(http://safeassign.blackboard.com/)

REVISTA - PROF. UNIV. DR. MANUELA EPURE

Prof. univ. dr. Manuela Epure
on Fri, Dec 30 2016, 3:23 PM

13% match
Submission ID: 127973990

Attachments (1)

6. Radu.docx ⓘ **null%**
Word Count: 1,064 Attachment ID: 148614270

6. Radu.docx

×

Plagiarism attempt possible

ⓘ SafeAssign found that some of the words in this submission contain characters from different languages, or character sets. Mixing character sets does not always mean plagiarism has occurred, but it can be used to avoid automatic detection. You may want to review this document manually to verify its originality.

1 **(ONLINE) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 JOURNAL OF ECONOMIC DEVELOPMENT, ENVIRONMENT AND PEOPLE VOLUME 5, ISSUE 2, 2016**

URL: 2 **HTTP://JEDEP.SPIRUHARET.RO E-MAIL:** 3 **OFFICE_JEDEP@SPIRUHARET.RO**

1 **(ONLINE) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 JOURNAL OF ECONOMIC DEVELOPMENT, ENVIRONMENT AND PEOPLE VOLUME 5, ISSUE 4, 2016**

URL: 2 **HTTP://JEDEP.SPIRUHARET.RO E-MAIL:** 3 **OFFICE_JEDEP@SPIRUHARET.RO**

Logical Querying of Relational Databases

Luminita Pistol 1, Radu Bucea-Manea-Tonis2 1 Spiru Haret University 2 Hyperion University Abstract. This paper aims to demonstrate the usefulness of formal logic and lambda calculus in database programming. After a short introduction in propositional and first order logic, we implement dynamically a small database and translate some SQL queries in filtered java 8 streams, enhanced with Tuples facilities from jOOλ library.

Keywords: logic query, propositional logic, predicate, relational database JEL Codes: M15 Introduction A database is a set of basic axioms corresponding to base relations and tuples plus deductive axioms or inference rules. Tuples are for the relationships what are nouns for sentences, each denote a true particular sentence [Date, 2005].

A logical query is the action of evaluating a Boolean expression concerning tuples and relations. Boolean operators in propositional logic are:

Table1: Boolean Operators

Operator name and meaning Example

negation (non) -φ

conjunction (and) (φ & ψ)

disjunction (or) (φ | ψ)

implication (if ..., then ...) (φ -> ψ)

equivalence (if and only if.) (φ <-> ψ)

A basic axiom is equivalent to a tuple of a database or a predicate. The predicate value is a function of truth that has a set of parameters. It should not be assigned a value to a database in order to determine the database predicate to take the truth value FALSE [Date, 2005]

Table 2: Basic axiom table

Parent Child

Caninae Canis

Canis Canis lupus

Corresponding to the exampe above, we can construct an open formula with two occurrences of the variable x:

Grandparent(x) <- Parent (x) & (Child(x) <-> Parent(y))

By placing an existential quantifier ∃ before x ("for some x") and an universal quantifier ∀ before y ("for all y"), we can bind these variables, as may be seen bellow [Bird, 2009]:

∃x. ∀y. Grandparent(x) <- Parent (x) & (Child(x) <-> Parent(y)) 1.1. Advantages of logical querying:

Uniform representation of operations and dependency constraints;

Improved semantics of the original data model;

Improve SQL facilities making possible to negate a where clause if we keep in mind the formal logic rules [StackOverflow, 2016,]:

A & B & (D | E) ↔ ⌐ (A & B & (D | E)) ↔ ⌐A | ⌐B | (⌐D & ⌐E)

Case study

Suppose that in our database the following schema has been defined [Moshe, 2006]:

Student (name, dorm, major, GPA), Professor (name, dept, salary, year hired) Chair (dept, name) We create a dynamic structure for this as the following:

```
Studenti = new ArrayList<Student>();

Profesori = new ArrayList<Professor>();

Decani = new ArrayList<Chair>();

Decani.add(new Chair("Iosipescu","Math"));

Decani.add(new Chair("Radulescu","CS"));
```

```
Profesori.add(new Professor("Georgescu","CS",5000,1999));

Profesori.add(new Professor("Iosipescu","Math",3000,2004));

Profesori.add(new Professor("Radulescu","CS",7000,2000));

Profesori.add(new Professor("Marinescu","Math",6000,1998));

Studenti.add(new Student("Ionescu", "A5", "CS", 9.5));

Studenti.add(new Student("Marinescu", "A3", "Math", 9.0));

Studenti.add(new Student("Popescu", "A4", "CS", 8.5));

Studenti.add(new Student("Vasilescu", "A5", "Math", 7.5));
```

1.2. List the name and dorm of Math students with a GPA of at least 8.0:

```
List<Student> result = db.Studenti.stream().filter(s -> s.major.equals("Math") &
s.GPA>=8.0).collect(Collectors.toList());
```

1.3. List the names of faculty members with a salary of at most 5000 who were hired after 1990:

```
List<Professor> result1 = db.Profesori.stream().filter(p -> p.salary<=5000 &
p.year>=1990).collect(Collectors.toList());
```

1.4. List the names of faculty whose salary is higher than their chair’s salary:

```
db.Profesori.stream() .sorted((p1, p2) -> Long.compare(p1.salary, p2.salary)) .flatMap(v1 ->   4
DB.DECANI.STREAM() .FILTER(V2 -> Objects.equals(v1.dept, v2.dept) &
db.Profesori.stream() .anyMatch(t ->  4  V1.SALARY>T.SALARY &
T.NAME.EQUALS(V2.NAME))) .MAP(V2 -> TUPLE(V1.NAME, V2.NAME)))
.FOREACH(SYSTEM.OUT::PRINTLN);
```

