The Journal of Systems and Software 84 (2011) 2253–2264

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Checking enforcement of integrity constraints in database applications based on code patterns

Hongyu Zhang[a,b,*], Hee Beng Kuan Tan[c], Lu Zhang[d], Xi Lin[a,b], Xiaoyin Wang[d], Chun Zhang[d], Hong Mei[d]

[a] Key Laboratory for Information System Security (Tsinghua University), Ministry of Education, Beijing 100084, China
[b] Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
[c] Nanyang Technological University, Singapore 639798, Singapore
[d] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China

ARTICLE INFO

Article history:
Received 5 July 2010
Received in revised form 4 June 2011
Accepted 16 June 2011
Available online 5 July 2011

Keywords:
Integrity constraint enforcement
Code patterns
PHP
Static analysis
Code quality

ABSTRACT

Integrity constraints (including key, referential and domain constraints) are unique features of database applications. Integrity constraints are crucial for ensuring accuracy and consistency of data in a database. It is important to perform integrity constraint enforcement (ICE) at the application level to reduce the risk of database corruption. We have conducted an empirical analysis of open-source PHP database applications and found that ICE does not receive enough attention in real-world programming practice. We propose an approach for automatic detection of ICE violations at the application level based on identification of code patterns. We define four patterns that characterize the structures of code implementing integrity constraint enforcement. Violations of these patterns indicate the missing of integrity constraint enforcement. Our work contributes to quality improvement of database applications. Our work also demonstrates that it is feasible to effectively identify bugs or problematic code by mining code patterns in a specific domain/application area.

1. Introduction

Database applications constitute one of the largest software application domains. A database application is a data-intensive software system that supports business processes or functions through maintaining a database using a database management system (DBMS). In database application development, integrity constraints are used to ensure integrity of a database (Ramakrishnan and Gehrke, 2000; Silberschatz et al., 2005). These constraints include key constraints, referential constraints and domain constraints. A key constraint specifies a key that uniquely identifies a record in a database table. A referential constraint (also called a foreign key constraint) ensures that two database tables maintain a primary-key-to-foreign-key relationship. A domain constraint defines the possible range of values of an attribute. Breaking these constraints may lead to the loss of data dependencies, runtime exceptions, security flaws or even corruption of the database. Therefore, it is vital to enforce integrity constraints for database applications.

There are two levels of integrity constraint enforcement (ICE): the DBMS level and the application level. Traditionally, most DBMS systems can only enforce a limited range of constraints. Referential constraint enforcement is often not supported due to performance concerns. In recent years, some DBMSs (such as MySQL+InnoDB) can be configured to enable full support of ICE. However, even though DBMSs support ICE, many developers fail to use it. Blaha (2001) studied about 50 database applications and found that 90% of them fail to use referential constraints.

The other level of ICE is at the application level, where ICE is implemented by program code before performing any database operation that may lead to a violation. At the application level, ICE is maintained by programmers and independent of the DBMS choice. Therefore, even if the DBMS does not support ICE, the data integrity can still be kept. The risk of database corruption is thus reduced. We believe that a good database application should be "defensive" to prevent violations from happening. The lack of the application level ICE in a database application often indicates bad programming practice or even bugs. In this paper, we propose a pattern based approach for checking enforcement of integrity constraints at the application level.

In recent years, many approaches have been proposed to detect bugs or problematic code based on code patterns (e.g., (FindBugs, 2011; Hovemeyer and Pugh, 2004; PMD, 2011)). We adopt a similar approach, but focus on only the domain of database applications. We find that in database applications, code that implements ICE generally exhibits certain empirical patterns. We have identified four such patterns, namely the key constraint enforcement pattern, the referential constraint enforcement pattern for insertion

* Corresponding author. Tel.: +86 10 62773275.
E-mail address: hongyu@tsinghua.edu.cn (H. Zhang).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.06.044

and update, the referential constraint enforcement pattern for deletion, and the input validation pattern. By detecting violations of these code patterns through static analysis, we can check whether constraint enforcement has been performed in a database application. We can then achieve a better understanding of the quality of the database application and reduce the risk of database corruption. Even if ICE is implemented at the DBMS level, knowing the absences of ICE at the application level could help software maintainers identify potential ICE problems during software evolution, where the database schema or even the DBMS may change.

We have conducted experiments on nine real-world open source PHP database applications. We find that, in reality, ICE does not receive enough attention in programming practice. Implementations of ICE are often neglected at both the DBMS level and the application level. We applied our code pattern based ICE detection approach to these applications and discovered many violations of integrity constraints with high accuracy.

We believe that our approach could help improve the quality of database applications. The organization of the rest of this paper is as follows. In Section 2, we introduce integrity constraint enforcement together with an illustrative example. In Section 3, we describe four ICE-related code patterns, which can be used to detect violations of ICE at the application level. We present our experiments on nine PHP database applications in Section 4. Section 5 discusses the related work and Section 6 concludes this paper.

