Forms Approach to Requirements Specification for Database Design

Nan C. Shu
IBM - Los Angeles Scientific Center

Harry K.T. Wong*
Lawrence Berkeley Laboratory

Vincent Y. Lum
IBM - Heidelberg Scientific Center

ABSTRACT

One of the most difficult tasks in database design is the collection of relevant information needed for database design. This information can be separated into two categories: information about the *processes* which use, modify or produce the data, and information about the *data* which is used, modified, or produced by the processes. This paper describes a "forms" oriented requirements specification facility as a formal means for capturing the information crucial for database design. Both kinds of information are expressible by the proposed structured forms specification. Furthermore, not only the application usage, but also the integrity constraints that must be maintained in the database can be captured in a concise, unambiguous and machine manipulable manner. The concepts underlying the forms approach are few and simple, hence it is easy to use for the database designer and can be understood by non-computer specialists. This facilitates the cooperation from users to participate in the specification of database application requirements.

1. INTRODUCTION

During the past decade, we have seen a dramatic advancement in the development and applications of database management systems. Today, not only the databases themselves have become very large, but the applications on databases have become very complex. Database design technology, however, has not advanced enough to keep pace with the rapid growth. As the 1978 New Orleans Data Base Design Workshop Report [Lum79] pointed out, in spite of the large number of papers published in this area (see [Lum79A] for a comprehensive list), database design currently is still "an art practiced by few with few tools other than the designer's experience and intuition. The sophistication of the applications has made their task more and more difficult."

This predicament is, in part, due to the fact that, until recently, most of the studies and tools were mainly concerned with the physical database design. Unfortunately, physical design is only the last step of the many aspects in the database design process. As a matter of fact, physical design tools generally assume the existence of a schema design and ignore the necessity of enforcing integrity constraints. They rely on the human database designer to provide the schema and depend on the application programs to enforce constraints. More recently, efforts from many universities and research laboratories have contributed to the understanding of schema design, as well as semantic and constraints issues (see [NYU SYM 78] and [ERA CONF 79] for example). As yet, very little attention has been given to ease the requirements specification, which is known according to industry experiences to be the most difficult and the most crucial phase in the database design efforts.

The reason for the difficulties in the requirements specification phase is that because database design in general involves trade-offs and decisions which require detailed knowledge not only about the characteristics of data but also about the existing and projected processes that operate on the data. Thus, the information gathered at the application specification stage has a direct and profound impact on the final product of the database design.

Currently, the common practices range from very informal interviews to document or code reading. These practices are time consuming and labor intensive. Ambiguities, fuzziness, misunderstandings, and either over-specification or under-specification of an application usually occur. The designer has to spend needless time trying to filter out the nonessential details and has to rely on intuition to guess the intents of the information suppliers.

The importance of requirements specification for information system design has long been recognized. Computerized tools have been built for this purpose. The best known work is perhaps the Problem Statement Language/Problem Statement Analyzer (PSL/PSA) developed at the University of Michigan [Teichroew71, Teichroew77]. TELL, a system developed at IBM San Jose Research Laboratory [Hebalkar79] is an example of more recent efforts. However, these works are primarily aimed at describing and documenting information systems from a very general standpoint. As such, they are not particularly suited for database design. A serious handicap is the lack of process description capability appropriate for both the users and the database designers. (In the context of this paper, users are the application specialists, application programmers and end users.) Another handicap is the lack of facility to state integrity constraints specific to databases, for instance, that an employee must have a department number; that a department number determines department name and location, etc. Since these constraints must be enforced in a database, they must be included in the information to be captured.

In this paper, we propose a facility for requirements specification that is designed to capture information relevant to database design in a concise and unambiguous manner. Section II outlines our approach. Section III introduces a forms-oriented application specification. Sections IV and V discuss the specification in detail. The final section summarizes the paper and discusses some future work.

* Supported by the Office of Energy Research, U.S. DOE under Contract No. DE-AC03-76SF00098.

2. APPROACH

Our approach to the requirements specification facility is to first examine the kinds of information that are essential for database design, then design a *formal* means of specification that can be used to encode these information. That is, the richness of the notation (or language) is determined by the kinds of information that has impact on the database design. Concurrency and recovery issues are not included in our consideration since their impact on database design is not as important as other information. Any aspects which are purely "system dependent" are also not considered. For instance, we are not interested in the capture of the characteristics of the hardware on which the database is to be implemented. Nor are we interested in the storage space management algorithms, the access path selection strategies, the availability of indexing capabilities, etc.

There are basically two groups of information we have identified to be needed for database design in a a system independent manner. The first group of information is about data and integrity constraints on data. Integrity constraints define "legal states" of a database which have to be maintained. The maintenance, however, requires effort either from the database system or the application programs. The estimate of these efforts should be part of the considerations for any database design.

The second is about anticipated processes which use, modify, or produce the data. This kind of information is important for the efficiency considerations as recognized by most database design tools.

More specifically, the following information are essential for database design.

-- on data:

  -- (1 to 1), (1 to m), (m to n) relationship
  -- integrity constraints
    -- keyness
    -- functional dependencies
    -- set inclusion and exclusion
    -- applicability of null values
    -- value constraints
  -- statistics
    -- data size
    -- data volume
    -- distribution of data values
  -- authorization constraints

-- on processes:

  -- What data items are needed for what processes?
  -- What data items are produced by what processes?
  -- How often is a collection of data items referenced together?
  -- What type of processing?
    -- produce reports
    -- retrieve
    -- update
    -- insert
    -- delete
  -- Is ordering of instances requested?
  -- Is combining of two or more data sets required?
  -- Is it an on-line or batch operation?
  -- What is the execution frequency?
  -- Is there any time constraint?

