Computer Languages, Systems & Structures 38 (2012) 123–157

Contents lists available at SciVerse ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Abstract interpretation of database query languages

Raju Halder, Agostino Cortesi *

Università Ca' Foscari Venezia, Italy

ARTICLE INFO

Article history:
Received 3 March 2011
Received in revised form
5 July 2011
Accepted 28 October 2011
Available online 12 November 2011

Keywords:
Databases
SQL
Program analysis
Abstract Interpretation

ABSTRACT

In this paper, we extend the Abstract Interpretation framework to the field of query languages for relational databases as a way to support sound approximation techniques. This way, the semantics of query languages can be tuned according to suitable abstractions of the concrete domain of data. The abstraction of relational database system has many interesting applications, in particular, for security purposes, such as fine grained access control, watermarking, etc.

1. Introduction

In the context of web-based services interacting with DBMS, there is a need of "sound approximation" of database query languages, in order to minimize the weight of database replicas on the web or in order to hide specific data values while giving them public access with larger granularity. There are many application areas where processing of database information at different level of abstraction plays important roles, like the applications where users are interested only in the query answers based on some properties of the database information rather than their exact values.

Given an exploratory nature of the applications, like decision support system, experiment management system, etc., many of the queries end up producing no result of particular interest to the user. Wasted time can be saved if users are able to quickly see an approximate answer to their query, and only proceed with the complete execution if the approximate answer indicates something interesting. The sound approximation of the database and its query languages may also serve as a formal foundation of answering queries approximately as a way to reduce query response times, when the precise answer is not necessary or early feedback is helpful.

Cooperative query answering [1,2] supports query relaxation and provides intelligent, approximate answers as well as exact answers. It provides neighborhood or generalized information relevant to the original query and within a certain semantic distance of the exact answer. Searching approximate values for a specialized value is equivalent to find an abstract value of the specialized value, since the specialized values of the same abstract value constitute approximate values of one another. Sound approximation of the database system provides a formal framework to the field of cooperative query answering, ensuring three key issues: soundness, relevancy and optimality which are crucial in this context.

When a database is being populated with tuples, all tuples must satisfy some properties which are represented in terms of integrity constraints. For instance, the ages of the employees must be positive and must lie between 18 and 62. Any

* Corresponding author.
E-mail addresses: halder@unive.it (R. Halder), cortesi@unive.it, cortesi@dsi.unive.it (A. Cortesi).

1477-8424/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2011.10.004

transaction over the database must satisfy all these integrity constraints as well. The dynamic checking for any transaction to ensure whether it violates the integrity constraints of the database can increase the run-time overhead significantly, while managing the integrity constraint verification statically may have a significant impact in terms of efficiency.

The traditional Fine Grained Access Control (FGAC) [3] provides only two extreme views to the database information: either public or private. There are many application areas where some partial or relaxed view of the confidential information is desirable. For instance, consider a database in an online transaction system containing the credit card numbers for its customers. According to the disclosure policy, the employees of the customer-care section are able to see the last four digits of the credit card numbers, whereas all the other digits are completely hidden. The traditional FGAC policy is unable to implement this type of security framework without changing the database structure.

An interesting solution to all these problems can be provided by extending to the database field a well known static analysis technique, called Abstract Interpretation [4–7]. Abstract Interpretation, in fact, has been proved, in other contexts, as the best way to provide a semantics-based approach to approximation. Its main idea is to relate concrete and abstract semantics where the later are focussing only on some properties of interest. It was originally developed by Cousot and Cousot as a unifying framework for designing and then validating static program analysis, and recently it becomes a general methodology for describing and formalizing approximate computation in many different areas of computer science, like model checking, verification of distributed memory systems, process calculi, security, type inference, constraint solving, etc. [7].

Relational databases enjoy mathematical formulations that yield to a semantic description using formal language like relational algebra or relational calculus. To handle the aggregate functions or NULL values, some extensions of existing relational algebra and relational calculus have been introduced [8–11]. However, this semantic description covers only a subset of SQL [8,11,12]. In particular, problems arise when dealing with UPDATE, INSERT or DELETE statements since operators originally proposed in relational algebra do not fully support them. This motivates our theoretical work aiming at defining a complete denotational semantics of SQL embedded applications, both at the concrete and at the abstract level, as a basis to develop an Abstract Interpretation of application programs embedded with SQL. In this setting, we represent all the syntactic elements in SQL statements (for example, GROUP BY, ORDER BY, DISTINCT clauses, etc.) as functions and the semantics is described as a partial functions on states which specify how expressions are evaluated and commands are executed. The functional representation of syntactic elements increases the power of expressibility of the semantics and facilitates us to provide a complete functional control on the corresponding domains of data. As far as we know, the impact of abstract interpretation for sound approximation of database query languages has not yet been investigated. This is the aim of this paper.

The underlying concepts is that the applications embedded with SQL code basically interact with two worlds or environments: *user world* and *database world*. Corresponding to these two worlds or environments we define two sets of variables: $\mathbb{V}_d$ and $\mathbb{V}_a$. The set $\mathbb{V}_d$ is the set of database variables (*i.e.* the set of database attributes) and $\mathbb{V}_a$ is a distinct set of variables called application variables defined in the application. Variables from $\mathbb{V}_d$ are involved only in the SQL commands, whereas variables in $\mathbb{V}_a$ may occur in all type of instructions of the application. We denote any SQL command by a tuple $C_{sql} \triangleq \langle A_{sql}, \phi \rangle$. We call the first component A_{sql} the *action part* and the second component ϕ the *pre-condition part* of C_{sql}. In an abstract sense, any SQL command C_{sql} first identifies an active data set from the database using the pre-condition ϕ and then performs the appropriate operations on that data set using the SQL action A_{sql}. The pre-condition ϕ appears in C_{sql} as a well-formed formula in first-order logic. The semantics defined this way can be lifted from the concrete domain of values to abstract representation of them by providing suitable abstract operators corresponding to the concrete ones.

The structure of this paper[1] is as follows: Section 2 recalls some preliminary concepts. Section 3 defines the abstract syntax of the SQL embedded application. In Section 4, we define environments and states associated with the application. Section 5 describes the semantics of the arithmetic and boolean expressions, whereas Sections 6 and 7 describe the formal semantics of atomic and composite statements respectively. The correspondence of the proposed denotational semantic approach with the relational algebra is discussed in Section 8. In Section 9, we lift the syntax and semantics of the query languages from concrete domain to an abstract domain of interest by discussing the soundness and completeness of the abstraction in details. In Section 10, we discuss the formal semantics of SQL statements with correlated and non-correlated subquery. The interesting applications of suitable abstraction of the relational databases are discussed in Section 11. Section 12 discusses the related work in the literature. Finally, in Section 13, we draw our conclusions.

2. Preliminaries

In this section, we recall some basic mathematical notation used in the literature, some ideas about Semantic Interpretation of First-Order Logic [14] and the Abstract Interpretation theory [4–7].

2.1. Basic mathematical notation

If S and T are sets, then $\wp(S)$ denotes the powerset of S, $|S|$ the cardinality of S, $S \backslash T$ the set-difference between S and T, $S \times T$ the Cartesian product. A poset P with ordering relation $\sqsubseteq$ is denoted as $\langle P, \sqsubseteq \rangle$, while $\langle C, \sqsubseteq, \sqcup, \sqcap, \top, \bot \rangle$ denotes the complete lattice C with ordering $\sqsubseteq$, lub $\sqcup$, glb $\sqcap$, greatest element $\top$, and least element $\bot$.

[1] The paper is a revised and extended version of [13].