Relational Database: A Practical Foundation

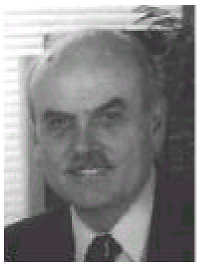
Francis Zinke

Moosglöckchenweg 16
D-14478 Potsdam

Abstract

I will present relational databases and explain some of its concepts. I will show the practicality and the offered improvements for productivity of this systems and prove this. In addition i will explain some concepts of Data Modeling, including ER-Models and normalization.

1. Introduction



Most of the materials of this lecture are from the 1981 ACM Turing Award Lecture by E. F. Codd. He was born in 1981 in Great Britain. E. F. Codd joined IBM in 1949 and worked in different areas. He received his B. A. and M. A. in Mathematics from Oxford University in England, and his M. Sc. and Ph. D. in Computer and Communication Sciences from the University of Michigan. His contributions for relational databases were enormous.

First I want to explain some terms: A Database is the concrete Data of the application area. A Database Management System is the Software which is controlling and checking the access to the Database.
In the early seventies the DBMS, as instruments of data- and file handling, failed as productivity boosters. They were thought for removing a lot of the problems of the application programmers, like the details of file- and datahandling. But why didn't they increase the productivity of the programmer?

E. F. Codd is restricting three reasons:

a) The systems forced the programmers to think and code at a low level of structural detail. They had to know numerous concepts for doing data retrieval and manipulation tasks.

b) DBMS did not support any commands for processing multiple records at a time, named set processing. So the programmers had to code in terms of iterative loops.

c) Query capabilitys for the needs of end users were not in. Simple direct particular interaction of the user with the database had to be added by the programmer.

In addition: The installation of these systems was incredible slow, because slight changes in the storage structure did lead to changes of all programs, lying on this structure. The programmers spent a lot of time in learning about the system and in planning the organization of data careful.

2. Motivation

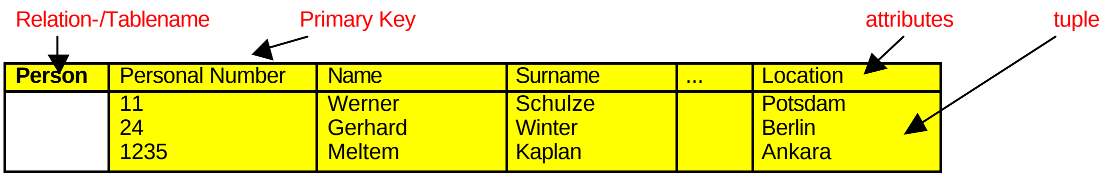
First of all E. F. Codd wanted to provide a sharp and clear boundary between the physical and logical aspects of database management (including the design, data retrival and manipulation). He is calling this data independence objective. Second point was the improving of the communication about the database, assuming a common understanding of the data, called communicapability objective. The last objective was the set processing, means: introducing high level language concepts to enable users to execute high specific operations at a time.

3. The Relational Model

For reaching this three aims it was useful to take a fresh look at the adressing of data. Adressing is the identification of the elements in a structure. They support the readable and writeable data. Different positional concepts are:

- symbolic adressing (you don't refer to the structure, only to the not ordered symbols)
- absolute numeric adressing (jumping to the required adress, relativ to the first/beginning adress)
- relative numeric adressing (jumping to the required adress, relativ to the actual adress) and
- plugboard adressing (jumping to the next or previous adress in the neighborhood)

In the relational model we use totally associative adressing, instead of positional adressing. The data is uniquely adressed by means of relation name, primary key value and attribute name. The details of placement of a new piece of information and the select of appropriate access paths, the system takes on. Programmers and end users adress data in the same way now. The mathematical foundation is the n-ary relation with appropriate operators and an appropriate conceptual representation, the table.