1.5. List the names of faculty members whose salary is highest in their department:

```
db.Profesori.stream().filter(p->db.Profesori.stream().anyMatch(t->t.salary<p.salary &
t.dept.equals(p.dept))).forEach(p->{System.out.println("name=" + p.name);});
```

We have employed the jOOλ library [GitHub, 2016], making the following mappings [Fusco, 2015]:

INNER JOIN - flatMap() with filter() WHERE - filter() GROUP BY - collect() HAVING - filter() SELECT - map()

The results are the following:

name=Marinescu dorm=A3//1 name=Georgescu name=Iosipescu//2 (Marinescu, Iosipescu)//3 name=Radulescu name=Marinescu//4.

Conclusions There are advantages evaluating expressions and functional programming has already given us the support for a declarative way of parsing collections of objects. Since relational databases cease way to noSQL ones, we have to discover a good substitute for SQL language. Beginning with Java 8 lambda expressions, streams and method references, we have to search no more.

References C.J. Date, Baze de date, Editura Plus, 2005, ISBN:973-861-90-1-7 S Bird, E. Klein, E. Loper, Natural Language Processing With Python, Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009, ISBN: 978-0-596-51649-9 http://victoria.lviv.ua/html/fl5/NaturalLanguageProcessingWithPython.pdf Moshe Y. Vardi, I. Barland, B. McMahan, Logic and Database Queries, August 31, 2006, https://www.cs.rice.edu/~tlogic/Database/all-lectures.pdf GitHub, 2016, https://github.com/jOOQ/jOOL M. Fusco, Common SQL Clauses and Their Equivalents in Java 8 Streams, 1 Mar 2015, https://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/ StackOverflow, SQL WHERE condition, not equal to?, 2016 http://stackoverflow.com/questions/6156979/sql-where-condition-not-equal-to

62

61

Citations (4/4)

1 Owner: Prof. univ. dr. Manuela Epure; Submitted: Wed, Jul 27 2016, 3:38 PM; Filename: Future economy .docx

2 Owner: Prof. univ. dr. Manuela Epure; Submitted: Mon, Dec 28 2015, 6:33 PM; Filename: JEDEP15_7Majid_p65-75.docx

3 Another user's paper

4 https://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/

Matched Text

Suspected Entry: **94% match**

Uploaded - 6. Radu.docx **(ONLINE) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 JOURNAL OF ECONOMIC DEVELOPMENT, ENVIRONMENT AND PEOPLE VOLUME 5, ISSUE 2, 2016**	**Source -** Owner: Prof. univ. dr. Manuela Epure; Submitted: Wed, Jul 27 2016, 3:38 PM; Filename: Future economy .docx (online) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 Journal of Economic Development, Environment and People Volume 5, Issue 3, 2016

Suspected Entry: **94% match**

Uploaded - 6. Radu.docx **(ONLINE) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 JOURNAL OF ECONOMIC DEVELOPMENT, ENVIRONMENT AND PEOPLE VOLUME 5, ISSUE 4, 2016**	**Source -** Owner: Prof. univ. dr. Manuela Epure; Submitted: Wed, Jul 27 2016, 3:38 PM; Filename: Future economy .docx (online) = ISSN 2285 – 3642 ISSN-L = 2285 – 3642 Journal of Economic Development, Environment and People Volume 5, Issue 3, 2016

Suspected Entry: **100% match**

Uploaded - 6. Radu.docx **HTTP://JEDEP.SPIRUHARET.RO E-MAIL**	**Source -** Owner: Prof. univ. dr. Manuela Epure; Submitted: Mon, Dec 28 2015, 6:33 PM; Filename: JEDEP15_7Majid_p65-75.docx http://jedep.spiruharet.ro e-mail

Suspected Entry: **100% match**

Uploaded - 6. Radu.docx **HTTP://JEDEP.SPIRUHARET.RO E-MAIL**	**Source -** Owner: Prof. univ. dr. Manuela Epure; Submitted: Mon, Dec 28 2015, 6:33 PM; Filename: JEDEP15_7Majid_p65-75.docx http://jedep.spiruharet.ro e-mail

Suspected Entry: **100% match**

Uploaded - 6. Radu.docx

OFFICE_JEDEP@SPIRUHARET.RO

Source - Another user's paper

office_jedep@spiruharet.ro

Suspected Entry: **100% match**

Uploaded - 6. Radu.docx

OFFICE_JEDEP@SPIRUHARET.RO

Source - Another user's paper

office_jedep@spiruharet.ro

Suspected Entry: **70% match**

Uploaded - 6. Radu.docx

DB.DECANI.STREAM() .FILTER(V2 ->

Source - https://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/

s2.stream() .filter(v2 ->

Suspected Entry: **65% match**

Uploaded - 6. Radu.docx

V1.SALARY>T.SALARY & T.NAME.EQUALS(V2.NAME))) .MAP(V2 ->

Source - https://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/

Objects.equals(v1, v2)) .map(v2 ->

Suspected Entry: **88% match**

Uploaded - 6. Radu.docx

TUPLE(V1.NAME, V2.NAME))) .FOREACH(SYSTEM.OUT::PRINTLN)

Source - https://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/

tuple(v1, v2))) .forEach(System.out::println)