"the Third Normal Form is usually adequate in dealing with most of the problems which can be encountered in database design."

Normalization in Database Design

by
Prof. Edgardo G. Atanacio*

ABSTRACT

The design of database files can be improved with the use of normalization. Normalization simplifies the structure of database files by grouping the components into simpler structures. Such simple structures adapt better to changes or modifications to the database which are usually necessary to meet changing needs. This paper describes the procedure of converting unnormalized files to the Third Normal Form.

INTRODUCTION

Databases are fast becoming indispensable in many of today's business and scientific operations. Their design seems to have been greatly simplified with the emergence of relational and semi-relational database management software. With the migration of this type of software to the easily accessible microcomputer, anybody can just set up database files for use in storing operational data. However, as applications and requirements become more complex, there is a compelling need for the proper design of the structure of these files. This article attempts to explain one of the most useful tools in database design.

DEFINITIONS

Normalization is a formal process which examines data and simplifies the data item groupings to better accommodate future changes. Normalization theory is built around the concept of normal forms. A data group or structure is in a particular normal form if it satisfies a certain set of constraints.

For our purposes, we will define a database simply as a collection of stored, interrelated data. We will use the terms "entity" and "attribute" to refer to the primary structural components in a database. An entity is anything about which information is stored. It may be tangible or intangible, and it may be a person, a concept, a place, or an object. An attribute is a characteristic or property of an entity. It is a data element describing an entity. An entity occurrence is a certain set of attributes of an entity. In a physical sense, an attribute is equivalent to a field, an entity occurrence is a record, and an entity can be regarded as a file. We will use these terms interchangeably throughout this paper.

*Assistant Professor, Department of Industrial Engineering and Operations Research, U.P. College of Engineering, Diliman, Quezon City, Philippines.

Each entity has to have a primary key. A primary key is an attribute which uniquely identifies a particular occurrence of an entity. For example, in a file containing information about employees, the unique employee number is a good choice as the primary key. Sometimes, a single attribute will not suffice to uniquely identify the records within the file. In such situations, two or more attributes are used together to identify every occurrence. We call such a set of attributes a concatenated key or compound key.

WHY DO WE HAVE TO NORMALIZE?

Normalization is done for the following reasons:

1. To prevent data update problems, referred to as anomalies.
2. To minimize data redundancy and inconsistency.
3. To improve the understanding of the interrelationship and interdependence of data.
4. To provide a basis for the sharing of data.

However, as noted by Kent, normalization will tend to penalize retrieval, and biasedly assumes that nonkey attributes will be updated frequently.

NORMALIZATION PROCEDURE

For all practical purposes, an entity is considered to be normalized when it is in at least the Third Normal Form. There are other higher normal forms, but the Third Normal Form is usually adequate in dealing with most of the problems which can be encountered.

In database design, the first step is to gather data about the data to be stored. In this step, the relevant entities and their attributes are identified. It is also critical that the relationships between the attributes are very well understood and properly documented. We will use the Purchase Order (PO) example to illustrate the steps we will be performing in normalization.

Let us assume that we need to keep data about purchase orders. When we look at our purchase order form, we find the following data in it:

PO-NUMBER: a unique four-digit sequence number which identifies the purchase order.
PO-DATE: the date the PO was written, in the format month/day/year.
BUYER-NAME: a unique three-character initial of the person who wrote the PO.
SUPPLIER-NUMBER: a unique three-digit code assigned to each supplier.
SUPPLIER-NAME: the name of a supplier.
PART-NUMBER: a unique three-digit code identifying a part. Many parts can be ordered in one PO.
PART-NAME: the name of a part.
PO-PART-QUANTITY: the number of units of a part ordered in a given PO.

As a start, we can group all of these attributes into a single entity which we will name PO. We can define this entity using the concise set form. In this notation, attributes are separated by commas, parentheses denote repeating groups, and underscoring denotes keys.

