanted a VS Code extension, I rebuilt it so the engine and the interface don't know which shell they are in.



`packages/core` and `packages/ui` are forbidden (by a lint rule) from importing `vscode`, touching the File API or using IndexedDB. Anything platform-specific goes through one interface, the `HostAdapter`: pick the theme, open files, read a file as bytes or a URL for range reads, save a file, persist state, set a status. The web app implements it with the browser; the extension implements it with the VS Code API. Each custom-editor webview gets its own DuckDB instance.

The stack is Svelte 5, Vite and strict TypeScript, MapLibre GL JS for the map, CodeMirror 6 for the SQL editor, Terra Draw for drawing, and DuckDB-WASM underneath.

4 Running DuckDB in a webview

DuckDB-WASM is the heart of it: a WebAssembly build of the DuckDB analytical engine, including the spatial extension that provides `ST_Within`, `ST_Intersects`, `ST_Transform` and friends.

In the first version I leaned on multi-threading, which needs `SharedArrayBuffer`, which needs special response headers that GitHub Pages cannot set. A service worker (`coi-serviceworker`) papered over that. VS Code webviews can't use that trick, so the rebuild uses the **single-threaded DuckDB-WASM bundle in both shells**. One engine configuration to test is worth more than the speed-up.

A few decisions followed from taking the engine seriously:

- **The engine is not bundled.** DuckDB-WASM, its extensions (spatial, parquet, json, and the optional full-text search, H3 and A5) and sql.js are downloaded once from pinned URLs and **checked against SHA-256 hashes** recorded in the repo, then cached locally (Cache Storage on the web, extension storage in VS Code), so later starts work offline.
- **There is a known ceiling.** The engine's memory limit is about 3.1 GiB in both shells, which I measured rather than assumed, and the design works around it.
- **Reads are ranged where possible**, so the whole file is not copied into the webview.

5 Big data without re-querying on every pan

Drawing a million features as GeoJSON would freeze any tab, and re-running a query on every pan is slow. The rule is a size threshold:

- Results under **100,000 rows** (and files under 100 MiB) are converted to GeoJSON once and drawn from memory.
- Larger ones are drawn from **vector tiles that DuckDB builds per tile** with `ST_AsMVT`, served to MapLibre through a custom protocol handler. A “Load all” link is there if you really want everything.

That is how files with millions of rows and several gigabytes stay pannable.

6 The query is the product

Several features that look like dialogs are really generators of SQL, and show you the SQL.

Grid cells bin a layer into H3 or A5 cells with a live preview. The SQL it writes is the same SQL a power user would write, so you can copy it, change it, and run it as a normal query. The result gets the same styling and export as any other.

Dot density is the best example. DuckDB has no point-in-polygon generator (`ST_GeneratePoints` takes a bounding box), so the query samples each polygon's box with a deterministic `hash(feature, column, k)` and keeps the first n points that are `ST_Within` the polygon. Because the hash is deterministic, the map looks the same every time instead of reshuffling the dots on each redraw.

7 Formats that fight back

File Geodatabases are the format I most wanted and the hardest to read. DuckDB-WASM's GDAL build has the OpenFileGDB driver turned off for WebAssembly, so the SQL engine cannot open a `.gdb` at all, and it cannot read inside a zip either. The workaround is `gdal3.js` (GDAL 3.8.4 compiled to WebAssembly), downloaded only the first time you open a geodatabase, which converts the chosen layer so DuckDB can query it. In a spike on an 818 MB geodatabase it listed layers in 87 ms and converted 4 million parcels to FlatGeobuf in 34 seconds. Compressed (CDF/SDC) geodatabases remain unreadable by any open-source reader, so that format stays marked *experimental*.

GTFS feeds are detected by what is inside the zip (stops, routes, trips, stop times and their columns), never by file name, so `GTFS.zip`, `google_transit.zip` and `data.zip` all work. A feed opens as two layers drawn together, `routes` (lines, in each route's own color) and `stops` (points), each its own table.

8 Styles that travel

Styles are saved as a **QGIS .qml sidecar**, so a styled file looks right in QGIS too, and the app imports and exports QML, SLD and Mapbox GL style JSON. Labels are drawn with system fonts (nothing is downloaded), saved with the style, and included when you export the map image.

9 What I'd say about scope

The project has an explicit list of things it will *not* do: feature editing, georeferencing, a processing toolbox, notebooks, plugins, raster analysis, stacked basemaps, multiple map layers. One layer is on the map at a time; multi-file work is done in SQL. Every exception (a globe projection, polygon extrusion, terrain) was a recorded decision. For a tool whose point is “never open QGIS for basic work”, staying a quick-look tool matters more than any one feature.

10 Privacy and network use

Your data stays on your machine. The network is used for the engine's code (downloaded once, then cached), for basemap tiles, and for an opt-out place search that sends only the text you type.

11 Wrapping up

The code is MIT-licensed on GitHub, and the changelog has the details of every release. The project owes a lot to GeoLibre and Qiusheng Wu, a constant reference for the style panel, basemap picker and data-loading design, and to Kyle Walker's original Spatial SQL Explorer for the idea. If you work with spatial data and want a scratchpad that doesn't need a server, give it a try. Issues and PRs welcome.