Chapter 9

Integrity Constraints Checking in a Distributed Database

Hamidah Ibrahim
Universiti Putra Malaysia, Malaysia

ABSTRACT

Preserving the accuracy and the integrity of information in a database is extremely important for the organization that is maintaining that database. Such an organization is likely to rely heavily upon that accuracy. Applications that consult and use the database expect a warranty that the database is supplying the correct information. Critical business decisions may be made assuming that information extracted from the database is correct. Thus, incorrect data can lead to incorrect business decisions which can have serious implications for the people and organizations using it (Codd, 1990).

INTRODUCTION

Preserving the accuracy and the integrity of information in a database is extremely important for the organization that is maintaining that database. Such an organization is likely to rely heavily upon that accuracy. Applications that consult and use the database expect a warranty that the database is supplying the correct information. Critical business decisions may be made assuming that information extracted from the database is correct. Thus, incorrect data can lead to incorrect business decisions which can have serious implications for the people and organizations using it (Codd, 1990).

An important problem for a database system is to guarantee 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 prevention of inconsistencies caused by concurrent access by multiple users or applications to a database. Security control deals with prevent-
Integrity Constraints Checking in a Distributed Database

...ing users from accessing and modifying data in a database in unauthorized ways. Reliability control deals with the prevention errors due to the malfunctioning of system hardware or software. Integrity control deals with the prevention of semantic errors made by the users due to their carelessness or lack of knowledge.

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 specify 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* on the database which verify whether the database is satisfying its constraints or not.

In a database system, a semantic integrity subsystem (SIS) is responsible for managing and enforcing integrity constraints to ensure that these rules are not violated by the database and the database is in a consistent state. An early proposal by Eswaran and Chamberlin (1975) described the functionality requirements for an integrity subsystem. The main tasks of this subsystem are to determine which constraints are to be checked after each database change and to trigger the appropriate actions when a constraint violation is detected. The crucial problem encountered in designing a complete integrity subsystem is the difficulty of devising an efficient algorithm for enforcing database integrity when updates occur. Many proposals for designing an integrity subsystem can be found in the database literature. In Grefen (1993) and McCarroll (1995) three ways to couple the integrity subsystem to the DBMS are described.

The first approach is known as the *decoupled subsystem* approach. This adds the subsystem as an additional layer on top of an existing DBMS. It was employed by the AIM project (Cremers & Domann, 1983) and the KBDTA system (Wang, 1992). In this approach, the responsibility for ensuring the consistency of the database when a transaction occurs is part of the transaction design process. The transaction designers are responsible for ensuring that the transactions are *safe*, i.e. when executed, the transactions are guaranteed to bring the database from one consistent state to another consistent state. Consequently, as transactions can get complex, a *transaction design tool* is usually incorporated into the subsystem to assist the transaction designers to construct safe transactions. Hence, in this approach, little or no support within the DBMS is needed for automatically enforcing database integrity constraints.

The second approach is known as the *loosely coupled subsystem*. It adds the subsystem as an extension to the DBMS. This is employed by the SABRE (Simon & Valduriez, 1987) and the PRISMA (Grefen, 1990) projects. In this approach, transactions have integrity tests embedded in them to perform the necessary integrity checking. The modified transactions can then be executed by the standard transaction facilities. This approach is based on *query modification* and *transaction modification* strategies, where an arbitrary query or transaction that may violate the integrity of a database is modified, such that the execution of the modified query or transaction is assured to leave the database in a consistent state.

In the third approach which is known as the *tightly coupled subsystem*, the subsystem is seen as part of the basic functionality of a database system, and is fully integrated into it. This approach, initially proposed by Hammer and McLeod (1975) and Eswaran and Chamberlin (1975), is employed by the Starburst project (Ceri, Fraternali, Paraboschi & Tanca, 1994), SICSDD project (Ibrahim, Gray & Fiddian, 1998) and the latest versions of commercial DBMSs such as INGRES.