Enforcing Integrity for Redundant and Contradiction Constraints in OODBs

Belal Zaqaibah
Faculty of Information Technology, Multimedia University, 63100 Cyberjaya, Malaysia belal@mmu.edu.my

Hamidah Ibrahim, Ali Mamat, and Md. Nasir Sulaiman
Department of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Selangor, Malaysia hamidah@fsktm.upm.edu.my

Abstract - Maintenance of redundant and contradicting constraints in object-oriented databases is as vital as maintaining an application on the run. An effective and much-organized maintenance of such databases relies on proper selection and usage of methods or techniques to maintain such databases. In this paper, a technique for enforcing and maintaining integrity constraints, more productively and efficiently, has been proposed. The technique supports user-defined constraints and keeps the derivation path along with the attributes’ relationships that are from composition and inheritance hierarchies. Technical methods are defined to implement the required functions that identify the attributes’ relationships, along with checking and maintaining the integrity constraints. By simply utilizing the available functionalities and flexibilities in object-oriented database architecture, our technique will be able to enforce and maintain redundant, duplicated, and contradictory constraints in object-oriented data model environment.

Keywords: Integrity constraints, structural integrity, logical integrity, OODB.

1 Introduction

At present, a fundamental problem in database design lies in maintaining and checking integrity constraints in Object-Oriented Databases (OODBs). Current OODB management systems lack the ability for an ad-hoc declarative specification of maintaining certain integrity constraints that are defined in a class composition or inheritance hierarchies. This paper will address an efficient and effective methodology for checking, enforcing, and maintaining such integrity constraints as it will support and maintain user-defined constraints using the Assertion Model of Integrity Constraints (AMIC). The AMIC is designed for both centralized integrity maintenance and application-oriented integrity maintenance. The AMIC’s maintenance techniques maintain redundant, contradiction, and duplicated constraints in both intra-object and inter-object constraints.

In general, there are two steps to be done while updating a requested object or constraint. First and foremost, constraints that would be violated must be identified. Secondly, if the constraint enforces an inconsistent state, the current transaction will be aborted or the constraint must be modified as such that it remains true in the new database state. A transaction is a collection of operations that performs a function in a database application [1]. Thus, transactions should not violate the database consistency as well.

Determining whether a transaction violates the database constraints or not is required to call the involved constraints from Object Meta Data (OMD) table [2], that’s when the AMIC’s optimization technique comes in handy by eliminating the unnecessary comparisons. Moreover, the optimization technique reduces the evaluation time as it calls only the involved constraints.

2 Background

A database state is said to be consistent if the database satisfies a set of statements, called semantic integrity constraints (or simply constraints) [3]. Handling semantic integrity constraints is an essential premise to manage semantically rich data [4, 5]. Also, checking the integrity constraints in OODBs is a fundamental problem in database design [4]. The integrity constraints enforcement is the responsibility of the database applications [6, 4] but the automated verification of constraints and their enforcement provided by current OODBMS is limited as mentioned in [7].

In this paper, we handle two parts of a database system namely: logical integrity (data) and structural integrity (constraints knowledge base). Typically, we consider the Object-Oriented Data Model (OODM) is the underlying data model. The maintenance methodology recommended here depends on fixed values domain and attributes domain.

The fixed values domain is a finite (e.g., set of integers between 1 and 12) or infinite set of values (set of characters), and the domain of an attribute is a finite set that includes data in a particular object. Attributes are members in a class and they represent data components that make up the content of a class. The term class refers