A COMPARISON OF DECLARATIVE AND PROCEDURAL CONSTRAINTS IN DATABASE MANAGEMENT SYSTEMS

Mohammad Badawy
Karel Richta

Department of Computer Science and Engineering, FEL, ČVUT, Karlovo nam. 13, 121 35 Praha 2, CZ, *{badawm1, richta}@cslab.felk.cvut.cz*

Abstract

Integrity constraints are used to ensure that the data in a database complies with rules that have been set to establish accurate and acceptable information for a database. Without integrity constraints, enforcing basic rules for a business data would be complex, if not impossible. Supporting integrity constraints is essential for database systems. Triggers are methods that are provided to the application programmer and database designer to ensure data integrity, and they can be used to define and enforce any type of integrity. Triggers are very useful for those databases that are going to be accessed from a multitude of different applications because they enable business rules to be enforced by the database instead of relying on the application software. This paper surveys the integrity constraints enforcement in the standard SQL: 1999 and the commercial database management systems, and compares the integrity features provided by each of them.

1. Introduction

Integrity constraints serve for several purposes in database design, implementation, and run-time. Besides specifying admissible data, they also often embed portions of the knowledge specific to an application domain. Furthermore, as database run-time integrity constraints can be exploited for query optimization purposes and thus can lead to a better performance for a query evaluation [16]. Integrity constraints enforcement is one of the most needed functionalities of advanced database applications [3].

Support for active data is crucial to the management of the world's information. Triggers provide a very powerful mechanism to realize effective constraints enforcing [16].

Declarative constraints and triggers are two essential features that have been introduced to support user requirements in relational database management systems. Given the differing expressive powers of declarative constraints and triggers, support for both are required for today's applications [5].

The specification of what data is semantically correct constitutes one of the most important tasks in the database design process [11,4]. In this process, data correctness requirements are gathered from users, applications developers, and business regulations, and are translated into integrity constraints. Integrity constraints, formulated in some language used in the database design, specify conditions on the database objects, which have to be satisfied in order to reflect the real world and universe of discourse correctly. Integrity constraint specifications are typically translated into constraint enforcing mechanisms provided by the database management system used to implement the database [16].

There are advantages to using both declarative constraints and procedural triggers, and both types of constructs are available in many commercial systems. It is not feasible to expect applications providers to either migrate their existing applications to use only triggers or partition the tables in their database according to the type of constraints and triggers that are required. It is therefore imperative to define and understand the interaction of constraints and triggers [5,2].

The text in this paper is structured as follows: In Section 2, we compare the integrity support in different database management systems by using declarative and procedural concepts. Design rules and guidelines for implementing constraints enforcing mechanisms in form of declarative and procedural (trigger) mechanisms are provided in Section 3. Finally Section 4 gives the conclusions.

2. Integrity Support in SQL: 1999 and Database Management Systems

In this section, we compare the integrity features of SQL: 1999 [10] with Oracle8i Server (Release 8.1.6) [14], IBM DB2 Universal Database (Version 7) [7], Informix Dynamic Server (Version 9.1) [8], Microsoft SQL Server (Version 7.0) [13], Sybase Adaptive Server (Version 11.5) [15], and Ingres II (Release 2.0) [9][16]. Table 1 presents a summary of the support of declarative integrity constraints in the various systems. The symbol `Y' states that the language construct is provided in the corresponding SQL dialect; `(Y)' marks the cases where the language construct is implicitly supported but not completely provided. The interesting issues of the comparison will be discussed briefly in the following:

