Teaching Database Modeling and Design: Areas of Confusion and Helpful Hints

George C. Philip, Ph. D.
College of Business, The University of Wisconsin – Oshkosh, Oshkosh, WI, USA

Philip@uwosh.edu

Executive Summary

This paper identifies several areas of database modeling and design that have been problematic for students and even are likely to confuse faculty. Major contributing factors are the lack of clarity and inaccuracies that persist in the presentation of some basic database concepts in textbooks. The paper analyzes the problems and discusses ways to minimize them. Specifically, the paper discusses the practical role of normalization in database design, addresses the confusion in representing attributes with repeating values, discusses how to remove inconsistencies in defining relations and first normal form, simplifies the process of identifying candidate keys to normalize relations, clarifies the conditions under which insertion and deletion anomalies may occur, and sheds light on the confusion in defining weak entities.

Normalization plays a vital role in both the theory and the practice of database design. The top-down approach popularly used in relational database design creates a conceptual schema that is represented by entity-relationship (E-R) models, and then uses mapping rules to convert the conceptual schema to relation schemas. Because E-R modeling is an intuitive process, errors could occur in identifying entities and their relationships, resulting in un-normalized relations. Un-normalized relations also could result from converting files in legacy systems and spreadsheets to relational tables. Normalization plays a key role in verifying the goodness of design of such relations and in improving the design.

The concept of repeating values in relations plays a major role in defining relations and first normal form. Yet, textbooks in general do not distinguish between multi-valued and single-valued attributes in a schema. This lack of clarity may result in conflicting interpretations of the schema. The paper presents a simple solution to the problem.

The lack of clarity in defining the terms tables, relations, and first normal form (1NF) in textbooks is another potential source of confusion. Some books define relation as a table with no duplicate tuples, and *only atomic values*. These books then redundantly define 1NF as a relation with *only atomic values*. Others define a relation as a table with columns and rows, and state that a relation is in 1NF if each value is atomic. These definitions fail to specify an important requirement of 1NF that there are no duplicate tuples. A third definition of 1NF that fails to include this property is that a table is in first normal form if each value is atomic.

A challenging task for many students during the normalization process is checking whether a determinant is a

Editor: Grandon Gill

candidate key. The standard method is to check whether every attribute of the relation is functionally dependent on the determinant. The paper presents a method that involves only the determinants, and therefore makes it easier to identify candidate keys. The paper also provides an alternate definition of Boyce-Codd Normal Form (BCNF), which is easier to apply.

Discussions in textbooks and other literature on the topic of normalization often give students the impression that data redundancy in un-normalized relations leads to all three types of anomalies – insertion, deletion, and update. The paper shows that though data redundancy generally results in insertion and deletion anomalies, that is not always the case. The conditions under which insertion/deletion anomalies don't occur are discussed.

Guidelines for mapping conceptual schema to relational tables often use the terms strong entity and weak entity to provide separate mapping rules for each. It is shown that the definition of weak entity as presented in many textbooks, however, is inaccurate. These books define weak entity using logical dependence rather than identifier dependence of entities.

The paper shows that several database design concepts and techniques commonly are presented inaccurately or ambiguously in textbooks and are problematic for students. However, as presented in the paper, simple solutions exist to minimize students' problems in these areas.

Keywords: Teaching database design, data modeling, multi-valued attributes, Boyce-Codd Normal Form, normalization, candidate key, weak entity, First Normal Form.

Introduction

Database design has been an integral part of MIS curricula for decades. This field is well-established, with an abundant supply of research papers and textbooks, many of which have undergone several revisions. Yet, lack of clarity and inaccuracies in the presentation of some basic database modeling and design concepts persist in textbooks. This, in turn, creates confusion and difficulty for students, and even for faculty. This paper identifies such problems in several areas and provides helpful hints for faculty and students. Specifically, the paper discusses the practical role of normalization in database design, addresses the confusion in representing attributes with repeating values, discusses how to remove inconsistencies in defining relations and first normal form, simplifies the process of identifying candidate keys to normalize relations, clarifies the conditions under which insertion and deletion anomalies may occur, and sheds light on the confusion in defining weak entities. The discussions here are based on an examination of seventeen database textbooks (Connolly & Begg, 2005; Date, 2004; Elmasri & Navathe, 2007; Frost & Van Slyke, 2006; Gillenson, 2005; Hoffer, Prescott, & McFadden, 2007; Kifer, Bernstein, & Lewis, 2006; Kroenke, 2006; Mannino, 2007; Post, 2005; Pratt & Adamski, 2008; Riccardi, 2003; Rob & Coronel, 2006; Rob & Semaan, 2004; Ullman & Widom, 2008; Umanath & Scamell, 2007; Watson, 2005).

Practical Significance of Normalization

Because several areas discussed here are related to normalization, we first examine the practical role of normalization and normal forms in designing databases. Database design may use a variety of inputs, including existing/proposed forms, reports, queries and transactions, data stored in spread sheets and legacy systems, and entities/attributes identified from knowledge of the business. Thus, for example, to develop a database for a new inventory control system to replace an existing spreadsheet-based system, the designer might use existing spreadsheets, reports, and additional data items represented by entities, such as vendors and product lines, that are identified from user requirements and knowledge of the business.

The Top-down Approach

In order to translate the data from various sources to a well-designed database, the top-down approach popularly used in relational database design uses three basic steps: 1) Create a conceptual schema that is represented by entity-relationship models, 2) Create a logical schema by mapping the conceptual schema to relation schemas using mapping rules, and 3) Apply normalization rules to test the goodness of the design and to improve the design, if necessary.

