Recommonmark Documentation

Release

Lu Zero, Eric Holscher, and contributors

Mar 05, 2017

Contents

AlaSQL is an open source project and we appreciate any and all contributions we can get. Please help out.

Got a question? Ask on Stack Overflow and tag with "alasql".

CHAPTER 1

AlaSQL

(à la SQL) [ælæ skju:l] - AlaSQL is a free and open source SQL database for Javascript with a strong focus on query speed and data source flexibility for relational data, schemaless data, and graph data. It works in your browser, Node.js, IO.js and Cordova.

The library is designed for:

- Fast SQL data processing in-memory for BI and ERP applications on fat clients
- Easy ETL and option for persistency by data import / manipulation / export for several formats
- All major browsers, Node.js, and mobile applications

We focus on speed by taking advantage of the dynamic nature of javascript when building up queries. Real world solutions demand flexibility regarding where data comes from and where it is to be stored. We focus on flexibility by making sure you can import/export and query directly on data stored in Excel (both `xls` and `.xlsx`), CSV, JSON, TAB, IndexedDB, LocalStorage, and SQLite files.

The library brings you the comfort of a full database engine to your javascript app. No, really - its working towards a full database engine complying with most of the SQL-99 spiced up with additional syntax for handling NoSQL (schema-less) data and graph networks.

```
// A) Traditional SQL
alasql("CREATE TABLE cities (city string, population number)");

alasql("INSERT INTO cities VALUES ('Rome',2863223),('Paris',2249975),('Berlin',
↪3517424),('Madrid',3041579)");

var res = alasql("SELECT * FROM cities WHERE population < 3500000 ORDER BY population␣
↪DESC");

console.log(res);

/*
[
  {
    "city": "Madrid",
```

```
    "population": 3041579
  },
  {
    "city": "Rome",
    "population": 2863223
  },
  {
    "city": "Paris",
    "population": 2249975
  }
]
*/
```

```
// B) SQL on array of objects
var data = [{a:1,b:10}, {a:2,b:20}, {a:1,b:30}];

var res = alasql('SELECT a, SUM(b) AS b FROM ? GROUP BY a',[data]);

console.log(res); // [{"a":1,"b":40},{"a":2,"b":20}]
```

```
// C) Promise notation + read from file example
alasql.promise('SELECT * FROM XLS("mydata.xls") WHERE lastname LIKE "A%" and city =
↪"London" GROUP BY name ')
      .then(function(res){
           console.log(res); // output depends on mydata.xls
      }).catch(function(err){
           console.log('Does the file exists? there was an error:', err);
      });
```

```
// D) Cheat and load your data directly

alasql("CREATE TABLE example1 (a INT, b INT)");

alasql.tables.example1.data = [               // Insert data directly from javascript␣
↪object...
    {a:2,b:6},
    {a:3,b:4}
];

alasql("INSERT INTO example1 VALUES (1,5)");  // ...or you insert data with normal SQL

var res = alasql("SELECT * FROM example1 ORDER BY b DESC");

console.log(res); // [{a:2,b:6},{a:1,b:5},{a:3,b:4}]
```

jsFiddle with example A) and example B)

If you are familiar with SQL it should come as no surprise that proper usage of indexes on your tables is essential to get good performance.

Install

```
npm install --save alasql      # node
bower install --save alasql    # bower
```

```
import alasql from 'alasql';   # meteor
npm install -g alasql          # command line
```

For the browser: include alasql.min.js

```
<script src="http://cdn.jsdelivr.net/alasql/0.3/alasql.min.js"></script>
```

Get started

The wiki has a great section on how to get started

When you feel you got the grip, you can check out the wiki section about data manipulation or get inspired by the list of Q&As

- Documentation: Github wiki
- Library CDN: jsDelivr.com
- Feedback: Open an issue
- Try online: Playground
- Website: alasql.org

Please note

All contributions are much welcome and greatly appreciated(!) - The project has never received any funding and is based on unpaid voluntary work: We really (really) love pull requests

AlaSQL project is very young and still in active development phase, therefore it may have bugs. Please, submit any bugs and suggestions as an issue.

AlaSQL uses Semantic Versioning so please note that major version is zero (0.y.z) and the API can not be considered 100% stable. Consider this before using the library in production and please checkout the limitations of the library

Performance

AlaSQL is very focused on speed, and we make sure to use all the tricks we can find to make javascript spit out your results as quick as possible. For example:

- Queries are cached as compiled functions.
- Joined tables are pre-indexed
- `WHERE` expressions are pre-filtered for joins

The results are good. Check out AlaSQL vs. other javaScript SQL databases:

- **3x speed** compared to SQL.js selecting with `SUM`, `JOIN`, and `GROUP BY`.
- **1x speed** compared to WebSQL selecting with `SUM`, `JOIN`, and `GROUP BY` (in-memory operations for WebSQL - see this discussion)
- **2x speed** compared to Linq for `GROUP BY` on 1,048,576 rows

Please remember to set indexes on your tables to speed up your queries. Have a look here if you are not familiar with this consept.

