A TUTORIAL ŒJ RELATIONAL DATABASES USING STRUCTURAL ENGINEERING EXAMPLES

by

H.J. Schaafer, D.R. Rshak 4 S.J. Fsnves

Dasambar, 1^82

DRC-12-09-82

A TUTORIAL ON RELATIONAL DATABASES USING STRUCTURAL ENGINEERING EXAMPLES

By Michael J. Schaefer[1], Daniel R. Rehak,[2]
and Steven J. Fenves,[3] M.ASCE

KEYWORDS:

Database; Relational database; Centralized control; Data integrity; Data independence; Information retrieval; Queries; Normalization; Normal forms; Partial dependencies; Transitive dependencies; Multi-valued dependencies; Join dependencies; Anomalies; Structural engineering; Engineering design; Civil engineering

ABSTRACT:

An introduction to the basic issues of database management and to the concepts of the relational database model are presented. Structural engineering examples are used to describe the principal relational operations for data retrieval. Examples of an available relational data retrieval language developed for a current database management system are provided. Also, the process of normalization is discussed, which includes the description of six different normal forms, to provide guidelines for database design.

---

[1] Research Assistant. Department of Civil Engineering. Carnegie-Mellon University. Pittsburgh. PA 15213.

[2] Assistant Professor of Civil Engineering. Carnegie-Mellon University. Pittsburgh. PA 15213.

[3] University Professor of Civil Engineering. Carnegie-Mellon University. Pittsburgh. PA 15213.

A TUTORIAL ON RELATIONAL DATABASES USING STRUCTURAL ENGINEERING EXAMPLES

By Michael J. Schaefer[1], Daniel R. Rehak,[2] and Steven J. Fenves,[3] M.ASCE

1. INTRODUCTION

During the design and construction of any civil engineering project, large amounts of information pertaining to all aspects of the project must be stored, accessed and operated upon. One way to store this information is in a database. A database offers two main advantages. First, it provides *centralized control* of the data [6], allowing all users to access the same data. Instead of each user having his own private files, which would probably store data similar to other user's files, all files can be integrated into one database, thus eliminating redundancy. Since the data are stored in only one place, centralized control also insures that the data are consistent. Once a piece of data is changed, all users access the most recent value. Centralized control also eases the process of insuring *data integrity,* that is, that all updates of the data are valid. Whenever an update is attempted, a predefined checking procedure can be invoked to determine the validity of the update.

A second advantage that a database provides is *data independence.* The way in which data is stored or accessed in the database is independent from its use. The user's view of the data is defined by the *conceptual schema* which provides the definition of information content only. No references to storage/access details are included in the schema [6]. The user need only specify which data items are needed, not how to find them or where to look for them. Therefore, even if one user needs the data in one form and another user needs the same data in a different form, the data is stored only once. It is then the database system's responsibility to supply the data in the appropriate form. The user specifies what data is needed through a retrieval language. An example of such language is shown in Section 3.2.

---

[1] Research Assistant Department of Civil Engineering. Carnegie-Mellon University, Pittsburgh. PA 15213.

[2] Assistant Professor of Civil Engineering. Carnegie-Mellon University. Pittsburgh. PA 15213.

[3] University Professor of Civil Engineering. Carnegie-Mellon University. Pittsburgh. PA 15213.

Most database systems offer access to the retrieval operations from two different modes. First, the *standalone* mode provides the user with interactive access to the data. The second mode is the *application program* mode. It supplies the retrieval operations as a data sublanguage which can be called by a host programming language in an application program.

One type of database is based on the relational model. A relational database stores information in a set of relations, which are basically two-dimensional tables. All information about each aspect of the project can be stored in a set of one or more relations. An example of a relational database, used throughout this paper, is shown in Figure 1-1. This database is made up of three relations; BEAMS, DESIGNATIONS and GRADES. The BEAMS relation is similar to a beam schedule for a construction project; it lists the LENGTH, DESIGNATION, GRADE of steel and quantity (QTY) of each specific shape and grade in the project. The DESIGNATIONS relation is similar to the structural shape tables in the AISC manual [1] and contains the DESIGNATION, SECTION MODULUS, AREA, moment of inertia (I) and depth (D) for each shape. Finally, the GRADES relation lists the GRADE and yield stress (FY) for different grades of steel.

Since, as stated by Codd [5], "relational database technology offers dramatic improvements in productivity", relational databases are of great interest to the developers of engineering design databases. Therefore, using examples similar to the database in Figure 1-1, this article will provide an introduction to the relational database approach by discussing some of the basic concepts involved, including guidelines for database design through normalization.

2. NOMENCLATURE

This section will describe a few of the basic terms in the relational database model.

The columns of the relations in Figure 1-1, such as LENGTH, DESIGNATION, SECTION MODULUS and GRADE, are called *attributes.* The actual values for each attribute are taken from the *domain* of the attribute. A domain is the set of all allowable values for a specific attribute. For example, the quantity (QTY) attribute in the BEAMS relation (Figure 1-1) has for its domain all integers greater than zero. In any given relation, the ordering of the columns is arbitrary.