Relation-/Tablename
Primary Key
attributes
tuple

Person	Personal Number	Name	Surname	...	Location
	11	Werner	Schulze		Potsdam
	24	Gerhard	Winter		Berlin
	1235	Meltem	Kaplan		Ankara

The set Person is a relation: Person $\subseteq$ W(Personal Number) • W(Name) • W(Surname) • W(Location) above the set of the attributes of the entity set Person.
A entity is a instance, like the name *Werner* or the Location *Ankara.* Entitys with same properties are the entity set. Attributes are this properties with a defined co-domain. Tuple is a set of attributes. And a relation is a set of tuples with the same attribute names. The mathematical definition of a relation follows: It is a subset of a cartesian product. A cartesian product is a combination of all elements of finite sets. Keys or primary keys are used to uniquely identify a row or tuple in a table. Think of it as house numbers in a street identifying each house. Without the visitors would not know where to go. And they enable tables in the database to be related with each other. Without a primary key a database engine can not identify individual rows. A primary key can consist of one or more columns or fields. All columns (or combination of columns) that contain unique values are called candidate keys and the primary key can be chosen from any of these. All other remaining candidate keys are called alternate keys. A primary key consisting of one column is called a simple key whilst a primary key made up of a number of columns is called a composite key.

But the relational model is not only a tabular model, because relations are at a higher abstraction level and the information content is independent of row order in tables.

This leads to the question what is a data model? It is not only a structure, it is a combination of three components:

a) A collection of data structure types (the database building blocks);
b) A collection of operators or rules, applied to any valid instances of the data types, to retrieve, derive, or modify data in any combinations;
c) A collection of general integrity rules, which implicitly or explicitly define the database set (called insert-update-delete rules)

In this definition a relational model is a data model.
Codd is differentiating here three parts of the relational model.
Its **structural** part consist of domains, relations of assorted degrees (tables), attributes, tuples, candidate keys and primary keys.
The **manipulative** part consists of the algebraic operators (select, project, join, etc.), which transform relations into relations (tables into tables).
Third part: a relational model must meet two **integrity** rules: entity integrity and referential integrity, defining, which sets of a cartesian product are allowed relations. The entity integrity rule states that the value of the primary key, can never be a null value (a null value is one that has no value and is not the same as a blank). Because a primary key is used to identify a unique row in a relational table, its value must always be specified and should never be unknown. The integrity rule requires that insert, update, and delete operations maintain the uniqueness and existence of all primary keys. The referential

integrity rule states that if a relational table has a foreign key, then every value of the foreign key must either be null or match the values in the relational table in which that foreign key is a primary key.

This three parts shouldn't be defined alone, because there is a strong coupling between them. For example CODASYL and ANSI tried to develop data definition language (DDL) and data manipulation language (DML) in separate committees, the foundation are this structures, but there has been a lot of misunderstandings.
In SQL DDL is for defining tables and views (create, delete, modify) , integrity conditions, access- and index administration. DML is for reading and updating queries, in the whole working with the data.

4. The Relational Processing Capability

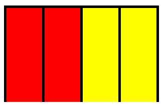
Relational processing is the use of whole relations as operands. The primary purpose is loop avoidence. The examples of the following operators are in SQL.

The SELECT operator takes one relation (table) and produces a new relation, consisting of selected tuples (rows) of the first.

Query: Put out all persons living in Berlin (select all tuple from table Person where the attribute Location ='Berlin'
select * from Person where Location='Berlin'
Solution: 24 Gerhard Winter ... Berlin

The PROJECT operator transforms one relation into a new one, consisting of selected attributes (columns), of the first.

Query: Put out the names and locations of all Persons
select distinct Name, Location from Person
Solution: Werner Gerhard Meltem, Potsdam Berlin Ankara



A JOIN operation combines the product, selection, and, possibly, projection. The join operator horizontally combines data from one row of a table with rows from another or the same table when certain criteria are met. The criteria involve a relationship among the columns in the join relational table. The EQUI JOIN concatenate two relations with matching attributes (same name and co-domain).

R	K	X	Y
	1	A	2
	2	B	4
	3	C	6
	4	D	8
	5	E	10

S	K	Z
	1	20
	4	22
	5	24
	7	26
	9	28

K	X	Y	K	Z
1	A	2	1	20
4	D	8	4	22
5	E	10	5	28

EQUI JOIN

select R.K,X,Y,Z from R,S
where R.K=S.K

K	X	Y	Z
1	A	2	20
4	D	8	22
5	E	10	28

NATURAL JOIN

select * from R,S
where R.K=S.K

The NATURAL JOIN takes two relations and produces a new in the same way, but the redundant columns are removed. The JOIN operators should allow matching any pair of attributes, providing that they are on the same domain or data type (unrestricted join capability).

