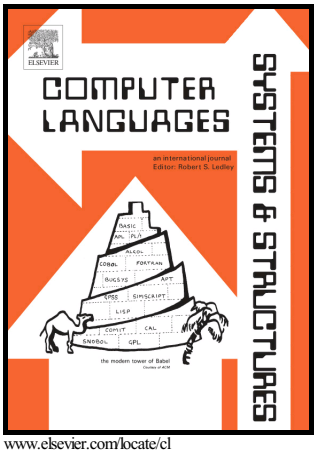

www.elsevier.com/locate/cl

Optimisation of Language-Integrated Queries by Query Unnesting

Tomasz Marek Kowalski, Radosław Adamus

PII: S1477-8424(16)30024-0
DOI: http://dx.doi.org/10.1016/j.cl.2016.09.002
Reference: COMLAN236

To appear in: *Computer Language*

Received date: 8 February 2016
Accepted date: 11 September 2016

Cite this article as: Tomasz Marek Kowalski and Radosław Adamus Optimisation of Language-Integrated Queries by Query Unnesting, *Compute Language,* http://dx.doi.org/10.1016/j.cl.2016.09.002

Optimisation of Language-Integrated Queries by Query Unnesting

Authors:
Tomasz Marek Kowalski, email: t.kowalski@iis.p.lodz.pl (corresponding author)
Radosław Adamus, email: r.adamus@iis.p.lodz.pl
Affiliation:
Institute of Applied Computer Science
Lodz University of Technology
ul. Stefanowskiego 18/22
90-924 Lodz, Poland

Abstract

Native functional-style querying extensions for programming languages (e.g., LINQ or Java 8 streams) are widely considered as declarative. However, their very limited degree of optimisation when dealing with local collection processing contradicts this statement. We show that developers constructing complex LINQ queries or combining queries expose themselves to the risk of severe performance deterioration. For an inexperienced programmer, a way of getting an appropriate query form can be too complicated. Also, a manual query transformation is justified by the need of improving performance, but achieved at the expense of reflecting an actual business goal. As a result, benefits from a declarative form and an increased level of abstraction are lost.

In this paper, we claim that moving of selected methods for automated optimisation elaborated for declarative query languages to the level of imperative programming languages is possible and desired. Our approach is based on the assumption that programmer is able distinguish whether a language-integrated query is intentionally used to introduce some side-effects or its sole purpose is to only query the data. We propose two optimisation procedures through query unnesting designed to avoid unnecessary multiple calculations in collection-processing constructs based on higher-order functions. We have implemented and verified this idea as a simple proof-of-concept LINQ optimiser library.

1 Introduction

Since the release of LINQ (Language-Integrated Query) for the Microsoft .NET platform in 2007, there has been a significant progress in the topic of extending programming languages with native querying capabilities [1]. Programming languages are mostly imperative; their semantics relies on the program stack concept. They operate on volatile data and the meaning of collections is rather secondary. On the other hand, query languages are usually declarative and their semantics often bases on some forms of algebras or logics; these languages operate mostly on collections of persistent data. Declarativity of a query language reveals itself mostly when considering operators for collections. In the case of an imperative language, operating on a collection takes a form of an explicit loop iterating over collection elements in a specified order, while in query languages one declares a desired result (e.g., a sub-collection containing elements of a base collection matching a given selection predicate) and an algorithm of filtration itself is not an element of an expression representing the query. Based on characteristics of data structures, a database state and existence of additional auxiliary structures (e.g., indices), an execution environment can choose the most promising algorithm (a plan) for evaluation of the query. Declarativity allows one to postpone selection of an algorithm even to the moment of an actual query execution. In this paper we discuss to what extent solutions for processing of collections within programming languages are actually declarative. To do so, we made an extensive research on query optimisation. In databases it is a