PO (PO-NUMBER, PO-DATE, BUYER-NAME, SUPPLIER-NUMBER, SUPPLIER-NAME, (PART-NUMBER, PART-NAME, PO-PART-QUANTITY))

ZEROTH AND FIRST NORMAL FORMS

We can say that this PO entity is in the Zeroth Normal Form. Figure 1 shows the structure and contents of the sample database.

Definition 1: Zeroth Normal Form

All entities are automatically in the Zeroth Normal Form (ONF).

PO

PO-NUMBER	PO-DATE	BUYER-NAME	SUPPLIER-NUMBER	SUPPLIER-NAME	PART-NUMBER	PART-NAME	PO-PART-QUANTITY
1273	02/05/87	ECD	335	Phoenix	PO2	Nut	30
					PO3	Bolt	20
					PO4	Screw	40
					PO5	Cam	20
					PO6	Cog	40
					PO7	Jig	50
1274	02/05/87	SMB	275	Apex	PO2	Nut	30
					PO3	Bolt	30
1275	02/06/87	LQF	223	Pacific	PO4	Screw	70
					PO6	Cog	40
1276	02/09/87	ECD	230	Vulcan	PO3	Bolt	20
					PO5	Cam	30
					PO6	Cog	40
1227	02/10/87	HIG	208	Super	PO6	Cog	50
1278	02/10/87	HIG	335	Phoenix	PO8	Rivet	30

Figure 1. Purchase Order Example, Zeroth Normal Form

Some entities in the Zeroth Normal Form will suffer some problems if implemented without any change in structure. If we disallow the use of variable repeating fields, the entity will have to be implemented using a fixed number of fields. If implemented as a single file, design problems will occur with respect to storage considerations. For instance, in our example, we will have to allocate a maximum number of the PART-NUMBER, PART-NAME, PO-PART-QUANTITY repeating group. A proper upper limit is hard to set. Too small an allocation value will result in the problem of some parts which cannot be accommodated. Too large a value will result in wastage of storage space. The solution to this predicament is to transform the entity from the Zeroth Normal Form to the First Normal Form.

Definition 2: First Normal Form

An entity is in First Normal Form (1NF) if all its attributes are atomic, that is, there are no repeating groups.

To transform our example from 0NF to 1NF, the repeating groups must be removed. We need to split the original entity into several entities, and we need to copy the primary key of the first entity into the new entities, and then determine what the new keys should be. For our example, we split the original PO entity into two entities PO and PO-PART. Entity PO-PART contains the items which existed as the repeating group in the original PO. Note that we have to have a concatenated primary key consisting of PO-NUMBER and PART-NUMBER to uniquely identify each occurrence of PO-PART.

First Normal Form:

PO (PO-NUMBER, PO-DATE, BUYER-NAME, SUPPLIER-NUMBER, SUPPLIER-NAME)
PO-PART (PO-NUMBER, PART-NUMBER, PART-NAME, PO-PART-QUANTITY)
Figure 2 shows the purchase order database in the First Normal Form.

1NF UPDATE ANOMALIES

The structure of our entity set is still undesirable in the current form. Consider the effects of the following operations to be performed:

- Insert: Add a new unordered part to the set of parts. Since the part is yet unordered, we cannot keep the part information in either PO or PO-PART entity since there is no PO-

PO

PO-NUMBER	PO-DATE	BUYER-NAME	SUPPLIER-NUMBER	SUPPLIER-NAME
1273	02/05/87	ECD	335	Phoenix
1274	02/05/87	SMB	275	Apex
1275	02/06/87	LQF	223	Pacific
1276	02/09/87	ECD	230	Vulcan
1277	02/10/87	HIG	208	Super
1278	02/10/87	HIG	335	Phoenix