This data sublanguage is a yardstick of power, if it is used without iteration or recursion statements, and not restricted by the implementation, having to do with predefinition of physical access paths.
A sublanguage has a relational processing capability, if the transformations SELECT, PROJECT and unrestricted JOIN of the relational algebra are specified without commands for iteration or recursion.

A database management system must support: Tables without user-visible navigation links between them and a data sublanguage with a least this (minimal) relational processing capability.

Codd distinct here the various kinds of relational and tabular models: A DBMS that does not support relational processing is called non-relational. If this system supports tables without user-visible navigation links between tables, it is a tabular system. The implementation complexity of a tabular system is high, because the programmer does his own navigation. A relational system provides automatic navigation. Therefore, a tabular system shouldn't be called semi-relational system.
All this definitions don't require, that the full relational algebra is supported. In that case the system is called fully relational. We need some more operators:

The UNION operation of two relational tables is formed by appending rows from one table with those of a second table to produce a third. Duplicate rows are eliminated. For using this operator the tables must have the same number of columns and corresponding columns must come from the same domain (union compatible).

A	K	X	Y
	1	A	2
	2	B	4
	3	C	6

B	K	X	Y
	1	A	2
	4	D	8
	5	E	10

UNION

K	X	Y
1	A	2
2	B	4
3	C	6
4	D	8
5	E	10

A-B
DIFFERENCE

K	X	Y
2	B	4
3	C	6

INTERSECT

K	X	Y
1	A	2

PRODUCT

AK	AX	AY	BK	BX	BY
1	A	2	1	B	2
1	A	2	4	D	8
1	A	2	5	E	10
2	B	4	1	B	2
2	B	4	4	D	8
2	B	4	5	E	10
3	C	6	1	B	2
3	C	6	4	D	8
3	C	6	5	E	10

The DIFFERENCE of two relational tables is a third that contains those rows that occur in the first table but not in the second. The Difference operation requires that the tables be union compatible. As with arithmetic, the order of subtraction matters.
The INTERSECTION of two relational tables is a third table that contains common rows. Both tables must be union compatible.
The PRODUCT of two relational tables, also called the Cartesian Product, is the concatenation of every row in one table with every row in the second.

Lets take a look at the prohibitions of the relational model again: user-visible navigation links between tables and the ordering of tuples within base relations must not be represented in the database. Database designers of non-relational systems sometimes don't understand or accept these prohibitions, in contrast users are enthusiastic of the use and results from them and the relational model.

5. The Uniform Relational Property

For a wide applicability a data sublanguage can be interfaced with one or more programming languages (e.g. Cobol, Fortran, PL/I, APL), they are named as host languages. Some DBMS do support more than one data sublanguage, string-oriented languages, like QUEL or SQL, or screen-oriented two-dimensional data sublanguages.
Some of this sublanguages are usable in two modes: interactively at terminal or embedded in a application program written in a host language.
Such a double-mode data sublanguage has a lot of advantages: application programmers can seperately debug at a terminal the database statements, they want to employ in their application programs; the communication among programmers, analysts, end users, administrators, etc. increases and the frivolous distinctions between the languages used in both modes decrease the deal of energy for user of them. This double-mode sublanguage DBMS is called uniform relational. The DBMS supporting relational processing in this way is a great productivity booster. All other DBMS are non-uniform relational.

6. Skepticism About Relational Systems

The skepticism about relational database management comes from a lack of understanding and the fear of the numerous theoretical investigations. Otherwise, there pose two questions: Has a relational DBMS the same service like others and is the performance of such a system as well as non-relational systems?