			SQL-99	Oracle	DB2	Informix	MSSQL	Sybase	Ingres
NOT NULL			Y	Y	Y	Y	Y	Y	Y
DEFAULT			Y	Y	Y	Y	Y	Y	Y
UNIQUE			Y	Y	Y	Y	Y	Y	Y
PRIMARY KEY			Y	Y	Y	Y	Y	Y	Y
FOREIGN KEY	MATCH	SIMPLE	Y	(Y)	(Y)	(Y)	(Y)	(Y)	(Y)
		PARTIAL	-	-	-	-	-	-	-
		FULL	-	-	-	-	-	-	-
	ON DELETE	NO ACTION	Y	(Y)	Y	(Y)	(Y)	(Y)	(Y)
		RESTRICT	Y	-	Y	-	-	-	-
		CASCADE	Y	Y	Y	Y	-	-	-
		SET NULL	Y	Y	Y	-	-	-	-
		SET DEFAULT	Y	-	-	-	-	-	-
	ON UPDATE	NO ACTION	Y	(Y)	Y	(Y)	(Y)	(Y)	(Y)
		RESTRICT	Y	-	Y	-	-	-	-
		CASCADE	Y	-	-	-	-	-	-
		SET NULL	Y	-	-	-	-	-	-
		SET DEFAULT	Y	-	-	-	-	-	-

CHECK	Column-level Row-level Table-level Database-level	Y Y Y Y	Y Y - -	Y Y - -	Y Y - -	Y Y - -	Y Y - -	Y Y - -
DOMAIN		Y	-	-	-	-	-	-
ASSERSION		Y	-	-	-	-	-	-

Table 1. Comparison of Declarative Integrity Constraints Support

2.1 Nullability and Default Values

In all considered systems, a not null constraint is defined either by using the keyword NOT NULL directly in the column specification or by specifying a check clause of the form CHECK (column IS NOT NULL). Whereas in nearly all of these systems columns are nullable by default unless null values are explicitly disallowed, in Sybase columns are implicitly defined as nonnullable. Null values have to be explicitly allowed using the keyword NULL in the column specification.

2.2 Unique/Primary Key Constraints

- *Semantics of unique/primary key constraints*. In DB2, Informix, MSSQL, Sybase, and Ingres, the semantics of a uniqueness constraint is defined as follows:

UNIQUE (u_1,…, u_n) holds on table $R \Leftrightarrow \forall r_1, r_2 \in R : \left(\bigvee_{i=1}^{n} r_1.u_i \neq r_2.u_i \right)$

That is, no two rows shall have the same value combination for the uniqueness columns u_1,..., u_n. Oracle defines the semantics of a uniqueness constraint less restrictive:

UNIQUE (u_1,…, u_n) holds on table

$$R \Leftrightarrow \forall r_1, r_2 \in R : \left(\bigvee_{i=1}^{n} r_1.u_i \neq NULL \vee r_2.u_i \neq NULL \right) \Rightarrow \left(\bigvee_{i=1}^{n} r_1.u_i \neq r_2.u_i \right)$$

Compared to the other systems, Oracle additionally allows that there may exist two rows whose uniqueness columns completely consist of null values.

- *Nullability of unique/primary key columns*. DB2, MSSQL, and Ingres do not allow unique/primary key constraints on nullable columns.

In Oracle, Sybase, and Informix, the definition of a primary key implicitly defines not null constraints on the corresponding uniqueness columns.

- *Integrity violation message in case of simultaneous violation of multiple uniqueness constraints*. Oracle, MSSQL, and Sybase print the name of the uniqueness constraint, which has the oldest timestamp (with respect to its recent enabling time). DB2, Informix, and Ingres only state that there is an integrity violation due to a uniqueness constraint.
- *Multiple uniqueness constraints on overlapping sets of columns*. In Oracle, Informix, and Sybase, the definition of a uniqueness constraint is rejected if there already exists a uniqueness constraint that is defined on the same order of columns. DB2 and Ingres are even more restricted. Here, as in SQL-99, the creation of a uniqueness constraint is rejected if there

already exists a uniqueness constraint that is defined on the same set of columns. MSSQL has no restrictions concerning the definition of uniqueness constraints on common Columns.