In general, if the attributes, entities, and their relationships are identified correctly, the proper application of mapping rules should result in normalized relations. However, because E-R modeling is an intuitive process, errors could occur in identifying entities and their relationships and in applying mapping rules, resulting in un-normalized relations. Consider a segment of an order form for a mail-order nursery, shown in Table 1.

Table 1: Segment of an Order Form

Martha Jones 2253 Lake Breeze Oshkosh, WI 54901		Cust#: 23476 Date: 9/24/2007		
Item#	**Description**	**Qty**	**Unit Price**	**Item Total**
65729	Blue Scabiosa	3	2.00	$6.00
14159	Mixed Hosta	2	3.33	$6.66
66711	Missouri Primrose	3	2.00	$6.00
For office use only: Order#: _____			Cost of Items	$18.66

Experience with student assignments indicates that inexperienced designers are likely to make mistakes in identifying entities and their relationships. Two common mistakes are:

1) Correctly recognize Customer as an entity, but incorrectly group the rest of the information as a single Order entity with a one-to-many relationship between Customer and Order. Applying the mapping rules to this conceptual schema would result in an un-normalized Order schema with data redundancy: **Order** (Order#, Date, Cust#, Item#, Description, Qty, Unit Price).

2) Correctly identify all three entities - Customer, Order, and Item - but incorrectly identify the relationship between Order and Item as one-to-many, resulting in an un-normalized Item schema: **Item**(Item#, Order#, Description, and Unit Price).

The number of entities erroneously represented in un-normalized relations resulting from the top-down approach tends to be small since some breakdown of the set of attributes already took place during creation of the conceptual schema by identifying at least some of the entities. Un-normalized relations also could result from converting spreadsheets and files in legacy systems to relational tables.

Verifying and Achieving Goodness of Design

A key role of normalization is to use a more scientific approach to verify the goodness of design of relation schemas that result from the top-down approach and from conversion of non-relational files. Thus, the goodness of design of the schema, **Order** (Order#, Date, Acct#, Item#, Description, Qty, Unit Price), which was mapped above from an incorrectly identified Order entity, may be tested by applying the definition of BCNF: a relation is in BCNF if, and only if, every candidate key is a determinant. Functional dependencies that represent the business rules

related to the Order entity indicate that Item# is one of the determinants because it uniquely determines description and unit price. But it is not a candidate key by itself because it doesn't uniquely determine Order# or Qty, leading to the conclusion that the design of Order is not good.

In addition to testing the goodness of design, the functional dependencies (FDs) derived from business rules during the normalization process also help to break down un-normalized relations to normalized relations. For example, in the relation **Order** (Order#, Date, Cust#, Item#, Description, Qty, Unit Price), the functional dependencies Item# -> (Description, Unit Price) help to recognize **Item** (Item#, Description, Unit Price) as a separate relation. Similarly, the dependency (Order#, Item#) -> Qty indicates another relation that consists of the three attributes.

Normalization also can be used in a second approach to database design called the bottom-up approach, which starts with a single large relation schema, often called the *universal relation,* that consists of the set of all attributes. The set may consist of attributes from multiple sources, such as reports, forms, transactions, and data files. As a result, the universal relation schema tends to represent a larger number of entities than the number of entities in a schema resulting from the top-down approach. In this approach, normalized relations are constructed by identifying the relationship among individual attributes as the starting point. As noted by Elmasri and Navathe (2007, p. 336), "this method is not very popular in practice because it suffers from the problem of having to collect a large number of binary relationship among attributes as the starting point." It would be simpler to first break down such large relations to smaller relations by identifying the entities, and then to map the entities to relations using the top-down approach. The discussions here are aimed at relations resulting from such a top-down approach.

Formal algorithms have been developed to break down un-normalized relations to normalized relations. An algorithm to synthesize 3NF relations by combining individual functional dependencies of an un-normalized relation is provided by Bernstein (1976). A different decomposition algorithm to break down an un-normalized relation to BCNF relations is provided in several textbooks (Elmasri & Navathe, 2007, p. 388; Kifer et al., 2006, p. 219; Ullman and Widom, 2008, p.92). Application of these algorithms, however, requires that the designer provide an accurate list of all the functional dependencies in the relation. This could be a tedious task for large relations. Further, these algorithms may produce different results depending on the order in which the input is provided and, also, may result in more relations than are desirable (Umanath & Scamell, 2007).

Because normalization beyond BCNF (4^{th}, 5^{th}, and Domain-Key normal forms) is of less practical significance, those higher normal forms are not included in this discussion. Since the requirements for a relation to be in BCNF includes the requirements for first, second, and third normal forms, normalizing a relation could be taught without teaching the first three normal forms. However, these three normal forms, which are presented in almost all textbooks that discuss BCNF, help to bring sharper focus on the specific types of undesirable FDs that contribute to data redundancy.

Dependency Preservation

Third normal form (3NF), and hence 1NF and 2NF that are included in its common definitions, is of further importance because some un-normalized relations cannot be decomposed to BCNF relations that preserve all functional dependencies (Elmasri & Navathe, 2007, p. 395; Ullman & Widom, 2008, p. 94). However, any relation can be decomposed to 3NF relations that preserve all the functional dependencies (Biskup, Dayal, & Bernstein, 1979). Functional dependencies represent business rules that are constraints on the database. Therefore, if a functional dependency is not represented in any one of the decomposed relations, enforcing the corresponding business rule through other means becomes more difficult.