BEAMS

LENGTH	DESIGNATION	GRADE	QTY
20	W36X300	A36	15
20	W36X300	A514	9
20	W33X241	A514	5
20	W33X241	A36	5
40	W30X211	A588	8
40	W30X211	A242	4
20	W27X114	A36	2
35	W16X57	A36	3
20	W27X114	A514	7

DESIGNATIONS

DESIGNATION	SECTION MODULUS	AREA	D
W36X300	1110	88.3	20300136.74
W33X241	829	70.9	14200134.18
W30X211	663	62.0	10300130.94
W27X114	299	33.5	4090|27.29
W16X57	92.2	16.8	758 116.43

GRADES

GRADE	Fy
A36	36
A514	100
A588	50
A242	50

Figure 1-1: Example of a Relational Database.

The individual rows of a relation are called *tuples.* No tuple is duplicated within any relation. A tuple thus represents some unique object or entity with its properties (attribute values). As is the case with domains, the ordering of the tuples within a relation is also arbitrary.

In each relation, a group of one or more of the attributes uniquely identifies each tuple. This group of attributes is called the *key* for the relation. The key for the BEAMS relation in Figure 1-1 is the attribute combination of LENGTH, DESIGNATION and GRADE: with these attributes specified, a specific tuple is uniquely determined. For the DESIGNATIONS relation, the key is DESIGNATION. The key for the GRADES relation is GRADE.

The overall configuration of the database is referred to as the *schema.* The schema is the layout of the relations themselves, without the data. The description of an individual relation is called the *relation schema.* Once all relations are described, such as the attribute names and types in each relation, the database schema is completely defined.

A *constraint* is a tool used to control the integrity of a relation. As stated above, the quantity (QTY) attribute in the BEAMS relation must be greater than zero. Therefore, the following constraint is specified:

QTY > 0

Also, if the value of one attribute is dependent upon another, a constraint can be specified to insure this dependency. For example, if a relation contains the height, width and area of a solid rectangular beam, the user may want to specify the following constraint:

AREA = WIDTH X HEIGHT

If a user tries to change one of these attribute values for a tuple without changing any other value, the database system would disallow the change and warn the user of the constraint violation.

3. DATA RETRIEVAL

Once a database is created, any authorized user must be able to access the data. Section 3.1 will discuss the three basic relational operators that support this process: *SELECT, PROJECT* and *JOIN.* Section 3.2 will demonstrate the implementation of these basic operators in a current relational database management system query

language.

The examples in this section show the DBMS commands in **BOLD FACE,** the relation names in UPPER CASE and the attribute names in lower case letters.

3.1. BASIC OPERATORS

This section presents the three basic relational operators. These three operators can be used individually or in any combination to retrieve data from the database. The PROJECT and JOIN operators will be used in Section 4 to describe the normalization process.

3.1.1. SELECT

The **SELECT** operator is used to produce a new relation from the rows of an existing relation for which a specified selection criterion is satisfied. The criterion is specified by attaching a boolean predicate to the **SELECT** command through the use of a **WHERE** clause. All tuples for which the **WHERE** clause evaluates to true are retrieved. For example, the following command forms a new relation from the BEAMS relation of Figure 1-1, containing all beams whose length equals 40 feet.

SELECT BEAMS **WHERE** length = 40

The result is:

LENGTH	DESIGNATION	GRADE	QTY
40	W30X211	1 A588	8
40	W30X211	I A242	4

3.1.2. PROJECT

The **PROJECT** operator is used to produce a new relation containing one or more of the columns of an existing relation. For example, the following command forms a new relation from the BEAMS relation in Figure 1-1, by eliminating the GRADE and quantity (QTY) columns.

PROJECT BEAMS **OVER** length **AND** designation

The result is:

LENGTH	DESIGNATION
20	W36X300
20	W33X241
40	W30X211
20	W27X114
35	W16X57

Notice that the duplicate tuples created by the projection are removed.

3.1.3. JOIN

The **JOIN** operator is used to produce a new relation by combining two or more relations over a comparison between attributes with common domains. Therefore, the relations to be joined must contain a common domain. For example, the following command forms a new relation containing the attributes LENGTH DESIGNATION, GRADE and quantity (QTY) from the BEAMS relation in Figure 1-1 and the attributes GRADE and FY from the GRADES relation in Figure 1-1.

JOIN BEAMS **AND** GRADES **OVER** grade

The result is:

LENGTH	DESIGNATION	GRADE	QTY	GRADE	Fy
20	W36X300	A36	15	A36	36
20	W36X300	A514	9	A514	100
20	W33X241	A514	5	A514	100
20	W33X241	A36	5	A36	36
40	W30X211	A588	8	A588	50
40	W30X211	A242	4	A242	50
20	W27X114	A36	2	A36	36
35	W16X57	A36	3	A36	36
20	W27X114	A514	7	A514	100

This is an example of an *equijoin;* the two relations are joined where the grade values are equal. Other types of joins are possible, such as, *greater than* and */ess than* joins. These joins combine the relations where a specific attribute in one relation is greater than (or less than) an attribute in another relation.