6.1 Range of Services

A full scale DBMS provides the following capabilities:

- data storage, retrieval, and update;
- a user-accessible catalog for data description;
- transaction support to ensure that all or none database changes are reflected in the pertinent databases;
- recovery services in case of failure;
- concurrency control (concurrent transactions run in some sequential order);
- authorization services (access to and manipulation of data only for specified users and programs)
- integration with support for data communication;
- integrity services (database states and changes of state conform to specified rules);

In the eighties several relational systems did provide all these services, except the last. Some relational DBMS support more complete data services than non-relational. Three examples follow.
In relational DBMS extraction of all meaningful relations is available, in non-relational only where exist statically predefined access paths.
Second example: some relational systems do support views. A view is a virtual relation (table), defined by an expression or sequence of commands. A view appears to a user like an additional table kept up-to-date and out of other base tables. They are useful for permitting application programs and users.
Third point: some systems permit changes of the physical and logical organization of the data dynamically - while transactions are in progress. This is a productivity booster for programmers. This capability is possible, because the system has full control over access path selection. In non-relational systems all transaction must be brought to halt for such organizational changes.

6.2 Performance

The performance of relational systems is concluded by the time it takes to execute a complex transaction. Comparing with the time a non-relational system takes for a simple transaction, is not fair. For arriving at a fair level, we compare them on the same tasks and applications.

If a system support performance-oriented physical data structures and if high level data requests are compiled into lower-level code sequences, which are as good as produced by hand, it is good performance.

Tandem Computer Corporation did install in 1981 sevenhundred systems with relational DBMS, written in COBOL, but with no support for links between database tables, user-visible (navigation) and user-invisible (access method) links.
Let's suppose now, we take this installation and replace the application programs by a database sublanguage with relational processing capability (like SQL). To obtain good perfomance we precompile (not interprete) the database statements, programs and queries from terminals (written in the high level sublanguage) into object code (CALLs) for optimizing the sequencing of the major operations and the selection of access paths. After this step the application programs are compiled into the host-language .

Now comparing the execution times of both system examples, the performance of the DBMS with relational processing capability is better, except the simplest of queries.
The price paid for extra compile time is compensated by the quite sophisticated optimization scheme of the precompiler that is normally not known by the programmers.

7. Data Modeling

In addition i explain some points of the Data Modeling. The modeling of a database follows a sequence of five steps – planning and analysis, design of the conceptual model, design of the logical model, design of the physical model and finally the implementation. We restrict to the the conceptual and logical data modeling. However, before getting down to the specifics it is important to note one thing - there are not many better ways of designing a database than with a pen and paper. A good

idea is to consider what outputs you need from the database (tuples, data files, etc.) and then decide what these outputs need to be made up of (customer information, sales figures, etc.).
Data modeling must be preceded by planning and analysis. Planning defines the goals of the database , explains why the goals are important, and sets out the path by which the goals will be reached. Analysis involves determining the requirements of the database. This is typically done by examining existing documentation and interviewing users.
An effective data model completely and accurately represents the data requirements of the end users. It is simple enough to be understood by the end user yet detailed enough to be used by a database designer to build the database. The model eliminates redundant data, it is independent of any hardware and software constraints, and can be adapted to changing requirements with a minimum of effort.

The **Entity-Relationship Model** is a conceptual data model that views the real world as consisting of entities and relationships. It was originally proposed by Peter Chen in 1976. The model visually represents these concepts by the **Entity-Relationship Diagram (ERD)**. The basic constructs of the ER model are entities, relationships, and attributes.

Entities are concepts, real or abstract, about which information is collected, such as person, places, things, or events which have relevance to the database. Specific examples of entities are Employees, Projects, Invoices. An entity is analogous to a table in the relational model.

Relationships are associations between the entities. An example would be: employees are assigned to projects; projects have subtasks; departments manage one or more projects.
Relationships are classified in terms of degree, connectivity, cardinality, and existence.

The **degree** of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Binary relationships (degree 2), the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself (example: some employees are married to other employees). A ternary relationship involves three entities. Because many modeling approaches recognize only binary relationships, ternary or n-ary relationships are decomposed into two or more binary relationships.

The **connectivity** of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The **cardinality** of a relationship is the actual number of related occurences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

A **one-to-one (1:1)** relationship is when at most one instance of a entity A is associated with one instance of entity B (example: employees in the company are each assigned their own office; for each employee there exists a unique office and for each office there exists a unique employee).

