SQLite Happy Hour Twitter Space

22nd March 2022 - 12:30pm PT / 1:30pm MT / 3:30pm ET

Welcome to the SQLite Happy Hour! This hour-long session will feature three projects that are doing interesting things with SQLite. Each project will provide a ten minute overview, followed by five minutes of discussion from the panel. The last 15 minutes of the hour will be an open discussion and general Q&A.

This document is open for anyone to edit. Please feel free to drop notes and questions in as we go along.

The recording of the space is available here: https://twitter.com/i/spaces/1ypKdEXvkMLGW

Riffle

Geoffrey Litt @geoffreylitt, Nicholas Schiefer @nschiefer

Riffle asks: what if you wrote your whole UI as a query over a local database? So far, we've built a prototype using SQLite and React. More background in this paper:

Building data-centric apps with a reactive relational database

Research project goal is to make development simpler, as opposed to the ongoing trend of more complexity.

Riffle looks at having a database-centric mechanism at the heart of the view. Declarative queries could make apps easier to understand and debug.

SQLite is the tool used for the prototype.

Local first architecture: Ink & Switch have been promoting this. Return to a world where you local client device serves as a source of truth - you can access data offline etc - and when the network is available your data gets synced to the cloud.

The prototype: a reactive layer that uses SQLite as a state management backend for React, using https://sql.js.org/ which compiles SQLite in WASM. Also built prototypes of desktop apps using https://github.com/tauri-apps/tauri - like Electron but using the system web browser instead of bundling its own.

Since they control the writes, they can re-execute every query after any writes happen. SQLite is so fast that this works fine, queries all take under a ms and even with a thousand queries you can still run them all.

ALL UI state is in the database - there's no local React component state - literally everything is in the database. This means all UI state is persistent by default.

IndexedDB is used for the in-browser persistence. The Tauri desktop app stores to a file on disk. Maybe SQL.js could do that with the new Chrome filesystem API stuff too?

Questions about Riffle:

- Will Riffle target vanilla JS, or Node.js?
    - It's running client-side, so vanilla JS
- From Stephen: What about browser-native UI state like scroll position, URL path, query string, multiple independent browser tabs, etc?
    - Great question. We do some syncing of browser-native state to put it in the DB: eg, to support virtualized list rendering we update scroll state in the DB with an event handler. But there's definitely some browser state that isn't being captured reliably. In the purest world, the pixels on your screen would be produced by a DB query :)
- From Predrag Gruevski: Would "query the queries" be a viable approach for narrowing the set of queries that need to be re-executed after a given write? Simple example: if table X gets modified, query for all queries that have table X in a FROM clause, then re-execute them.
    - yeah, that's roughly the direction we're headed. It's a little trickier than that if you start having subqueries / materialized view, but good general idea
- From Longwei Su: Right now, each db update will cause a whole refresh. Is there any plan to refine the binding? So that any db update will only trigger UI component that "subscribe" to this section of the data. Sqlite have trigger, which can have callback on record update. How to construct that "publisher"-> "subscriber" mapping from sql query?

Comments for Riffle:

- From Jesse - http://web.dev/file-system-access/ isn't a very rich api - I think you could persist to it, but I don't think you can seek/update/…/all the posix stuff sqlite probably needs
- Hasura documented how they do reactive queries with Postgres, might be useful for minimising refetch overhead?

Datasette

Simon Willison @simonw

Datasette is an open source multi-tool for exploring and publishing data. It explores SQLite as a read-only mechanism for publishing structured data online in as flexible a manner as possible,

and aims to build an ecosystem of plugins that can handle a wide range of exploratory data analysis challenges.

Video introduction here: https://simonwillison.net/2021/Feb/7/video/

Questions about Datasette:

- How does it compares with https://github.com/dinedal/textql, it seems the same but instead of sqlite binaries, just raw csv files which are more ubiquitous, and easier to view and edit with with office software (msf excel, libreoffice calc) ?
    - sqlite-utils memory provides similar functionality: https://simonwillison.net/2021/Jun/19/sqlite-utils-memory/
- Does Datasette need to worry about SQLite's Defense Against the Dark Arts security guidelines?
    - Yes, absolutely! I've put a lot of work in there. Most importantly, Datasette enforces a time limit on queries, which cuts them off if they take more than a second.
- The SQLite3 docs are sometimes light on examples for the tricky stuff (e.g., enabling WAL). What's your best sort of info beyond the official docs?
    - I've been publishing my own notes here: https://til.simonwillison.net/sqlite
    - The SQLite Forum is **amazing** - I ask questions on there and often get a reply from the maintainers within a few hours: https://sqlite.org/forum/forummain
- From Predrag Gruevski: Regarding learning curve, is a GraphQL web IDE (with syntax highlighting / autocomplete etc.) sufficiently user-friendly for folks more comfortable with a spreadsheet than a CLI tool or SQL?
    - Probably not! GraphQL requires thinking like a programmer too. I'm interested in helping people who aren't yet ready to learn any kind of programming language
    - I have a plugin for Datasette that adds GraphQL with the GraphiQL user interface - demo here: datasette-graphql-demo.datasette.io
        - Thanks! Would love to compare notes on this – my experience from working with analysts at my employer was that they were able to master GraphiQL very quickly. In a sense, it was more intimidating than actually difficult, so working with them directly to get them over the initial difficulty hump via examples and targeted exercises made a huge positive impact.
- … your questions here …

Litestream

Ben Johnson @benbjohnson

Litestream adds replication to SQLite, allowing databases to be cheaply replicated to storage systems such as S3. Litestream also now implements live read-replication, where many read replicas can be run against a single leader database.

https://www.sqlite.org/np1queryprob.html - Many Small Queries Are Efficient in SQLite

Questions about Litestream:

- What does the planned hot standby feature look like, especially regarding durability guarantees during fail-over?
    - *BJ: Hot standby is a tough issue to generalize. The database-as-a-service version of Litestream that's coming will handle this but it's not necessarily planned for Litestream)*
        - Will DBaaS be hosted, OSS, or both?
            - *It'll be both*
- From Longwei Su: I assume offline update will be commit locally then sync with the online storage. If there is a offline commit that conflict with the online version(that already committed in). How to resolve the conflict?
- Not sure if this relates to Litestream but; how big is sql.js — how much does it cost (in kilobytes) to load sqlite in the browser?
    - *BJ: I think sql.js is 1.2MB so the cost depends on how much your provider charges for bandwidth*
        - *Thanks! Meant "cost" in the sense of bytes transferred over wire — this answers it :)*
- … your questions here …

GraphQL

- https://github.com/simonw/help-scraper is scraping GraphQL schemas

Thanks!