2.3 Foreign Key Constraints

As depicted in Table 1, all reference systems implement the simple match rule for referential constraints. Besides, MSSQL, Sybase, and Ingres purely rely on the no action rules for deletions from and updates of the referenced table. The `(Y)' marked fields in Table 1 state that the corresponding systems implement this semantics but support neither the keyword ON DELETE NO ACTION nor the keyword ON UPDATE NO ACTION. The delete cascade rule is supported by Oracle, DB2, and Informix, the delete set null rule only by Oracle and DB2, and the restrict rule solely by DB2.

If in DB2 a table contains more than one foreign key, then all foreign keys must have the same delete rule as the first foreign key defined in this table. Surprisingly, in the case, when the first foreign key is based on the ON DELETE SET NULL rule, DB2 disallows the definition of multiple foreign keys.

2.4 CHECK Constraints

All reference systems support CHECK constraints at the row level. That is, the Boolean expression of the check clause must not contain subqueries, aggregate functions, host variables, stored procedure calls, pseudo-columns (like ROWNUM or ROWID), columns of different tables, and certain function calls (like USER or TODAY).

2.5 Comparison of Trigger Features

Sybase was the first commercial database system that provided the concept of a trigger. Currently, triggers are available in all reference systems. However, since the concept of a trigger was not standardized until the recent SQL: 1999 standard, the implementations in the various systems differ with respect to syntax (and semantics) and capabilities. The grammars of trigger definitions in the various systems are so different that no trigger specification that is formulated in one system can be used without any modifications in another system. Particularly, the action parts differ in the various approaches because each system has its own "trigger programming language". Table 2 compares the capabilities of the supported trigger mechanisms [16]. The comparison is based on the following properties [16]:

	SQL-99	Oracle	DB2	Informix	MSSQL	Sybase	Ingres
Number of Events per Trigger	1	n	1	1	1	n	n
Number of Triggers per Event	n	n	n	1/n	n	1	n
Trigger Granularity	R/S	R/S	R/S	R/S	S	S	R/S
Activation Time	B/A	B/A	B/A	B/A	A	A	A
Activation Condition	Y	Y	Y	Y	Y	Y	Y

Table 2. Trigger Features

- *Number of events per trigger*. Potential events that can lead to the firing of a trigger are insert, update, and delete operations. In SQL-99, DB2, Informix, and MSSQL, a trigger is based on exactly one event. Oracle, Sybase, and Ingres, in contrast, allow defining a trigger on multiple events on the same table. Informix does not allow defining a delete trigger on a table that contains a foreign key with the ON DELETE CASCADE option.
- *Number of triggers per events***.** Ingres and Informix allow defining at most one trigger for each event. In case of an update event, multiple triggers are possible if the column lists of the update events are mutually exclusive. The other systems do not have such a restriction.
- *Trigger granularity*. The granularity of a trigger can be FOR EACH ROW (R) or FOR EACH STATEMENT (S). MSSQL and Sybase only provide statement level triggers, whereas the other systems support both kinds of triggers. The default granularity is FOR EACH STATEMENT in Oracle and Informix, whereas it is FOR EACH ROW in Ingres. In DB2 the granularity has to be specified explicitly.
- *Trigger activation time*. A trigger is activated BEFORE (B) or AFTER (A) the specified event is handled. MSSQL, Sybase, and Ingres only provide AFTER triggers, whereas the other systems support both kinds of triggers. However, DB2 has the restriction that FOR EACH STATEMENT cannot be specified for BEFORE triggers.
- *Trigger activation condition*. The activation of a trigger can be associated with a condition. All reference systems support such conditional triggers, although they differ in the way they specify such a condition.

3. Declarative versus Procedural Specifications

This section provides several design rules and guidelines for implementing constraint-enforcing mechanisms in the form of declarative and procedural (triggers) mechanisms.

