Assignment 2: Macros in Clojure

The Assignment

This assignment is a modified version of an assignment developed by Isak Karlsson and Beatrice Åkerblom in 2013.

The assignment is about implementing macros in Clojure and write some reflections.

Safe Macro

The first macro should function similar to try in Java as in the following example:

```
try (Socket s = new Socket()) {
  ...
}
```

In the code above the socket 's' is automatically closed, just as if you explicitly would have written a finally clause:

```
try (Socket s = new Socket()) {
  ...
}
finally {
  if (s != null) s.close();
}
```

You should define a macro 'safe' with two arguments. The first argument should be a vector with two elements, one variable and one value (corresponding to 's' and 'new Socket()' in the example above). The second argument should be a form (expression) that should be evaluated. The macro 'safe' should either return the return value of the second argument or an exception. In both cases the variable should be closed (if it is an instance of java.io.Closeable; hint use 'instance?'). It should be possible to use the variable inside the form (expression) as in the following example:

```
=> (import java.io.FileReader java.io.File)
...
=> (safe [s (new FileReader (new File "file.txt"))] (.read s))
104
```

The return value 104 is the first byte in the file "file.txt".

SQL-like Macro

The second macro should implement an SQL-like syntax for searching in lists of maps, as illustrated by the following example:

```
=> (def persons '({:id 1 :name "olle"} {:id 2 :name "anna"} {:id 3 :name
"isak"} {:id 4 :name "beatrice"}))
...
=> (select [:id :name] from persons where [:id > 2] orderby :name)
({:id 4 :name "beatrice"} {:id 3 :name "isak"})
```

The query format should be:

```
(select [columns] from #{table} where [:column op value] orderby :column)
```

where the condition operator 'op' is one of the following: `=` `<` `>` `<>`

Reflections

Argue whether these two macros can be implemented as functions or not (max 500 words).

Grading

Each part, safe macro, SQL-like macro and reflection, will be given a valuation with respect to the quality of the implementation/argumentation:
a) NotOk means it is missing or not functioning,
b) Ok means it is substantially functioning and the code/argumentation is ok written and ok structured.
c) Good means it is completely functioning and the code/argumentation is well written and well structured.

The following table will be used for grading:

Grade	**Safe Macro**	**SQL-like Macro**	**Reflection**
A	Good	Good	Good
B	Good	Ok	Ok
C	Good	NotOk	Ok
D	Ok	NotOk	Ok
E	Ok	NotOk	NotOk

Good luck!