Dokumentace k semestrální práci z předmětu KIV/DBM2

Bc. Tomáš Vyleta

Obsah

Úvod

Problémem či zadáním semestrální práce je nalézt a popsat knihovnu pro zpracování množiny souborů ve formátu JSON, které většinou vrací endpointy REST API. Využít ji na úrovni jednoho programu (bez nutnosti vkládání dat do jiných SŘBD pro zpracování objektových souborů). Knihovna by měla být schopna poskytovat funkce obsažené v jazyce SQL jako FROM, GROUP BY, JOIN, atd. Cíle práce jsou:

- Nalezněte existující metody dotazováním se nad množinou JSON souborů.
- Popište rozsah nabízených operací ve vybraném jazyce/nástroji.
- Analyzujte možnosti jazyka/nástroje v kontrastu s klasickou relační databází resp. SQL.

1 Knihovna AlaSQL

AlaSQL (čti à la SQL)je *open source* SQL databáze pro JavaScript, operující na straně klienta. Implementuje mnoho funkcí ze čtvrté verze jazyka SQL (SQL:1999) a také některé funkce navíc pro snažší manipulaci s NoSQL a grafovými sítěmi. Také podporuje asynchronní volání pomocí metody promise. Knihovnu lze využít pro webové aplikace, aplikace založené na Node.js nebo v mobilních aplikacích.

Podporuje import/export formátů, jako jsou např. tabulky programu Excel (.xls), CSV - Comma-separated values (.csv), již zmiňovaný JSON - JavaScript Object Notation (.json), TAB - Tab Separated Data File (.tab), IndexedDB, LocalStorage a SQLite.

Obrázek 1: Logo AlaSQL

Užitečné odkazy:

- WEB: http://alasql.org

- Github: https://github.com/agershun/alasql

2 Instalace

Knihovna je dostupná v několika JavaScriptových správcích balíčků (package management) kterými jsou npm, Bover a Meteor nebo si ji můžete přímo naklonovat/stáhnout z officiálního GitHub repozitáře.

Pokud nechcete stahovat knihovnu na fyzické úložiště, nejjednodušší varianta je získat AlaSQL do své aplikace pomocí cloudové služby cdnjs.com, která je založena na principu CDN (Content Delivery Network). Stačí tedy do našeho projektu vložit odkaz na tuto knihovnu, např. do hlavičky HTML souboru:

```
<script src="//cdn.jsdelivr.net/alasql/0.2/alasql.min.js"></script>
```

3 Použití

Po úspěšném nainstalovaní/naimportování knihovny jsme schopni ji začít používat. Nyní je několik možností jak pracovat s databází.

AlaSQL je rozšířitelná a umožňuje nám vytvářet vlastní funkce s použitím JavaScriptu. - lenght v array

```
alasql.aggr.CONCAT = function(v,s) {
    return (s||[]).concat(v);
};
```

3.1 Funkcí

Všechny dotazy (queries) se vkládají jako atribut metody *alasql(stringWithSQL)* nebo jiné proměnné, pokud si je nazvete, třeba v Node.js.

3.1.1 NodeJS

```
var alasql = require('alasql');
alasql('CREATE TABLE one (two INT)');
```

3.2 Nová databáze

Můžeme do proměnné založit novou databázi a nad vytvořenou proměnnou zavolat funkci *exec()* do které vložíme dotaz jako atribut jako v předchozí sekci. Můžeme také vkládat více dotazů oddělených ;, kde poté navrácená hodnota bude jako pole výsledků.

```
var mybase = new alasql.Database();
mybase.exec('CREATE TABLE one (two INT)');
```

Funkce *alasql()* je zkrácenou verzí *alasql.exec()*.

Také k ním můžete přistupovat přes globální objekt *alasql*. Obecně se všechny databáze nacházejí v alasql.databases.[název databáze].

```
var mybase = new alasql.Database('mybase');
console.log(alasql.databases.mybase);
```

3.3 Nová tabulka

Založení nové tabulky je podobné, můžeme ji vytvořit přímo v globálním objektu alasql nebo vytvořit k nějaké databázi. Budu předpokládat, že máme databázi uloženou v proměnné *var mybase*. Založení tabulky je pak následující:

```
mybase.exec("CREATE TABLE cities (city string, population number)");
console.log(alasql.databases.mybase.tables.cities);
console.log(mybase.exec("SELECT * FROM cities"));
```

Jak můžete vidět, výpis celé tabulky může být uskutečněn opět přes objekt alasql.databases.[název databáze].tables.[název tabulky] nebo jako SQL dotaz nad databází.