Also, notice that the equijoin produces a relation with two identical columns. This redundancy must be eliminated by projecting out one of the duplicate attributes. However, since the removal of the duplicate attribute is a very common operation after a join, a *natural Join* can be specified, which is the same as an equijoin with the duplicate attribute eliminated.

3.2. IMPLEMENTATION OF RELATIONAL OPERATORS

Many languages have been developed to ease the data retrieval operations for the inexperienced user. One such language is called SEQUEL (Structured English QUEry Language) [3]. After many extensions and improvements, SEQUEL 2 was developed [4] and implemented in the Database Management System (DBMS) entitled SYSTEM R [2]. This language is used in the following examples to query, manipulate, define and control the database in Figure **1-1.**

3.2.1. QUERYING THE DATABASE

The query facility of SEQUEL 2 allows the user to ask the database questions about the data. The **SELECT** command is used to specify which attributes are to be returned. The **FROM** command is used to specify the relation to use for the **SELECT** command. The **WHERE** command is used to compare attributes to values or to other attributes. The comparisons may be connected by the operators **AND** and **OR** to form more complex comparisons. Also, the result of a query may be used in the **WHERE** command of another query. The following examples are typical queries.

1. List the DESIGNATION, depth (D) and SECTION MODULUS of all beams with a depth less than 34 inches, sorted in order of increasing section modulus.

```
SELECT designation, D, section modulus
FROM DESIGNATIONS
WHERE D < 34
ORDER BY section modulus
```

The result is:

DESIGNATION	D	SECTION MODULUS
W16X57	16..43!	92.2
W27X114	27,.29!	299
W30X211	30.94!	663

The ORDER BY command orders the tuples of a relation by the attribute specified. However, the ordering is done specifically for the user and does not effect the actual storage of the relation, since the order of tuples is completely arbitrary.

2. List the average area of the 20 foot beams.

```
SELECT AVERAGE(area)
FROM DESIGNATIONS
WHERE designation IN
        SELECT designation
        FROM BEAMS
        WHERE length = 20
```

The result is:

AREA
64.2

This command is the same as the following list of operations using the basic operators, defined in Section 3.1.

- SELECT BEAMS **WHERE** (length.EQ.20)
- **JOIN** RESULT **AND** DESIGNATIONS **OVER** designation
- **PROJECT** RESULT **OVER** area
- Compute average area using the result of the above command.

3. Find all beams in the project made of A514 steel.

```
SELECT UNIQUE designation
FROM BEAMS
WHERE grade = 'a514'
```

The result is:

DESIGNATION
W36X300
W33X241
W27X114

Notice that the word 'unique' follows the select command. This is because SEQUEL 2 allows duplicate tuples and only removes them upon request.

4. Find the yield stress for all W36x300 beams in the project.

```
SELECT UNIQUE Fy
FROM GRADES
WHERE grade IN
        SELECT grade
        FROM BEAMS
        WHERE designation = 'W36x300'
```

The result is:

FY
36
100

5. List the DESIGNATION, SECTION MODULUS and the values of 2*I/D for all beams with a 2*I/D value less than 830. Notice that the derived value of 2*I/D is calculated by the database system.

```
SELECT designation, section modulus, 2*I/D
FROM beams
WHERE 2*I/D < 830
```

The result is:

DESIGNATION	SECTION MODULUS	2*I/D
W30X211	663	665.80
W27x114	299	299..74
W16X57	92.2	92..27

6. How many different beam designations are in the project?

SELECT COUNT (UNIQUE designation)
FROM BEAMS

The result is:

5

3.2.2. **MANIPULATION OF THE DATABASE**

The manipulation aspect of SEQUEL 2 allows the user to change, add and/or delete values in the database, as shown in the following examples.

1. Insert a new beam into the BEAMS relation with a length of 60 feet, a designation of W27x114 and a grade of A36, leaving the quantity null until it can be determined.

INSERT INTO BEAMS (length, designation, grade);
<60, W27x114, A36>

This command causes the system to create a new tuple in the BEAMS relation with the given data in the appropriate attribute columns.

2. Create a new relation called SMALL DEPTHS, with the same attribute names as DESIGNATIONS, that includes beams from the DESIGNATIONS relation that have depth values less than 30.