PO-PART

PO-NUMBER	PART-NUMBER	PART-NAME	PO-PART-QUANTITY
1273	PO2	Nut	30
1273	PO3	Bolt	20
1273	PO4	Screw	40
1273	PO5	Cam	20
1273	PO6	Cog	40
1273	PO7	Jig	50
1274	PO2	Nut	30
1274	PO3	Bolt	30
1275	PO4	Screw	70
1275	PO6	Cog	40
1276	PO3	Bolt	20
1276	PO5	Cam	30
1276	PO6	Cog	40
1277	PO6	Cog	50
1278	PO8	Rivet	30

Figure 2. Purchase Order Example, First Normal Form

NUMBER which can be used because no PO has been issued for that part. For instance, we cannot add a part with part number "PO1" and part name "Washer" if that part has not been ordered yet.

- Delete: Remove the PO-PART record for a part already received. This may not always create a problem. However, let us try to delete the record with PO-NUMBER "1273" and PART-NUMBER "PO7". When the delete operation has been carried out, all information about the deleted part will have been lost, since this was the only occurrence of that part.
- Change: Change the part name. Let us consider changing the part name of part number "PO6" from "Cog" to "Gear". Since this part occurs several times within the PO-PART entity, we have to look for and change all occurrences of the part name. If we overlook even just a single occurrence, we lose the integrity of our data.

The preceding problems are called update anomalies, and to be able to remove such anomalies present in the 1NF, we have to understand first the concept of functional dependence.

Definition 3: Functional Dependence

For any entity R, attribute Y is functionally dependent on attribute (or collection of attributes) X if the value of X determines the value of Y. In short, X identifies Y.

Attribute Y is functionally dependent on X if it is invalid to have more than one occurrence with the same X value but different Y values. That is, a given X value must always occur with the same Y value.

For example, EMPLOYEE-NUMBER, EMPLOYEE-NAME, EMPLOYEE-ADDRESS, and EMPLOYEE-SALARY are attributes of the entity EMPLOYEE. If, given a particular value of EMPLOYEE-NUMBER, there exists precisely one corresponding value for each of EMPLOYEE-NAME, EMPLOYEE-ADDRESS, and EMPLOYEE-SALARY, then EMPLOYEE-NAME, EMPLOYEE-ADDRESS, and EMPLOYEE-SALARY are said to be functionally dependent on EMPLOYEE-NUMBER.

It is easier to understand functional dependencies if a device called a functional dependency diagram (FDD) is used. The relationships between the attributes of each entity are examined and a diagram is drawn for each entity structure. An attribute is shown within a rectangular box. A directed arrow is used to connect related attributes. The tail end comes from the attribute upon which the attribute being pointed to is dependent. The functional dependency diagram of PO-PART is shown in Figure 3. Note the use of the larger box to designate the whole concatenated key and of the underscore to identify the primary key attributes.

Definition 4: Full Functional Dependency

An attribute (or a collection of attributes) Y of an entity R is fully functionally dependent on another collection of attributes X of R, if Y is functionally dependent on the whole of X but not on any subset of X.

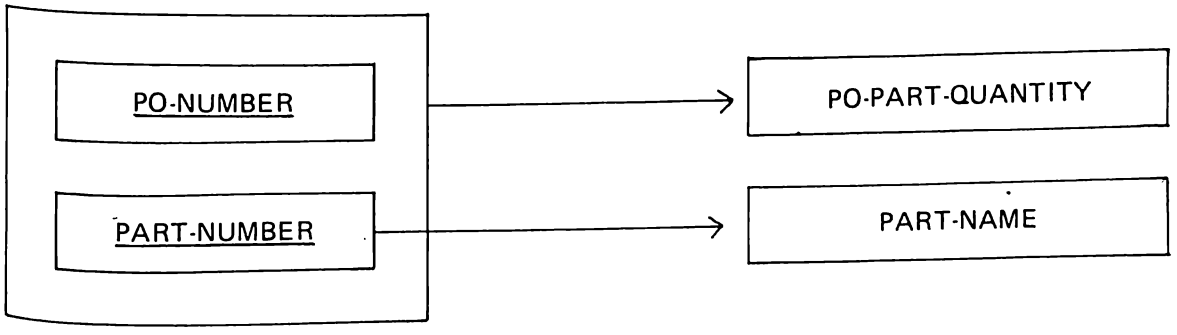

Figure 3. Functional Dependency Diagram. PO-PART

SECOND NORMAL FORM

Definition 5: Second Normal Form

An entity is in Second Normal Form (2NF) if:

