

Dokumentace k semestrální práci z předmětu
KIV/DBM2

Bc. Tomáš Vyleta

Obsah

1	Knihovna AlaSQL	3
2	Stack	4
3	Instalace	4
4	Použití	4
4.1	Pomocí funkce	4
4.1.1	NodeJS	4
4.2	Nová databáze	5
4.3	Nová tabulka	5
4.3.1	Datové typy	5
4.4	Manipulace dat v tabulce	6
4.5	Načítání souborů	7
5	Synchronní/Asynchronní přístup	7
5.1	HTML	8
6	SQL dotazy	8
6.1	Nejlépe funguje s JS	8
7	Datasety	8
7.1	Rohlík.cz	8
7.2	Starcraft	9
8	Důležitá je cesta, ne cíl	9
8.1	Rohlík.cz	9
8.1.1	Jaké je nejbližší okno s volnou kapacitou?	9
8.1.2	Kolik volných časových intervalů je k dispozici?	10
8.1.3	Jaké je nejbližší nejlevnější okno s volnou kapacitou?	10
8.2	Starcraft	11
8.2.1	Kolik hráčů je v jednotlivých divizích/úrovních?	11
8.2.2	Jsou konzistentní počty hráčů mezi JSON typ A a B?	11
8.2.3	Seřazení hráčů v rámci jednoho ladderu.	11
8.2.4	Jací hráči vybočují z hranic divizí?	11
8.2.5	Kolik hráčů odehrálo v poslední době nějakou hru?	11
8.2.6	Jací hráči jsou v jednom ladderu vícekrát a za jaké rasy?	12
8.2.7	Jací hráči přibyli/ubyli v daných ladderech?	13
8.2.8	Kolik zástupců mají týmy v jednotlivých úrovních?	13
8.3	Objevené zajímavé funkce	14
9	Chyby	15
10	Popis přiložených souborů	15

11 Porovnání s SQL	15
12 Na závěr	16

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 snazší 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) a import/export databází IndexedDB, LocalStorage a SQLite.



Obrázek 1: Logo AlaSQL

Užitečné odkazy:

- WEB: <http://alasql.org>

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

2 Stack

Pro práci s knihovnou jsem zvolil *single page* aplikaci, kde všechny dotazy běží na klientovi s podporou JavaScriptu. Aplikace neběží na žádném serveru, tudíž nelze načítat soubory pomocí běžných metod, proto jsou všechna data uložena jako proměnné v JavaScriptu. Pokud ale pracujete s knihovnou v prostředí Node.JS, můžete soubory načítat pomocí *File system* modulu nebo pokud Node.JS nevyužíváte a aplikace vám běží na serveru, tak je zde možnost načítat soubory např. pomocí JQuery (asynchronně) nebo pomocí dalších externích knihoven a frameworků. Alasql dokáže načítat soubory asynchronně, ale metody kterými to jde, mají divné ukládání dat viz kapitola načítání dat.

3 Instalace

Knihovna je dostupná v několika JavaScriptových správcích balíčků (package management) kterými jsou npm, Bower a Meteor nebo si ji můžete přímo nainstalovat/stáhnout z oficiálního 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>
```

4 Použití

Po úspěšném nainstalování/nainportování knihovny jsme schopni ji začít používat. Nyní je několik možností jak s knihovnou pracovat.

4.1 Pomocí funkce

Všechny dotazy (queries) se vkládají jako atribut metody *alasql(stringWithSQL)* nebo jiné proměnné, pokud si je nazvete, třeba v Node.js. Dotazy lze vykonávat také nad jednotlivou tabulkou nějaké databáze pomocí funkce *exec()* (viz dále).

4.1.1 NodeJS

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

4.2 Nová databáze

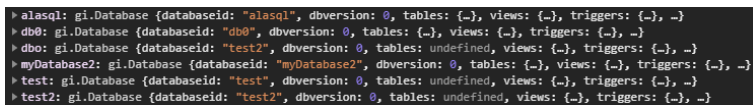
Můžeme do proměnné založit novou databázi a nad vytvořenou proměnnou zavolat funkci *exec()* do které vložíme dotaz jako atribut jako v předchozí sekci. Můžeme také vkládat více dotazů oddělených `;;` kde poté navracená hodnota bude jako pole výsledků.

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

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

Také k nim můžete přistupovat přes globální objekt *alasql*. Obecně se všechny databáze nacházejí na cestě *alasql.databases.[název databáze]*.

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



