REPRESENTING MULTIVALUED ATTRIBUTES IN DATABASE DESIGN

Mark I. Hwang, Central Michigan University, mark.hwang@cmich.edu
Jack D. Becker, University of North Texas, becker@unt.edu
Jerry W. Lin, University of Minnesota, Duluth, jlin@d.umn.edu

ABSTRACT

An important aspect of data modeling is the identification of entities or objects and their associated attributes. An attribute may be single-valued or multivalued. During database design, the designer usually has to represent all attributes in relational tables that only allow single-valued cells. A number of authors (1, 3, 5, 6) advocate the notion that multivalued attributes are essentially separate entities and, therefore, should be represented as a separate table. We argue that some multivalued attributes should be treated as entities, but others simply are not and should, therefore, not be treated as entities. Assuming that all multivalued attributes are entities creates confusion and results in an inefficient design. We discuss alternative designs, such as storing multiple values in one cell, that should be used for different types of multivalued attributes. We also present a decision table for easy reference by designers in dealing with multivalued attributes.

Keywords: Database design, multivalued attributes, multivalued dependencies.

INTRODUCTION

An important aspect of data modeling is the identification of entities or objects and their associated attributes. In the seminal paper that describes semantic data modeling, Hammer and McLeod (3) first distinguished two types of attributes: single-valued and multivalued. In their model, a multivalued attribute is defined as an entity class, a "collection of entities" (3, p. 363). In other words, a multivalued attribute is also an entity that describes certain aspects of other entities. This definition of multivalued attributes extends the meaning of attributes to include entities. Such attributes were later termed "semantic objects" by some authors (e.g., 2, 4) to represent things that are important to users.

Subsequently the term "multivalued attributes" has been used by many authors to represent not only entities or objects but also "simple attributes". For example, citing that an employee may have several skills, McFadden, Hoffer, and Prescott (5) defined "skill" as a multivalued attribute because it is "an attribute that may take on more than one value for a given entity instance." (p. 95) Similarly, Connolly and Begg (1, p.154) used telephone number to illustrate multivalued attributes because in their example a branch entity may have several telephone numbers. Both McFadden, Hoffer, and Prescott (5) and Connolly and Begg (1) treated simple attributes that are multivalued no differently than entity attributes that are multivalued. We will show that these two types of attributes are distinct and, thus, should be represented differently in a database. In the next section we discuss in detail different types of attributes. We then explore the proper way to represent different types of multivalued attributes in database design.

ATTRIBUTE TOPOLOGY

Attributes can be classified in several ways. Most authors distinguish simple attributes from group attributes (e.g., 1, 4, 5). A simple attribute contains one piece of data while a group attribute is composed of two or more attributes. Employee number is an example of a simple attribute whereas employee name is an example of a group attribute, which is made of first name, middle initial and last name. Several authors (e.g., 2, 4) follow the semantic object modeling approach and recognize another type of attributes, object attributes, which are themselves objects or entities. Assuming that a company has issued certain employees a corporate charge card, then charge card is an attribute of employee, and charge card in turn has its own attributes such as the issuing company, balance, etc.

An attribute may also be single- or multivalued. For instance, most companies only need to know a single name for each of its customers. However, a mortgage company may require a potential customer to supply all aliases in a mortgage application. Customer name, therefore, may be a single-valued or multivalued attribute. A company that issues its employees corporate charge cards may have a policy to limit every employee to one card. In that case, charge card is a single-valued attribute of employee. On the other hand, if an employee is allowed to have more than one card, then charge card is multivalued.

Figure 1 illustrates an employee object with different types of multivalued attributes. Title is a simple attribute that is multivalued, as an employee may hold several titles. Telephone number is modeled as a group attribute consisting of area code and phone number. It is multivalued as an employee may have several telephone numbers. Finally, charge card is a multivalued object attribute as an employee is allowed to have multiple cards.

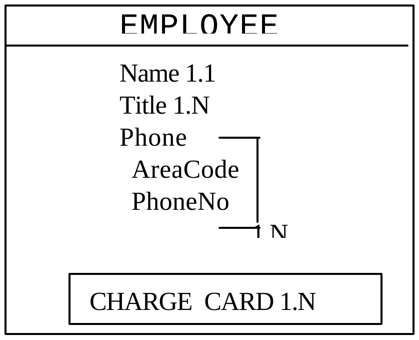

Figure 1. An Employee Object with Three Multivalued Attributes

REPRESENTATION OF MULTIVALUED ATTRIBUTES

Conventional Approach

In most cases, database design involves the definition of relational tables. An important characteristic of tables is that each cell must be single-valued, thus presenting problems for multivalued attributes. As mentioned above, some authors treat all multivalued attributes the same and advocate creating a separate table for each multivalued attribute. The rationale, as pointed out by Sanders (6, p. 24), is that in many instances a multivalued attribute *is* an entity.