1. It is in First Normal Form, and
2. All the nonkey attributes are functionally dependent on the whole primary key (full functional dependency upon the primary key).

If we analyze our purchase order entity set, we find that entity PO is automatically in the Second Normal Form since it has only a single attribute as the primary key. The PO-PART entity is not in the Second Normal Form because PART-NAME is functionally dependent on just PART-

NUMBER and not on the whole concatenated key PO-NUMBER and PART-NUMBER. To convert this to the Second Normal Form, we have to split the PO-PART entity into two entities PO-PART and PART, giving the following structures:

Second Normal Form:

PO (PO-NUMBER, PO-DATE, BUYER-NAME, SUPPLIER-NUMBER, SUPPLIER-NAME)
PO-PART (PO-NUMBER, PART-NUMBER, PO-PART-QUANTITY)
PART (PART-NUMBER, PART-NAME)

The purchase order database in the Second Normal Form is shown in Figure 4.

In converting an entity from 1NF to 2NF, we have to transfer to a new entity the attribute or attributes which are dependent on just part of the concatenated key, copy into this entity the attribute they are dependent on, and make this attribute the primary key of the new entity.

PO

PO-NUMBER	PO-DATE	BUYER-NAME	SUPPLIER-NUMBER	SUPPLIER-NAME
1273	02/05/87	ECD	335	Phoenix
1274	02/05/87	SMB	275	Apex
1275	02/06/87	LQF	223	Pacific
1276	02/09/87	ECD	230	Vulcan
1277	02/10/87	HIG	208	Super
1278	02/10/87	HIG	335	Phoenix

PART

PART-NUMBER	PART-NAME
PO1	Washer
PO2	Nut
PO3	Bolt
PO4	Screw
PO5	Cam
PO6	Cog
PO7	Jig
PO8	Rivet

PO-PART

PO-NUMBER	PART-NUMBER	PO-PART-QUANTITY
1273	PO2	30
1273	PO3	20
1273	PO4	40
1273	PO5	20
1273	PO6	40
1273	PO7	50
1274	PO2	30
1274	PO3	30
1275	PO4	70
1275	PO6	40
1276	PO3	20
1276	PO5	30
1276	PO6	40
1277	PO6	50
1278	PO8	30

Figure 4. Purchase Order Example, Second Normal Form

2NF UPDATE ANOMALIES

Looking over our entity set, we find that there are still some update anomalies remaining. The following operations cannot be serviced properly yet:

- Insert: Add a new supplier. Again, unless a supplier has been specified in a PO, details about that supplier cannot be made part of the database. Thus, only suppliers currently supplying parts in POs can be included in the database.
- Delete: Remove a purchase order. If we delete a purchase order containing the last occurrence of a supplier, we will lose all information about that supplier once the delete operation has been carried out.
- Change: Change supplier name. If we try changing the name of a supplier who has supplied a lot of parts, then we have to search for and change all occurrences of that supplier name. Neglecting to change all of the occurrences will inject inconsistencies within the database.

These update anomalies are due to a relationship known as transitive dependence.

Definition 6: Transitive Dependence

Given three attributes (or collections of attributes) X, Y, and Z, if Z is functionally dependent on Y, and Y is functionally dependent on X, then Z is functionally dependent on X. Z is said to be transitively dependent on X.