See more speed related info on the wiki

Features you might like

Traditional SQL

Use "good old" SQL on your data with multiple levels of: `JOIN`, `VIEW`, `GROUP BY`, `UNION`, `PRIMARY KEY`, `ANY`, `ALL`, `IN`, `ROLLUP()`, `CUBE()`, `GROUPING SETS()`, `CROSS APPLY`, `OUTER APPLY`, `WITH SELECT`, and subqueries. See the wiki to compare supported features with SQL standards.

User defined functions in your SQL

You can use all benefits of SQL and JavaScript together by defining you own costume functions. Just add new functions to the alasql.fn object:

```
alasql.fn.myfn = function(a,b) {
    return a*b+1;
}
var res = alasql('SELECT myfn(a,b) FROM one');
```

You can also make user defined aggregator functions (like your own `SUM(...)`). See more in the wiki

Compiled statements and functions

```
var ins = alasql.compile('INSERT INTO one VALUES (?,?)');
ins(1,10);
ins(2,20);
```

See more in the wiki

SELECT directly on your javascript data

Group your JavaScript array of objects by field and count number of records in each group:

```
var data = [{a:1,b:1,c:1},{a:1,b:2,c:1},{a:1,b:3,c:1}, {a:2,b:1,c:1}];
var res = alasql('SELECT a, COUNT(*) AS b FROM ? GROUP BY a',[data]);
console.log(res);
```

See more ideas of creative datamanipulation in the wiki

JavaScript Sugar

AlaSQL extends "good old" SQL to make it closer to JavaScript. The "sugar" includes:

- Write Json objects - `{a:'1',b:@['1','2','3']}`
- Acesss object propertires - `obj->property->subproperty`

- Access Ooject and arrays elements - `obj->(a*1)`
- Access JavaScript functions - `obj->valueOf()`
- Format output format with SELECT VALUE, ROW, COLUMN, MATRIX to format results of query
- ES5 multiline sql with `var SQL = function(){/*select 'MY MULTILINE SQL'*/}` and pass instead of SQL string. (will not work if you compress your code)

Read and write Excel, and raw data files

You can import from and export to CSV, TAB, TXT, and JSON files. Calls to files will always be [[async]] so the approach is to chain the queries if you have more than one:

```
var tabFile = 'mydata.tab'

alasql.promise([
        "select * from txt('mytext.txt') where [0] like 'M%'",
        ["select * from tab(?) order by [1]", [tabFile]],   // note how to pass␣
↪parameter when promises are chained
        "select [3] as city,[4] as population from csv('cities.csv')",
        "select * from json('array.json')"
    ]).then(function(results){
        console.log(results)
    }).catch(console.error)
```

Read SQLite database files

AlaSQL can read (not write) SQLite data files if you include the SQL.js library:

```
    <script src="alasql.js"></script>
    <script src="sql.js"></script>
    <script>
        alasql('ATTACH SQLITE DATABASE Chinook("Chinook_Sqlite.sqlite");\
            USE Chinook; \
            SELECT * FROM Genre',[],function(res){
                console.log("Genres:",res.pop());
        });
    </script>
```

sql.js calls will always be async.

AlaSQL works in the console - CLI

After globally installing AlaSQL `npm install alasql -g` you can access AlaSQL via the commandline

```
> alasql "SET @data = @[{a:'1',b:?},{a:'2',b:?}]; SELECT a, b FROM @data;" 10 20
[ 1, [ { a: 1, b: 10 }, { a: 2, b: 20 } ] ]

> alasql "VALUE OF SELECT COUNT(*) as abc FROM TXT('README.md') WHERE LENGTH([0]) > ?
↪" 140
// Number of lines with more than 140 characters in README.md
```

See more in the wiki

Features you might love

AlaSQL D3.js

AlaSQL plays nice with d3.js and gives you a convenient way to integrate a specific subset of your data vis the visual powers of d3. See more about D3.js and AlaSQL in the wiki

AlaSQL Excel

AlaSQL can export data to both Excel 2003 (.xls) and Excel 2007 (.xlsx) with coloring of cells and other Excel formatting functions.

