

15 Seconds : Introduction to Relational Databases - Part 1: Theoretical Foundation

Tore Bostrup
05/22 /2002

An important aspect of most every business is record keeping. In our information society, this has become an important aspect of business, and much of the world's computing power is dedicated to maintaining and using databases.

Databases of all kinds pervade almost every business. All kinds of data, from emails and contact information to financial data and records of sales, are stored in some form of a database. The quest is on for meaningful storage of less-structured information, such as subject knowledge.

This is the first of a two-part article that will provide an introduction to **relational databases** and the **SQL language**. This first part describes some of the key elements of the technology with an emphasis on database normalization. The second part will describe a less theoretical approach to database design, as well as provide an introduction to the SQL language.

History

The concept of relational databases was first described by Edgar Frank Codd (almost exclusively referenced as E. F. Codd in technical literature) in the IBM research report RJ599, dated August 19th, 1969.¹ However, the article that is usually considered the cornerstone of this technology is "A Relational Model of Data for Large Shared Data Banks," published in *Communications of the ACM* (Vol. 13, No. 6, June 1970, pp. 377-87). Only the first part of the article is available online.

Additional articles by E. F. Codd throughout the 1970s and 80s are still considered gospel for relational database implementations. His famous "Twelve Rules for Relational Databases"² were published in two *Computerworld* articles "Is Your DBMS Really Relational?" and "Does Your DBMS Run By the Rules?" on October 14, 1985, and October 21, 1985, respectively. He has since expanded on the 12 rules, and they now number 333, as published in his book "[The Relational Model for Database Management, Version 2](#)" (Addison-Wesley, 1990).

Codd's twelve rules call for a language that can be used to define, manipulate, and query the data in the database, expressed as a string of characters. The language, SQL, was originally developed in the research division of IBM (initially at Yorktown Heights, N.Y., and later at San Jose, Calif., and Raymond Boyce and Donald Chamberlin were the original designers.)³ and has been adopted by all major relational database vendors. The name SQL originally stood for Structured Query Language. The first commercially available implementation of the language was named SEQUEL (for Sequential English QUery Language) and was part of IBM's SEQUEL/DS product. The name was later changed for legal reasons. Thus, many long-time database developers use the pronunciation "see-quell."

SQL has been adopted as an ANSI/ISO standard. Although revised in 1999 (usually referenced as SQL99 or SQL3), most vendors are still not fully compliant with the 1992 version of the standard. The 1992 standard is smaller and simpler to reference for a user, and since only some of the 1999-specific requirements are typically implemented at this time, it may be a better starting point for learning the language.

Need-to-Know for the Database Developer

The Rules of the Game

Codd's Twelve Rules

Many references to the twelve rules include a thirteenth rule - or rule zero:

A relational database management system (DBMS) must manage its stored data using only its relational capabilities.

This is basically a corollary or companion requirement to rule #4

1. Information Rule
All information in the database should be represented in one and only one way -- as values in a table.
2. Guaranteed Access Rule
Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.
3. Systematic Treatment of Null Values
Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.
4. Dynamic Online Catalog Based on the Relational Model
The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.
5. Comprehensive Data Sublanguage Rule
A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:
 - a. data definition
 - b. view definition
 - c. data manipulation (interactive and by program)
 - d. integrity constraints
 - e. authorization
 - f. transaction boundaries (begin, commit, and rollback).
6. View Updating Rule
All views that are theoretically updateable are also updateable by the system.
7. High-Level Insert, Update, and Delete
The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data, but also to the insertion, update, and deletion of data.
8. Physical Data Independence
Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.
9. Logical Data Independence
Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.
10. Integrity Independence
Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.
11. Distribution Independence
The data manipulation sublanguage of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically

centralized or distributed.

12. Nonsubversion Rule

If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

(**Author's Note:** I do not have access to E.F. Codd's original text, but the above wording appears to be commonly used in other documents. See source 2.)

The rules primarily address implementation requirements for relational database management system (RDBMS) vendors. However, some of them also have an impact on application design.