A **one-to-many (1:N)** relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A (a department has many employees and each employee is assigned to one department)

A **many-to-many (M:N)** relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A (example: employees can be assigned to no more than two projects at the same time; projects must have assigned at least three employees)

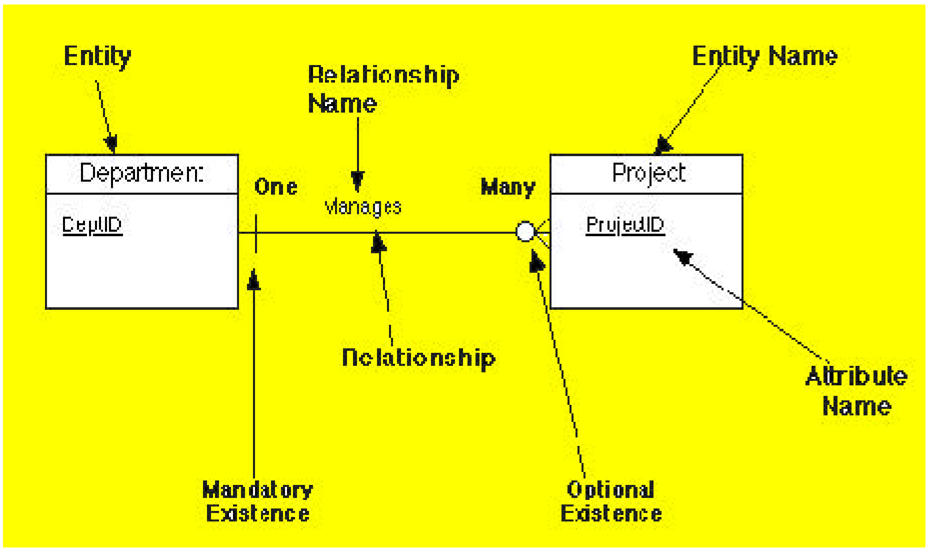
Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either mandatory or optional. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory (every project must be managed by a single department). If the instance of the entity is not required, it is optional (employees may be assigned to work on projects).

A **generalization hierarchy** is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a supertype or

generic entity. The lower-level of entities become the subtype, or categories, to the supertype. Subtypes are dependent entities.
Generalization occurs when two or more entities represent categories of the same real-world object. For example, Wages_Employees and Classified_Employees represent categories of the same entity, Employees. In this example, Employees would be the supertype; Wages_Employees and Classified_Employees would be the subtypes.

Attributes are properties which describe the entities. A particular instance of an attribute is a value. For example, "Schulze" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.
Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

The ER model views the real world as a construct of entities and association between entities.
Example of a ER-Diagram:



While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies specify a bottom-up development process were the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users. The normally used sequence is:

1. Identification of data objects and relationships
2. Drafting the initial ER diagram with entities and relationships
3. Refining the ER diagram
4. Add key attributes to the diagram
5. Adding non-key attributes
6. Diagramming Generalization Hierarchies
7. Validating the model through normalization
8. Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. The requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may uncover problems or missing information which require more information gathering and analysis.

Before transforming the ER-Diagram into tables, we must know that Relational tables have six properties:

1. Values are atomic.
2. Column values are of the same kind.
3. Each row is unique.
4. The sequence of columns is insignificant.
5. The sequence of rows is insignificant.
6. Each column must have a unique name.

For satisfying them we use **Normalization**. It is essentially a two step process that puts data into tabular form by removing repeating groups and then removes duplicated from the relational tables. There are currently five normal forms that have been defined. In this section, we will cover the first three normal forms that were defined by E. F. Codd.

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are (a) mutually independent and (b) fully dependent upon the primary key. Mutual independence means that no non-key column is dependent upon any combination of the other columns. An attribute (or group of attributes) Y, in a relation is said to be functionally dependent on another attribute (or group) X, if knowing the value of X is sufficient to determine the value of Y.The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF.
Simply stated, normalization is the process of removing redundant data from relational tables by decomposing (splitting) a relational table into smaller tables by projection. The goal is to have only primary keys on the left hand side of a functional dependency. In order to be correct, decomposition must be lossless. That is, the new tables can be recombined by a natural join to recreate the original table without creating any spurious or redundant data.

