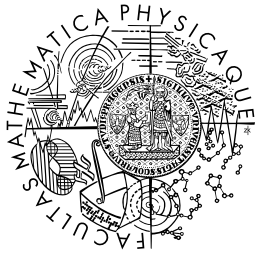

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

# BACHELOR THESIS

Přemysl Šťastný

# lsql-csv

Department of Applied Mathematics

Supervisor of the bachelor thesis: doc. Mgr. Jan Hubička, Ph.D.
Study programme: study programme
Study branch: study branch

Prague 2024

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  .....

Author's signature

Dedication.

Title: lsql-csv

Author: Přemysl Šťastný

Department: Department of Applied Mathematics

Supervisor: doc. Mgr. Jan Hubička, Ph.D., Department of Applied Mathematics

Abstract: Abstract.

Keywords: key words

# Contents

# Introduction

Database refers to a set of related data accessed through the use of a database management system [1]. CSV files (Comma Separated Value files) are common way of exchanging and converting data between various spreadsheet programs [2]. Through this definition, we can see even a simple collection of CSV files accessed through some programs may be seen as an database itself.

SQL (Structured Query Language) is a language used to manage data, especially in a relational database management system [3]. It was first introduced at 1970s [3] and is one of the most used query languages. Despite being standardized at 1987 by International Organization for Standardization [4], there are virtually no implementations adhere to it fully [3]. Standard SQL is a typed language (Every data value belongs to some data type) [5] and the language design is therefore not much suitable for type-less databases like a collection of CSV files. Despite that, there are some implementations of SQL (eg. *q* [6], *CSV SQL* [7], *trdsql* [8] or *csvq* [9]), which tries to implement SQL on CSV files.

SQL itself requires large amount of text to be written for running even a simple queries and Unix ecosystem misses a tool[1], which would allowed running a short enough queries over CSV files with similar semantics like SQL has. And this is the reason, why *lsql-csv* was created.

*lsql-csv* is a tool for small CSV files data querying from shell with short queries. It makes possible to work with small CSV files like with read-only relational database. The tool implements a new language LSQL similar to SQL, which is type-less, specifically designed for working with CSV files in shell.

Haskell is a language with great features for working with text [10] and therefore it was selected for the task of implementation of lsql-csv.

[1] Or author don't know about it

# 1. User documentation

## 1.1 Installation

## 1.2 Title of the second subchapter of the first chapter

# 2. Developer documentation

This chapter is for potential developers of the project.

## 2.1 Project building and testing

The project have two ways of building. One way is through Makefile and second through cabal. By running the following command it is generated build folder and lsql-csv binary under it.

```
make
```

It is necessary to have all Haskell dependencies (*Parsec* ($\geq 3.1 < 3.2$), *Glob* ($\geq 0.10 < 0.11$), *base* ($\geq 4.9 < 4.17$), *text* ($\geq 1.2 < 2.1$) and *containers* ($\geq 0.5 < 0.7$)) installed. The package boundaries given are identical to cabal bounderies. Also it is necessary, you had *GHC* ($\geq 8 < 9.29$) installed.

Second way of building is through cabal, which handles all dependencies for you.

```
cabal build
```

The project unit tests require building through *Makefile* and is called by:

```
make test
```

It should always succeed before any commit is made.

It is possible to generate Haddock developer documentation by calling:

```
cabal haddock
```

The documentation contains comments on all exported functions. The documentation can be alternatively generated by running:

```
make docs
```

It generates html documentation under build folder.

## 2.2 *String* vs *Data.Text*

As there is a long term discussion in Haskell community whether *String* or *Data.Text* should be used as the primary representation for text, I would like to emphasize that in this project, String is used as a primary representation for text.

## 2.3 Project layout

The project is split to:

1. a library, which contains almost all the logic and is placed under *src* folder of the project
2. the *main*, which contains one source file with *Main*, which is the entry point for *lsql-csv* binary. It parses the arguments, checks, whether help optional argument was called or no argument at all was given, and either shows help or further call run from *Lsql.Csv.Main* from library.