3.4 Manipulace dat v tabulce

Jako jsme zvyklí z SQL, data jsou provádět různé úkony, jako je INSERT, UPDATE, DELETE. Syntaxe je stejná jako při založení nové tabulky. Akorát se změní dotaz. Jako příklad mohu uvést:

```
mybase.exec("INSERT INTO cities VALUES ('Rome',2863223), ('Paris',2249975), ('Be
('Madrid',3041579)");
```

INSERT jako JS funkce

V úvodu jsem zmínil že jde vytvářet vlastní metody a jedna z možností je je přes funkci compile() knihovny. Která umožňuje překompilovat příkazy a přidá je do cache dané databáze.

```
var insert1 = db.compile('INSERT INTO one (?,?)');
var insert2 = db.compile('INSERT INTO one (:a,:b)');

insert1([1,2]);
insert2({a:3,b:4});
```

Přidělení dat jako JS objekt

Určitě vás napadlo, že jako přistupujeme k databázím a tabulkám přes globální objekt, proč by nešlo přistupovat i k datům tabulky a opravdu to jde. Obecně se všechna data tabulky nacházejí v alasql.databases.[název databáze].tables.[název tabulky].data. Příklad na přidělení dat může být následující:

```
const ceskaMesta = [
    {name: 'Prague', population: 1324277},
    {name: 'Brno', population: 381346},
    {name: 'Ostrava', population: 287968},
    {name: 'Plzen', population: 174842}
];
alasql.databases.mybase.tables.cities.data = ceskaMesta;
```

Je zde ale riziko, přidělená data neprochází přes žádnou funkci, tudíž data vůbec nemusejí odpovídat schématu tabulky, při přidělení dat žádná vyjímka nevyhodí, ale pokud by jsme nad daty zavolali nějaký dotaz, vyskočila by podmínka nebo by data byly *undefiend*.

4 Synchronní/Asynchronní přístup

Za normálního běhu knihovna vykonává procesy synchronně, ale jdou také volat callbacky nebo asynchronní volání pomocí metody *promise()*. Pokud pracujete se soubory, načítáte je, pak knihovna funguje asynchronně a je doporučeno používat již zmíněnou metodu *promise()*.

```
alasql('SELECT * FROM cities', [],          // callback
    function (res) {
        console.log(res);
    }
);
```

5 Import/Export dat

Zmíním jen zajímavé funkce, které mi přišli relevantní, ostatní jen zmíním, že existují.

5.1 HTML

Knihovna dokáže číst data z HTML tabulky <table>...</table> a výsledek opět vygenerovat do HTML tabulky.

5.1.1 Čtení z tabulky

```
alasql('SELECT * FROM HTML("#MyTable", {headers:true})');
```

5.1.2 Zápis do tabulky

```
alasql('SELECT * INTO HTML("#MyTable", {headers:true}) FROM ?',[data]);
```

5.2 Ostatní

Ostatní import/export funkce jsou z/do souborů typu Textový soubor, CSVm TAB a XLS.

5.3 JSON

Hlavní náplň této knihovny.

6 SQL dotazy

Knihovna podporuje všechny základní funkce SQL jazyka, jako jsou funkce JOIN. GROUP, UNION, ANY, ALL, IN, podotazy a také okleštěnou správu transakcí. Dále také knihovna podporuje agregační data mining??? funkce, kterými jsou ROLLUP, CUBE a GROUPING SETS.

6.1 Nejlépe funguje s JS

Tato knihovna je v podstatě SQL databáze v JavaScriptu. Slouží zejména jako podpora zpracování, usnadnění délky kódu, lepší práci s daty nebo jako samostatná databáze - nejlépe však funguje SPOLEČNĚ s JavaScriptem. Knihovna nevrací rovnou tabulky, jak je běžné u relačních databází, ale defaultně vrací data právě jako JSON a další práce s ním je už jako práce s objektem.

```
// vypis poctu objektu ve vracenem poli
var db = new alasql.Database();
db.exec('select * from one', function(data) {          // callback
   console.log(data.length);
});

// novy datovy typ
alasql.fn.Date = Date;
    alasql('CREATE order (orderno INT, orderdate Date)');
```

7 Datasety

7.1 Rohlík.cz

K dispozici máme data z webového portálu Rohlik.cz, který se zaměřuje na obchod s potravinami. Data obsahují dva soubory ze dne 2.11.2020, jeden z pohledu obyčejného uživatele (*rohlik2.json*) a druhý z pohledu prémiového uživatel (*rohlik1.json*)e, hlavní rozdíl mezi těmito dvěma datasety je v ceně za dopravu. Nejvíce nás zajímá atribut *availabilityDays*, který se sestává ze čtyř dalších objektů, popisující dnešek a další tři dny. V nich nalezneme atribut *slots*, ve kterém jsou objekty popisující hodiny daný den. V dané hodině nás zajímá atribut *timeSlotCapacityDTO*, který obsahuje konečný atribut *totalFreeCapacityPercent*

popisující obsazenost danou hodinu. O úroveň víš je pak atribut *price* značící cenu dovozu.