```
> alasql: gi.Database {databaseid: "alasql", dbversion: 0, tables: {}, views: {}, triggers: {}, -}
> db0: gi.Database {databaseid: "db0", dbversion: 0, tables: {}, views: {}, triggers: {}, -}
> db1: gi.Database {databaseid: "test2", dbversion: 0, tables: undefined, views: {}, triggers: {}, -}
> myDatabase2: gi.Database {databaseid: "myDatabase2", dbversion: 0, tables: {}, views: {}, triggers: {}, -}
> test: gi.Database {databaseid: "test", dbversion: 0, tables: undefined, views: {}, triggers: {}, -}
> test2: gi.Database {databaseid: "test2", dbversion: 0, tables: undefined, views: {}, triggers: {}, -}
```

Obrázek 2: Inicializované databáze v objektu *alasql* (příklad)

4.3 Nová tabulka

Založení nové tabulky je podobné, můžeme ji vytvořit přímo v globálním objektu *alasql* nebo vytvořit do nějaké databáze. 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 pomocí cesty *alasql.databases.[název databáze].tables.[název tabulky]* nebo jako SQL dotaz nad databází.

4.3.1 Datové typy

Datové typy, které jsou dostupné od začátku, tak se dají nalézt v *alasql* objektu pod cestou *alasql.fn.[název nového dat. typu] = [Datový typ]*. V základu jsou k dispozici tyto datové typy:

Příklad přidání nového datového typu do objektu databáze:

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

```

▶ Boolean: f Boolean()
▶ Date: f Date()
▶ Number: f Number()
▶ String: f String()

```

Obrázek 3: Základní datové typy

4.4 Manipulace dat v tabulce

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

```

mybase.exec("INSERT INTO cities VALUES ('Rome',2863223),
('Paris',2249975), ('Berlin',3517424),
('Madrid',3041579)");

```

INSERT jako JS funkce

Knihovna umožňuje také vytvářet vlastní metody a jedna z možností 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 nebo je rovnou přiřadit a opravdu to jde. Obecně se všechna data tabulky nacházejí v *alasql.databases.[název databáze].tables.[název tabulky].data*. Příklad objekt 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 neprocházejí přes žádnou funkci, tudíž data vůbec nemusí odpovídat schématu tabulky, při přidělení dat žádná výjimka nevyhodí, ale pokud by jsme nad daty zavolali nějaký dotaz, vyskočila by podmínka nebo by data byla *undefined*.

4.5 Načítání souborů

V dokumentaci je uvedené, že lze zřídit databázi z lokálního souborového systému nebo ze souboru SQLite (místo výroku FILESTORAGE -> SQLITE) a IndexedDB (místo výroku FILESTORAGE -> INDEXEDDB)

```
alasql('ATTACH FILESTORAGE DATABASE
test("./js/file.json")', function () {
    console.log(alasql.databases.test.data);
});
```

Vytvoří se databáze s názvem *test* s daty, která jsou v atributu data v dané databázi. Tento atribut pro jinak založené databáze není běžný, jelikož atribut data neobsahují, mají je až obsaženy v tabulkách databáze. Nepřišel jsem na způsob jak nad těmito daty použít dotazy, pouze jak přes adresu proměnné vypsat všechna data nebo dotazovat se pomocí placeholderu s odkazem na data. Je možné, že podporované databáze mají v souboru založené tabulky a Alasql je sám namapuje, pak lze nad nimi vyvolávat dotazy bez vyhození chyby.

5 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()*.

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

// promise
alasql.promise('SELECT * FROM cities')
    .then(function(res){
        // zpracování response
    }).catch(function(err){
        // zpracování chyby
    });
```