We believe in a means of formal specification to capture this information because a formal specification would be unambiguous. But more importantly, the information encoded in such a specification would be machine manipulable and more amendable to analysis.

In addition, specifications should be easy to formulate by the database designer and easy to understand by the users since the specification usually involves communication with users who are not necessarily computer experts. Furthermore, requirements specification is a starting point of the database design process -- a process which is iterative in nature. At each iteration some of the specifications may have to be revised; therefore, it is important that the specifications be easily modifiable. Finally, in order to provide a sound basis for estimation of processing costs, process specifications should be transformable into corresponding data manipulation actions in the underlying database system.

The formal specification that we have developed to capture the required information is forms oriented. The decision to take the forms oriented approach is prompted in part by the recommendations in the February 1979 Status Report of the CODASYL End-User Facilities Committee (EUFC) [Lefkovits79]. The committee has reported that "The forms approach was considered the most natural interface between an end user and data because a large number of end users employ forms (e.g., purchase order forms, expense report forms, etc.) or versions of forms (e.g., reports, memos, etc.) in their daily work activities as well as in their personal lives (e.g., tax forms, employment application forms, etc.)." An example of one such usage is the query language, Query-By-Example [QBE78].

3. FORMS ORIENTED REQUIREMENTS SPECIFICATIONS

In this section, we present a formal means of requirements specification that is forms oriented. This facility has been applied to the area of office automation as described in [Shu82]. The example of the distribution company in that paper will also be used to illustrate how the required information for database design can be captured.

Before we delve into details of our proposal, it is important to note that we view each *process* as a "basic application unit." To clarify this notion, we introduce the following terminologies.

FORMS

A FORM is an information holding object consisting of two parts: (1) a *form heading* which describes form name, form structure and component names; and (2) one or more *form instances* (or form occurrences). The form heading assumes a role that is commonly known as data structure definition or schema definition in the data processing community. It is a natural way to represent hierarchies. The collection of all form instances belonging to a particular form of a given name is known as a *file* of that names (e.g., collection of all instances of EMPLOYEE form is known as EMPLOYEE file). When there is no ambiguity, we use the terms form and file interchangeably. A basic concept underlying our approach to requirements specification is the premise that most of the common data processing applications can be viewed as manipulations of forms.

BASIC APPLICATION UNITS

A BASIC APPLICATION UNIT is defined as an activity which either produces or modifies a form. Input to a basic application unit, however, is not restricted to one form. A simple application consists of only one basic application unit. A more complex application may consist of several.

To illustrate this point, let us examine some of the applications of a distribution company. This company purchases products from vendors, stores them and resells them to customers. The database maintains customer, vendor and product information, and keeps track of stocks, orders, etc. Some of the applications are simple. For instance, a monthly report that is generated to show the suppliers and storage locations of each product is a basic application unit. It produces one form. Inserting a new customer into customer information file is another basic application unit. It modifies one form. Sending out a product to a customer, on the other hand, involves two processes: (1) reduce the quantity on hand for that product in the stock file according to the quantity sent out; (2) issue an invoice to the customer. The first process modifies a form, the second produces a form. It is not hard to envision applications that are

more complex than the ones we have just described. In these cases, the applications will simply have to be decomposed into a set of basic application units.

Decomposing a complex application into basic application units is not a difficult task. We can depend on application specialists to do the decomposition since, intentionally or not, it is a mental process that an application specialist normally goes through when an application is analyzed. One may even argue that the requirement to decompose a complex application into basic application units imposes a discipline on structured approach to application analysis.

Earlier we have mentioned that the process specifications should be translatable into data manipulation actions in the underlying database system so that an estimate of processing costs can be made. Our experience ([Wong & Shu 80]) suggests that basic application units correspond rather closely to queries and data manipulation actions in languages such as SQL [Chamberlin79] or CONVERT [Shu75].

Even more important than the possibility of translation is the ease of specification when there is only a single form to be described for each process. Since a person normally deals with one form at a time, this notion of having a basic application unit as a unit of specification is supported quite naturally in the forms oriented approach.

FORM HEADING

FORM HEADING plays an important role in the application specification. It represents the form name, the components of the form, and the structure of the form.

Figure 1 shows the form heading of a form named PRODUCT. The top line contains the form name. Components of the form are represented by their names. The form structure is represented as follows: item names are represented in the columns; group names are placed on top of their components; parentheses are used to denote repeating components; nesting of repeating groups are represented as levels of parenthesized group names; parenthesizing the name of the outer-most group (i.e., the form name) indicated that the form may have more than one instance. As a convention, we use a double line at the bottom of the form heading.

A corresponding hierarchy graph (used by some database designers to show structural relationships) is also included in Figure 1. It should be obvious that the hierarchical relationships are exhibited in the form heading via parenthesized expressions. The mapping into the corresponding hierarchy graph is straightforward. One must not get the wrong impression, however, that forms are only for hierarchies. Relational tables are flat forms (i.e., one-level forms), and a CODASYL DBTG [DBTG71] type of network can be represented as a collection of forms, where each form representing a record type in the network, and the relationships between forms can be established by referring to forms by their unique names.