ASSIGN TO SMALL DEPTHS (designation, section modulus, area, **I, DE-**
SELECT designation, section modulus, area, I, D>
FROM DESIGNATIONS
WHERE D < 30

The following relation is created by this command:

SMALL DEPTHS

DESIGNATION	SECTION MODULUS	AREA	I	D
W27X114	299	33.5	4090	27.29
W16X57	92.2	16.8	758	16.43

3. Add to the SMALL DEPTHS relation all beams in the DESIGNATION relation with depths less than 34 (for this example it is assumed that the beams with depths less than 30 are already contained in the SMALL DEPTHS relation).

```
INSERT INTO SMALL DEPTHS
        SELECT (designation, section modulus, area, I, D)
        FROM DESIGNATIONS
        WHERE D < 34 AND D > 30
```

The result is:

SMALL DEPTHS

DESIGNATION	SECTION MODULUS	AREA	I	D
W27X114	299	33.5	4090	27.29
W16X57	92.2	16.8	758	16.43
W30X211	663	62.0	110300	30.941

4. Delete all beams in the SMALL DEPTHS relation with depths greater than 30.

```
DELETE SMALL DEPTHS
        WHERE D > 30
```

This command would return the SMALL DEPTHS relation to its original form.

5. Update the BEAMS relation by adding 10 feet to the beams with a length equal to 35 feet.

```
UPDATE BEAMS
SET length = length + 10
WHERE length = 35
```

This command changes the eighth tuple in the BEAMS relation to:

45	W16x57	A36	3

3.2.3. DATA DEFINITION

The data definition aspect of SEQUEL 2 allows the user to create, delete and/or change the structure of relations in the database.

1. The following command would be used to define (create) the DESIGNATIONS relation:

   CREATE TABLE DESIGNATIONS (designation(**CHAR**(7), **NONULL**),
   section modulus(**DECIMAL**(1)), area(**DECIMAL**(1)),
   I(**DECIMAL**(1)), D(**DECIMAL**(2)))

   The **CHAR**(n) specification means that the value for the appropriate attribute is always a character string of at most n characters. **NONULL** means that the attribute must always be specified for each tuple. **DECIMAL**(n) means that the value for the attribute is a real number with at most n decimal places.

2. Add an attribute (column) to the designation relation to store the web thickness (TW).

   EXPAND COLUMN TW(**DECIMAL**(3))

3. Delete the GRADE relation.

   DROP TABLE GRADE

3.2.4. CONTROL OF THE DATABASE

This aspect of SEQUEL 2 allows the users to control the access of their data to other users and to exercise control over the integrity of data values. Access is controlled by the **GRANT** and **REVOKE** commands. Data integrity is controlled by specifying assertions on the data, using the **ASSERT** command. Each assertion is given a name by the user who specifies it and is referenced by this name whenever it is violated.

1. Allow Smith complete access (read, insert and change) to the BEAMS relation, including the option to give access to someone else.

   GRANT READ, INSERT, UPDATE(length, designation, grade, qty)
   ON BEAMS **TO** Smith **WITH GRANT OPTION**

2. Disallow Smith's access to the BEAMS relation.

   REVOKE ALL RIGHTS ON BEAMS **FROM** Smith

The **ALL RIGHTS** command can always be substituted by a list of rights as seen in the above example.

3. Require that all quantities in the BEAMS relation are greater than zero.

```
ASSERT A1 ON BEAMS: qty > 0.0
```

4. Require all beam lengths in the BEAMS relation to be greater than 5 but less than 100.

```
ASSERT A2 ON BEAMS: length BETWEEN 5 AND 100
```

5. Require each beam in the BEAMS relation to have a designation equal to one of the designations in the DESIGNATIONS relation.

```
ASSERT A3:
    (SELECT designation FROM BEAMS)
    IS IN
    (SELECT designation FROM DESIGNATIONS)
```

4. NORMALIZATION

Normalization is a method for insuring that the organization of the database has certain desirable properties. It is a tool for determining the best possible arrangement of the data in the database. Six different normal forms will be discussed; *first, second, third fourth, fifth,* and *domain-key* normal form. The description of first, second, third, fourth and fifth normal forms is based on the discussion of normalization by Date [6].

4.1. FIRST NORMAL FORM

All relations in a relational database must be in first normal form (1NF). There are two reasons for this: first, storage is easier, and second, the data manipulation operators are not as complicated as they would be for unnormalized relations. A relation not in 1NF is called an unnormalized relation. Figure 4-1 is an example of an unnormalized relation.

The key for this relation, beam information, is made up of the sub-attributes LENGTH, DESIGNATION and GRADE. This relation is considered unnormalized, because not all rows and columns contain single values. Figure 4-2 shows an equivalent normalized relation. The key for this relation is a composite key consisting of LENGTH, DESIGNATION and GRADE. Notice that all values are simple, that is, they are all single valued.

BEAM INFORMATION			Fy	SECTION MODULUS	AREA	I	D	QTY
LENGTH	DESIGNATION	GRADE						
20	W36X300	A36 A514	36 100	1110	88.3	20300	36.74	15 9
20	W33X241	A514 A36	100 36	829	70.9	14200	34.18	5
40	W30X211	A588 A242	50 50	663	62.0	10300	30.94	8 4
35	W16X57	A36	36	92.2	16.8	758	16.43	3
20	W27X114	A36 A514	36 100	299	33.5	4090	27.29	2 7

Figure 4-1: Relation BEAM: an unnormalized relation.