5.1 HTML

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

```
// ctení z tabulky
alasql('SELECT_*_FROM_HTML("#MyTable",{headers:true})');

// zápis do tabulky
alasql('SELECT_*_INTO_HTML("#MyTable",{headers:true})
FROM_?'',[data]);
```

6 SQL dotazy

Knihovna podporuje všechny základní výroky SQL jazyka, jako jsou 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. Sami autoři ve své dokumentaci uvádějí nějaké příklady společně s JavaScriptem a pokud příliš neznáte knihovnu nebo nevíte že nějaké funkce obsahuje, je snazší zpracovat dotaz pomocí JavaScriptu.

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

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

7 Datasetsy

7.1 Rohlík.cz

K dispozici máme data z webového portálu Rohlík.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 uživatele (*rohlik1.json*), 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 následující 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 výš je pak atribut *price* značící cenu dovozu.

7.2 Starcraft

Druhým datasetem jsou data ze hry Starcraft, máme k dispozici soubory dvou typů. Prvním typem (A) je popis jednotlivých **tierů**, kterých je dohromady 6, ale k dispozici máme pouze 5. a 6. Dále každý tier obsahuje pole po třech **divizích (úrovních)** kromě poslední 6. - ta obsahuje pouze jeden. Division má atributy *min_rating* a *max_rating* určující rozmezí hráčů. Každý division má pole obsahující pole s několika **laddery**, které již obsahují jednotlivé hráče. Tyto laddery se v průběhu času mění a maximální počet hráčů v jednom ladderu je 100. Laddery popisují soubory typu B, které jsou ve dvou různých datech. Jednotlivé laddery se dají rozeznat podle id a timestampu. V souborech je uložené pole *team* které obsahuje data o jednotlivých hráčích, muhu uvést některé, které budu používat, např. *last_played_time_stamp* - udávající poslední aktivitu hráče v ms, *member[0].legacy_link.name* - nick hráče s rozeznávacím tagem, *member[0].played_race_count.race* - udávající rasu, za kterou hráč hrál, atd.

8 Důležitá je cesta, ne cíl

V této sekci popíši všechny dotazy, které jsem pomocí Alasql řešil, jak jsem postupoval a popř. co se mi nepovedlo nebo knihovna nenabízí.

8.1 Rohlík.cz

8.1.1 Jaké je nejbližší okno s volnou kapacitou?

Tento dotaz jsem musel řešit s pomocí JavaScriptu, jelikož struktura dat Rohlík.cz v *slots* (rozvozová okénka) jsou atributy a tudíž nelze procházet tento atribut v Alasql jako pole, ale pouze přímými adresami na jednotlivé objekty, pokud chceme projít všechny rozvozová okénka, musíme použít k iteraci JavaScript. V tomto dotazu nám jde o nalezení nejbližšího volného okénka, což je nalezení takové hodnoty, kdy *totalFreeCapacityPercent* je větší než 0:

```
(alasql('SELECT slots FROM ?'
,[rohlikPremium.data.availabilityDays])).forEach(day => {
  for (const [key, value] of Object.entries(day.slots)) {
    const freeCapacity =
      (alasql('SELECT timeSlotCapacityDTO->totalFreeCapacityPercent
```

```

    AS freeCapacity FROM ?',[value]));
    if (freeCapacity[0].freeCapacity > 0) {
        console.log('První volné okénko dne je: '
            + value[0].since + ' do ' + value[0].till) ;
        break;
    }
}
}));