AlaSQL Meteor

Meteor is amazing. You can query directly on your Meteor collections with SQL - simple and easy. See more about Meteor and AlaSQL in the wiki

AlaSQL Angular.js

Angular is great. Besides using AlaSQL for normal data manipulation it works like a charm for exporting you present scope to Excel. See more about Angular and AlaSQL in the wiki

AlaSQL Google Maps

Pinpointing data on a map should be easy. AlaSQL is great to prepare source data for Google Maps from for example Excel or CSV making a one unit of work for fetching and identifying whats relevant. See more about Google Maps and AlaSQL in the wiki

AlaSQL Google Spreadsheets

AlaSQL can query data directly from a google spreadsheet. A good "partnership" for easy editing and powerfull data manipulation. See more about Google Spreadsheets and AlaSQL in the wiki

Miss a feature?

Take charge and add your idea or vote on your favorite feature to be implemented:

Limitations

Please be aware that AlaSQL ~~may~~ have bugs. Beside the bugs there are a number of limitations

1. AlaSQL has a (long) list of keywords that must be escaped if used for column names. When selecting a field named `key` please write `SELECT `key` FROM ...` instead. This is also the case for words like `` `value` ``, `` `read` ``, `` `count` ``, `` `by` ``, `` `top` ``, `` `path` ``, `` `deleted` ``, `` `work` `` and `` `offset` ``. Please consult the full list of keywords.

1. It is Ok with select for 1000000 records or to join two tables by 10000 records in each (You can use streaming functions to work with longer datasources - see test/test143.js) but be aware that the workload is multiplied so selecting from more than 8 tables with just 100 rows in each will show bad performance. This is one of our top priorities to make better.
2. Limited functionality for transactions (supports only for localStorage) - Sorry, transactions are limited, because AlaSQL started to use more complex approach for PRIMARY KEYS / FOREIGN KEYS. Transactions will be fully turned on again in future version.
3. A `(FULL) OUTER JOIN` and `RIGHT JOIN` on more than 2 tables will not give the expected results. `INNER JOIN` and `LEFT JOIN` are ok.
4. Please use alias when you want fields with same name from different tables (`SELECT a.id as a_id, b.id as b_id FROM ?`).
5. At the moment Alasql does not work with jszip 3.0.0 - please use version 2.x

1. JOINing a sub-SELECT does not work. Please store your sub-select in a temporary table (or fetch the sub-select and pass it as an argument)
2. AlaSQL uses FileSaver.js library for saving files locally from the browser. Please be aware that it does not save files in Safari 8.0.

Probably, there are many of others. Please, help us to fix them by submitting it as an issue. Thank you!

How To

Use AlaSQL to convert data from CSV to Excel

ETL example:

```
    alasql('CREATE TABLE IF NOT EXISTS geo.country; \
            SELECT * INTO geo.country FROM CSV("country.csv",{headers:true}); \
            SELECT * INTO XLSX("asia.xlsx") FROM geo.country WHERE continent_name =
↪"Asia"');
```

Use AlaSQL as a WebWorker

AlaSQL can work as a webworker.. Pleaes be aware that all interaction with AlaSQL when running must be async.

In the browser you can include `alasql-worker.min.js` instead of `alasql.min.js` and AlaSQL will figure out the rest:

```
<script src="alasql-worker.min.js"></script>
<script>
var arr = [{a:1},{a:2},{a:1}];
    alasql('SELECT * FROM ?',[arr],function(data){
        console.log(data);
    });
</script>
```

Try the example at jsFiddle.

Another option is to include the normal file but call alasql.worker() as the first thing yourself:

```
<script src="alasql.min.js"></script>
<script>
     alasql.worker();
     var res = alasql('select value 10',[],function(res){
          console.log(res);
     });
</script>
```

Try this example in jsFiddle.

If using AlaSQL from a webworker, you can importing it traditionally as a script:

```
    importScripts('alasql.min.js');
```

Use Webpack and Browserify

When targeting the browser, several code bundlers like Webpack and Browserify will pick up modules you might not want.

Here's a list of modules that alasql requires

- fs
- cptable
- jszip
- xlsx
- xls
- cpexcel
- path
- es6-promise
- net
- tls

Webpack

There are several ways to handled alasql with webpack

IgnorePlugin

Ideal when you want to control which modules you want to import.

```
var IgnorePlugin =  require("webpack").IgnorePlugin;

module.exports = {
  ...
  //Will ignore the modules fs, path, xlsx, xls
  plugins:[new IgnorePlugin(/(^fs$|cptable|jszip|xlsx|xls|^es6-promise$|^net$|^tls$|^
↪forever-agent$|^tough-cookie$|cpexcel|^path$)/)]
};
```

module.noParse

As of alasql 0.3.5, you can simply tell webpack not to parse alasql, which avoids all the dynamic require warnings and avoids using eval/clashing with CSP with script-loader.Read the webpack docs about noParse

```
...
//Don't parse alasql
{module:noParse:[/alasql/]}
```

script-loader

If both of the solutions above fail to meet your requirements, you can load alasql with script-loader.

```
//Load alasql in the global scope with script-loader
import "script!alasql"
```

This can cause issues if you have a CSP that doesn't allow eval.

