Chapter II
Bounded Cardinality and Symmetric Relationships

Norman Pendegraft
University of Idaho, USA

INTRODUCTION

Bounded cardinality occurs when the cardinality of a relationship is within a specified range. Bounded cardinality is closely linked to symmetric relationships. This article describes these two notions, notes some of the problems they present, and discusses their implementation in a relational database.

BOUNDED CARDINALITY AND SYMMETRIC RELATIONSHIPS

Bounded Cardinality

An entity relationship diagram (ERD) shows the cardinality of each entity in a relationship. In an ERD, minimum cardinalities can be either 0 or 1, and maximum cardinalities can be 1 or infinity. Bounded cardinality occurs when a relationship between entities has cardinality within a specified range. Problems displaying bounded cardinality might include team rosters that must have exactly 5, 9, 11, or some other number of players. Figure 1 illustrates how UML (unified modeling language) provides for modeling specified-range relationships in a class diagram (Dennis, Wixom, & Tegarden, 2005). ERD, as described by Chen (1976), does not, although there are extensions to the ERD model that do (Webre, 1981). The SQL-92 standard provides for such constraints, but many relational database management systems (RDBMSs) do not support these features, and consequently do not allow for easy implementation of such a constraint (Lewis, Bernstein, & Kifer, 2002).

Bounded cardinality presents some interesting problems. For example, Boufares and Kraïem (2001) point out that cardinality constraints may result in conflicts. Figure 2 illustrates one of their examples. In Figure 2, if we let e_i be the number of instances of entity E_i and r_i be the number of instances of relationship R_i, then we get the following constraints.

$r_1 = e_1$ $\qquad$ $r_1 > e_2$
$r_2 < e_2$ $\qquad$ $r_2 > 2\ e_1$

These lead in turn to e1 > e2 and e2 > 2 e1. Clearly these allow only the solution e1 = e2 =0,

Figure 1. Specified range / bounded cardinality

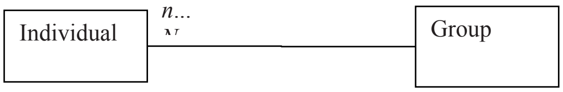
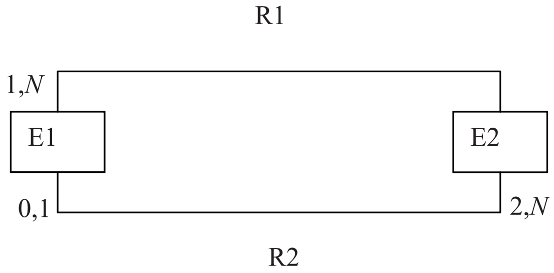


Figure 2. Inconsistent cardinality constraints



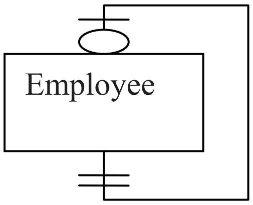
that is, an empty database. Boufares and Bennaceur (2004) offer a mathematical programming technique to detect inconsistent constraints.

Symmetric Relationships

Symmetric relationships require that if a relationship R(*x*,*y*) holds for *x* and *y* in S, then R(*y*,*x*) must also hold (Dean, 1966). For example, if George is married to Martha, then Martha must also be married to George. While these relations are common, they may be difficult to model and to implement in an RDBMS. In particular, there is no way in an ERD to easily show a symmetric relationship. The relational database model does not provide for any way to impose a constraint requiring that a relationship be symmetric.

Bounded cardinality also arises from symmetric relationships. As Ross and Stoyanovich (2004, p. 913) note, there is a "natural isomorphism between symmetric relationships among k entities and k-element multi-sets." Multisets may be represented using bounded cardinality. In other words, a group may also be viewed as a relationship amongst its members. For example, a marriage may be viewed as a group of size 2. SQL-2003 provides for a multiset data type (Eisenberg, Melton, Kulkarni, & Zemke, 2004). Similarly, a team roster represents a symmetric relationship. Suppose an athletic league maintains a roster of players. Each player is on one team, and each team must have from 5 to 10 players. In Figure 1 we would have *n* = 5 and *N* = 10. Note that the business problem requires that the relationship be symmetric: If Able is on a team with Baker and Charlie, then Baker must be on a team with Able and Charlie as well.

Figure 3. A simple design for a marriage



Constraints

Business rules are generally imposed in a database design via constraints. These constraints may be implemented declaratively or procedurally. Declarative constraints are created in the database definition. An example is "NOT NULL" to require that an attribute have some value. Another is "REFERENCES," which imposes a referential integrity constraint on a tuple.

Procedural constraints are imposed in the logic of applications or in triggers. For example, the procedure that allows us to record a marriage could be designed so as to ensure that the data entered would be symmetric. In contrast, a declarative constraint would build that requirement into the database itself and simply forbid entering nonsymmetric data. Date (2000) points out that declarative constraints are superior because they relieve programmers of additional programming tasks. Since the code would be more difficult to write, test, and debug, there is a greater chance of an error that will further introduce errors into the data. Türker and Gertz (2001) point out that declarative constraints are less costly to execute.