In the early days of relational database products, these twelve rules were often used to evaluate RDBMSs. In the academic community, the discussions of full compliance of RDBMS versus the Relational Model continues, as does the discussion about whether the SQL language satisfies all the requirements. But we will stick to more practical matters.

If you would like more information about what these twelve rules mean, see http://www.frick-cpa.com/ss7/Theory_RelationalDB.asp#Codd.

How to Design the Database

Theoretical Foundation

Purpose

The problem with data is that it changes. Not just its individual items' values change, but their structure and use, especially when kept over extended periods of time. Even for public records that may have been kept for hundreds of years, there are occasionally changes in what data elements are captured and recorded and how.

Therefore, a method to avoid problems due to duplication of data values and modification of structure and content has been developed. This method is called *normalization*.

You normalize a database in order to ensure data consistency and stability, to minimize data redundancy, and to ensure consistent updatability and maintainability of the data, and avoid update and delete anomalies that result in ambiguous data or inconsistent results.

Some Key Concepts

Before we continue, I would like to show the correlation between the formal names of Tables, Rows, and Columns in Relational Theory and their more common counterparts:

<i>Formal Name</i>	<i>Common Name</i>	<i>Also Known As</i>
<i>Relation</i>	Table	Entity
<i>Tuple</i>	Row	Record
<i>Attribute</i>	Column	Field

A **Primary Key** is one or more columns whose values uniquely identify a row in a table (See rule #2 above).

A **Candidate Key** is one or more columns whose values could be used to uniquely identify a row in a table. The Primary Key is chosen among a table's Candidate Keys.

Normalization

Normalization is the formalization of the design process of making a database compliant with the concept of a **Normal Form**. It addresses various ways in which we may look for repeating data values in a table. There are several levels of the Normal Form, and each level requires that the previous level be satisfied. I have used the wording (indicated in italicized text) for each normalization rule from the *Handbook of Relational Database Design* by Candace C. Fleming and Barbara von Halle.⁴

The normalization process is based on collecting an exhaustive list of all data items to be maintained in the database and starting the design with a few "superset" tables. Theoretically, it may be possible, although not very practical, to start by placing all the attributes in a single table. For best results, start with a reasonable breakdown.

First Normal Form

Reduce entities to first normal form (1NF) by removing repeating or multivalued attributes to another, child entity.

Basically, make sure that the data is represented as a (proper) table. While key to the relational principles, this is somewhat a motherhood statement. However, there are six properties of a relational table (the formal name for "table" is "relation"):

- Property 1: Entries in columns are single-valued.
- Property 2: Entries in columns are of the same kind.
- Property 3: Each row is unique.
- Property 4: Sequence of columns is insignificant.
- Property 5: Sequence of rows is insignificant.
- Property 6: Each column has a unique name.

The most common sins against the first normal form (1NF) are the lack of a Primary Key and the use of "repeating columns." This is where multiple values of the same type are stored in multiple columns. Take, for example, a database used by a company's order system. If the order items were implemented as multiple columns in the Orders table, the database would not be 1NF:

OrderNo	Line1Item	Line1Qty	Line1Price	Line2Item	Line2Qty	Line2Price
245	PN768	1	\$35	PN656	3	\$15

To make this first normal form, we would have to create a child entity of Orders (Order Items) where we would store the information about the line items on the order. Each order could then have multiple Order Items *related* to it.

OrderNo	OrderNo	Item	Qty	Price
245	245	PN768	1	\$35
	245	PN656	3	\$15

Second Normal Form

Reduce first normal form entities to second normal form (2NF) by removing attributes that are not dependent on the whole primary key.

The purpose here is to make sure that each column is defined in the correct table. Using the more formal names may make this a little clearer. Make sure each attribute is kept with the entity that it describes.