The lack of dependency preservation in decomposition of certain relations to BCNF is illustrated by the following relation schema that represents information on student internships.

INTERNSHIP (StudentId, Organization, SupervisorPhone, PerformanceRating)

A student is allowed to have only one internship with the same organization, but may have multiple internships if they are at different organizations. SupervisorPhone uniquely identifies the student's internship supervisor who is an employee of the sponsor organization. The business rules are represented by the following FDs:

fd1: (StudentId, Organization) -> SupervisorPhone

fd2: (StudentId, Organization) -> PerformanceRating

fd3: SupervisorPhone -> Organization

Relation INTERNSHIP is in 3NF because there are no functional dependencies between non-key attributes. However, Internship is not in BCNF because fd3 implies that SupervisorPhone is a determinant, but it is not a candidate key. The practical problem is that the relation allows the inconsistency of having two or more different organizations associated with the same phone number in different rows, thus violating fd3.

There are two possible decompositions of INTERNSHIP into BCNF relations:

Option 1: One option is to decompose INTERNSHIP to two relations, shown below:

ST_SUP (StudentId, SupervisorPhone, PerformanceRating)

SUP_ORG (SupervisorPhone, Organization)

This option preserves fd3, but it doesn't preserve fd1 and fd2. An effect of not preserving fd1: (StudentId, Organization) -> SupervisorPhone is that the set of two relations allows two different phone numbers for the same (StudentId, Organization) combination. Similarly, not preserving fd2: (StudentId, Organization) -> PerformanceRating allows two different PerformanceRatings for the same (StudentId, Organization) combination. Sample records in the two tables ST_SUP and SUP_ORG illustrate these problems.

ST_SUP	Student Id	SupervisorPhone	PerformacneRating
	112344	424-3151	4
	112344	233-4121	5

SUP_ORG	SupervisorPhone	Organization
	424-3151	Acme Inc
	233-4121	Acme Inc

A natural join of the two tables would show the student-organization combination (112233, Acme Inc) associated with two different phone numbers 424-3151 and 424-4121, violating the fd1, and two different PerformanceRatings 4 and 5, violating fd2.

Join of ST_SUP and SUP_ORG	Student ID	Organization	SupervisorPhone	PerformanceRating
	112344	Acme Inc	424-3151	4
	112344	Acme Inc	424-4121	5

Option 2: A second way to decompose INTERNSHIP to BCNF relations is:

ST_ORG (StudentId, Organization, PerformanceRating)

PHONE_ORG (SupervisorPhone, Organization).

These relation schemas preserve fd2 and fd3; but do not preserve
fd1: (StudentId, Organization)-> SupervisorPhone. Again, the effect of not preserving fd1 would be to allow two different SupervisorPhone numbers for the same student-organiztion combination, as explained above.

This option also suffers from a more serious problem of failing to produce a *lossless join* decomposition. That is, a natural join of any state of the two relations may fail to yield the original relation state. The join may result in additional spurious tuples. Since lossless join is considered to be an extremely important property (Umanath & Scamell, 2007, p. 366), option 1 is preferred. It should be noted that every un-normalized relation can be decomposed to 3NF relations or BCNF relations, which meet the lossless join condition (Biskup et al., 1979; Ullman & Widom, 2008).

In the case of un-normalized relations that cannot be decomposed to BCNF relations that preserve the functional dependencies, the designer needs to choose between the redundancy of the 3NF and the lack of dependency preservation in BCNF. Thus, in the current example, one needs to choose between two designs: 1) the 3NF relation INTERNSHIP that suffers from potential inconsistencies of having two or more organizations associated with the same phone number; 2) the BCF relations in ST_SUP and SUP_ORG, which suffer from problems in enforcing the functional dependencies fd1 and fd2. In such cases, understanding 3NF and lower normal forms that are included in its common definitions is important to the designer. The problem due to lack of dependency preservation may be minimized by using materialized views that store data separate from the base tables and refresh the data when the base tables change (Umanath & Scamell, 2007)

The problem with dependency preservation discussed above indicates the practical significance of applying the normalization theory along with the top-down approach to data modeling to produce well-designed relations, and to choose between designs that are not optimal. Next we address the lack of clarity in several areas related to data modeling and normalization.

Representing Repeating Values

A core concept in the definition of 1NF is “repeating values” or “multi-valued” attributes. Typical definitions include: 1) a table is in first normal form if it doesn’t have any multi-valued or composite attributes; and 2) a table is in first normal form if each value in a tuple is atomic. Tables with multi-valued attributes may result from converting legacy systems. such as COBOL files, and from incorrect mapping of entities with multi-valued attributes. Though the concept of multi-valued attributes (or, repeating values) is simple, the lack of clarity in representing repeating values could be a source of confusion. Attributes with repeating values commonly are illustrated in textbooks by showing data (the relation state) along with relation schema. For example, the table EMPLOYEE with a multi-valued composite attribute, skills (skill_id, skill_level), would be represented as shown in Table 2. Skill_id and Skill_level represent the ID and level of an employee’s skills, respectively.