Transitive dependence occurs when a nonkey attribute identifies other attributes.

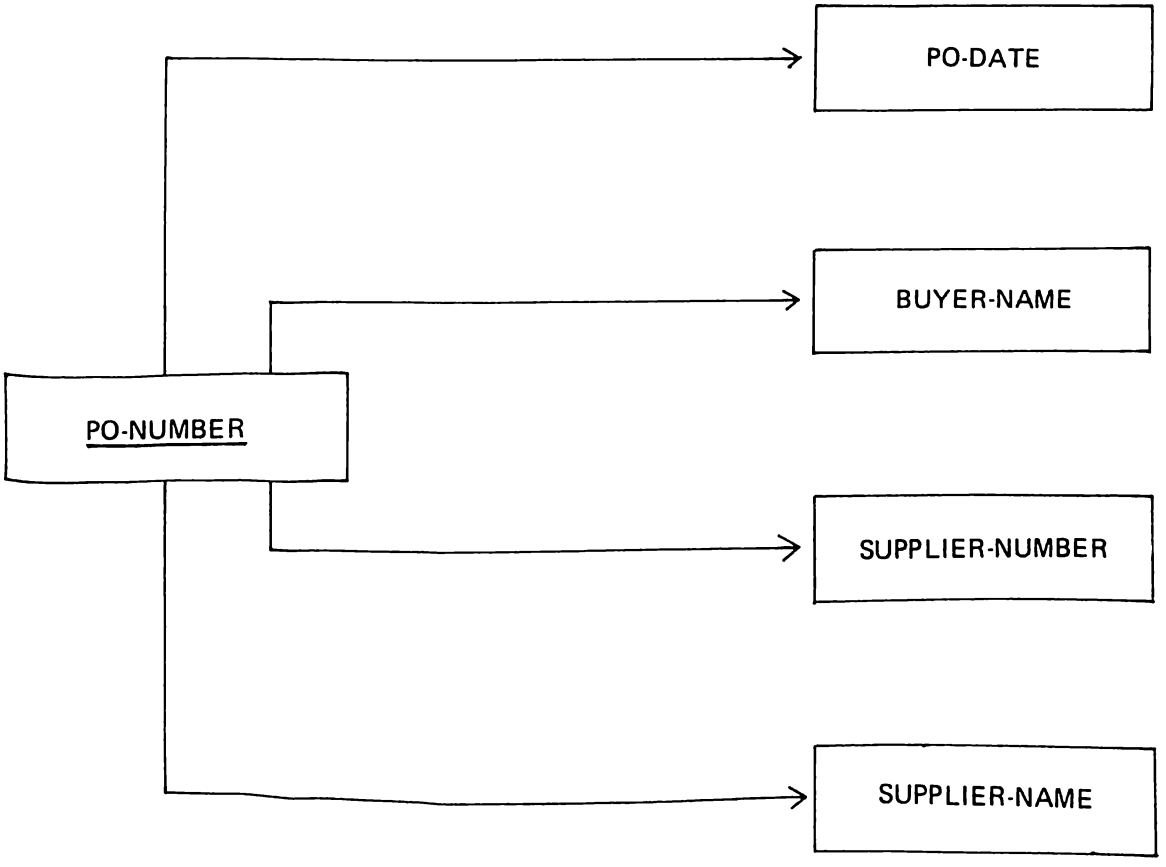

Figure 5. Functional Dependency diagram, PO

PO

PO-NUMBER	PO-DATE	BUYER-NAME	SUPPLIER-NUMBER
1273	02/05/87	ECD	335
1274	02/05/87	SMB	275
1275	02/06/87	LQF	223
1276	02/09/87	ECD	230
1277	02/10/87	HIG	208
1278	02/10/87	HIG	335

SUPPLIER

SUPPLIER-NUMBER	SUPPLIER-NAME
208	Super
223	Pacific
230	Vulcan
275	Apex
335	Phoenix