Consider the Order Items table that we established above. If we place Customer reference in the Order Items table (Order Number, Line Item Number, Item, Qty, Price, Customer) and assume that we use Order Number and Line Item Number as the Primary Key, it quickly becomes obvious that the Customer reference becomes repeated in the table because it is only dependent on a portion of the Primary Key - namely the Order Number. Therefore, it is defined as an attribute of the wrong entity. In such an obvious case, it should be immediately clear that the Customer reference should be in the Orders table, not the Order Items table.

So instead of:

OrderNo	ItemNo	Customer	Item	Qty	Price
245	1	SteelCo	PN768	1	\$35
245	2	SteelCo	PN656	3	\$15
246	1	Acme Corp	PN371	1	\$2.99
246	2	Acme Corp	PN015	7	\$5

We get:

OrderNo	Customer	OrderNo	ItemNo	Item	Qty	Price
245	SteelCo	245	1	PN768	1	\$35
245	SteelCo	245	2	PN656	3	\$15
246	Acme Corp	246	1	PN371	1	\$2.99
		246	2	PN015	7	\$5

Third Normal Form

Reduce second normal form entities to third normal form (3NF) by removing attributes that depend on other, nonkey attributes (other than alternative keys).

This basically means that we shouldn't store any data that can either be derived from other columns or belong in another table. Again, as an example of derived data, if our Order Items table includes both Unit Price, Quantity, and Extended Price, the table would not be 3NF. So we would remove the Extended Price (= Qty * Unit Price), unless, of course, the value saved is a manually modified (rebate) price, but the Unit Price reflects the quoted list price for the items at the time of order.

Also, when we established that the Customer reference did not belong in the Order Items table, we said to move it to the Orders table. Now if we included customer information, such as company name, address, etc., in the Orders table, we would see that this information is dependent not so much on the Order per se, but on the Customer reference, which is a nonkey (not Primary Key) column in the Orders table. Therefore, we need to create another table (Customers) to hold information about the customer. Each Customer could then have multiple Orders related to it.

OrderNo	Customer	Address	City
245	SteelCo	Works Blvd	Vinings
246	Acme Corp	North Drive	South Bend
247	SteelCo	Works Blvd	Vinings

OrderNo	Customer	Customer	Address	City
245	SteelCo	SteelCo	Works Blvd	Vinings
246	Acme Corp	Acme Corp	North Drive	South Bend
247	SteelCo			

Why Stop Here?

Many database designers stop at 3NF, and those first three levels of normalization do provide the most bang for the buck. Indeed, these were the original normal forms described in E. F. Codd's first papers. However, there are currently four additional levels of normalization, so read on. Be aware of what you don't do, even if you stop with 3NF. In some cases, you may even need to de-normalize some for performance reasons.

Boyce/Codd Normal Form

Reduce third normal form entities to Boyce/Codd normal form (BCNF) by ensuring that they are in third normal form for any feasible choice of candidate key as primary key.

In short, Boyce/Codd normal form (BCNF) addresses dependencies between columns that are part of a Candidate Key.

Some of the normalizations performed above may depend on our choice of the Primary Key. BCNF addresses those cases where applying the normalization rules to a Candidate Key other than the one chosen as the Primary Key would give a different result. In actuality, if we substitute any Candidate Key for Primary Key in 2NF and 3NF, 3NF would be equivalent with BCNF.

In a way, the BCNF is only necessary because the formal definitions center around the Primary Key rather than an entity item abstraction. If we define an entity item as an object or information instance that correlates to a row, and consider the normalization rules to refer to entity items, this normal form would not be required.

In our example for 2NF above, we assumed that we used a composite Primary Key consisting of Order Number and Line Item Number, and we showed that the customer reference was only dependent on a portion of the Primary Key - the Order Number. If we had assigned a unique identifier to every Order Item independent of the Order Number, and used that as a single column Primary Key, the normalization rule itself would not have made it clear that it was necessary to move the Customer reference.

There are some less obvious situations for this normalization rule where a set of data actually contains more than one relation, which the following example should illustrate.