Suppose in our sample application, we have to manage information about employees. Employees are uniquely identified by id. We assume that every employee has a first name and a last name. Every employee has a supervisor, except the top manager who supervises himself. Also in our sample application two types of jobs are distinguished: ‘manager‘ and ‘employee‘. The declaration of employee (emp) is as follow:

emp (id, lastname, firstname, job, supervisor, salary, bonus)

Before going into the details of implementing constraint enforcing mechanisms, it is worth mentioning that enforcing integrity constraints and rules identified in the application domain with declarative constraints (and even triggers) often is less costly than enforcing the equivalent rules by issuing SQL statements in an application.

For best performance, it is advisable to define and enable integrity constraints in the database system, and to develop applications to rely on them, rather than on (verifying) SQL statements in the applications. Another important reason to embed integrity constraints in the database schema rather than in applications is to preserve logical data independence. That is, if integrity constraints are added and modified in the database schema, applications do not need to be modified.

In the rest we focus on implementing constraint-enforcing mechanisms for the following four types of integrity constraints: row, table, inter-table, and state transition constraints.

Constraint Type		Declarative	Procedural (Triggers)
Row	NOT NULL	Y	-
	DOMAIN	Y	-
Table	PRIMARY KEY	Y	-
	UNIQUE	Y	-
	FOREIGN KEY	Y	(Y)
	Aggregate	-	Y
	Others	-	Y
Inter-table	FOREIGN KEY	Y	(Y)
	Dependencies	-	Y
	Aggregate	-	Y
	Others	-	Y
Transition		-	(Y)

Table. 3 Declarative versus Procedural (Triggers) Mechanisms

Although all types of integrity constraints can be implemented by only using triggers, because of performance reasons it is advisable to use declarative mechanisms whenever possible. Table 3 gives an overview of what mechanisms should be used for the different types of integrity constraints.

- *Not null*. In all reference systems, a not null constraint can be specified by declarative constraints either directly using the NOT NULL clause or within a CHECK clause.
- *Domain*. Domain constraints can either be specified in a CHECK clause by explicitly listing admissible values or restricting an existing domain, or a domain can be explicitly specified and used as column type.
- *Primary key*. All reference systems provide declarative language constructs to specify a primary key for a table.
- *Unique*. This type of integrity constraint is supported by all reference systems through the UNIQUE clause. Candidate keys for a table that have not been chosen as the primary key always should be specified as unique.
- *Aggregate* (*table and inter-table*). Integrity constraints that include the aggregate functions MIN, MAX, SUM, COUNT, or AVG cannot be specified in a declarative fashion. This includes even very simple integrity constraints such as "every department must have at least two employees". Although the SQL standard supports complex CHECK conditions that can be associated with a table, none of the reference systems supports declarative specifications for this type of integrity constraint. They have to be implemented using triggers.
- *Others* (*table*). These types of integrity constraints include all constraints that cannot be specified for a table using the above declarative mechanisms. Typical examples are column value comparisons of pairs of rows from the same table. For example, the integrity constraints "A manager's salary must be greater than an employee's salary ", or ``An employee must work in the same department as her/his supervisor". Again, these types of integrity constraints can be formulated in the SQL standard using a complex CHECK condition, which is not supported by any of the reference systems. Thus they must be formulated using triggers. In the next section we will illustrate how to formulate the first example above by using triggers.
- *Foreign key*. In all reference systems, a foreign key can be specified either using the REFERENCES clause, referring to a single column primary key, or the FOREIGN KEY clause, referring to a composite (multi-column) primary key. Note that if many tables have a foreign key referencing the same table, it is important to choose a "good" primary key for that

table. However, since many database management systems do not provide the various rules of update and delete, these have to be implemented by triggers.

- *Dependencies*. Such integrity constraints include all types of dependencies among rows from different tables that cannot be expressed through foreign key constraints. For example, the integrity constraint "The location of a project must be the same as the location of the department managing the project " specifies an inclusion dependency.

The above descriptions should give some good pointers on what type of constraint enforcing mechanism is applicable to what type of integrity constraint. In order to classify an integrity constraint into one of the above categories, it is important to have them decomposed. If it seems like two "portions" of an integrity constraint can be enforced by different mechanisms, it is likely that this integrity constraint should be decomposed accordingly.