SCHEMATIC DIAGRAM FOR FORMS ORIENTED SPECIFICATION

As mentioned earlier, there are two categories of information to be captured for database design purposes. One category is on data description, the other is on process description. In general, these two categories of information can be provided by two different classes of people. The first category, i.e., information on data, can be specified by database administrators or persons who are familiar with data (but not necessarily processes). The second category, i.e., information on processes, are specified by application specialists, application programmers, and end users. They know what they want to accomplish with the process, but are not knowledgeable about integrity constraints and statistics about data. We, therefore, separate these two categories of specifications to accommodate the two classes of people, but provide a unified representation for both.

Figure 2 contains schematic diagrams for both categories of specifications. Note that the space above the double line and to the left of the form heading is used to indicate whether the specification is pertaining to data (DESCRIPTION) or to process (operations such as INSERT, DELETE, UPDATE, QUERY and PRINT). The space below the double line is used to specify constraints and statistics in case of data specification or to specify the process environment and qualifications for the selected operation in case of process specifications. The applications of these diagrams to example data and processes are described in subsequent sections.

To illustrate the forms oriented application specifications, we use the distribution company (described earlier) as an example. Five files are currently maintained by this company: CUST_INFO, VENDOR_INFO, PRODUCT, STOCK and ORDERLIST. To cope with increasing demand on data processing activities, a database management system is being installed, and a database is being designed. Essentially, the data about customers, vendors, products, stock, and orders that exist at present will be maintained in the database. In addition, integrity and authority constraints on the data will be enforced, and expansion of the applications (especially in the area of on-line queries and report generations) are anticipated.

In the following two sections, we shall discuss how these specifications can be made so that a database designer will have concrete and necessary knowledge upon which to base his design.

4. DATA SPECIFICATIONS

The first step of data specification is to formulate the form heading for each file. Formulation of form headings is quite straightforward. Figure 3 shows the form headings of the five files maintained by the example distribution company.

As a second step, the form headings are augmented with specifications on data constraints and statistics. Figure 2(a) shows a list of keywords designated to describe the constraints and statistics. Figure 4 illustrates how these are applied in a specific example. Discussions are given in subsequent subsections.

KEY

A KEY denotes an item or a collection of items whose value uniquely identifies an instance within a group (keeping in mind that a form can be viewed as an outer-most group). Thus, when applied to the top level item(s) of a form, the value of the key uniquely identifies a form instance. When applied to the item(s) of a repeaing group at lower level, it uniquely identifies an instance of the group under the parent instance. In the example of Figure 4, an "*" under PROD_NO in the KEY row specifies that PROD_NO is the unique identifier of PRODUCT form instances. The example also shows that for a given instance of PRODUCT (the parent of both SUPPLIER and STORAGE), VNAME uniquely identifies an instance of SUPPLIER, while BIN_NO uniquely identifies an instance of STORAGE.

There are situations where alternate keys are possible. In that case, each of the alternatives can be specified in one row. In Figure 5, SS_NO and ENO are alternate keys for EMPLOYEE instances while the combination of DNO and PJNO serves as the key for PROJ instances within an EMPLOYEE instance.

UNIQUENESS

UNIQUENESS conveys the fact that every instance of the specified item in the form has a distinct value. Thus, an "*" under PROD_NO (in Figure 4) in the UNIQUENESS row indicates that every PROD_NO instance in the PRODUCT form has a unique value. This agrees with the specification that PROD_NO is the KEY of the PRODUCT form instances.

For items of repeating groups below the top level, however, this constraint does not necessarily hold. Since the KEY of a repeating group merely serves as a unique identifier for the group occurrences within a parent instance, values of the repeating group key may or may not be unique across all of the form instances.

In the example shown in Figure 4, we observe that BIN_NO is marked a key (of STORAGE group). Since STORAGE is a subordinate group, the relationship between PROD_NO and BIN_NO (the KEY items of the pertinent hierarchy) can be either (1:n) or (m:n). The fact that BIN_NO is unique across all PRODUCT form instances (as shown by "*" under BIN_NO in the UNIQUENESS row) narrows down the relationship to (1:n). In the case of SUPPLIER, on the other hand, we observe that VNAME is the KEY of SUPPLIER group for a PRODUCT instance, yet there is no "*" under VNAME in the UNIQUENESS row. This tells the database designer that the same value of VNAME may appear in different instances of PRODUCT. A conclusion can be drawn that there is a (m:n) relationship between the key items PROD_NO and VNAME.

DETERMINACY

Inherent in the hierarchical relationship is the implication that a group key, concatenated with its ancestor keys if any, determines the non-key items of the same group. Take the PRODUCT form (in Figure 4) for example. It is inherent in the hierarchical structure (together with its keys) that the key item PROD_NO determines PNAME, TYPE, and PRICE. Similarly, the hierarchy and KEY information implies that PROD_NO concatenated with BIN_NO determines LOC. DETERMINACY specification can be used to supplement or override the implied information. More than one row can be used for that purpose, if necessary. In each DETERMINACY row, the item(s) marked with "*" determines the item(s) marked with "->". This specification corresponds to the notation commonly used to express functional dependency, i.e., BIN_NO -> LOC. In our example, the DETERMINACY row specifies that BIN_NO alone (regardless of the value of PROD_NO) determines LOC. Note that this information can be derived from the UNIQUENESS specification in this particular example. Nevertheless, the DETERMINACY specification is a general facility to explicitly capture this information.

SET INCLUSION

SET INCLUSION is used to make known the fact that the values of an item in the pertinent form *must exist* in the file specified in the SET INCLUSION row. The example in Figure 4 specifies that all instances of VNAME occurring in PRODUCT file must exist in VENDOR_INFO file. This specification imposes another type of integrity constraint to be enforced on the database.