Consider a scenario of a large development organization, where the projects are organized in project groups, each with a team leader acting as a liaison between the overall project and a group of developers in a matrix organization. Assume we have the following situation:

- Each Project can have many Developers.
- Each Developer can have many Projects.
- For a given Project, each Developer only works for one Lead Developer.
- Each Lead Developer only works on one Project.
- A given Project can have many Lead Developers.

In this case, we could theoretically design a table in two different ways:

ProjectNo	Developer	Lead Developer
-----------	-----------	----------------

20020123	John Doe	Elmer Fudd
20020123	Jane Doe	Sylvester
20020123	Jimbo	Elmer Fudd
20020124	John Doe	Ms. Depesto

Case 1: Project Number and Developer as a Candidate Key can be used to determine the Lead Developer. In this case, the Lead Developer depends on both attributes of the key, and the table is 3NF if we consider that our Primary Key.

Lead Developer	Developer	ProjectNo
Elmer Fudd	John Doe	20020123
Sylvester	Jane Doe	20020123
Elmer Fudd	Jimbo	20020123
Ms. Depesto	John Doe	20020124

Case 2: Lead Developer and Developer is another Candidate Key, but in this case, the Project Number is determined by the Lead Developer alone. Thus it would not be 3NF if we consider that our Primary Key.

In reality, these three data items contain more than one relation (Project - Lead Developer and Lead Developer - Developer). To normalize to BCNF, we would remove the second relation and represent it in a second table. (This also illustrates why a *table* is formally named a *relation*.)

ProjectNo	Lead Developer
20020123	Elmer Fudd
20020123	Sylvester
20020123	Elmer Fudd
20020124	Ms. Depesto

Lead Developer	Developer
Elmer Fudd	John Doe
Elmer Fudd	Jimbo
Sylvester	Jane Doe
Ms. Depesto	John Doe

Fourth Normal Form

Reduce Boyce/Codd normal form entities to fourth normal form (4NF) by removing any independently multivalued components of the primary key to two new parent entities. Retain the original (now child) entity only if it contains other, nonkey attributes.

Where BCNF deals with dependents of dependents, 4NF deals with multiple, independent dependents of the Primary Key. This is a bit easier to illustrate.

Let us say we wanted to represent the following data: Manager, Manager Awards, and Direct Reports. Here, a Manager could have multiple Awards, as well as multiple Direct Reports. 4NF requires that these be split into two separate tables, one for Manager - Awards, and one for Manager - Direct Reports. We may need to

maintain a Managers table for other Manager attributes.

This table:

Manager	Awards	Direct Reports
Scrooge McDuck	Stingy John	Donald Duck
Minnie Mouse	Mouse of the Month	Mickey Mouse
Minnie Mouse	Mouse of the Year	Pluto
Clara		Goofy

becomes two tables:

Manager Awards Table

Manager	Awards
Scrooge McDuck	Stingy John
Minnie Mouse	Mouse of the Month
Minnie Mouse	Mouse of the Year
Clara	

Direct Reports Table

Manager	Direct Reports
Scrooge McDuck	Donald Duck
Minnie Mouse	Mickey Mouse
Minnie Mouse	Pluto
Clara	Goofy

Fifth Normal Form

Reduce fourth normal form entities to fifth normal form (5NF) by removing pairwise cyclic dependencies (appearing within composite primary keys with three or more component attributes) to three or more parent entities.