```

Prvním dotazem vybereme pomocí Alasql atribut *slots*, což je pole (dotaz je zde zbytečný, stačilo pouze přistoupit k poli pomocí JavaScriptu, ale chtěl jsem co nejvíce využívat Alasql). Procházíme každý den. Po té iterujeme každým časovým okénkem pomocí metody *for* pro iteraci objektů. V každém okénku si necháme pomocí dotazu napsaného v Alasql najít zanořený atribut *totalFreeCapacityPercent* a pomocí podmínky zjistit, jestli je větší jak 0, pokud ne opakujeme a pokud ano, tak necháme časové okénko vypsát a danou iteraci zrušíme pomocí *break* a iterujeme na další den. Tím se nám vypíší všechny okénka v každém dni:

```

První volné okénko dne je: 2020-11-02 13:00 do 2020-11-02 14:00
První volné okénko dne je: 2020-11-03 08:00 do 2020-11-03 09:00
První volné okénko dne je: 2020-11-04 08:00 do 2020-11-04 09:00
První volné okénko dne je: 2020-11-05 08:00 do 2020-11-05 09:00

```

Obrázek 4: Logo Výsledek dotazu Rohlik.cz - 1

8.1.2 Kolik volných časových intervalů je k dispozici?

Tento dotaz je podobný jako první, akorát nechceme pouze najít první volné časové okénko, ale všechny v ten daný den. Tudíž v podmínce nebudeme chtít iteraci přerušit, ale pouze aby pokračovala a pokud tam je volné okénko, tak ho přičíst k nějakému counteru a vypsát pro jednotlivý den, výsledek je:

```

Den ma 8 volnych dodavkovych okenek.
3 Den ma 13 volnych dodavkovych okenek.

```

Obrázek 5: Logo Výsledek dotazu Rohlik.cz - 2

8.1.3 Jaké je nejbližší nejlevnější okno s volnou kapacitou?

TODO

8.2 Starcraft

8.2.1 Kolik hráčů je v jednotlivých divizích/úrovních?

TODO

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

TODO

8.2.3 Seřazení hráčů v rámci jednoho ladderu.

S touto otázkou si knihovna poradí celkem snadno (bez použití JavaScriptu), pokud chceme seřadit hráče v jednom ladderu a známe id tohoto ladderu. S přiloženými daty si například vezmeme soubor *ladders-eu-230898.json*, což je ladder s *ladder_id=230898* a pomocí JS můžeme přistupovat k jeho atributům. Všechny hráči daného ladderu jsou uloženy v poli *team*. Chceme zobrazit id hráče, k případnému dalšímu dotazování, jméno hráče a rating, což bude seřazená hodnota. Vytvoříme si AlaSQL dotaz:

```
alasql('SELECT id , rating , member->(0)->character_link->battle_tag  
FROM ? ORDER BY rating DESC' , [ladderJson.team])
```

Vybereme všechny zmíněné atributy. Atribut jména hráče je zanořený, proto musíme volit přístup přes další atributy nebo pole a zobrazíme ho jako *battle_tag*, jinak by se celý atribut nazýval podle cesty, které k němu přistupujeme (v tomto případě by se vybraný atribut jmenoval 'member->(0)->character_link->battle_tag'). Vybraná data můžeme seřadit podle atributu *rating*, pomocí výrazu *ORDER BY* sestupně nebo vstoupně. Jako výsledek dotazu nad ladderem s *ladder_id=230898* nám dotaz sestrojí následující výsledek:

Jediné na co jsem během tohoto dotazu narazil je, že jsem chtěl, aby si uživatel volil metodu seřazení, jestli ASC nebo DESC přes placeholder pomocí *?*, bohužel ale při překladu aplikace padala a vyžadovala již definovanou metodu seřazení vloženou přímo jako string v dotazu.