Table 2: EMPLOYEE with repeating values; not in 1NF			
Emp_Id	Emp_Name	Skill_id	Skill_level
10110	John Smith	22	1
		25	3
10120	Mark Adams	15	2
		22	4

Although the use of data along with the schema identifies multi-valued attributes, very few text-books present any method to identify a multi-valued attribute using schema alone. Consider the schema shown above without the data:

EMPLOYEE (Emp_Id, Emp_Name, Skill_id, Skill_level)

Given the above schema (with no data), and the business rule that an employee may have multiple skills, students have difficulty deciding whether EMPLOYEE is a table with repeating values (not in 1NF) as shown in Table 2, or whether it is a relation in 1NF but not in second normal form, as shown in Table 3. To make this decision with no additional information in the schema, one needs to know how the data is represented. If the data is represented as shown in Table 2, then Emp_Id would be a key and Skill_id and Skill_level are repeating values. On the other hand, if the data is represented as shown in Table 3, then the combination of Emp_Id and Skill_Id would be the key and Skill_id and Skill_level are not multi-valued. In this case, EMPLOYEE is in 1NF, but not in 2NF.

Table 3: EMPLOYEE in 1NF			
Emp_Id	Emp_Name	Skill_id	Skill_level
10110	John Smith	22	1
10110	John Smith	25	3
10120	Mark Adams	15	2
10120	Mark Adams	22	4

As relations often are represented using schema without showing the data, it is important that the schema clearly distinguishes multi-valued attributes from single-valued attributes. The distinction between the schemas represented in Table 2 and Table 3 can be made clear without using data, by employing a notation like braces {} to represent multi-valued attributes (Elmasri & Navathe, 2007).

EMPLOYEE (Emp_Id, Emp_Name, {Skills(Skill_id, Skill_level)})

A simplified version would be:

EMPLOYEE (Emp_Id, Emp_Name, {Skill_id, Skill_level})

The use of braces indicates that a tuple may have multiple Skill_id and Skill_level. On the other hand, the representation of EMPLOYEE without braces,

EMPLOYEE (Emp_Id, Emp_Name, Skill_id, Skill_level),

indicates that each tuple has only one Skill_id and Skill_level.

Almost all textbooks do not provide any method to represent multi-valued attributes in the schema.

It should be noted that specifying the key(s) and the functional dependencies would help students distinguish between tables in 1NF and those not in 1NF. For example, given the business rule that an employee may have multiple skills, EMPLOYEE (Emp_Id, Emp_Name, Skill_id, Skill_level) is not in 1NF because Skill_id and Skill_level are multi-valued attributes. On the other hand, EMPLOYEE (Emp_Id, Emp_Name, Skill_id, Skill_level) is in 1NF because each tuple has only one value for Skill_id and Skill_level. However, if the intent is to let students determine FDs and keys based on business rules, it defeats the purpose to provide such information.

Tables, Relations, and First Normal Form

The definitions of the terms tables, relations, and first normal form are of fundamental importance to teaching normalization. The lack of clarity in defining these terms in textbooks is a potential source of confusion for students of database design. One group of textbooks defines normal form using the term table, and another group defines normal form using the term relation, which is more consistent with mainstream database literature. Inaccuracies are prevalent in defining normal forms using both these terms.

Some textbooks that use the term relation define relation as a table with the following characteristics: 1) Each tuple (row) represents an instance of an entity or relationship, that is, there are no duplicate tuples; or, in other words, a relation has at least one key, 2) Each value in a tuple is atomic, that is, there are no multi-valued or composite attributes, 3) The order of tuples and attributes is not significant. This definition is consistent with the database literature. But, these books then define first normal form as follows: *a relation is in first normal form if each value in a tuple is atomic*. The above definitions of relation and first normal form are somewhat confusing. If the definition of a relation includes the characteristics of having atomic values, then any table that is a relation is already in first normal form. There is no need to impose the requirement of atomic values again on a relation for it to be in 1NF. As stated by Elmasri and Navathe (2007, p. 358), "First normal form (1NF) is now considered to be part of the definition of a relation in the basic (flat) relational model." Date (2004) points out that in mathematics, a relation need not be in 1NF, but in the relational model, a relation by definition is in 1NF. Student confusion could be reduced by making it clear that the term relation, as defined above, is in 1NF because a relation has only atomic values. Some books use the term "un-normalized relation" to refer to relations that have non-atomic values. Again, this is not consistent with the database literature.

Other textbooks define relation as a table with columns and rows. Then, the first normal form is defined the same way as stated above: a relation is in first normal form if each value in a tuple is atomic. Without a more formal definition of table, the combination of the above two definitions fails to specify an important property of a relation in first normal form: that there are no duplicate tuples, that is, it has at least one key. These definitions would classify the table shown in Table 4 as a relation in 1NF, though it doesn't qualify as a relation because it has duplicate tuples that represent two different students with the same name, John B. Adams.

Table 4: A Table not in First Normal Form

Course#	Student Name	Grade
Bus 311	John B. Adams	B
Bus 311	Sarah Mueller	A
Bus 311	John B. Adams	B

A couple of books attempt to specify this requirement by stating that for a relation to be in 1NF, each attribute must be dependent on the primary key. This statement is confusing because a primary key is not a primary key if each attribute is not dependent on it. A better way of stating this is that the relation must have at least one key.

A third group of textbooks uses the term table to define normal forms inaccurately. A typical definition is: a table is in first normal form if each value is atomic. Again, this definition fails to specify the property that a relation in 1NF has no duplicate tuples. Applying this definition would lead to the wrong conclusion that Table 4 is in 1NF.