LENGTH	DESIGNATION	GRADE	Fy	SECTION MODULUS	AREA	I	D	QTY
20	W36X300	A36	36	1110	88.3	20300	36.74	15
20	W36X300	A514	100	1110	88.3	20300	36.74	9
20	W33X241	A514	100	829	70.9	14200	34.18	5
20	W33X241	A36	36	829	70.9	14200	34.18	5
40	W30X211	A588	50	663	62.0	10300	30.94	8
40	W30X211	A242	50	663	62.0	10300	30.94	4
35	W16X57	A36	36	92.2	16.8	758	16.43	3
20	W27X114	A36	36	299	33.5	4090	27.29	2
20	W27X114	A514	100	299	33.5	4090	27.29	7

Figure 4-2: Relation Beami: first normal form.

This leads to the definition of a normalized relation:

> A relation whose attributes or tuples cannot be broken down into two or more attributes or tuples is considered normalized.

4.2. SECOND NORMAL FORM

After examining the relation in figure 4-2, one may observe that certain problems arise because of the *partial* dependency of the non-key attributes on the attributes in the composite key. As shown in figure 4-3, Fy is uniquely determined by GRADE. Also, SECTION MODULUS, AREA, I and D are uniquely determined by DESIGNATION. These partial dependencies exist even though the entire key of LENGTH, GRADE and DESIGNATION is necessary to identify a specific tuple. Notice also that section modulus is determined by I and D. This is called a transitive dependency and will be described in Section 4.3.

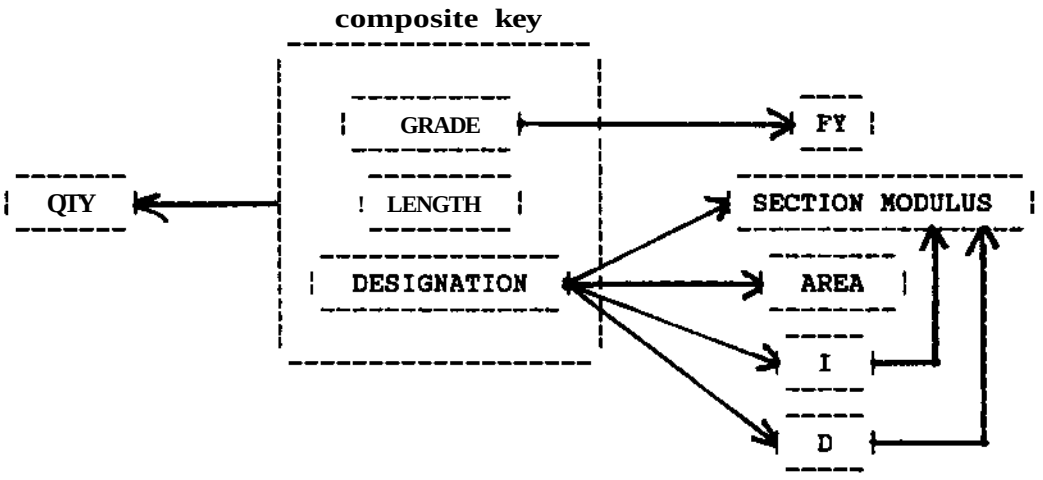

Figure 4-3: Dependencies in BEAM1 relation.

The partial dependencies cause the following problems:

- There is no way to store the yield stress (Fy) of a grade A529 steel since none of the beams are made of grade A529 steel.
- If the W16x57 beam is deleted, the relation no longer contains the I, D, area or section modulus for this shape.
- If it was discovered, after the data was entered, that the area of the W36x300 beam is not 88.3, then all tuples would have to be searched to find all occurrences of W36x300 to correct the error.

These problems are eliminated by creating three new relations from the BEAM1 relation. As shown in Section 3.1.2, this can be done by the PROJECT operator.

BEAM2

LENGTH	DESIGNATION	GRADE	QTY
20	W36X300	A36	4
20	W36X300	A514	2
20	W33X241	A514	5
20	W33X241	A36	5
40	W30X211	A588	8
40	W30X211	A242	4
20	W27X114	A36	2
35	W16X57	A36	3
20	W27X114	A514	7

DESIGNATION

DESIGNATION	SECTION MODULUS	AREA	I	D
W36X300	1110	88.3	20300	36.74
W33X241	829	70.9	14200	34.18
W30X211	663	62.0	10300	30.94
W27X114	299	33.5	4090	27.29
W16X57	92.2	16.8	758	16.43

GRADE

GRADE	Fy
A36	36
A514	100
A588	50
A242	50

Figure 4-4: BEAM2, DESIGNATION and GRADE relations: second normal form.