Browserify

Read up on excluding, ignoring, and shimming

Example (using excluding)

```
var browserify = require("browserify");
var b = browserify("./main.js").bundle();
//Will ignore the modules fs, path, xlsx, xls
["fs","path","xlsx",  ... , "xls"].map(ignore => b.ignore(ignore));
```

jQuery

Please remember to send the original event, and not the jQuery event, for elements. (use `event.originalEvent` instead of `myEvent`)

JSON-object

You can use JSON objects in your databases (do not forget use == and !== operators for deep comparision of objects):

```
alasql> SELECT VALUE {a:'1',b:'2'}

{a:1,b:2}

alasql> SELECT VALUE {a:'1',b:'2'} == {a:'1',b:'2'}

true

alasql> SELECT VALUE {a:'1',b:'2'}->b

2

alasql> SELECT VALUE {a:'1',b:(2*2)}->b
```

```
4
```

Try AlaSQL JSON objects in Console [sample](http://alasql.org/console?drop table if exists one;create table one;insert into one values {a:@[1,2,3],c:{e:23}}, {a:@[{b:@[1,2,3]}]};select * from one)

Experimental

Usefull stuff, but there might be dragons

Graphs

AlaSQL is a multi-paradigm database with support for graphs that can be searched or manipulated.

```
// Who loves lovers of Alice?
var res = alasql('SEARCH / ANY(>> >> #Alice) name');
console.log(res) // ['Olga','Helen']
```

See more at the wiki

localStorage and DOM-storage

You can use browser localStorage and DOM-storage as a data storage. Here is a sample:

```
alasql('CREATE localStorage DATABASE IF NOT EXISTS Atlas');
alasql('ATTACH localStorage DATABASE Atlas AS MyAtlas');
alasql('CREATE TABLE IF NOT EXISTS MyAtlas.City (city string, population number)');
alasql('SELECT * INTO MyAtlas.City FROM ?',[[{city:'Vienna', population:1731000},
        {city:'Budapest', population:1728000}]]);
var res = alasql('SELECT * FROM MyAtlas.City');
console.log(res);
```

Try this sample in jsFiddle. Run this sample two or three times, and AlaSQL store more and more data in localStorage. Here, "Atlas" is the name of localStorage database, where "MyAtlas" is a memory AlaSQL database.

You can use localStorage in two modes: SET AUTOCOMMIT ON to immediate save data to localStorage after each statement or SET AUTOCOMMIT OFF. In this case, you need to use COMMIT statement to save all data from in-memory mirror to localStorage.

AlaSQL supports plugins

AlaSQL supports plugins. To install the plugin you need to use the `REQUIRE` statement. See more at the wiki

Alaserver - simple database server

Yes, you can even use AlaSQL as a very simple server for tests.

To run enter the command:

```
alaserver [port]
```

then type in browser something like "http://127.0.0.1:1337/?SELECT VALUE 2*2"

Warning: Alaserver is not multi-thread, not concurrent, and not secured.

Tests

Regression tests

AlaSQL have more than 1200 regresion tests, but they only cover of the codebase.

AlaSQL uses `mocha` for regression tests. Install mocha and run

```
> npm test
```

or run test/index.html for tests in browser (Please serve via localhost with for example `http-server`).

Tests with AlaSQL ASSERT from SQL

You can use AlaSQL ASSERT operator to test results of previous operation:

```
CREATE TABLE one (a INT);
ASSERT 1;
INSERT INTO one VALUES (1),(2),(3);
ASSERT 3;
SELECT * FROM one ORDER BY a DESC;
ASSERT [{a:3},{a:2},{a:1}];
```

SQLLOGICTEST

AlaSQL uses SQLLOGICTEST to test it compatibility with SQL-99. The tests include about 2.000.000 queries and statements.

The testruns can be found in the testlog.

Bleeding edge

If you want to try the last development version of the library please download this file or visit the testbench to play around in the browser console.

License

MIT - see MIT licence information

Main contributors

AlaSQL is an open source project and we appreciate any and all contributions we can get. If you feel like contributing, have a look at CONTRIBUTING.md.

Credits

Many thanks to Zach Carter for Jison parser generator, to the author of FileSaver.js, Andrew Kent for his SQL Parser, authors of XLSX library, and other people for useful tools, which make our work much easier.

Related projects that have inspired us

- AlaX - Export to Excel with colors and formats
- WebSQLShim - WebSQL shim over IndexedDB (work in progress)
- AlaMDX - JavaScript MDX OLAP library (work in progress)
- Other similar projects - list of databases on JavaScript