SET EXCLUSION

Contrary to SET INCLUSION, SET EXCLUSION represents the fact that the values of an item in the pertinent form *must not exist* in the file specified in the SET EXCLUSION row. The example in Figure 4 specifies that the instances of PROD_NO in PRODUCT form must not exist in the OUTDATED_PROD form. Similar to SET INCLUSION, SET EXCLUSION also imposes an integrity constraint to be enforced.

NULL

An "*" under an item in the NULL row specifies that a null value is allowed for the item. In the example shown in Figure 4, NULL is allowed for PNAME and TYPE, but not for other items. Disallowance of NULL value imposes a constraint that must be enforced on the database.

VALUE

VALUE specification can be used to provide information on data distribution (if a "%" value is present), or on data value constraints (if a "%" value is absent).

In Figure 4, the entry in the VALUE row under LOC specifies that the value of LOC is constrained to be either "SFO" or "SJC". Any other value is invalid. Like all other integrity constraints that we have discussed so far, value constraints impose an enforcement cost.

Under PROD_NO, the distribution of value is specified. Ten percent of PROD_NO value is less that 100, 10% is larger than 999, and the other 80% is assumed to have the remaining values. This specification gives the database designer a better basis to compute the cost of supporting applications than just assuming, for example, a uniform distribution.

OCCURRENCE

The OCCURRENCE specification is used to provide data volume information. When applied to a top level item, the specified number represents the number of instances in the file. When applied to an item in a repeating group, the specified number represents the average number of occurrences within a parent instance. The example in Figure 4 shows that there are 1000 occurrences of PROD_NO in the PRODUCT form; there are, on the average, 16 instances of VNAME and 10 instance of BIN_NO for each occurrence of PRODUCT.

DATA TYPE

Data size information can be derived from DATA TYPE specifications. Various notations can be adopted to describe different types of data. As an example, Figure 4 shows the notation adopted from the COBOL PICTURE description. Note that if a corporation has data dictionary facility available, data type information can normally be kept there. We are including data type specification in our example merely to show that this information should be included in the requirements specification for database design purpose since data size is one of the many factors impacting database schema design.

AUTHORITY LEVEL FOR READ OR WRITE

Specification of AUTHORITY LEVEL for READ (or WRITE) signifies which user(s) or application program(s) can read (or write) a particular item in the form. The example in Figure 4 shows that only persons or application programs having authority level greater than 5 can modify or write PRICE values. Authorization constraints have an impact on grouping items into records. For instance, if a frequently executed process is authorized to access A and B, but not C, it may be advantageous to store (A,B) and (A,C) as separate records even though both B and C are functionally dependent on A.

5. PROCESS SPECIFICATIONS

To define a process, the first step is to formulate the form heading for the target (output) form. The form heading is then augmented with process descriptions.

Process Description includes the specification of a process name, operation, qualifications on the operation, and process environment.

While we discuss the process specification, it is important to note that the data restructuring type of operations which normally account for a large percentage of data processing activities are *implied* in the transformation of input to output form heading. Most common types of data restructuring operations include creating new items, flattening or expanding levels of hierarchies, rearranging the juxtaposition of items, combining different items from two or more files into one file, extracting some components from a file, etc. The implied transformation of form structures greatly simplifies the mechanics of process specification.

Figure 2(b) shows a schematic diagram for what can be specified for a process. An explanation of each of the specification clauses is given in the following discussion.

5.1. Process Name

The process name is used to uniquely identify a process. A process name can be the same as its output form name as long as no two process names are the same.

5.2. Operations

Above the double line and to the left of the form heading, the type of operation which produces or modifies the form instances is specified. Five types of operations are possible. They are: INSERT, DELETE, UPDATE, PRINT, and QUERY. A brief description of each type of operation is given below. Examples of these operations follow the discussion on qualifications.

INSERT

INSERT is the operation that inserts form, group or item instances into a file. Note that the INSERT operation is used in a broad sense. Not only can one insert instances into an existing file, one can also create a new form by inserting instances into an empty form.

DELETE

DELETE is the operation that deletes form or group instances, or non-key item instances from a file.

UPDATE

UPDATE is the operation that modifies some item values in a file.

PRINT

PRINT causes a hard copy print-out of a form, which can be either extracted from a form existing in the database (e.g., VENDOR_REPORT in Figure 6(d)), or from a transient form created on the spot for the purpose of printing or reporting (e.g., INVOICE form in Figure 6(c)).

QUERY

QUERY causes the contents of a form to be displayed at a terminal. Similar to PRINT, the pertinent form can be either in existence or can be created temporarily for the purpose of display.

5.3. Qualifications

Under the operation specification, qualifications on the operation and process environment are described. The purpose of the qualifications is to provide more specific descriptions to the operation specified. Qualifications may include information such as the source of data (SOURCE), ordering of form instances within the file and/or ordering of group instances within its parent instance (ORDER), retaining or elimination of duplications (NODUP), conditions applied to input/output values while making up a form instance (CONDITION), and deletions effected (DELETION).

To illustrate these various qualifications, we again use the distribution company as an example. In particular, we concentrate on the activities related to order entries. There are approximately 1000 new orders from customers every day. When a customer order is received, it is entered in a form called NEWORDER (Figure 6(a)). It is the company's policy that each item on an order should be picked up from one storage bin. A PICKUP list is generated for that item (Figure 6(b)), and the same item is deleted from NEWORDER. After the customer picked up the ordered item, the STOCK file is updated to reflect the reduction in the quantity of the item (Figure 6(c)), and the item is deleted from the PICKUP list (Figure 6(c)).

