INCORPORATING DATABASE DESIGN IN WARNIER METHOD

Donald R. Chand
CIS Department
Bentley College
Waltham, MA 02452
dchand@bentley.edu

ABSTRACT

The Warnier method, a highly prescriptive program design approach for file-oriented solutions, has been criticized for its lack of a database design component. This paper addresses this weakness by incorporating a logical database design step in Warnier method. Specifically, the paper presents rules for transforming the information in a Warnier diagram into a set of relations. With this extension the Warnier method complements the entity-relationship approach for data analysis and logical database design.

INTRODUCTION

Warnier [(1974), (1978), (1981)] design method, first introduced as Logical Construction of Programs (LCP) and later elaborated as Logical Construction of System (LCS), is a highly prescriptive approach of program design for file-based, report-oriented solutions of data-processing problems. Because it is highly prescriptive, different programmers produce almost identical programs for the same problem. Warnier method is also self -documentary because the design is documented as the prescribed steps of Warnier method are applied. These two attributes, self-documentation and highly prescriptive design steps, aid in synchronizing the mental and physical efforts of a programming team.
Despite these desirable attributes and outcomes, the Warnier method was not widely used outside France. The major criticism of Warnier method [McClure & Martin (1981)] is that it lacks a database design component, which is central in the design of business applications. This paper addresses this problem by incorporating in Warnier method an additional step of transforming the data in a Warnier diagram into a set of normalized relations. The transformation rules and the incorporation of logical database design considerations in Warnier method are illustrated in section 3. Section 2 reviews Warnier method and section 4 discusses the broader implications this extension of Warnier method.

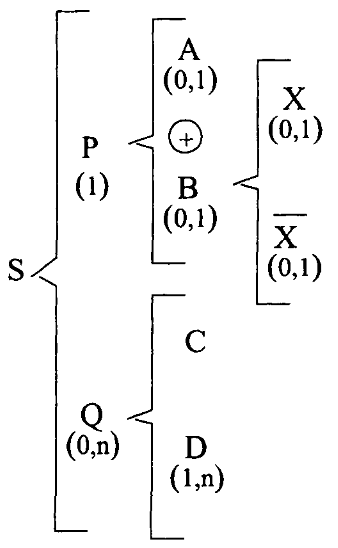
THE WARNIER METHOD

J. D. Warnier and his colleagues at Honeywell-Bull in Paris, France developed a systematic approach for deriving structured programs from the logical data structures of the output reports. The premise of Warnier approach is that it is necessary and feasible to derive the structure of the application program from the hierarchical structure of the data. It provides a graphical tool, called the Warnier diagram, for modelling and documenting the data and program as a hierarchy. The design method consists of following five steps.

Step1: Model the layout of the output report into a hierarchical data structure and document this structure in a Warnier diagram.
Step2: Using the structure of the output file developed in step1, postulate a compatible input file. The compatibility of the input file ensures that sequential processing of the records in the logical input file yields the output records in the order prescribed on the report. Model the data structure of the postulated compatible input file as a hierarchy in a Warnier diagram.
Step3: Using the hierarchical model of the compatible input file, conceptualised in step 2, derive a hierarchical program structure which is again documented in a Warnier diagram.
Step4: Using the output-report of step1 as a reference, discover the operations needed to produce the report.
Step5: Allocate at the appropriate locations in the program structure of step3 the operations discovered in step4.

The tool used for documenting the outcome of these steps is the Warnier diagram. It is a hierarchy chart laid on its side, where each level of hierarchy is composed of sequence, selection and repetition structures, shown in figure (1).

WARNIER DIAGRAM



Meaning

S is composed of one occurrence of P and zero to many occurrences of Q

P is composed of either A or B but not A and B. A occurs zero or one time and B occurs zero or one time.

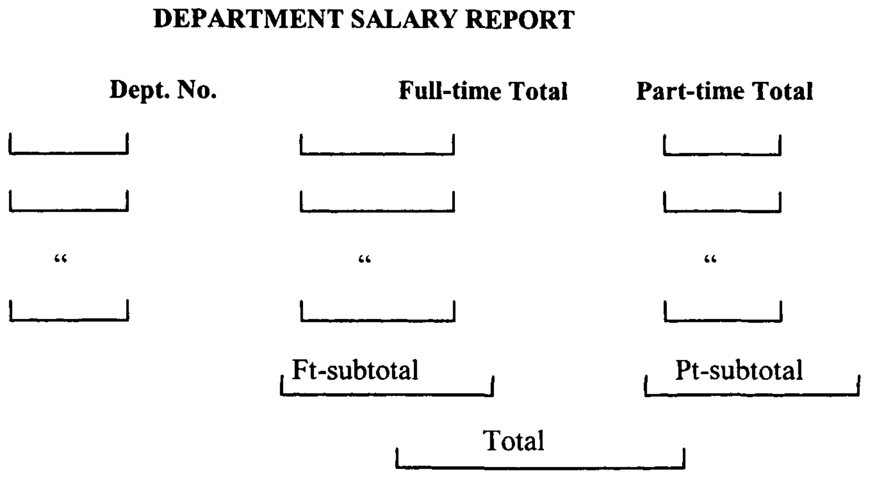
Q is composed of one occurrence of C and one to many occurrences of D.