PART

PART-NUMBER	PART-NAME
PO1	Washer
PO2	Nut
PO3	Bolt
PO4	Screw
PO5	Cam
PO6	Cog
PO7	Jig
PO8	Rivet

PO-PART

PO-NUMBER	PART-NUMBER	PO-PART-QUANTITY
1273	PO2	30
1273	PO3	20
1273	PO4	40
1273	PO5	20
1273	PO6	40
1273	PO7	50
1274	PO2	30
1274	PO3	30
1275	PO4	70
1275	PO6	40
1276	PO3	20
1276	PO5	30
1276	PO6	40
1277	PO6	50
1278	PO8	30

Figure 6. Purchase Order Example, Third Normal Form

THIRD NORMAL FORM

Definition 7: Third Normal Form

An entity is in Third Normal Form (3NF) if:

1. It is in Second Normal Form, and
2. No nonkey attribute is functionally dependent on any other nonkey attribute (no transitive dependence).

Again, looking at our purchase order example, we find that the entities PART and PO-PART are already in the Third Normal Form. However, the PO entity has a complexity remaining due to the relationship between the nonkey attributes SUPPLIER-NUMBER and SUPPLIER-NAME. This is clearly shown in the functional dependency diagram of the entity PO in Figure 5. Given just the SUPPLIER-NUMBER we can determine the SUPPLIER-NAME. To remedy the situation, we can split the entity PO into two entities PO and SUPPLIER.

Third Normal Form:

PO (PO-NUMBER, PO-DATE, BUYER-NAME, SUPPLIER-NUMBER)
SUPPLIER (SUPPLIER-NUMBER, SUPPLIER-NAME)
PO-PART (PO-NUMBER, PART-NUMBER, PO-PART-QUANTITY)
PART (PART-NUMBER, PART-NAME)
The sample database in its final form is shown in Figure 6.

CONSOLIDATION STEP

In more complex situations, there is a process of consolidation after every transformation. The purpose of consolidation is to remove the redundancies which can appear when the entities are split. Our purchase order example is, however, too simple to illustrate this step. But in general, to consolidate the entities after every stage of the transformation:

1. Examine all entities and determine which entities have exactly the same primary key (for both singular or concatenated keys).
2. Delete all entities which are exactly identical to other existing entities.
3. Combine all remaining entities with the same primary keys into new entities.
4. Remove all redundant attributes within the same entity.

As we can see in the final set of entities which are in at least the Third Normal Form, we can simplify the structure of database files by systematically grouping them into smaller and simpler entities or files. With such simpler structures, we can adequately cope with modifications due to changing future requirements. The cost of this flexibility will be in terms of access and retrieval effort and time. In our example, we ended up with four entities coming from a single entity. If we implement this physically, we may have to access all four files to get and present the information required while if we use the original one-file design we will have to look at only one file. However, such tradeoff will be necessary if we want to be able to service future requirements with the minimum of problems.

REFERENCES

BRADLEY, J. (1982), *File and Data Base Techniques,* Holt, Rinehart and Winston, New York, N.Y.

DATE, C. J. (1981), *An Introduction to Database Systems,* 3rd ed., Addison-Wesley Pub. Co., Reading, Mass.

KENT, W. (1983), "A Simple Guide to Five Normal Forms in Relational Database Theory." *Communications of the ACM* (February 1983), pp. 120-25.

KROENKE, D. (1977), *Database Processing: Fundamentals, Modelling, Applications,* Science Research Associates, Inc., Chicago.

MARTIN, J. (1983), *Managing the Data-Base Environment,* Prentice-Hall, Inc., Englewood Cliffs, N. J.

ROBINSON, H. M. (1981), *Database Analysis and Design,* Chartwell-Bratt, Kent, England.

ULLMAN, J. D. (1980), *Principles of Database Systems,* Computer Science Press, Inc., Potomac, Maryland.