This rather involved order entry application can be decomposed into seven basic application units. They are named NEWORDER, ENTER_ORDER, PICKUP, REDUCE_STOCK, INVOICE, DELETE_ORDER, and DELETE_PICKUP, respectively.

There are, of course, many other applications in this company. All applications (existing or projected) will be taken into consideration by the database designer. For the purpose of illustration, however, we shall only cite two more:

(1) Vendor inquiry (on the average of 100/day):
For a given vendor, list its address, and the price of a particular product the vendor supplies (Figure 6(d)).

(2) Vendor report (once per year):
For each product in the VENDOR_INFO file, list vendor name, address and vendor price on each vendor that can supply the product (Figure 6(d)).

Specifications of the above mentioned processes (except DELETE_ORDER, which does not add any new concept) are shown in Figures 6(a) through 6(d).

We shall now discuss the qualifications on operations in more detail.

SOURCE

SOURCE specifies how or where to obtain the instances relevant to the operation. For PRINT and QUERY, source must be specified for all items of the form. For INSERT and UPDATE, source denotes respectively by the *new* instances to be inserted and the instances to be used as *replacements*. For DELETE, the source need not be specified. The following are the various types of sources that can be specified.

(1) Dashes (i.e., "--") under an item indicate that the value is to be supplied as input (see NEWORDER in Figure 6(a)).

(2) Form name under one or more components specifies that the values of these components are to be obtained from the corresponding components of the specified form. As an example, new instances of CUST_NO, CADDRESS, PROD_NO, and QTY in the PICKUP process (Figure 6(b)) are obtained from the NEWORDER form. Alternatively, one can specify NEWORDER.CUST_NO under the CUST_NO column, NEWORDER.CADDRESS under the CADDRESS column, etc.

(3) An expression involving arithmetic operations specifies how the new values are to be derived. Take the REDUCE_STOCK process (Figure 6(b)) for example. Both BQTY and QOH are decremented by the corresponding value of QTY in the PICKUP form. Note that the corresponding instances from STOCK and PICKUP are determined by the criteria stated in the CONDITION row (below).

(4) A set of built-in functions are provided for more complex source of data. Functions such as SUM, COUNT, MAX, MIN, and AVG can be used to specify an aggregate value as source of data (e.g. TOTAL in INVOICE, Figure 6(c)). FIRST and LAST can be used to specify whether the first or the last instances of the group if to be used as source (e.g., BIN_NO in the PICKUP process, Figure 6(b)).

CONDITION

One purpose of the CONDITION specification is to provide criteria for selecting instances out of a form. Another purpose of the CONDITION specification is to provide criteria for combining instances from two or more forms. Take the PICKUP process (Figure 6(b)) for example again. Data to be inserted into the PICKUP from are extracted from NEWORDER and STOCK forms. The condition is to match PROD_NO obtained from NEWORDER with that from STOCK.

ORDER

The purpose of the ORDER specification is to request the desired ordering of instances within a target (output) form or ordering of group instances within a parent instance. In Figure 6(d), it is requested that the VENDOR_REPORT form be in descending (DES) order of PROD_NO; and for each instance of PROD_NO, the SUPPLIERS group instances be in ascending (ASC) order of VNAME. In case that there are more than one sort field at any level, the specification of sort direction (ASC or DES) can be suffixed with a digit to indicate the significance of each (e.g., ASC1 under ITEM1 and DES2 under ITEM9 specifies that ITEM1 is the most significant sort field and ITEM9 is the next significant).

NODUP

NODUP provided a facility to eliminate duplication in a form or in a group. Asterisk ("*") under pertinent item(s) in the NODUP row is used to control the duplications. In the example of VENDOR_INQUIRY process (Figure 6(d)), if two or more instances contain identical values of the triplets (VNAME, ADDRESS, VPRICE), only one instance will be listed.

DELETION

DELETION specifies what is to be deleted when the specified condition is met. In the DELETE_PICKUP (Figure 6(c)), when the matching instances of INVOICE, CUST_INFO, and PICKUP are found, the corresponding (PROD_NO, QTY, BIN_NO) triplet is deleted. Furthermore, the DELETION row also specifies that (CUST_NO, CADDRESS) pair is to be deleted when there is no existence of ORDER group instance for the pertinent (CUST_NO, CADDRESS) pair. If "DELETE WHEN COUNT(ORDER)=0" has been omitted from this example, the parent instance will survive even though all children instances may be deleted.

5.4. Process Environment

Process environment supplies information such as on-line or batch (MODE), execution frequency (FREQUENCY), and time constraints (RESPONSE TIME) if any. Since they have an obvious interpretation, no elaboration is necessary.

In section 2, we listed the information needed for database schema design. Looking back at the list, it is not hard to see that all the information needed can be captured readily by the forms oriented specification as we proposed.

The specification notation is easily extendible. For example, we do not see the impact of concurrence or sequencing of processes on database design. Therefore, we did not include them in the specification. If a database designer can indeed justify the need to acquire this information, adding them to the specification is straightforward. As an example, a SEQUENCE specification is added in figure 6(e) to indicate that the UPDATE_STOCK process follows PICKUP process and precedes DELETE_PICKUP.