To eliminate the confusion and make the definitions consistent with the literature, a relation should be defined as a table with the three major properties specified earlier: each value is atomic, there are no duplicate tuples, and the order of tuples and attributes is not significant. In addition, it should be stated that a relation, by virtue of its property that it has only atomic values, is in first normal form.

Normalizing Relations

This section identifies the problem students encounter in normalizing relations when they use the approach commonly presented in textbooks to identify candidate keys. A simplified approach to identify candidate keys is presented.

BCNF: Conventional Method

A relation is in *BCNF,* if and only if, every determinant is a candidate key. A *determinant* is any set of attributes on which another set of attributes is fully functionally dependent. A set of attributes Y is *fully functionally dependent* on another set of attributes X if it is functionally dependent on X and not functionally dependent on any subset of X. A set of attributes Y is *functionally dependent* on another set of attributes X, that is, X -> Y, if for every valid instance of X, the values of X uniquely determine the values of Y.

A set of attributes X is a candidate key if all other attributes of the relation are fully functionally dependent on X. That is, X uniquely determines all other attributes.

A common method used in textbooks to check whether a relation is in BCNF is to identify the determinants and verify that every determinant is a candidate key. An area that is particularly difficult for students is identifying the candidate keys, because the standard definition of candidate key, presented earlier, involves checking whether all other attributes of the relation are dependent on it.

Consider the un-normalized relation, ORDER, similar to the one discussed earlier under the top-down approach:

ORDER (OrderNo, OrderDate, CustNo, PmtMethod, ItemNo, ItemName, UnitPrice, OrderQty)

UnitPrice represents the standard price per unit, which is assumed to be the same for a product in all orders. ItemName is not unique. PmtMethod represents the single method of payment for an order.

When students are provided with the business rules and asked to normalize relations like ORDER and explain their rationale, their answers and comments indicate that it is relatively easier for them to identify the determinants than to identify the candidate keys. Identifying a determinant is simpler because, to be a determinant, only one attribute needs to be dependent on it. For example, OrderNo is identified as a determinant typically by noting that there is only one OderDate associated with an OrderNo, that is, OrderDate is dependent on OrderNo. Similarly, the single

dependency (OrderNo + ItemNo) -> OrderQty establishes that (OrderNo + ItemNo) is a determinant.

Student comments, however, show that they are more confused about identifying the candidate key(s), and they make more mistakes because it involves checking whether each attribute of the relation is functionally dependent on a determinant. Thus, to identify (OrderNo + ItemNo) as a candidate key, students need to establish seven dependencies involving seven attributes, such as (OrderNo + ItemNo) -> PmtMethod and (OrderNo + ItemNo) -> UnitPrice. The larger the number of dependencies to check, the more time-consuming and complex the process becomes, increasing the chance for errors.

A simpler method to identify candidate keys is presented. This method is then used to simplify the process of checking whether a relation is in BCNF.

A Simpler Method to Identify Candidate Keys

The proposed approach is based on the assertion that, after the determinants are identified, checking whether a determinant is a candidate key doesn't have to involve all the attributes of a relation. The transitivity property of functional dependency (Armstrong, 1974) states that:

> If A -> B, and B -> C, then A -> C

Thus if a determinant uniquely determines all other determinants, then the determinant uniquely determines all other attributes of a relation. That is, *a determinant is a candidate key if it uniquely determines all other determinants*.

An equivalent definition is: *a determinant is a candidate key if all other determinants are functionally dependent on it*. Using this property to identify candidate keys eliminates the need to check the dependency between a determinant and all other attributes of the relation. It is necessary to check only the dependency between a determinant and all other determinants, thus reducing the number of attributes to work with.

Using the proposed definition to verify that (OrderNo + ItemNo) is a candidate key, it is necessary to check only whether the other two determinants, OrderNo and ItemNo, are functionally dependent on (OrderNo + ItemNo). That is, verify that

> (OrderNo + ItemNo) -> OrderNo, and
>
> (OrderNo + ItemNo) -> ItemNo.

Both are trivial dependencies since the attribute on the right side is a subset of the attributes on the left side, leading to the conclusion that {OrderNo, ItemNo} is a candidate key. Thus the proposed definition eliminates the need to check whether all other attributes are dependent on (OrderNo + ItemNo).

Similarly, for OrderNo to be a candidate key, ItemNo and (OrderNo + ItemNo) must be dependent on OrderNo. That is, we need to check whether

> OrderNo -> (?) ItemNo, and
>
> OrderNo -> (?) (OrderNo + ItemNo).

Because the first dependency does not hold in this relation, OrderNo is not a candidate key, though it is a determinant.

It should be noted that even when using the conventional definition of a candidate key, one could show that a determinant like OrderNo is not a candidate key, by examining the single dependency OrderNo -> ItemNo that doesn't hold. But, the advantage of using the proposed definition that uses only determinants is that it helps students to focus their attention on determinants, so that

they don't have to decide which one of the entire set of attributes should be checked for lack of dependency. For example, using the conventional definition of candidate keys, students might examine any one of the dependencies (OrderNo -> UnitPrice), (OrderNo -> PmtMethod), (OrderNo -> OrderQty), etc., in any order. Because determinants are typically identifiers of entities (or relationships between entities), it would be easier to examine the dependency between two determinants than the dependency between a determinant that is the identifier of one entity and an attribute of another entity. For example, it would be easer to check the direct dependency OrderNo -> ItemNo than the indirect dependency OrderNo -> UnitPrice, which is based on two relationships OrderNo to ItemNo, and ItemNo to UnitPrice.