Figure 4-4 shows these new relations. With the database in this form, all of the previous problems are solved.

- Any beam designation or grade can be stored, even if those designations or grades are not currently in the BEAM2 relation.
- A particular beam occurrence can be deleted, without losing its related dimensions.
- Information about any designation or grade can be changed without having to search for multiple occurrences.

By eliminating the partial dependencies on the key, the relations in figure 4-4 are now considered to be in at least second normal form[4] (2NF). The definition of second normal form is then as follows:

> A 1NF relation is in second normal form if all attributes depend on the entire key.

4.3. THIRD NORMAL FORM

Problems can also arise in a relation in second normal form, due to the possibility of *transitive* dependencies of two or more non-key attributes. An example of a transitive dependency is shown in Figure 4-5 for the DESIGNATION relation in Figure 4-4.

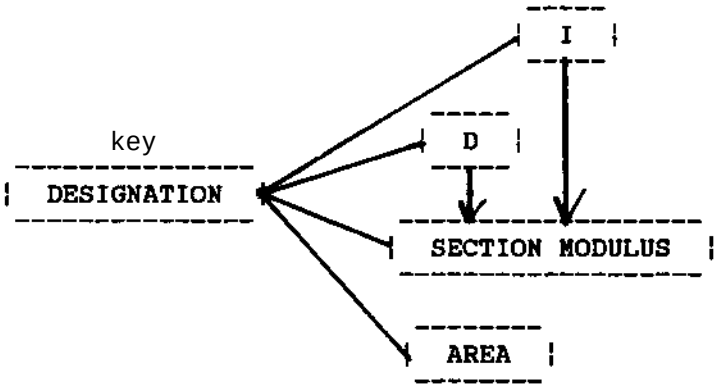

Figure 4-5: Transitive dependencies between I, D and section modulus.

Although all attributes depend on the key, DESIGNATION, another dependency exists between SECTION MODULUS, I and D, since, theoretically, section modulus equals I/(D/2).

---

[4] **Actually, the BEAM2 and GRADE relations *are* also in third and fourth normal forms.**

This kind of transitive dependency can cause problems similar to the ones in a relation in 1NF. However, in this case none of these problems are present. This is because a SECTION MODULUS relation containing SECTION MODULUS (as the key for the relation), I and D could not stand alone; two beam designations could have the same section modulus even if the I's and D's were different, thus forcing the key of SECTION MODULUS to be non-unique. Ignoring the lack of an actual problem in this particular relation, normalization would force it to be split, so that the transitive dependency is removed. Figure 4-6 shows two new relations that remove this dependency. These new relations are in at least third normal form.[5]

DESIGNSA

DESIGNATION	SECTION MODULUS	AREA
W36X300	1110	88.3
W33X241	829	70.9
W30X211	663	62.0
W27X114	299	33.5
W16X57	92.2	16.8

DESIGNID

DESIGNATION	I	D
W36X300	20300	36.74
W33X241	14200	34.18
W30X211	10300	30.94
W27X114	4090	27.29
W16X57	758	16.43

Figure 4-6: DESIGNSA and DESIGNID relations: third normal form.

This leads to the definition of third normal form.

> A 2NF relation is in third normal form if all attributes depend on the key and no other attribute.

[5] Actually, these relations are also in fourth normal form, as explained in Section 4.4.

4.4. FOURTH NORMAL FORM

A close look at the BEAM2 relation in figure 4-4 uncovers another problem. To provide a clearer example of this problem, a new relation similar to BEAM2 is created by removing the quantity attribute and changing a few tuples. The new relation (BEAM3) is *all key,* since all attributes must be specified to uniquely identify any specific tuple. Figure 4-7 shows the new relation.

BEAM3

LENGTH	DESIGNATION	GRADE
20	W36X300	A36
20	W36X300	A514
20	W33X241	A514
20	W33X241	A36
40	W30X211	A588
40	W30X211	A242
20	W27X114	A36
20	W27X114	A514
20	W30X211	A36
20	W30X211	A514

Figure 4-7: BEAM3 relation.

Assume that this relation is a list of all the possible types of beams that can be provided. This means that only 20 and 40 feet lengths are allowed: that a 20 foot beam can only have a grade of A36 or $A514_r$ and that the 40 foot beams can only be made of A588 or A242 steel. This dependency between LENGTH and GRADE is called a *multi-valued* dependency (MVD). A multi-valued dependency is similar to the transitive dependency: in a relation with a transitive dependency, one or more attributes uniquely determine the value of another attribute, while in a relation with a multi-valued dependency, one or more attributes determine a well-defined set of values for another attribute.

This multi-valued dependency causes the following two problems:

- There is a large amount of redundancy in this relation. This redundancy causes the same updating problem as before; if a designation is incorrect, all occurrences have to be searched and changed.

- If a new grade of steel can be provided for 20 foot beams, three new tuples must be entered, one for each distinct designation. This adds to the redundancy.