6. DISCUSSION AND SUMMARY

We have presented in this paper a forms oriented requirements specification facility. It is important to note that this facility is data model independent. A user's conception of data is exhibited in the form heading. This conception is independent of the data model supported by the underlying database management system (DBMS). Yet, when augmented with other information specified by the user (also independent of data model) it provides relevant information needed for designing a database supportable by a particular DBMS.

To show that the same specifications can be used for different data models, let us use the PRODUCT form (Figure 4) as an example.

In a hierarchical model such as the one implemented on IMS [IMS79], the PRODUCT form can be supported by the structure in Figure 7.

In a relational model, on the other hand, decomposition of the hierarchical structure into several relationships is required. The following relations are the results of decomposing PRODUCT form based on the KEY, UNIQUENESS and DETERMINACY specifications in conjunction with the hierarchical relationships derived from the form heading:

PRODUCT (*PROD_NO*, PNAME, TYPE, PRICE)
SUPPLIER (*PROD_NO, VNAME*)
STORAGE (*PROD_NO, BIN_NO*)
BIN_LOC (*BIN_NO*, LOC)

Similarly, on a DBTG type of network data model, a schema may be as in Figure 8.

In any case, the schema derived from the specification is determined by the data model, but the specification itself is independent of the data model and the underlying DBMS. This is important because the user should only be concerned with the description of his/her requirements and should not be forced to express them in a specific data model.

Earlier, we have mentioned the transformability of the requirements specifications into execution units. Ideally, one would want to translate the specification directly into a program supported by the database designer given the "best possible" basis to compute the cost of running the applications, but the user is also given a very high level programming facility.

Preliminary studies have shown that, using the requirements specification as we proposed, translation from specification to high level languages such as SEQUEL [Chamberlin76] and CONVERT [Shu75] is feasible. Take the ENTER_ORDER process in Figure 6(a), for example. The following CONVERT statement can be used in order to: to transform the instances of NEWORDER to conform with the form heading specified for ORDERLIST,

```
ORDERLIST   (CUST_NO,
             ORDER  (ORDER_NO, CADDRESS, DATE
                     PROD (PROD_NO, QTY) ) )
    = CONSOLIDATE (NEWORDER);
```

At the same time, the following SEQUEL statement can be used to simulate this effect:

```
SELECT  CUST_NO,  ORDER_NO,  CADDRESS,  DATE,
PROD_NO, QTY
FROM NEWORDER
ORDER BY 1, 2, 5
```

Translation of forms-oriented specifications into executable programs requires more research and development. At IBM Scientific Center in Los Angeles, California, a compiler is being implemented for the forms processing language as an experimental vehicle. To be realistic, we do not expect a translator for every system in every system organization. However, even before a program translator is available, the requirements specification (using our approach) will provide the database designer a sound and realistic basis to compute the costs. In this case, the "execution unit," instead of being an executable program, can be a skeleton that serves as a holder for the outstanding operations that must be accounted for; operations such as accessing two or more files in one process, extraction and rearranging of components in a file, testing for conditions, aggregation that requires accessing many records (e.g., SUM, MAX, etc.), sorting, updating, etc. These operations have a direct and profound bearing on the database design. Using the forms approach as we proposed here, applications specified can be transformed relatively easily into a set of skeleton execution units that brings the operational characteristics into focus.

In summary, we have presented a forms oriented application specification, motivated by the need to collect information crucial for database design. The technique is based on a few simple concepts, hence it is easy to learn and apply by the database designer, and can be understood by the information suppliers, who are usually not computer specialists. It also lends itself to a methodology of capturing requirements for database design in which the application users can concentrate on describing his/her data processing activities and requirements in a highly iterative manner based on a familiar data structure -- forms. A wide variety of database applications can be specified in this simple manner. Since the specification is formal, the information is machine manipulable. Using the information thus collected, algorithms can be developed to compute the relative importance of individual applications, the cost of running all applications against a database, and the cost of maintaining the integrity of the database. An example of an application for designing a relational schema based on the information gathered with forms specification is given in [Wong & Shu80]. The forms approach has been used in describing

requirements for database design for a dentist office application and a beer brewery company application. In addition, about 50 database design projects by Master students on both real and contrived applications have been performed. The feedback has been encouraging.

ACKNOWLEDGMENT

The authors are grateful to Meri Jones for her excellent editing of this paper. The comments by Dr. Arie Shoshani are very much appreciated.

REFERENCES

[Astrahan76]

Astrahan, M.M. et al., "System R: A Relational Approach to Database Management, *ACM Transactions on Database Systems*, Vol. 1, No. 2, pp. 97-137, June 1976.

[Blasgen79]

Blasgen, M.W., et al., "System R: An Architectural Update, IBM Research Report, RJ2581, July 1979.

[Chamberlin76]

Chamberlin, D.D., et al., "SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal of Research and Development*, Vol. 20, No. 6, November 1976.

[DBTG71]

CODASYL Data Base Task Group April 1971 Report, ACM, New York.

[ERA CONF 79]

Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, California, (Edited by P. Chen), December 10-12, 1979.

[Hebalkar79]

Hebalkar, P.G. and S.M. Zilles., "Graphical Representation and Analysis of Information System Design," IBM Research Report, RJ2465, January 1979.

[IMS79]

IMS/VS Version 1 General Reference Manual, IBM Reference Manual Number GH20-1260-8, Ninth Edition (April 1979).

[Lefkovits79]

Lefkovits, H.C., et al., "A Status Report on the Activities of the CODASYL End User Facilities Committee (EUFC)," (Issued August 1979), SIGMOD Record Volume 10, Nos. 2 and 3, February 1979.

