Chapter XL
The Challenges of Checking Integrity Constraints in Centralized, Distributed, and Parallel Databases

Hamidah Ibrahim
Universiti Putra Malaysia, Malaysia

INTRODUCTION

A vital problem that should be tackled in today's database system is guaranteeing *database consistency*. Many techniques and tools have been devised to fulfill this requirement in many interrelated research areas such as concurrency control, security control, reliability control, and integrity control (Eswaran & Chamberlin, 1975; Grefen, 1993). *Concurrency control* deals with the prevention of inconsistencies caused by concurrent access by multiple users or applications to a database. *Security control* deals with preventing users from accessing and modifying data in a database in unauthorized ways. *Reliability control* deals with the prevention of errors due to the malfunctioning of system hardware or software. *Integrity control* deals with the prevention of semantic errors made by users due to their carelessness or lack of knowledge. This chapter is concerned only with integrity control.

A database state is said to be consistent if the database satisfies a set of statements called *semantic integrity constraints* (or simply *constraints*). Integrity constraints stipulate those configurations of the data that are considered semantically correct. Any update operation (insert, delete, or modify) or transaction (sequence of updates) that

occurs must not result in a state that violates these constraints. Thus, a fundamental issue concerning integrity constraints is *constraint checking*; that is, the process of ensuring that the integrity constraints are satisfied by the database after it has been updated. Checking the consistency of a database state will generally involve the execution of *integrity tests* (query that returns the value true or false) on the database, which verify whether or not the database is satisfying its constraints.

Integrity checking is primarily implemented in one of the following ways, depending on who or what is responsible for ensuring the consistency of a database. The responsibility for constraint checking is either allocated to the users of the database or to a database management system component called *semantic integrity subsystem* (SIS). In the former approach, the users are responsible for specifying a set of integrity constraints that the update or transaction may violate, and they must include checks or integrity tests into update transactions to ensure that none of these constraints will be violated. In the alternative approach, a complete set of integrity constraints is identified once by the users and then applied to all updates or transactions of the database. Obviously, the first approach is less friendly and more error-prone because the efficiency and correctness of manually written integrity tests rely on the users' skills; and changes to constraint definitions require modification of all transactions, which include integrity control with respect to these definitions. However, integrity constraint checking is not fully supported by current database technology (Martinenghi, 2005).

The growing complexity of modern database applications plus the need to support multiple users has further increased the need for a powerful integrity subsystem to be incorporated into these systems. Therefore, a complete integrity subsystem is considered an important part of any modern DBMS. The crucial problem is the difficulty of devising an efficient algorithm for enforcing database integrity against updates. A naive approach is to perform the update and then check whether the integrity constraints are satisfied in the new database state. This method, termed *brute force checking*, is very expensive and impractical, and can lead to prohibitive processing costs because the evaluation of integrity constraints requires accessing large amounts of data that are not involved in the database update transition (Simon & Valduriez, 1987). Hence, improvements to this approach have been reported in many research papers.

Many factors need to be well thought out before an enforcement mechanism can be devised. These factors include the following:

- The type of environment considered (i.e., whether it is centralized, distributed, parallel, or even mobile. Different environments require different schemes of enforcing the integrity of the system due to different architectures, components, function of the components, and so forth.
- The type of integrity constraints considered. There are many classifications of integrity constraints ranging from simple to complex. The classifications of integrity constraints are based on some of their characteristics; for instance, scope space, data model, time scope, definiteness of the constraints (hard condition or fuzzy condition), computational complexity, or even properties of the constraints such as soundness and completeness. Most of the approaches proposed for checking constraints are limited to certain types of integrity constraints that are generally used and referred to in theory and practice, such as implicit, inherent, and explicit constraints of a data model, and state and transition constraints of a database application, which include value set (domain) constraints (e.g., the age of an employee must not be less than 20), key (uniqueness) constraints (e.g., every employee has a unique employee number), structural constraints (e.g., null value is not