The MVD is eliminated by creating two new relations. The new relations are shown in Figure 4-8.

DESIGNATIONL

LENGTH	DESIGNATION
20	W36x300
20	W33x241
20	W27x114
40	W30x211
20	W30x211

GRADEL

LENGTH	GRADE
20	A36
20	A514
40	A588
40	A242

Figure 4-8: DESIGNATIONL and GRADEL relations: fourth normal form.

These relations are now considered to be in fourth normal form (4NF), which is defined in the following way:

> A 3NF relation is in fourth normal form if, when a multi-valued dependency exists, one attribute upon another, then all other attributes depend on this same attribute.

4.5. FIFTH NORMAL FORM

After examining the discussion on fourth normal form in the previous section, it could be decided to ignore the redundancy problem in the BEAM3 relation shown in Figure 4-7. If this is done, the BEAM3 relation will remain intact and the user has to be aware of the redundancy. This is a possible alternative, since any relation that is at least in first normal form is a valid relation. However, there is a different type of problem associated with a relation like the BEAM3 relation that could cause more than just a redundancy problem. This new problem arises from the *join dependency* of the attributes. To show this join dependency, assume that, at some time, the BEAM3 relation is broken into its three projections: DESIGNATIONL(containing the length and designation), GRADEL(containing the length and grade) and DESIGRADE(containing designation and grade). Then, at some other time, only two of these relations, such as DESIGNATIONL and DESIGRADE, are joined to form the original relation. Figure 4-9 shows that this process causes four new tuples to appear in the new BEAM3 relation (called NBEAM3) that did not appear in the original

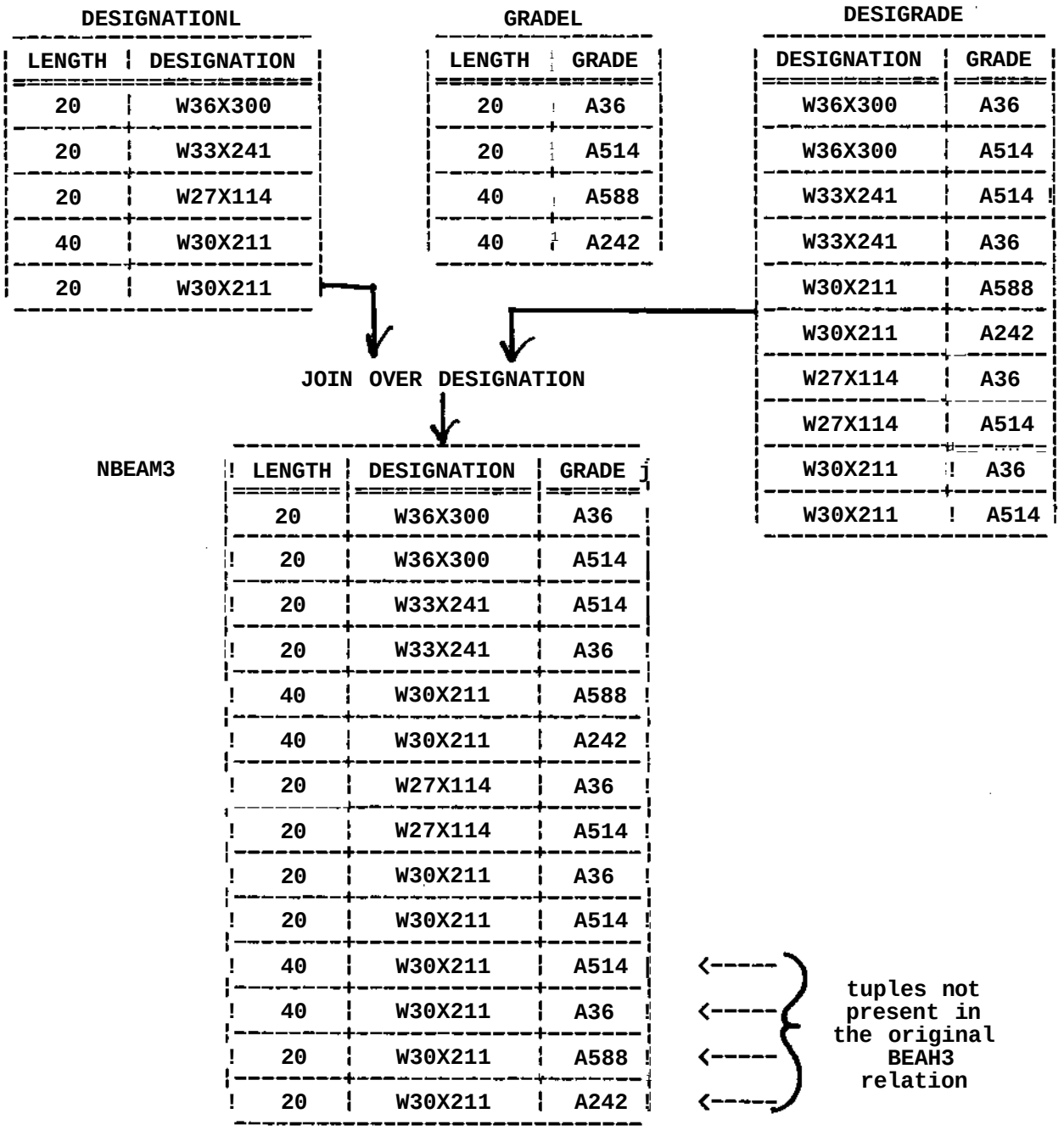

Figure 4-9: DESIGNATIONL, GRADEL, DESIGRADE and NBEAM3 relations: Example of join dependency.