[Lum79] Lum, V.Y., et al., "1978 New Orleans Data Base Design Workshop Report," *Proceedings of the Fifth International Conference on Very Large Data Bases*, Brazil, October 1979.

[Lum79A]

Lum, V.Y., et al., "1978 New Orleans Data Base Design Workshop Report," IBM Research Report, RJ2554, July 1979.

[NYU SYM78]

Proceedings of NYU Symposium on Database Design, May 18-19, 1978.

[QBE78]

"Query-by-Example Terminal User's Guide," IBM Installed User Program, Program Number 5796-PKT, Publication Number SH20-2078-0.

[Shu75]

Shu, N.C., B.C. Housel and V.Y. Lum, "CONVERT: A High Level Translation Definition Language for Data Conversion," *Communications ACM*, Vol. 18, No. 10, pp. 557-567, October 1975.

[Shu82]

Shu, N.C., et al., "Specification of Forms Processing and Business Procedures for Office Automation," *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 5, September 1982.

[Teichroew71]

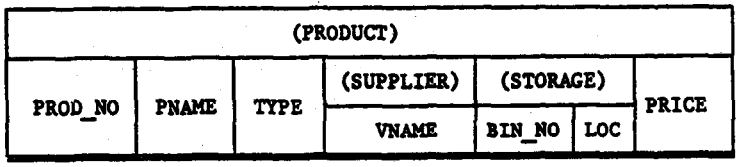
Teichroew, D. and H. Sayani, "Automation of System Building," *Datamation*, pp. 25-30, August 1971.

[Teichroew77]

Teichroew, D. and E.A. Hershey, III, "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977.

[Wong & Shu80]

Wong, H.K.T. and N.C. Shu. "Relational Database Scheme Design," IBM Research Report, RJ2688.



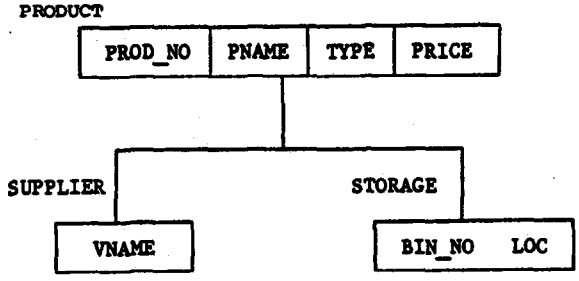


Figure 1. Example of a Form Heading and its Corresponding Hierarchy Graph

DESCRIPTION	Form Heading						
KEY							
UNIQUENESS							
DETERMINACY							
SET INCLUSION							
SET EXCLUSION							
NULL							
VALUE							
OCCURRENCE							
DATA TYPE							
AUTHORITY LEVEL(READ)							
AUTHORITY LEVEL(WRITE)							

Figure 2(a). Summary of Data Specification

INSERT UPDATE DELETE PRINT/QUERY	Form Heading						
SOURCE							
ORDER							
MODUP							
DELETION							
CONDITION							
MODE							
FREQUENCY							
RESPONSE TIME							

Figure 2(b). Summary of Process Specifications

(CUST_INFO)					
CUST_NO	CNAME	(CADDRESS)			CREDIT_LIMIT
		ST	CITY	STATE	

(VENDOR_INFO)					
VNAME	ADDRESS	(SUPPLIES)			
		CATALOG	(PROD)		
			PROD_NO	PNAME	VPRICE

(PRODUCT)						
PROD_NO	PNAME	TYPE	(SUPPLIER)	(STORAGE)		PRICE
			VNAME	BIN_NO	LOC	

(STOCK)				
PROD_NO	(STORAGE)			QOH
	BIN_NO	BQTY	CAPACITY	

(ORDERLIST)					
CUST_NO	(ORDER)				
	ORDER_NO	CADDRESS	DATE	(PROD)	
				PROD_NO	QTY

Figure 3. Form Headings for Files Maintained by the Example Distribution Company

DESCRIPTION	(PRODUCT)						
	PROD_NO	PNAME	TYPE	(SUPPLIER)	(STORAGE)		PRICE
				VNAME	BIN_NO	LOC	
KEY	*			*	*		
UNIQUENESS	*				*		
DETERMINACY					*	->	
SET INCLUSION				VENDOR_INFO			
SET EXCLUSION	OUTDATED_ PROD.PNO						
NULL		*	*				
VALUE	10%(<100) 10%(>999)					SJC SFO	
OCCURRENCE	1000			16	10		
DATA TYPE	9(4)	X(6)	99	X(8)	99	XXX	9999v99
AUTHORITY LEVEL(WRITE)							>5

Figure 4. Example of Specification on Data

DESCRIPTION	(EMPLOYEE)					
	NAME	SS_NO	ENO	(PROJ)		
				DNO	PJNO	TIME_ALLOTTED
KEY		*		*	*	
			*			

Figure 5. Example of Alternate Keys

PROC NAME: NEWORDER

INSERT	(NEWORDER)					
	CUST_NO	ORDER_NO	CADDRESS	DATE	(PROD)	
					PROD_NO	QTY
SOURCE	—	—	—	—	—	—
FREQUENCY	1000/DAY					
MODE	ON_LINE					

PROC NAME: ENTER_ORDER

INSERT	(ORDERLIST)					
	CUST_NO	(ORDER)				
		ORDER_NO	CADDRESS	DATE	(PROD)	
					PROD_NO	QTY
SOURCE	NEWORDER FORM					
FREQUENCY	1000/DAY					
MODE	ON_LINE					