8.2.4 Jací hráči vybočují z hranic divizí?

TODO

8.2.5 Kolik hráčů odehrálo v poslední době nějakou hru?

K tomuto dotazu knihovně opět postačí pouze jeden dotaz, se dvěma vstupními parametry, které nahradíme místo placeholderů a to určitý ladder a konstanta času v ms proti které budeme hráče porovnávat. AlaSQL dotaz je:

```
alasql.exec('SELECT COUNT(*) FROM ? WHERE  
last_played_time_stamp > ?', [ladderJson.team, timeFromInMilis])
```

```

▶ 0: {id: "15100607019867963392", rating: 6015, battle_tag: "Morris#21456"}
▶ 1: {id: 6131688361959621000, rating: 5881, battle_tag: "llllllllll#1769"}
▶ 2: {id: "17590824934920159232", rating: 5867, battle_tag: "hodor#2879"}
▶ 3: {id: "16861226402123350016", rating: 5800, battle_tag: "WingsOfFire#21508"}
▶ 4: {id: 8640186757335220000, rating: 5771, battle_tag: "JunO#1480"}
▶ 5: {id: 6695757618218009000, rating: 5641, battle_tag: "Tartem#2108"}
▶ 6: {id: 5761260695538631000, rating: 5620, battle_tag: "JustLeaveNow#2171"}
▶ 7: {id: 3229681352072757000, rating: 5592, battle_tag: "Progamer#22909"}
▶ 8: {id: "12505976182107275264", rating: 5560, battle_tag: "froz#2369"}
▶ 9: {id: 8720978871744201000, rating: 5558, battle_tag: "endrju#2412"}
▶ 10: {id: 2602275828007960600, rating: 5550, battle_tag: "Irbis#1136"}
▶ 11: {id: 1010527138127806500, rating: 5547, battle_tag: "Tranzistor#21474"}
▶ 12: {id: "15357872949558247424", rating: 5481, battle_tag: "llllllllll#21429"}
▶ 13: {id: 5004659256675271000, rating: 5414, battle_tag: "Ditrih#21911"}
▶ 14: {id: 4827608197789385000, rating: 5378, battle_tag: "digbickdaddy#2501"}
▶ 15: {id: "13837623301809111040", rating: 5370, battle_tag: "Mookashade#2499"}
▶ 16: {id: "15953199122371051520", rating: 5355, battle_tag: "Surprise#2673"}
▶ 17: {id: 7345120389489492000, rating: 5355, battle_tag: "Mantaza#2752"}
▶ 18: {id: "10226861201053188096", rating: 5337, battle_tag: "Shoes#2851"}
▶ 19: {id: "11166708347313324032", rating: 5331, battle_tag: "Goomba#1920"}
▶ 20: {id: 4725715355731755000, rating: 5317, battle_tag: "Protoss#21530"}
▶ 21: {id: 8569255063204135000, rating: 5310, battle_tag: "NEEK#2928"}
▶ 22: {id: "10303985344671907840", rating: 5305, battle_tag: "Seracis#2371"}
▶ 23: {id: "13903487346847776768", rating: 5294, battle_tag: "JustLeaveNow#2171"}
▶ 24: {id: 4411589281722663000, rating: 5251, battle_tag: "Project#21119"}
▶ 25: {id: 8641312657242063000, rating: 5250, battle_tag: "Bloop#2209"}
▶ 26: {id: "15722103768445091840", rating: 5238, battle_tag: "RhCn#2502"}
▶ 27: {id: "16455626458241433600", rating: 5230, battle_tag: "HànTattoo49#2314"}
▶ 28: {id: 5441506221506953000, rating: 5204, battle_tag: "Dext#2339"}
▶ 29: {id: "1145324764581516288", rating: 5204, battle_tag: "DasDuelon#2923"}
▶ 30: {id: 8205594890852106000, rating: 5168, battle_tag: "Fortuna#2755"}

```

Obrázek 6: Logo Výsledek dotazu Starcraft - 3

Chceme zobrazit číslo, tedy počet hráčů, kolik hrálo Starcraft od časové značky. Použijeme agregační funkci COUNT buď s nějakým atributem nebo hvězdičkou, což je v tomto případě jedno. Musíme přidat také klauzuli WHERE a říci, že hledáme hráče, kteří mají *last_played_time_stamp* větší než naše časové razítko. Výsledkem dotazu je:

```
▶ 0: {COUNT(*): 16}
```

