Checking Integrity Constraints in a Distributed Database

Hamidah Ibrahim
Universiti Putra Malaysia, Malaysia

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 guarantee 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 preventing 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 database management system (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 and ORACLE. In this approach, integrity tests are general rather than transaction specific and thus no knowledge of the internal structure of a transaction is required. Typically, this requires rule mechanisms to implement integrity constraint enforcement (Ibrahim, 2002a).

BACKGROUND

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 to be an important part of any modern DBMS (Grefen, 1993). The crucial problem in designing a complete integrity subsystem is the difficulty of devising an efficient algorithm for enforcing database integrity against updates. Thus, it is not surprising that much attention has been paid to the maintenance of integrity in centralized databases over the last decade. 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, impractical and can lead to prohibitive processing costs (Embry, Gray, & Basiliades, 1993; Hsu & Imieliinski, 1985; Mazumdar, 1993, Plexousakis, 1993; Qian, 1988, 1989; Sheard & Stemple, 1989). Enforcement is costly because the evaluation of integrity constraints requires accessing large amounts of data which are not involved in the database update transition (Simon & Valduriez, 1987). Hence, improvements to this approach have been reported in many research papers (Bernstein & Blaustein, 1981; Blaustein, 1981; Henschen, McCune, & Naqvi, 1984; Hsu & Imieliinski, 1985; McCune & Henschen, 1989; Nicolas, 1982; Qian, 1989). Although this research effort has yielded fruitful results that have given centralized systems a substantial level of reliability and robustness with respect to the integrity of their data, there has so far been little research carried out on integrity issues for distributed databases. The problem of devising an efficient enforcement mechanism is more crucial in a distributed environment. This is due to the following facts (Barbara & Garcia-Molina, 1992; Mazumdar, 1993; Qian 1989; Simon & Valduriez, 1987):

- Integrity constraints are general statements about sets of data elements which may spread over several sites in a distributed database. A large amount of data may therefore need to be transferred around the network in order to determine the truth of such statements.
- Owing to the possibility of fragmentation of relations with the fragments stored at different locations, the integrity constraints must be transformed into constraints on the fragments so that they can be straightforwardly used for constructing efficient enforcement algorithms. Thus, there are usually more integrity constraints in an equivalent distributed database than a centralized database, all of which need to be maintained. In addition, replication of data imposes an additional constraint that the replicas must have equivalent values at all times.
- Frequent updates can lead to frequent executions of expensive violation testing operations.
- If some constraints are violated, the whole update transaction which causes the state transition must be aborted and the database must be restored to the previous state, which can be a very costly operation in a distributed system.

The brute force strategy of checking constraints is worse in the distributed context since the checking would typically require data transfer as well as computation leading to complex algorithms to determine the most efficient approach. Allowing an update to execute with the intention of aborting it at commit time in the event of constraint violation is also inefficient since rollback and recovery must occur at all sites which participated in the update. Thus, the question of interest is how to efficiently check integrity constraints in a distributed environment.

Many researchers have studied the problem of maintaining the consistency of a database and not surprisingly many different approaches have been proposed. For centralized databases, researchers have suggested that constraint checking can be optimized by exploiting the fact that the constraints are known to be satisfied prior to an update, and by reducing the number of integrity constraints that need checking by only checking the subset of integrity constraints that may be vio-