Library is split into 5 different namespaces. Its usage is not strictly defined, but this can be said

- *Lsql.Csv* – This namespace contains the entry point
- *Lsql.Csv.Core* – This namespace contains the logic of evaluation
- *Lsql.Csv.Lang* – This namespace contains parsers
- *Lsql.Csv.Lang.From* – This namespace cointains parsers for from block
- *Lsql.Csv.Utils* – This namespace contains helper functions

## 2.4 Modules

The following section is a summary of all modules of the library.

- *Lsql.Csv.Core.BlockOps* – This module contains the *Block* definition and functions for getting specific types of blocks from list of *Block*.
- *Lsql.Csv.Core.Evaluator* – This module contains the evaluator of *lsql-csv* program.
- *Lsql.Csv.Core.Functions* – This module contains the syntactic tree definition and helper functions for its evaluation.
- *Lsql.Csv.Core.Symbols* – This module contains the definition of *Symbol*, *SymbolMap* and helper functions.
- *Lsql.Csv.Core.Tables* – This module contains the definition of *lsql-csv* data types, classes over them and types *Table* and *Column* and functions over them for manipulation of them.
- *Lsql.Csv.Lang.Args* – A module for command line argument parsing.
- *Lsql.Csv.Lang.BlockChain* – This module contains a main parser of blocks other than from block.
- *Lsql.Csv.Lang.BlockSeparator* – This module contain a preprocessor parser, which splits command into list of strings - one string per one block.

- *Lsql.Csv.Lang.Options* – This module implements a common *Option* type and it is parsers for from blocks and command line optional arguments.
- *Lsql.Csv.Lang.Selector* – This module implements the selector expression and arithmetic expression parsers.
- *Lsql.Csv.Lang.From.Block* – This module contains the from block parser. It loads initial *SymbolMap*.
- *Lsql.Csv.Lang.From.CsvParser* – This module contains *CsvParser* called by *parseFile*, which loads input CSV files.
- *Lsql.Csv.Main* – This module contains the starting point for *lsql-csv* evaluation.
- *Lsql.Csv.Utils.BracketExpansion* – This module contains the curly bracket (braces) expansion implementation.
- *Lsql.Csv.Utils.CsvGenerator* – This module contains the CSV generator for the output.

## 2.5 Entry point

The entry point is in module *Lsql.Csv.Main* in function *run*. The function first calls preprocesor in *Lsql.Csv.Lang.BlockSeparator*, which splits command into list of strings. Then *SymbolMap* with input is loaded using *Lsql.Csv.Lang.From.Block* and after that rest of blocks are parsed using *Lsql.Csv.Lang.BlockChain*. Program is then evaluated using *Lsql.Csv.Core.Evaluator* and finally output generated by *Lsql.Csv.Utils.CsvGenerator*.

# 3. Analysis

Why the language have been made the way it is? Why it is so inspired by SQL and is not just next implementation of SQL? Why it is implemented the way it is? Further chapter is about design decision of the language.

## 3.1 Why SQL at first place?

Why we talk so much about SQL at first place? Since it was introduced in 1970s [3], it has become de facto standard for many major databases. Just for illustration name a few of them.

- Oracle DB have used it since 1979 as a first commercially available implementation [11].
- MySQL have used it since 1994, since its original development started [12]
- PostgreSQL have used it since 1996, since it was created [13].
- MSSQL have used it since 1989, since its initial release [14].

SQL is so much known, that there is a widely used term NoSQL databases as databases opposed to SQL databases.

The main point of making language inspired by SQL is that it brings the advantage of getting large user base which only needs to understand the difference between SQL and the new language to start using the new language. This is the starting point, from which we further argument about design decisions made.

## 3.2 Why not implement just another SQL for CSV files?

As mentioned in the introduction, standard SQL is a typed language [4], which implied many design choices. Furthermore, SQL itself require large amount of text to be written, before it can be executed.

One of the ambition of *lsql-csv* is to allow a user to write shorter queries to get the result. For the example, consider

```
select A.dataX from data A where A.dataX > 1000
```

This simple SQL query shows $dataX > 1000$ from table *data*. Now, if we have a CSV file *data.txt*, where we know, that *dataX* is a second column, the same query can be written with *lsql-csv* as

```
data.txt, &1.2, if &1.2 > 1000
```

The length difference is about 35% off the original query. It is simply said the another reason, why we will not just implement another SQL implementation.