Obrázek 7: Logo Výsledek dotazu Starcraft - 5

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

Tento dotaz jsem se snažil vyřešit pouze s pomocí Alasql, ale k výsledku jsem se dobral pouze s pomocí JavaScriptu. Vstupním parametrem je jeden ladder.

```

result = '';
temp = alasql('SELECT member->(0)->
character_link->battle_tag as name, member->(0)->
played_race_count->(0)->race as race FROM ?',
[ladderJson.team]);

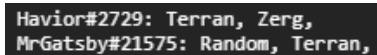
```

```

    alasql('SELECT name FROM ? GROUP BY name
    HAVING COUNT(*) > 1 ORDER BY name',
    [temp]).forEach(dupPlayer => {
        result += dupPlayer.name + ': '
        temp.forEach(player => {
            if (dupPlayer.name === player.name) {
                result += player.race + ', ';
            }
        });
        result += '\n';
    });
});

```

Nejdříve inicializuji prázdný string, který mi bude pomáhat s vypsáním výsledků dotazů. Nejdříve si do proměnné *temp* zobrazím *battle_tag* hráčů jako *name* a rasu jako *race* za kterou hráči hráli. V dalším dotazu zobrazím *name* hráče, který se vyskytl v daném ladderu vícekrát. Po té si pomocí JavaScriptu pro každého hráče nechám vypsat jméno a rasy, za které hrál. Výsledek nad ladderem s *ladder_id=230882* vypadá takto:



Obrázek 8: Logo Výsledek dotazu Starcraft - 6

8.2.7 Jací hráči přibyli/ubyli v daných ladderech?

V tomto dotazu chceme porovnat dvě instance stejného ladderu v jiných časových obdobích, tudíž vstupem budou dva zmíněné parametry. Tímto dotazem si Alasql poradí pomocí dvou dotazů nebo jedním zanořeným:

```

union = (alasql('SELECT member->(0)->character_link->
battle_tag as name FROM ? UNION ALL SELECT member->(0)->
character_link->battle_tag as name FROM ?',
[ladderJson1.team, ladderJson2.team]));

```

```

    alasql('SELECT name FROM ? GROUP BY name
    HAVING COUNT(*)=1',[union])

```

Prvním krokem se sestavení UNION výroku, který vezme z obou instancí *battle_tag* hráčů jako *name* a vytvoří z nich pouze jednu tabulku. Nad touto tabulkou zobrazíme pouze ty hráče, kteří se v obou instancích nacházejí pouze jednou. Nevíme ale, jestli daní hráči přibili nebo ubyli. Výsledek dotazů mezi ladderem s *ladder_id=230882* je:

8.2.8 Kolik zástupců mají týmy v jednotlivých úrovních?

TODO

```

▶ 0: {name: "Intebasnygg#2196"}
▶ 1: {name: "siao#2503"}
▶ 2: {name: "LuPin#2455"}

```

Obrázek 9: Logo Výsledek dotazu Starcraft - 7

8.3 Objevené zajímavé funkce

Placeholder

Otazník v dotazech je tzv. placeholder. Všechny dotazy, které jsou volány přes Alasql mohou mít v dotazu otazník, který je pak nahrazen hodnotou v poli za výrazem, které obsahuje odkazy na zdroje.

```
alasql('SELECT a->b FROM ?',[data]);
```

Vlastní funkce

Alasql je rozšiřitelná a umožňuje nám vytvářet vlastní funkce s použitím JavaScriptu, které pak můžeme zavolat přímo v SQL dotazu.

```

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