Potential Problems

We agree with Sanders that as long as a multivalued attribute is also an entity, it deserves a table of its own. However, as discussed previously, a simple or group attribute can also be multivalued. To create a table for multivalued attributes that are *not* entities causes potential confusion and results in an inefficient design. To understand why, consider the example used by Connolly and Begg (1), in which a branch has several telephone numbers. If a separate table is created for the multivalued attribute telephone number, then information about each branch is scattered into two tables as shown in Figure 2:

BRANCH

Branch_No	Name	Address

BRANCH_TELEPHONE

Branch_No	Telephone_No

Figure 2. A Two-Table Design for A Multivalued Attribute

By convention, the primary key is underlined in each table. The BRANCH_TELEPHONE table has a composite key, since each branch may have multiple telephones.

This design is prescribed by Connolly and Begg (1) and the same approach is recommended by almost all authors including Kroenke (4), McFadden, Hoffer, and Prescott (5) and Sanders (6). However, the design is problematic since the primary key is unnecessarily duplicated. Inefficiency is also introduced as two tables rather than one has to be accessed whenever the complete information about a branch is needed for processing. Finally, as will be shown later, a more serious problem with this design is that it may be wrong even if one can make a case that Telephone_No is an entity and thus deserves a table of its own.

The inefficiency problem can quickly multiply when an entity has more than one multivalued attribute. The employee object in Figure 1, for example, has three multivalued attributes and, therefore, employee information may be spread across four tables. Similarly, if a branch serves several geographical areas then a separate table may need to be created just to keep track of the areas served by each branch. Also, a branch may have, say, several salesmen. Should another table be created just to keep track of which salesman belongs to which branch?

A closer look at the previous example reveals that some multivalued attributes are distinct entities and, therefore, deserve a separate table. For example, a salesman is also an employee and will be part of an employee table. Other attributes are potential entities; for example, service areas. If only the name of different areas is important (e.g., North America versus Europe), then area is an attribute. If, on the other hand, the company is interested in related attributes such as the population, the median income, etc. of each area, then area becomes an entity. Sanders (6) argued that multivalued attributes should be elevated to entities because related attributes *may need* to be collected in the future. However, some non-entity attributes will probably always stay that way. A good example is the aliases of customers. It is inconceivable to consider aliases as an entity that is separate from the customer entity. Similarly, a bank or financial institution generally has joint accounts. When an account is registered with several customer names, it is inconceivable to put the names into a table that is separate from the rest of the account information. In fact, doing so will likely create confusion that customer name is an entity that is independent of the customer entity.

Finally, even if one can show that a multivalued attribute is an entity, the design in Figure 2 may be fundamentally wrong. The reason is that the multivalued attribute may have either a many-to-one relationship or a many-to-many relationship with the other attribute. The proper design is dictated by the relationship that exists between the two attributes, which is related to multivalued dependency (MVD), as discussed next.

The Multivalued Dependency Connection

MVD is an important concept in the relational model that describes a constraint that a particular attribute has multiple values for every instance of another attribute. Sanders (6, p. 133) distinguished three types of MVD: simple, independent, and transitive. Simple MVD involves two attributes whereas both transitive and independent MVD involve three attributes. The latter is usually discussed in normalizing relations into third and fourth normal form, respectively. It is the simple MVD that is of interest here.

A simple MVD involves *at least* one multivalued attribute. For example, if a company allows an employee to have more than one charge card, then an MVD exists between, say, employee number and card number, which is usually depicted as follows:

EMPLOYEE_NUMBER --> CARD_NUMBER

CARD_NUMBER is a multivalued attribute, which has a many-to-one relationship with EMPLOYEE_NUMBER. However, if, at the same time, a card may be shared by employees of the same department, then a recursive MVD also exists as follows:

CARD_NUMBER -->EMPLOYEE_NUMBER

The relationship between EMPLOYEE_NUMBER and CARD_NUMBER becomes many-to-many and EMPLOYEE_NUMBER becomes a multivalued attribute too. In summary, a simple MVD involves two attributes, one of which or both are multivalued. The next section presents a complete treatment of representation of multivalued attributes. The proper design for non-entity attributes is discussed first, followed by the design based on the relationship involved between two (or more) entity attributes.

COMPLETE TREATMENT

Non-Entity Attributes

As mentioned above, separating non-entity multivalued attributes into a different table is confusing and inefficient. Therefore, multivalued attributes that are not potential entities should be left in one table along with related attributes. The designer can identify all possible values of the attribute and separate them into different columns. For example, it is a common practice to ask for both the home telephone and work telephone of a new customer. Instead of treating telephone as a multivalued attribute, the designer should create a separate column for home telephone and another for work telephone. Similarly, for a joint account, instead of treating account name as a multivalued attribute, the designer should create a separate column for the primary account holder name and another for the secondary account holder name.

A problem arises when it is difficult to identify all possible values of an attribute or the number of possible values is unknown. For example, a company may wish to track the skills of its employees so that it can assign employees to different projects. In this case, further consultation with the users may enable the designer to arrive at a best estimate of the total number of skills in need of tracking. Another solution is to simply enter all skills into a single column (with a varying length field). An employee who is proficient in COBOL, C ++, and JAVA will have an entry such as: “COBOL/C++/JAVA”. This is a simple yet robust solution as it can accommodate any number of values of an attribute.