## 3.3 Why number references?

Where the 35% difference happened. One and the main reason, why is, that we allowed referencing *dataX* as *.2*. This is also possible due to nature of CSV file, where columns have their index [1]. Normally, in SQL database, indexes of columns are not considered as something, which should decide about query meaning. This is because the SQL database itself may change, new columns may be added or removed. Just because somebody removed a column, it is not wanted, that developers needed to change some of the queries so they comply with the new database layout.

On the other hand, CSV files usually are not altered as much as databases are. If developer (or an user) is not sure about the columns of the CSV file, it is one of the first signs, that he or she should use rather SQL database than simple CSV file.

Also *lsql-csv* is rather a tool for daily life and simple scripts rather than tool for development of medium sized or large sized project, like SQL is. This adds much more flexibility in what can language do (like number references).

## 3.4 Why renaming standard keywords?

Why have we renamed *where* for *if*, *group by* for *by* and *sort by* for *sort*? Simply said, because the renamed variants are shorter and still do not block user in understanding, what the query does.

## 3.5 Why some features like descending sort are missing?

*lsql-csv* is a shell utility and as such, it tries to comply with UNIX philosophy. The summarized version from Doug McIlroy is: 'This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.' [15].

The tool for reverting the order of sorted output already exist: *tac*. Piped together it creates the wanted output.

Similar case, the function for second filtering of grouped by (in SQL named *having*) output have been not added, because the wanted output may be received by piping the output with another instance of *lsql-csv*.

And why there is no support for creating output with first line names of columns? Because the wanted output may be simply made with usage of *echo* called before *lsql-csv* if needed.

## 3.6 Why blocks are delimited by comma?

The author thinks, it is more readable like this. Developers in SQL usually use upper case and new line writing to delimiter the blocks and comma is just more

[1] Do not confuse with SQL database index for performance

easy to write than switching on and off caps lock, holding shift key or making multiline input.

## 3.7 Why there are two types of expression?

Why there are arithmetic and select expression with different grammars? We think, that addition of special select expression further allows the user to write shorter queries. It allows us to introduce wildcard and curly brackets expansion for the user, which will not be possible otherwise.

## 3.8 Why there is support only for cross join and not other types of join?

The tool is supposed to be simple. We recommend user to import CSV data to SQL database and use standard SQL if he needs more complicated joins.

The other reason is, that CSV have no standardized NULL value, which is needed for left or right outer joins.

## 3.9 Why there is no package used for CSV parsing and generating?

Very simply said, to limit number of dependencies. The more dependencies is used, the harder is to compile it, maintain it and add to any Linux distribution.

As CSV parsing is not hard job, it was decided so.

## 3.10 Why *String* is used as primary text representation?

As the performance gain from using *Data.Text* would not be significant and *String* does not add any more complexity to the code like *Data.Text* does, it was decided that *String* will be primary text representation.

## 3.11 Why joins have $\mathcal{O}(nm)$ complexity?

*lsql-csv* is a tool for small dataset data querying. As such, it implements only simple algorithm for joining the tables – cross join.

For larger dataset users are encouraged to use standard SQL database.

# 4. Alternative solutions

What other solutions are there for the given problem? What other approaches can we use, when we are dealing with queries over CSV files? The following chapter is about alternative approaches to the problem.

## 4.1 Using SQL database

The first obvious solution is importing the dataset to some of standard SQL database and do the queries over it. This approach requires definition of schema to which the data will be imported. This is an overhead, which isn't always advantageous to pay as we might need only a simple query to be run over it.

But it might be vary favorable solution, if we have large dataset, need complex query or a large amount of simple queries run over it.