```

```

alasql.aggr.LENGTH = function(array) {
    return array.length;
}

```

Abstract szntax tree

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

```

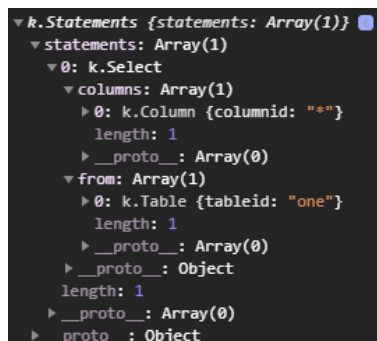
var ast = alasql.parse("SELECT * FROM one");
console.log(ast.toString()); // Vypise puvodni SQL dotaz

```

Tato funkce se může použít různě, například pro rychlejší sestavování větších dotazů. Pro představu uvedu strukturu rozloženého objektu:

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.



Obrázek 10: Struktura AST stromu v JavaScriptu

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

```
alasql.value(sql, params, callback)
alasql.log(sql, params)
```

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

TODO nelze prochazet promenne nefunkci metody - napr. arrayOfArray

10 Popis příložených souborů

K této dokumentaci také příkládám soubory s příklady a s řešením popsaných otázek. TODO - popis souborů + struktura

11 Porovnání s SQL

Knihovna je primárně navržena aby se s ní co nejvíce pracovalo jako s SQL. Obsahuje všechny základní výroky (SELECT, DISTINCT, FIRST, FROM, WHERE, ORDER BY, atd.) agregační funkce (SUM(), AVG(), GROUP BY, atd.), stringové, numerické, logické a datové funkce (ABS(), UPPER(), YEAR(), ...), JOIN operace (INNER JOIN, LEFT JOIN, OUTER JOIN, ...) a řízení transakcí příkazy (BEGIN, COMMIT a ROLLBACK). TODO - SQL99 standart Pokud si vytvoříte databázi pomocí Alasql, chová se stejně jako SQL databáze, akorát ukládání dat je jiné. Data jsou uloženy v JavaScriptovém objektu a máme k nim

přístup i bez dotazovacího jazyka přes objektovou cestu. Data je možné ukládat i do Localstorage nebo jiné podporované databáze.

12 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 trochu 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 nějaký projekt 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.

Reference

- [1] *Github - AlaSQL* [online]. [cit. 2020-11-16]. Dostupné z: <https://github.com/agershun/alasql>
- [2] *Github - AlaSQL/Wiki* [online]. [cit. 2020-11-16]. Dostupné z: <https://github.com/agershun/alasql/wiki>
- [3] *W3C - SQL* [online]. [cit. 2020-11-16]. Dostupné z: <https://www.w3schools.com/sql/default.asp>
- [4] *cdnjs - alasql* [online]. [cit. 2020-11-16]. Dostupné z: <https://cdnjs.com/libraries/alasql>
- [5] *DZone - alasql-in-action-the-javascript-sql-database* [online]. [cit. 2020-11-16]. Dostupné z: <https://dzone.com/articles/alasql-in-action-the-javascript-sql-database>

- [6] *DEV* - *alasql-a-real-database-for-web-browsers-and-nodejs* [online].
[cit. 2020-11-16]. Dostupné z: [https://dev.to/jorge_rockr/
alasql-a-real-database-for-web-browsers-and-nodejs-24gj](https://dev.to/jorge_rockr/alasql-a-real-database-for-web-browsers-and-nodejs-24gj)

Seznam obrázků

1	Logo AlaSQL	3
2	Inicializované databáze v objektu <i>alasql</i> (příklad)	5
3	Základní datové typy	6
4	Logo Výsledek dotazu Rohlik.cz - 1	10
5	Logo Výsledek dotazu Rohlik.cz - 2	10
6	Logo Výsledek dotazu Starcraft - 3	12
7	Logo Výsledek dotazu Starcraft - 5	12
8	Logo Výsledek dotazu Starcraft - 6	13
9	Logo Výsledek dotazu Starcraft - 7	14
10	Struktura AST stromu v JavaScriptu	15