For reaching this aim you should proceed one entity at a time in the normalization steps.

A table is in **First Normal Form (1NF)**, if each attribute has a single well-defined atomic value, not a set of several/repeating values. So each row can be identified uniquely. You must eliminate all repeating groups, i.e., make a separate table for each set of related attributes, and give each table a primary key.

The definition of second normal form states that only tables with composite primary keys can be in 1NF but not in 2NF. A relational table is in **Second Normal Form (2NF)** if it is in 1NF and every non-key column is fully dependent upon the primary key. That is, every non-key column must be dependent upon the entire primary key.
The process for transforming a 1NF table to 2NF is:

a) Identify any determinants other than the composite key, and the columns they determine.
b) Create and name a new table for each determinant and the unique columns it determines.
c) Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
d) Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
e) The original table may be renamed to maintain semantic meaning.

The **Third Normal Form** requires that all columns in a relational table are dependent only upon the primary key. A more formal definition is: A relational table is in third normal form (3NF) if it is already in 2NF and every non-key column is non transitively dependent upon its primary key. In other words, all nonkey attributes are functionally dependent only upon the primary key.
The process of transforming a table into 3NF is:

a) Identify any determinants, other the primary key, and the columns they determine.
b) Create and name a new table for each determinant and the unique columns it determines.
c) Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
d) Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
e) The original table may be renamed to maintain semantic meaning.

The advantages to having relational tables in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies.

After 3NF, all normalization problems involve only tables which have three or more columns and all the columns are keys. Many practitioners argue that placing entities in 3NF is generally sufficient because it is rare that entities that are in 3NF are not also in 4NF and 5NF. They further argue that the benefits gained from transforming entities into 4NF and 5NF are so slight that it is not worth the effort. However, advanced normal forms are presented because there are cases where they are required.

Boyce-Codd Normal Form (BCNF) is a more rigorous version of the 3NF deal with relational tables that had (a) multiple candidate keys, (b) composite candidate keys, and (c) candidate keys that overlapped.
BCNF is based on the concept of determinants. A determinant column is one on which some of the columns are fully functionally dependent.
A relational table is in BCNF if and only if every determinant is a candidate key.
A relational table is in the **Fourth Normal Form (4NF)** if it is in BCNF and all multivalued dependencies are also functional dependencies.
Fourth Normal Form (4NF) is based on the concept of multivalued dependencies (MVD). A Multivalued dependency occurs when in a relational table containing at least three columns, one column has multiple rows whose values match a value of a single row of one of the other columns. So you must isolate the independent multiple relationships, i.e., no table may contain two or more 1:n or n:m relationships that are not directly related.

A table is in the **Fifth Normal Form (5NF)** if it cannot have a lossless decomposition into any number of smaller tables.
While the first four normal forms are based on the concept of functional dependence, the fifth normal form is based on the concept of join dependence. Join dependency means that an table, after it has been decomposed into three or more smaller tables, must be capable of being joined again on common keys to form the original table. Stated another way, 5NF indicates when an entity cannot be further decomposed. 5NF is complex and not intuitive.

After the normalization you can implement the database.

8. Conclusions

Relational databases offer dramatic improvements in productivity for end users and application programmers. The arguments has been data independence, structural simplicity, and relational processing defined in the relational model and implemented in the DBMS. Additional first argument keeps the programs viable in the case of changes. The practicality of the relational system has been proven by the test and production installations. In addition i did explain the concepts of ER-Models and the Normalization for data modeling. With this words I end my letter.

9. References

1. 1981 ACM Turing Award Lecture by E.F.Codd
2. Script Databases by Prof. Dr. Liggesmeyer Sept. 1999
3. Script Databases by Prof. Dr. Weske Sept. 2001
4. http://www.cs.sfu.ca/CC/354/zaiane/material/notes/Chapter7/node1.html
5. http://qsilver.queensu.ca/~comm390/c390lc11.htm
6. http://www.4guysfromrolla.com/webtech/042699-1.shtml
7. http://www.surfermall.com/relational/lesson_4.htm