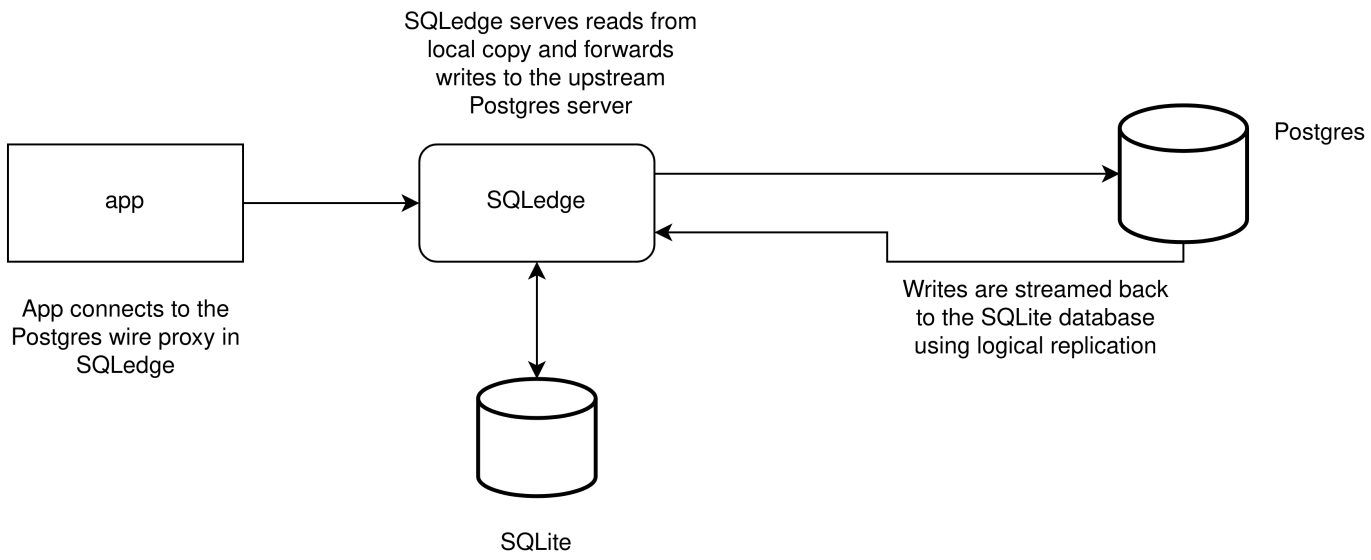
README.md

SQLedge

[State: alpha]

SQLedge uses Postgres logical replication to stream the changes in a source Postgres database to a SQLite database that can run on the edge. SQLedge serves reads from its local SQLite database, and forwards writes to the upstream Postgres server that it's replicating from.

This lets you run your apps on the edge, and have local, fast, and eventually consistent access to your data.



SQL generation

The `pkg/sqlgen` package has an SQL generator in it, which will generate the SQLite insert, update, delete statements based on the logical replication messages received.

SQL parsing

When the database is started, we look at which tables already exist in the sqlite copy, and make sure new tables are created automatically on the fly.

Postgres wire proxy

SQLedge contains a Postgres wire proxy, default on `localhost:5433`. This proxy uses the local SQlite database for reads, and forwards writes to the upstream Postgres server.

Compatibility

When running, the SQL statements interact with two databases; Postgres (for writes) and SQLite (for reads).

The Postgres wire proxy (which forwards reads to SQLite) doesn't currently translate any of the SQL statements from the Postgres query format/functions to the SQLite format/functions. Read queries issued against the Postgres wire proxy need to be compatible with SQLite directly. This is fine for simple `SELECT` queries, but you will have trouble with Postgres-specific query functions or syntax.

Copy on startup

SQLEdge maintains a table called `postgres_pos`, this tracks the LSN (log sequence number) of the received logical replication messages so it can pick up processing where it left off.

If no LSN is found, SQLedge will start a postgres `COPY` of all tables in the `public` schema. Creating the appropriate SQLite tables, and inserting data.

When the replication slot is first created, it exports a transaction snapshot. This snapshot is used for the initial copy. This means that the `COPY` command will read the data from the transaction at the moment the replication slot was created.

Trying it out

1. Create a database

```
create database myappdatabase;
```

2. Create a user -- must be a super user because we create a publication on all tables

```
create user sqledger with login superuser password 'secret';
```

3. Run the example

```
SQLEDGE_UPSTREAM_USER=sqledger SQLEDGE_UPSTREAM_PASSWORD=secret
SQLEDGE_UPSTREAM_NAME=myappdatabase go run ./cmd/sqledge/main.go
```

4. Connect to the postgres wire proxy

```
psql -h localhost -p 5433

$ CREATE TABLE my_table (id serial not null primary key, names text);
$ INSERT INTO my_table (names) VALUES ('Jane'), ('John');

$ SELECT * FROM my_table;
```

The read will be served from the local database

5. Connect to the local sqlite db

```
sqlite3 ./sqledge.db

.schema
```

Config

All config is read from environment variables. The full list is available in the struct tags on the fields in `pkg/config/config.go`

Releases

No releases published

Packages

No packages published

Contributors 2


zknill zak


The-Alchemist The Alchemist

Languages

- **Go** 100.0%