THE UNIVERSITY OF HONG KONG

Data Processing Domain-Specific Language in Clojure

CHOI CHONG HING, PARRY

ABSTRACT

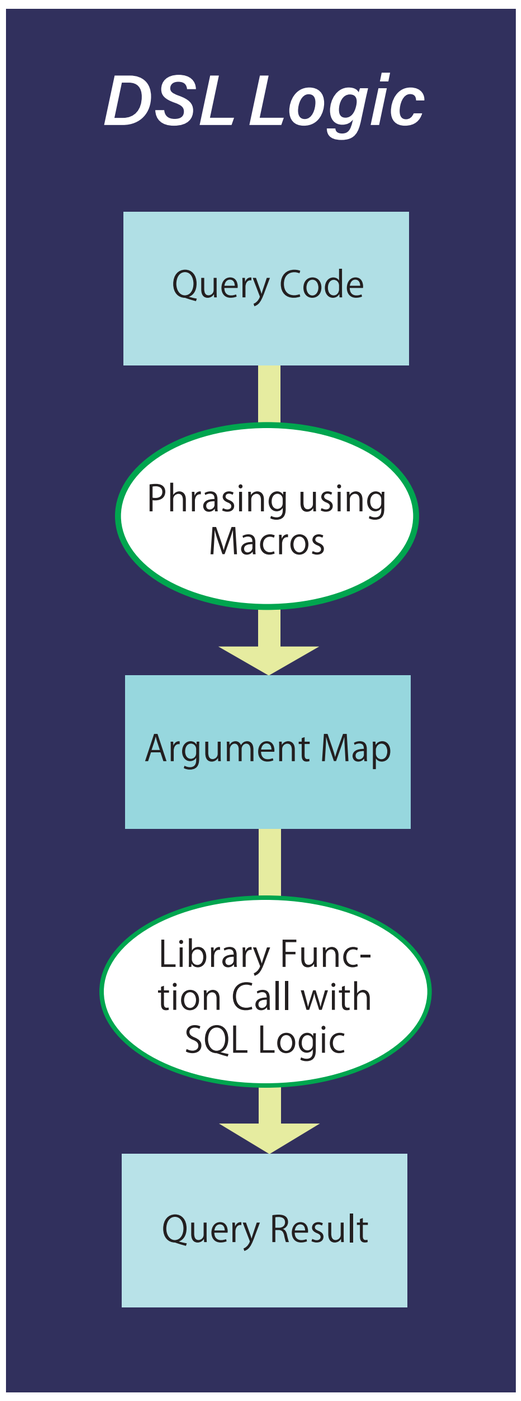
Clojure is one of the dialects of Lisp, developed with the code-as-data philosophy. Together with the powerful macro system, a variety of Domain-specific Languages (DSL) could be developed conveniently. This research project focus on integrating panda style data processing DSL into Clojure, exploring the limits of Clojure. The developed data processing DSL is based on an existing data processing library, tech.ml.dataset. It provides the basic data processing operations for the project. This development of the DSL has three stages: conceptual, fundament and syntax design. The conceptual phase involves experimenting with Clojure macro system and defining the project goal. The Fundament stage involves the development of the fundamental pipeline of the DSL, following the logical processing order of the SELECT statement in Structured Query Language (SQL). Lastly, during the syntax design part, the limits of Clojure syntax was explored.

USE OF MACROS

The macro system in Clojure allows the compiler to be extended by code. It reads the input code as data, phrasing it into a different code for execution. The transformation using the macros could be defined by the user.

SQL

Structured Query Language is a declarative query language designed for managing the data in a Relational Database Management System (RDBMS). The SELECT statement query of the SQL is very similar to the query of the DSL developed in this project. The logical processing order of the SELECT statement has been adopted in the DSL.



DSL LOGICAL PROCESSING ORDER

1	WHERE	Specifies the search condition for the rows
2	ROW	Specifies the row index for the rows
3	GROUP BY	Divides the query into groups of rows
4	HAVING	Specifies the search condition for a group or an anggregate
5	SELECT	Specifies the columns to be returned
6	ORDERED BY	Sorts the data returned

RESULTS

Query Syntax

```
dt-get data ‘[ROW-SELECTION-SECTION &
          COL-SELECTION-SECTION & OPTIONS]
```

Row Selection Syntax

```
[col filter-function]  OR  row-index
```

Column Selection Syntax

```
col
```

Optional Syntax

```
operation-keyword operation-args
```

Group by Sort by

```
:group-by col  :sort-by col sorting-funtion
```

Aggregate Column

```
aggregate-keyword col
```

EXAMPLES

Select rows with salary > 300, age < 20

```
dt-get data ‘[[:salary #(> % 300)] [:age #(< % 20)] & :*]
```

Group rows by age with sum of salary > 1000

```
dt-get data ‘[[:sum :salary #(> % 1000)] & :age :sum
:salary & :group-by :age]
```

Group rows by age, sort by SD of salary in descending order

```
dt-get data ‘[:* & :age :sd :salary & :group-by :age
:sort-by :sd :salary >]
```

Group rows by age and name

```
dt-get data ‘[:* & :age :name :sum :salary &:group-by
:age :name]
```