Figure 6(a). Sample Process Specifications: NEWORDER and ENTER_ORDER Processes

PROC NAME: PICKUP

	(PICKUP)				
INSERT	CUST_NO	CADDRESS	(ORDER)		
			PROD_NO	QTY	BIN_NO
SOURCE	NEWORDER FORM				FIRST (STOCK.BIN_NO WHERE STOCK.BQTY > NEWORDER.QTY)
CONDITION	NEWORDER.PROD_NO = STOCK.PROD_NO				
FREQUENCY	1000/DAY				
MODE	ON_LINE				

PROC NAME: REDUCT_STOCK

	(STOCK)				
UPDATE	PROD_NO	(STORAGE)			QOH
		BIN_NO	BQTY	CAPACITY	
SOURCE			BQTY - PICKUP.QTY		QOH-PICKUP.QTY
CONDITION	STOCK.PROD_NO = PICKUP.PROD_NO AND STOCK.BIN_NO = PICKUP.BIN_NO				
MODE	ON_LINE				
FREQUENCY	1000/DAY				

Figure 6(b). Sample Process Specifications: PICKUP and REDUCT_STOCK Processes

PROC NAME: INVOICE

	(INVOICE)						
PRINT	CNAME	CADDRESS	(ORDER)				TOTAL
			PROD_NO	QTY	PRICE	SUBTOTAL	
SOURCE	CUST_INFO FORM	PICKUP FORM	PICKUP FORM		PRODUCT FORM	QTY * PRICE	SUM (SUBTOTAL)
CONDITION	CUST_INFO.CUST_NO = PICK.CUST_NO AND PICKUP.PROD_NO = PRODUCT.PROD_NO						
FREQUENCY	1000/DAY						
MODE	ON_LINE						

PROC NAME: DELETE_PICKUP

	(PICKUP)				
DELETE	CUST_NO	CADDRESS	(ORDER)		
			PROD_NO	QTY	BIN_NO
DELETION	DELETE WHEN COUNT(ORDER)=0		*	*	*
CONDITION	INVOICE.CNAME = CUST_INFO.CNAME AND CUST_INFO.CUST_NO = PICKUP.CUST_NO AND INVOICE.PROD_NO = PICKUP.PROD_NO				
FREQUENCY	1000/DAY				
MODE	ON_LINE				

Figure 6(c). Sample Process Specifications: INVOICE and DELETE_PICKUP Processes

PROC NAME: VENDOR_INQUIRY

	(VENDOR_INQUIRY)		
QUERY	VNAME	ADDRESS	VPRICE
NODUP	*	*	*
SOURCE	VENDOR_INFO FORM		
CONDITION	VENDOR_INFO.PROD_NO = —		
MODE	ON_LINE		
FREQUENCY	100/DAY		

PROC NAME: VENDOR_REPORT

	(VENDOR_REPORT)			
	PROD_NO	(SUPPLIERS)		
PRINT		VNAME	ADDRESS	VPRICE
ORDER	DES	ASC		
NODUP		*	*	
SOURCE	VENDOR_INFO FORM			
MODE	OFF_LINE			
FREQUENCY	1/365DAY			

Figure 6(d). Sample Process Specifications: VENDOR_INQUIRY and VENDOR_REPORT Processes

PROC NAME: UPDATE_STOCK

	(STOCK)				
UPDATE	PROD_NO	(STORAGE)			QOH
		BIN_NO	BQTY	CAPACITY	
SOURCE			BQTY - PICKUP.QTY		QOH-PICKUP.QTY
CONDITION	STOCK.PROD_NO = PICKUP.PROD_NO AND STOCK.BIN_NO = PICKUP.BIN_NO				
MODE	ON_LINE				
FREQUENCY	1000/DAY				
SEQUENCE	FOLLOWS PICKUP, PRECEDES DELETE_PICKUP				

Figure 6(e). Example of a SEQUENCE Specification.

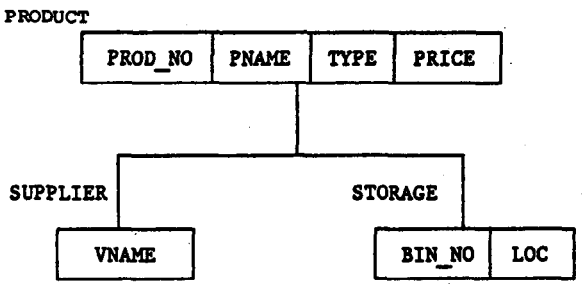


Figure 7. Hierarchy for the PRODUCT Form

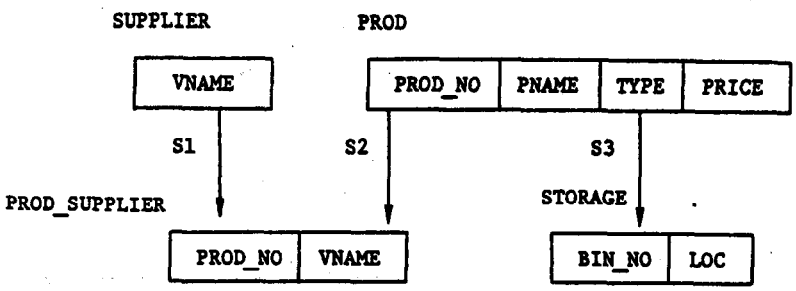


Figure 8. DBTG schema for the PRODUCT Form