7.2 Starcraft

8 Důležitá je cesta, ne cíl

8.1 Rohlík.cz

8.1.1 Jaké je nejbližší okno s volnou kapacitou?

8.1.2 Kolik volných časových intervalů je k dispozici?

8.1.3 Jaké je nejbližší nejlevnější okno s volnou kapacitou?

8.2 Starcraft

8.2.1 Kolik hráčů je v jednotlivých divizích/úrovních?

8.2.2 Jsou konzistentní počty hráčů mezi JSON typ A a B?

8.2.3 Seřazení hráčů v rámci jednoho ladderu.

S touto otázkou si knihovna poradí celkem snadno, pokud chceme seřadit hráče v jedno ladderu a známe ID tohoto ladderu. S přiloženými daty si například vezmeme soubor *ladders-eu-230898.json*, což je ladder s ladderId=230898 a pomocí JS můžeme přistupovat k jeho atributům. Všechny hráči daného ladderu jsou uložené v atributu *team*. Vytvoříme si AlaSQL dotaz:

```
alasql('SELECT * FROM ? ORDER BY rating DESC', [ladder230898.team])
```

Vybereme všechny atributy a pomocí Funkce ORDER BY je můžeme seřadit sestupně nebo vzestupně.

8.2.4 Jací hráči vybočují z hranic divizí?

8.2.5 Kolik hráčů odehrálo v poslední době nějakou hru?

8.2.6 Jací hráči jsou v jednom ladderu vícekrát a za jaké rasy?

8.2.7 Jací hráči přibyli/ubyli v daných ladderech?

8.2.8 Kolik zástupců mají týmy v jednotlivých úrovních?

8.3 Objevené zajímavé funkce

Placeholder

The question mark is a placeholder. The queries all start off with 'alasql' followed by parens containing a string and an optional array containing placeholders.

AST

Umožňuje převádět jazyk SQL na AST (Abstract Syntax Tree), který můžeme programově interpretovat do našeho datového modelu.

```
var ast = alasql.parse("SELECT * FROM one");
console.log(ast.toString());                    // Vypise puvodni SQL dotaz
```

Tato funkce se může použít různě, například pro rychlejší sestavování větších dotazů.

pretty(sql)

Interně, pokud se ve svých dotazech vyznáme, je tato funkce asi zbytečná, ale je dobrá ji mít při vypisování, speciálně dlouhých, dotazů do konzole nebo kamkoliv jinam. Funkce totiž zkrášlí náš stringový dotaz.

Místo funkcí *alasql()* nebo *exec()* můžete využít také jednu z následujících funkcí:

```
alasql.value(sql, params, callback) - execute SQL statement but return only one
alasql.log(sql, params) - execute SQL statement and log the results to console o
```

První funkce *value()* vykoná dotaz, ale místo objektu nebo více hodnot vrátí pouze jednu. Druhá funkce *log()* opět vykoná dotaz a rovnou výsledek vypíše do konzole nebo přímo do HTML elementu. Což se velice hodí při debugování.

9 Chyby

10 Na závěr

Celkově se mi s knihovnou dělalo relativně dobře, ALE... pouze při práci společně s JavaScriptem. Dokumentace knihovny pokrývá základy a místy je vidět, že autoři mají připravené odkazy a stránky pro její rozšíření, bohužel je v ní takový celkem zmatek, některé věci jsou na více místech a většina z "live example"nefungují. Odkaz na knihovnu z wiki je ve verzi 0.3.9, což je starší verze, přitom píší, že verze v npm je verzi 0.6.4. Jediné relevantní stránky jsou ty na GitHubu, ostatní stránky v silné většině kopírují obsah a nelze načíst další relevantní zdroj. Bohužel, což mě hodně zklamalo a trošku odradilo jsou funkce, které uvádějí se zajímavými, ne-li potřebnými funkcemi a bohužel, když si je naimplementujete do svého kódu, tak vám to většinou vyhodí exception, že požadovaná funkce zavolaná nad objektem *alasql* není funkcí a nefunguje.

Na druhou stranu, knihovna nabízí opravdu mnoho zajímavých funkcí a pomocí JavaScriptu jde udělat prakticky cokoliv. Je zde vždy několik cest, jak s knihovnou pracovat, to znamená, že je možné různé problémy řešit různými cestami. Knihovna je velmi populární, podle statistik si ji stáhne cca 13 000 lidí týdně. Leč na knihovně se stále pracuje (soudě podle commitů) a věřím, že autoři

opraví stávající problémy a přidají ještě další funkce. Knihovna je opensource, takže kdyby jste potřebovali něco na ní založit, můžete. Také pokud vám tam něco chybí, není těžké si udělat vlastní funkci, která toto vykoná. Knihovna se z velké části používá jako normální SQL dotazy a uživatel JavaScriptu s ní nebude mít také problémy.

Seznam obrázků