Alternate Definition of BCNF

The definition of candidate key presented earlier states that a determinant is a candidate key if all other determinants are functionally dependent on it. Applying this definition to BCNF yields an alternate definition of BCNF:

A relation is in BCNF if every determinant determines every other determinant; that is, if every determinant is functionally dependent on every other determinant.

In the special case when a relation has only one determinant, then the determinant is a candidate key, because every relation, by definition, has at least one candidate key. Hence, a relation with only one determinant is in BCNF.

In summary, *a relation is in BCNF if*

1) *it has only one determinant, or*
2) *it has multiple determinants, and every determinant is functionally dependent on every other determinant*.

Thus, to test whether ORDER is in BCNF, it is necessary to consider only the dependency between the three determinants:

1) OrderNo, 2) ItemNo, and 3) {OrderNo, ItemNo}.

Though, as discussed earlier, OrderNo and ItemNo are dependent on (OrderNo + ItemNo), there is no functional dependency between OrderNo and ItemNo, leading to the conclusion that ORDER is not in BCNF.

Multiple Candidate Keys

An additional example is presented to illustrate the application of the proposed approach when there are multiple candidate keys. Consider a normalized relation, CLASS, which stores information on each class offered in a semester in a university:

CLASS (ClassId, Course#, Sec#, Bldg, Room#, Day, Time, FacultyId, #OfStudents).

ClassId is a unique identifier of each class offered. It is assumed that, for a given class, there is only a single value for each of the attributes Course#, Sec#, Bldg, Room, Day, Time, FacultyId, and #OfStudents.

ClassId could be identified as a determinant based on a single dependency like ClassId->Course#. Similarly, (Course# + Sec#) could be identified as a key based on a dependency like (Course#+Sec#) -> ClassId. The test for BCNF could be done using either its standard definition or the proposed definition. Applying the standard definition requires checking whether ClassId and (Course# + Sec#) each determine all other attributes. To use the alternate definition, it is necessary to verify only that ClassId -> {Course# + Sec#}, and that {Course# + Sec#} -> ClassId.

Because both dependencies hold, CLASS is in BCNF. Again, the simplified approach does not require verifying that all attributes are dependent on each determinant.

Insertion and Deletion Anomalies

Discussions on normalization in textbooks and other literature often give students the impression that data redundancy in un-normalized relations leads to all three types of anomalies: insertion, deletion, and update. A typical statement is: "Data redundancy and consequent modification (insertion, deletion, and update) anomalies can be traced to "undesirable" functional dependencies in a relational schema" (Umanath & Scamell, 2007, p. 345). The three anomalies are commonly defined as:

1. Insertion anomaly: Information on an instance of one entity cannot be inserted without inserting a tuple representing an instance of a related entity. This is the most common description of insertion anomaly.
2. Deletion anomaly: information on one entity cannot be deleted under certain situations without losing information on another entity.
3. Update anomaly: a change in the value of an attribute may have to be made in multiple tuples.

Although data redundancy in relations always results in update anomalies, some un-normalized relations that have data redundancy do not suffer from insertion anomaly, or from deletion anomaly.

Consider the relation AUTHOR, shown in Table 5, which keeps information on authors and books for a publishing company:

Table 5: AUTHOR

Contract#	AuthorID	AuthorName	BookId	BookTitle	Royalty
10001	1010	A. Adams	10010	Intro Database	4%
10002	1020	B. Brown	10010	Intro Database	5%
10003	1020	B. Brown	10020	Intro Java	8%

The relationship between authors and books is assumed to be many-to-many. The royalty may vary with the author and the book. The combination of AuthorId and BookId is a candidate key. A contract# is assigned for each author and book combination to serve as a surrogate key.

AUTHOR is clearly not in BCNF. Both AuthorName and BookTitle are stored redundantly. This results in update anomaly, as expected, making it necessary to make any changes in AuthorName and BookTitle in multiple tuples. However, the relation AUTHOR does not suffer from insertion anomaly as defined in 1(a), or from deletion anomaly, if Contract# is selected as the primary key. Information on an author can be inserted without inserting information on any book written by the author, by assigning a contract#, so that the book information can be added later. The same is true about inserting author information without inserting book information. Similarly, the information on the single author of a book can be deleted without deleting the information on the book, since Contract# is the primary key.

Consider a second example involving a relation CLASS, shown in Table 6, which contains information on courses and sections, with a one-to-many relationship between them.

Table 6: CLASS					
ClassId	Course#	Course Name	Section#	Room#	FacultyId
1001	Bus211	Intro MIS	01	Clow 241	403355
1002	Bus211	Intro MIS	02	Clow 105	403355
1003	Bus315	Database	01	HS 107	403355

The un-normalized relation suffers from data redundancy. However it does not suffer from insertion or deletion anomalies. Deleting the single section of the course Bus315, for example, doesn't require deleting the Course# and Course Name for Bus315 since the surrogate key, ClassId, is the primary key. Similarly, a course without sections can be inserted without any problem.

Insertion and deletion anomalies also may be encountered in relations that do not suffer from data redundancy or update anomalies (Philip, 2002). Consider a schema that shows the current assignment of faculty members to office rooms:

FACULTY_OFFICE (Fac_id, Fac_name, Fac_dept, Office_Id, Bldg, Room#, SqFt).