2. Background

2.1. Integrity constraint enforcement (ICE)

A unique feature of database applications is the enforcement of integrity constraints. Integrity constraints are conditions specified on a database schema (Ramakrishnan and Gehrke, 2000; Silberschatz et al., 2005). They restrict the data that can be stored in an instance of the database. Key constraints, referential constraints and domain constraints are three major forms of integrity constraints.

A key constraint specifies a unique key (an attribute or a set of attributes) that uniquely identifies each record in a database table. A Primary Key (PK) is a unique key whose value cannot be null. According to key constraints, no two distinct records in a table can have the same value for the PK attribute.

A referential constraint (also called foreign key constraint) identifies an attribute (or a set of attributes) in one (referencing) table that refers to an attribute (or a set of attributes) in another (referenced) table. It establishes a relationship between the two tables. For an attribute defined as a Foreign Key (FK), it should refer to a PK in the referenced table. Thus, a row in the referencing table cannot contain key values that do not exist in the referenced table.

In a database, a domain constraint is a basic form of integrity constraint that defines the possible range of values of an attribute. It includes rules for the allowed data type and length, the NULL value acceptance, etc. All of these rules are used to ensure the validity and consistency of data.

Integrity constraints should be enforced when the database is to be modified. For key constraints, whenever a record containing a PK value is to be inserted or updated, a check for key duplication is required. If the table already contains a record having the same PK value, the key constraint is violated and the operation should be rejected. To enforce referential constraints, whenever a record containing a FK value is to be inserted or updated, we need to check whether the referenced table contains the associated PK value. Whenever a record containing a PK value is to be deleted, a typical action is to delete any associated record containing a FK with the same PK value.

Domain constraints contain domain-specific rules for preserving data validity. It is not easy to use one specific approach to check all forms of the rules in a domain. However, since ultimately the data to be stored in the database comes from the external environment such as user input, we can implement input validation to help ensure domain constraints. Input validation enforces that the input submitted from the external environment must satisfy the required domain constraints before it is accepted to the database. Input validations are especially important for Web-based database applications, where security vulnerabilities such as SQL injection are real concerns.

ICE can be performed by a DBMS such as SQL Server 2005 or MySQL+InnoDB. For example, the deletion of a PK record may cause the deletion of the corresponding FK records. Changes to the data that violates the constraints can be automatically rejected. However, if the constraints are not explicitly specified in the database schema or the DBMS is not configured to enable automatic checking of constraints, the integrity of the database would be in danger. Empirical studies (Blaha, 2001, 2004) have shown that DBMS-level ICE, especially the referential constraints, is seldom enforced in real-world practice. Furthermore, many DBMSs (such as MySQL's default setting MySQL+MyISAM) do not fully support automatic checking of ICE.

We believe that a good database application should be "defensive" to prevent violations from happening. If the integrity constraints are missing from the database schema or if the DBMS is not configured properly to enable automatic enforcement of integrity constraints, we need to rely on application-level ICE. During long-term software evolution, the database schema, DBMS configuration or even the type of DBMS could be changed by different maintainers. Application-level ICE checking could help identify possible ICE-related problems and warn the maintainers when such changes occur. Furthermore, implementation of ICE at the application level can help detect errors in user input earlier, and therefore help provide better feedback to users and achieve better software usability.

To check the absence of ICE at the DBMS level is relatively easy. We just need to check whether the constraints are explicitly specified in the database schema and whether the DBMS is configured to enable automatic checking of integrity constraints. However, manually checking ICE at the application level is usually difficult, as the code base to be examined could be large. In Section 3, we propose a code pattern based approach for checking enforcement of integrity constraints at the application level.

2.2. An illustrative example of ICE at the application level

We now give an example to illustrate the concepts of ICE at the application level. Fig. 1 shows the conceptual model of a small customer-order management system, which involves three entities: `Customer`, `Order`, and `Item`. In relational database design, the attributes of the `Customer` table and their domain constraints are as follows: *custID* (not null, int type), *name* (not null, varchar type), *address* (not null, varchar type), and *phone* (not null, int type with 8 digits). Attribute *custID* is defined as the PK of table `Customer`, *orderID* as the PK of table `Order`, and *itemID* as the PK of table `Item`. As each order is associated with a customer, attribute *custID* in table `Order` is defined as a FK. Similarly, *orderID* in table `Item` is also a FK.

To enforce database integrity, a program needs to check whether the constraints are broken before performing any database operation. As an example, the PHP code in Fig. 2 implements input validation and key constraint enforcement. The code from line 6 to line 10 uses *if* structures to implement input validation, which checks the values of user input before they are accepted to the database. In line 18, a record with key value $*custID* is inserted