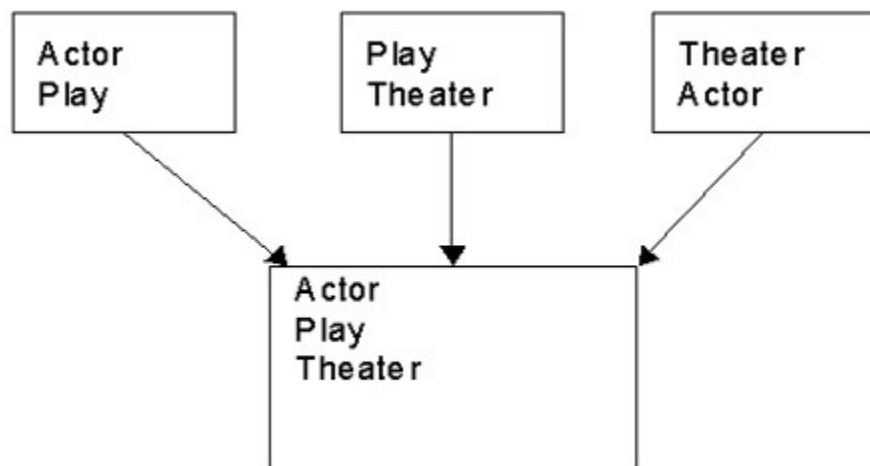
This addresses problems that arise from representing associations between multiple entities with interdependencies. Making it 5NF consists of adding parent tables, one for each meaningful combination that has children in the original table.

A table with such information is 5NF if the information cannot be represented in multiple smaller entities alone.

An example of such a situation may be the representation of Actors, Plays, and Theaters. In order to know who plays what and where, we need the combination of these three attributes. However, they each relate to each other cyclically. So to resolve this, we would need to establish parent tables with Actor - Play, Play - Theater, and Theater - Actor. These would each contain a portion of the Primary Key in the Actor, Play, and Theater table.

Actor	Play	Theater
-------	------	---------

Billy Bob	Catcher in the Rye	West 42nd
Ann	Catcher in the Rye	West 42nd
John	Catch-22	Broadway
Lily	Hamlet	Broadway
Lisa	Cats	West 42nd
Andy	Cats	Darlington



Domain Key Normal Form

(Not defined in "Handbook of Relational Database Design."⁵).

The simplest description I have found is at Search Database.com at http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci212669,00.html:

"A key uniquely identifies each row in a table. A domain is the set of permissible values for an attribute. By enforcing key and domain restrictions, the database is assured of being freed from modification anomalies."

This appears to differ from the other normal forms in that it does not seek to introduce additional tables, but rather ensures that columns are restricted to valid values.

According to http://www.cba.nau.edu/morgan-j/class/subtop2_3/tsld023.htm, "...there is no known process for ensuring that tables are in Domain Key Normal Form."

Conclusion

While we may not always observe all the rules or normalize our databases to the fifth and domain key normal form, it is important to have a basic understanding of the theoretical principles of database design. It will help us not only design normalized databases, but to build more powerful and flexible applications. Also, it will help us ensure that our data remains usable. Now that we have laid the theoretical foundation and defined the formal database design methods for normalization, it may be time to take a break. I need one anyway. :->

When we resume with Part 2 of this article, I will show how we can design a fairly well normalized database using nothing but some common sense, a few simple rules, and a piece of string, so get ready! Part 2 of this article will address a different approach to designing the database, normalization through synthesis, and will describe the SQL language.

Oh, and never mind the piece of string. :->

Sources

1. C. J. Date, "There's Only One Relational Model!" (see <http://www.pgro.uk7.net/cjd6a.htm>).
2. Dr. E. F. Codd's 12 rules for defining a fully relational database (see <http://www.cis.ohio-state.edu/~sgomori/570/coddsrules.html>).
3. C. J. Date
4. *Handbook of Relational Database Design* by Candace C. Fleming and Barbara von Halle (Addison Wesley, 1989).
5. *Ibid.*

References

1. Characteristics of a Relational Database by David R. Frick & Co., CPA.
2. Dr. Morgan at Northern Arizona University - College of Business Administration
3. SearchDatabase.com

About the Author

Tore Bostrup is a software developer for Team IA, Inc., in Columbia, South Carolina. Born in Oslo, Norway, he has a master's degree in computer science from the University of Oslo. He has twenty-five years of experience in software support, management, consulting, and development. Tore can be reached at tbostrup@telocity.com.

[Back to article](#)

Copyright 2005 Jupitermedia Corp. All Rights Reserved.
[Legal Notices](#), [Licensing](#), [Reprints](#), & [Permissions](#), [Privacy Policy](#).
<http://www.internet.com>