A faculty member is assumed to have zero or one office, and an office is assigned to zero or one faculty member, resulting in a (zero-or-one)-to-(zero-or-one) relationship. Fac_name and Fac_dept are single valued attributes of faculty. Bldg, Room#, and SqFt are single valued attributes of office. As shown by Philip (2002), such a table meets the requirements of a relation.

FACULTY_OFFICE does not suffer from duplication of data or resulting update anomalies, because the relationship between office and faculty is 1:1. However, if Fac_id, for example, is selected as the primary key, then information on offices that are not assigned to any faculty cannot be inserted into the relation, resulting in insertion anomaly. Similarly, if a faculty member leaves, his/her information cannot be deleted without losing the information on his/her office, if any, resulting in deletion anomaly. The same anomalies exist if Office_Id, or the combination of (Fac_id + Office_id), is selected as the primary key.

Student confusion can be minimized by making it clear that though data redundancy generally results in insertion and deletion anomalies, that is not always the case. When a surrogate key is used in place of a composite key, data redundancy may not result in insertion and deletion anomalies, as in the examples shown above. Similarly, insertion and deletion anomalies may be encountered in relations that do not suffer from update anomalies, when a relation represents two entities with a 1:1 relationship.

Weak Entities

The term weak entity often is used to refer to a certain type of entity to facilitate prescribing rules to map them to relations. The definition of weak entity as presented in many textbooks, however, could be an area of confusion for students and faculty. Two common definitions of weak entity found in textbooks are:

Definition 1: *A weak entity is an entity that doesn't have a complete identifier of its own*. The identifier of the weak entity is formed by combining the identifier of another entity with attribute(s) of the weak entity. An entity that doesn't have a complete identifier of its own is also called an ID-dependent entity.

Definition 2: *A weak entity is an entity whose existence depends on some other entity*. This definition is commonly based on the logical dependence of an entity on another entity. For example, Hoffer et al. (2007, p. 98) state: "the weak entity has no business meaning in an E-R diagram."

Pratt and Adamski (2008, p. 216) state: "…because an order cannot exist without a customer, the relationship between customers and orders is existence dependency." It should be noted that an order typically has its own identifier (an order number); but this statement considers order to be a weak entity because it is logically dependent on a customer. Statements by Kifer et al. (2006, p. 84) support this view: "Sometimes designers choose to strip weak entity types of their key attributes and have the entities identified through their relationship with the master entity…. For example, the DEPENDENT entity might have the Name attribute, but not the SSN attribute." These statements indicate that the definition of weak entity based on logical dependence allows its own complete identifier (SSN) for a weak entity (DEPENDENT), thus contradicting Definition 1.

To further illustrate that ID-dependence and logical dependence are not the same, consider two entities, APPLICANT and APPLICATION with a 1:Many relationship, as shown in Figure 1. Application# is a key for APPLICATION.

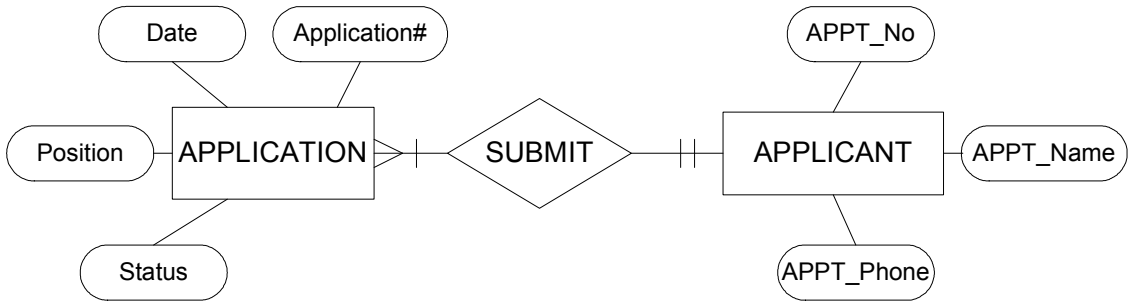

Figure 1: Strong/weak Entities

The first definition of weak entity would not classify APPLICATION as a weak entity because APPLICATION has its own identifier, Application#. On the other hand, the second definition, which generally is based on the logical dependence of an entity on another entity, would classify APPLICATION as a weak entity, because an application cannot logically exist without an applicant.

All logically dependent entities are not ID-dependent, as shown in the above example where APPLICATION is logically dependent on APPLICANT, yet it is not ID-dependent. But, all ID-dependent entities are logically dependent on another entity because an ID-dependent entity depends on another entity to have an identifier. For example, an alternate version of APPLICATION that doesn't have the identifier Application# wouldn't have a complete identifier of its own, and therefore it would be classified as an ID-dependent entity. In this case, the identifier of APPLICATION would include the identifier of APPLICANT, and therefore, APPLICATON would be logically dependent on APPLICANT.

A significant majority of the textbooks use definition 1 (ID-dependency) for weak entity. However, most others continue to use definition 2 (logical dependency), which is an earlier definition of weak entity. It should be noted that a very small number of books use the term existence dependence, but describe the term essentially as ID-dependence. It is important to note that a major reason for defining weak entity is to help specify mapping rules that are unique to ID-dependent entities, like how to form the primary key by combining the key of the strong entity(ies) with attribute(s) of the weak entity. These special rules do not apply to entities like "Order" that are only logically dependent and not ID-dependent on another entity like "Customer." Therefore, a weak entity should be defined as an ID-dependent entity, as done in a majority of the books, to bring more consistency. If the term weak entity is used to represent logically dependent entities, then it should be made clear that the mapping rules unique to ID-dependent entities do not apply to all logically dependent entities.