Using a single column to store a multivalued attribute is a natural choice in some cases. For example, the title of Bill Gates is “Chairman and CEO”. Rather than putting “Chairman” and “CEO” into two columns, or worse yet into two rows in a separate title table, a natural design is to put “Chairman and CEO” or “Chairman/CEO” in a single title column. Similarly, while most students have a single major but some have double major. Rather than creating a separate column for the second major, a more natural design is simply record a double major as, say, “MIS/Accounting”.

To summarize, a non-entity multivalued attribute should not be represented in a separate table as conventionally prescribed. A possible solution is to arbitrarily define the maximum number of values allowed and reserve the same number of columns, one for each value. A simpler and yet more robust solution is to use a single column to store the multivalued attribute. A sample employee table keeping track of skills using this approach is shown in Table 1.

Name	**Phone**	**Skills**
Andre Agassi	6388	COBOL
Michael Chang	2053	COBOL/JAVA
Jim Courier	9985	C++/JAVA
Pete Sampras	1566	COBOL

Table 1. An Employee Table with One Multivalued Attribute

Processing of this table presents no problems in SQL. For example, to find which employees are proficient in, say, COBOL:

```
SQL> Select * from Employee where skills like ‘%COBOL%’;
SQL>            Name                Phone           Skills
                Andre Agassi        6388            COBOL
                Michael Chang 2053          COBOL
                Pete Sampras        1566            COBOL
```

Similarly, we can find out the number of skill sets and their membership as:

```
SQL> Select skills, count(*) from Employee group by skills;
SQL>            Skills                      Count(*)
                C++/JAVA                           1
                COBOL                              2
                COBOL/JAVA                         1
```

Entity Attributes

The proper design for entity attributes depends on the relationship between the two attributes. If the relationship is many-to-many, the two attributes can be stored in a separate table with a composite key. As noted previously, this design is advocated by many authors, but it really only applies to entity attributes having a many-to-many relationship. If the relationship is many-to-one, the two attributes can also be stored in a separate table, but with a simple key, which is the multivalued attribute. For example, if an employee can have multiple charge cards and a charge card can belong to multiple employees, then a many-to-many relationship exists between two independent entities, EMPLOYEE and CHARGE CARD. The proper design is shown in Figure 3:

EMPLOYEE	Emp_No	Name	Address
CHARGE_CARD	Card_No	Balance	
EMPLOYEE_CARD	Emp_No	Card_No	

Figure 3. Design for Multivalued Attributes in Many-to-Many Relationships

If, on the other hand, an employee can have multiple cards but a card can belong to only one employee, then a many-to-one relationship exists between CHARGE CARD and EMPLOYEE. The proper design for this case is shown in Figure 4:

EMPLOYEE	Emp_No	Name	Address
CHARGE_CARD	Card_No	Balance	Emp_No

Figure 4. Design for Multivalued Attributes in Many-to-One Relationships

The above discussion can be summarized in a decision table as shown in Table 2 below. The current literature only focuses on the third case, and we have shown two other cases where different designs are called for.

IF A --> B AND B Is A(n)	Number of Tables	Attributes & Key
Simple or group attribute	One	T (<u>A,</u> B , …)
Entity attribute (N:1 relationship)	Two	T (<u>A,</u> …), T (<u>B</u>, … A)
Entity attribute (N:M relationship)	Three	T (<u>A,</u> …), T (<u>B</u>, …), T(<u>A,B</u>, …)

Table 2. Decision Table for Representing Multivalued Attributes

CONCLUSION

Database design is still as much an art as a science. It is not uncommon to have several alternative designs for a given situation. However, the current literature seems to suggest an oversimplified solution to the problem of representing multivalued attributes in relational tables. Whereas most authors argue that every multivalued attribute should be treated as a separate table in a relational database, there are different types of multivalued attributes and each type should be represented differently. A multivalued attribute deserves a separate table only when it is an entity *and* when it has a many-to-many relationship with the other entity. If the multivalued attribute is not an entity or when the relationship involved is not many-to-many, alternative designs should be adopted.

REFERENCES

1. Connolly, T. and Begg, C. (1999). *Database Systems: A Practical Approach to Design, Implementation, and Management*, 2nd edition, Addison-Wesley.
2. Dewitz, S. and Olson, M. (1994). *Semantic Object Modeling with Salsa: A Case Book*, Mitchell McGraw-Hill.
3. Hammer, M. and McLeod, D. (1981) Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, (6): 3, 351-386.
4. Kroenke, M. D. (2000). *Database Processing: Fundamentals, Design, and Implementation,* 7th edition, Prentice Hall.
5. McFadden, F.R., Hoffer, J.A. and Prescott, M.B. (1999). *Modern Database Management*, 5th edition, Addison-Wesley.
6. Sanders, G.L. (1995). *Data Modeling*. Boyd & Fraser.