By using a standard SQL database you gain the advantage of better performance, typed dataset, indexes and large amount of build-in functions

## 4.2 Using standard Unix tools

It is possible to do large amount of work just by using *awk*, *join*, *sort*,... For example:

```
lsql-csv -d: '-, &1.*, if &1.3>=1000' < /etc/passwd
```

This query might be rewritten using *awk* to

```
awk -F: '{ if($3 >= 1000){ print $0 }}' </etc/passwd
```

The main advantage of *lsql-csv* is that it handles some more complex queries more easily. For example:

```
lsql-csv -d: '/etc/{passwd,group}, &1.1 &2.1, if &1.4 == &2.3'
```

This is the simple join query. When written using standard Unix tools, it is:

```
sort -t: -k3,3 /etc/group > /tmp/group.sort
sort -t: -k4,4 /etc/passwd > /tmp/passwd.sort
join -t: -14 -23 /tmp/passwd.sort /tmp/group.sort | cut -d: -f2,8
```

As demonstrated, the *lsql-csv* variant is more readable and shorter. [1]

It should be also noted, that *lsql-csv* join have $\mathcal{O}(nm)$ time complexity, while standard Unix tools have for join written above $\mathcal{O}(n \log n + m \log m)$ time complexity, so for larger dataset, it might be more beneficial to use them.

[1] It is possible, it can be written in shorter and more readable form, but not more than the *lsql-csv* variant.

## 4.3 By using SQL implementation for CSV files

There are many projects implementing SQL on CSV files. For example:

- *q* [6]
- *CSV SQL* [7]
- *trdsql* [8]
- *csvq* [9]

It is possible to use them to do the job directly with SQL.

The advantage of it is you don't have to learn new language if you already know standard SQL. The disadvantage is that the queries will be probably longer than would be with *lsql-csv*.

## 4.4 By using general purpose programming language

It is not hard to parse and process CSV files by general purpose programming language (for example Python).

The advantage of this solution is a much greater flexibility of what you can do with the CSV files. The large disadvantage is, it will take to much code to be written for any query.

# Conclusion

# Bibliography


[1] Wikipedia contributors. Database — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Database&oldid=1200665358, 2024. [Online; accessed 16-February-2024].

[2] Yakov Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180, October 2005. Available at https://www.rfc-editor.org/info/rfc4180 [Online; accessed 16-February-2024].

[3] Wikipedia contributors. Sql — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=SQL&oldid=1205226098, 2024. [Online; accessed 16-February-2024].

[4] Iso 9075:1987: Information technology – database languages sql – part 1: Framework (sql/framework). Standard, International Organization for Standardization, Geneva, CH, June 1987.

[5] Iso 9075-1:2023: Information technology – database languages sql – part 1: Framework (sql/framework). Standard, International Organization for Standardization, Geneva, CH, June 2023.

[6] q - text as data. https://github.com/harelba/q, 2022. [Online; accessed 18-February-2024].

[7] Csv sql. https://github.com/alex/csv-sql, 2021. [Online; accessed 29-February-2024].

[8] trdsql. https://github.com/noborus/trdsql, 2024. [Online; accessed 29-February-2024].

[9] csvq. https://github.com/mithrandie/csvq, 2023. [Online; accessed 29-February-2024].

[10] Alejandro Serrano Mena. *Practical Haskell - A real world guide to programming*. Second Edition. Apress, New York, 2019.

[11] et al. Usha Krishnamurthy. *Oracle Database SQL Language Reference, 19c*. Oracle, 2023. Available at https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/sql-language-reference.pdf [Online; accessed 23-February-2024].

[12] Wikipedia contributors. Mysql — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=MySQL&oldid=1205045759, 2024. [Online; accessed 23-February-2024].

[13] https://www.postgresql.org/about/news/happy-birthday-postgresql-978/, July 2008. [Online; accessed 23-February-2024].

[14] Wikipedia contributors. Microsoft sql server — Wikipedia, the free encyclopedia. "https://en.wikipedia.org/w/index.php?title=Microsoft_SQL_Server&oldid=1210428592", 2024. [Online; accessed 29-February-2024].

[15] Eric S. Raymond. http://www.catb.org/~esr/writings/taoup/html/ch01s06.html, September 2003. [Online; accessed 24-February-2024].

# List of Figures

# List of Tables

# List of Abbreviations

# A. Attachments

## A.1 First Attachment