Review of Textbooks

Table 7 presents a summary of the treatment of four areas in each textbook that was considered in this study: representation of multivalued attributes, definition of relation, definition of 1NF, and description of weak entity. Some areas discussed earlier in this paper are not included in Table 7. The alternate definitions of candidate key and BCNF provided in this paper are not presented in any of the books. So, separate data on individual books on this topic is not included in Table 7. Similarly, none of the books discuss relations in which data redundancy may not lead to insertion and deletion anomalies, or relations that suffer from insertion and deletion anomalies without any data redundancy. These areas also are not included in Table 7.

Table 7: Summary of Textbook Reviews

Textbook (Authors)	Multivalued Attributes Represented by:	Definition of Relation	Definition of First Normal Form	Weak Entity Described as
Connolly & Breggs	Schema+data	Accurate	Confusing	Existence/ID dependence*
Date	None	Accurate	Accurate	Existence/ID dependence
Elmasri & Navathe	Schema alone	Accurate	Accurate	ID dependence
Frost & Van Slyke	Schema+data	Not Provided	Incomplete	ID dependence
Gillenson	Schema+data	Accurate	Incomplete	Not defined
Hoffer et al	Schema+data	Accurate	Confusing	Logical dependence
Kifer et al.	None	Accurate	Accurate	Logical dependence
Kroenke	None	Accurate	Accurate	Logical dependence
Mannino	Schema+data	Not Provided	Accurate	ID dependence
Post	Schema+data	Incomplete	Incomplete	Not defined
Pratt & Adamski	Schema alone	Accurate	Confusing	Existence Dependence**
Riccardi	None	Incomplete	Confusing	ID dependence
Rob & Coronel	Schema+data	Incomplete	Accurate	ID dependence
Rob & Semaan	Schema+data	Not Provided	Confusing	ID dependence
Ullman & Widom	None	Accurate	Confusing	ID dependence
Umanath & Scamell	Schema+data	Accurate	Confusing	ID dependence
Watson	None	Incomplete	Incomplete	ID dependence

* The term existence dependency is used in the book to represent ID-dependence.
** Not clear whether existence dependence means logical or ID dependence.

Summary

This paper showed that several database design concepts and techniques are commonly presented inaccurately or ambiguously in textbooks and are problematic for students. These areas include representation of repeating values, definitions of table, relation, and first normal form, identifying candidate keys, the relationship between data redundancy and insertion/deletion anomalies, and the definition of weak entities. The paper described the source of the problems and discussed simple solutions to make concepts more clear and techniques more efficient.

References

Armstrong, W. (1974). Dependency structure of data base relationships. *Proceedings of the IFIP Congress*.

Bernstein, P. A. (1976). Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems, 1*(4), 277-298.

Biskup, J., Dayal, U., & Bernstein, P. A. (1979). Synthesizing independent database schemas. *ACM SIG-MOD International Conference on Management of Data*, 143-152.

Connolly, T., & Begg, C. (2005). *Database systems: A practical guide to design, implementation, and management*. Reading, MA: Addison Wesley.

Date, C. J. (2004). *An introduction to database systems*. Reading, MA: Addison-Wesley.

Elmasri, R., & Navathe, S. (2007). *Fundamentals of database systems.* Reading, MA: Addison-Wesley.

Frost, R., & Van Slyke C. (2006). *Database design and development: A visual approach*. Upper Saddle River, NJ: Prentice Hall.

Gillenson, M. L. (2005). *Fundamentals of database management systems*. Danvers, MA: Wiley.

Hoffer, J. A., Prescott, M. B., & McFadden, F. R. (2007). *Modern database management*. Reading, MA: Addison-Wesley.

Kifer, M., Bernstein, A., & Lewis, P. (2006). *Database systems: An application-oriented approach.* Reading, MA: Addison-Wesley.

Kroenke, D.M. (2006). *Database processing: Fundamentals, design, and implementation*. Upper Saddle River, NJ: Prentice Hall.

Mannino, M.V. (2007). *Database: Design, application development, and administration*. New York: McGraw-Hill.

Philip, G. (2002). Normalizing relations with nulls in candidate keys. *The Journal of Database Management*, *13*(3), 35-45.

Post, G. V. (2005). *Database management systems*. New York: McGraw-Hill.

Pratt, P., & Adamski, J. (2008).*Concepts of database management*. Boston: Course Technology.

Riccardi, G. (2003). *Database management with web site development applications*. Reading, MA: Addison Wesley.

Rob, P., & Coronel, C. (2006). *Database systems: Design, implementation, & management*. Boston: Course Technology.

Rob, P., & Semaan, E. (2004). *Database: Design, development & deployment*. New York: McGraw-Hill.

Ullman, J., & Widom, J. (2008). *A first course in database systems*. Upper Saddle River, NJ: Prentice Hall.

Umanath, N. & Scamell, R. (2007). *Data modeling and database design*, Boston: Course Technology.

Watson, R. T. (2005). *Data management: Databases and organizations*. Danvers, MA: Wiley.

Biography

Dr. **George C. Philip** is a professor in Information Systems in the College of Business, University of Wisconsin – Oshkosh. He obtained his Ph. D. from the University of Iowa. His work experience includes seven years as Director of M.S. in Information Systems program and Team Leader of MIS program. His areas of publication and teaching include Business Intelligence, software design and development, and database design. He teaches seminars and provides consulting services in these areas.