BEAM3 relation. This is called a join dependency. It is only when the third projection is also joined that the NBEAM3 relation is the same as the original BEAM3 relation. Therefore, in order to eliminate this join dependency, the BEAM3 relation should be broken into three new relations, each with a different key corresponding to the number of possible or "candidate" keys in BEAM3, and left in this form.

Fifth normal form can be defined in the following way:

> A relation is in fifth normal form or projection-join normal form if and only if all join dependencies are eliminated.

4.6. DOMAIN-KEY NORMAL FORM

After reading about all the other normal forms, one may wonder if there is a limit to the number of possible normal forms. It is true that as long as new problems are found, new normal forms could be defined. However, Fagin [7] has defined a different normal form that encompasses all of the other normal forms described in this article. It is called *domain-key* normal form (DK/NF). In order for a relation to be in DK/NF all *insertion* and *deletion* anomalies must be removed. Insertion and deletion anomalies are defined by Fagin [7] in the following way:

> "An insertion anomaly occurs when a seemingly legal insertion of a single tuple into a valid instance of the relation schema causes the resulting relation to violate one of the constraints of the relation schema. Here 'seemingly legal' means that every entry of the new tuple is in the appropriate domain and that the new tuple differs from all previous tuples".
>
> "A deletion anomaly occurs when the deletion of a single tuple from a valid instance of the relation schema causes a constraint to be violated".

All of the problems described in this article for the other normal forms can be described as an insertion and/or deletion anomaly, once the concept of *domain dependencies* is included in their definitions. A domain dependency is the dependency between the attribute and the domain. If the domain is infinite, there is no domain dependency. A bounded domain creates a domain dependency. For example, assume that the LENGTH domain in the BEAM3 relation in Figure 4-7 is bounded by the values 0 and 100. The insertion of a beam of length 200 feet would violate the domain dependency.- Therefore, this is a special case of a insertion anomaly (a constraint is violated upon insertion).

Therefore, removal of these anomalies, by whatever means necessary, causes the

relation to be in DK/NF which implies that the relation is also in first, second, third, fourth and fifth normal forms. The reader is directed to Fagin [7] for a proof of this statement and a more detailed description of DK/NF.

5. CONCLUSION

This article has defined some of the basic concepts in the relational approach to database management, using structural engineering examples. It has shown that databases, especially relational databases, are of great potential benefit to the engineering profession, due to increased data independence. Also, a detailed description of normalization has been provided. Normalization can best be summarized by the following statement:

> Once the user decides on the configuration (schema) of the database, he or she must also be sure that this schema guards against all possible insertion and deletion anomalies (problems) that might arise. If the schema does not do this, the user can choose one of the following alternatives:
>
> - Change the schema so that all possible problems are eliminated. If this is done, the database is in the "best" configuration.
>
> - Use this schema and manually check for the occurrence of the insertion and/or deletion problems. Under normal circumstances this alternative should not be chosen, since the insertion and deletion problems are usually very difficult to detect.

It is hoped that this article has provided the reader with an informative summary of database issues and has shown the importance of careful planning during the design of a relational database for engineering.

REFERENCES

[1] American Institute of Steel Construction.
Manual of Steel Construction
Eighth edition, American Institute of Steel Construction Inc., Chicago, Illinois, 1980.

[2] Blasgen, M. W., et. al.
SYSTEM R: An Architectural Overview.
IBM Systems Journal 20(1):41-61, 1981.

[3] Chamberlin, D. D. and Boyce, R. F.
SEQUEL: A Structured English Query Language.
Proceedings of the ACM - SIGFIDET Workshop , May, 1974.

[4] Chamberlin, D. D., et. al.
SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control.
IBM Journal of Research and Development 20(6):560-575, November, 1976.

[5] Codd, E. F.
Relational Database: A Practical Foundation for Productivity.
Communications of the ACM 25(2): 109-117, February, 1982.

[6] Date, C. J.
An Introduction to Database Systems.
Addision-Wesley, Reading, Massachusetts, 1981.
Third Edition.

[7] Fagin, Roland.
A Normal Form for Relational Databases That Is Based on Domains and Keys.
ACM Transactions on Database Systems 6(3), September, 1981.