Finally, we briefly point out that the interplay of various integrity constraints can lead to different implementations of the integrity constraints. These different implementations can result in different performances. Therefore, it is an important design rule to consider all related integrity constraints when implementing one.

4. Conclusions

We have provided in this paper a comparison of integrity features in the standard SQL: 1999 and the major relational database management systems. We have provided general guidelines for design and implementation of integrity constraints in form of declarative and procedural mechanisms.

Briefly we observe that the relational database management systems basically support the entry level of the SQL: 1999 [6,16,12]. This mean that with regard to the specification of declarative constraints these systems support the specification of default values, not null constraints, primary and foreign keys, uniqueness constraints, and check constraints on the columns level. The biggest discrepancy between the current standard SQL: 1999 and the reference systems considered in this paper is on the specification and usage of database triggers. The reason of this is, triggers have been provided by most systems prior to the introduction of SQL: 1999.

When using declarative integrity constraints there are many problems. SQL: 1999 defines unlimited CHECK clause, but current implementations do not allow it. In our point of view the best way is to convert the CHECK clause into triggers. We suggest that integrity constraints can be specified into a more conceptual specification (on conceptual schema), and then converted into triggers, our paper [1] has provided a general deriving rules for triggers from constraints specifications.

References:

1. Badawy, M., Richta, K. Deriving Triggers from Integrity Constraint Specifications in the Database Management Systems. Journal of Advanced Engineering: Acta Polytechnica, Vol. 41, No. 6/2001, Prague, CZ, pages 39-44
2. Badawy, M., Richta, K. Integrity Enforcement in a Database Using Constraints and Triggers. The 16th International Conference on Production Research (ICPR), 29.7 - 3.8.2001, Prague, CZ, ISBN 80-02-01438-3

3. Baralis, E., Ceri, S., Paraboschi, S. Declarative Specification of Constraint Maintenance. In International Conference on ER Approach, pages 205-222, Springer-Verlag, LNCS 881, 1994
4. Biskup, J. Achievements of Relational Database Schema Design Theory Revisited. In B. Thalheim and L. Libkin, editors, Semantics in Databases, Lecture Notes in Computer Science, Vol.1358, pages 29--54. Springer-Verlag, Berlin, 1998
5. Cochrane, R., Pirahesh, H., Mattos, N. Integrating Triggers and Declarative Constraints in SQL Database Systems. In Proceedings of the 22nd VLDB Conference, Bombay, India, 1996
6. Date, C.J., Darwen, H. A Guide to the SQL Standard. Addison-Wesley, Reading, MA, 4 edition, 1997
7. IBM Corporation. IBM DB2 Universal Database: SQL Reference, Version 6, 1999
8. Informix Software, Inc., Menlo Park, CA. INFORMIX-Universal Server: Informix Guide to SQL: Syntax, Version 9.1, March 1997
9. Ingres Corporation. Ingres Database Administrator's Guide, Version II, 1999
10. International Organization for Standardization (ISO) & American National Standards Institute (ANSI), ANSI/ISO/IEC 9075-2:99. ISO International Standard: Database Language SQL - Part 2: Foundation (SQL/Foundation), September 1999
11. Maier, D. The Theory of Relational Databases. Computer Science Press, Rockville, MD, 1983
12. Melton, J., Simon, A.R. Understanding the New SQL - A Complete Guide. Morgan Kaufmann Publishers, San Mateo, CA, 1993
13. Microsoft Corporation. Microsoft SQL Server, Version 7.0, 1999
14. Oracle Corporation. Oracle8i SQL Reference, Release 8.1.6, December 1999
15. Sybase Inc. Transact-SQL User Guide, Version 11.0, 1999
16. Turker, C., Gertz, M. Semantic Integrity Support in SQL-99 and Commercial (Object) Relational Database Management Systems. Computer Science Technical Report, University of California, Davis, USA, CSE-2000-11, 2000