B is composed of zero or one occurrence of X and zero or one occurrence of not X

Figure 1

An Illustration of Warnier Method

A simple single-output problem is used to illustrate the five steps of Warnier method. The problem is to devise a program that processes the salary records of the full-time and part-time faculty at a small college to produce a department-wise summary report of faculty salaries. The layout of the prescribed report is shown in figure (2).


DEPARTMENT SALARY REPORT

Dept. No. **Full-time Total** **Part-time Total**

" " "

Ft-subtotal Pt-subtotal

Total

Figure (2)

Step1: Model the output layout as a hierarchy and document the model in a Warnier diagram. The logical data structure of the output report is shown in figure (3).

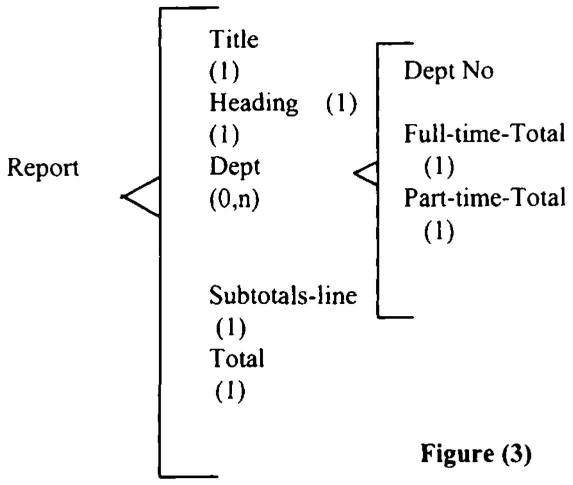


Figure (3)

Step2: Postulate a logical input-file whose data structure is compatible to the logical data structure of output report. A compatible input file and its data structure are shown in figures (4) and (5).

COMPATIBLE INPUT FILE

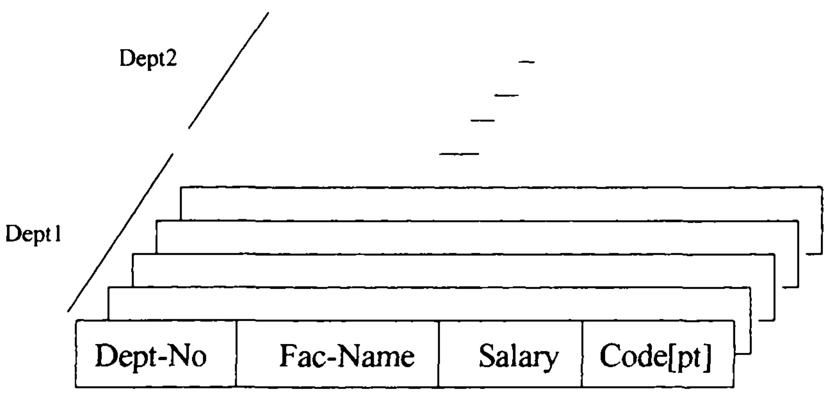


Figure (4)

INPUT FILE STRUCTURE

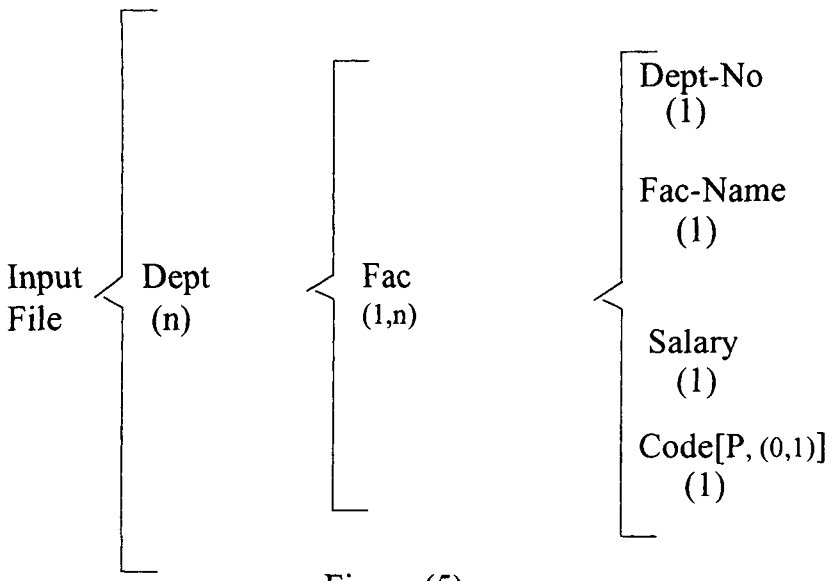


Figure (5)

PROGRAM STRUCTURE

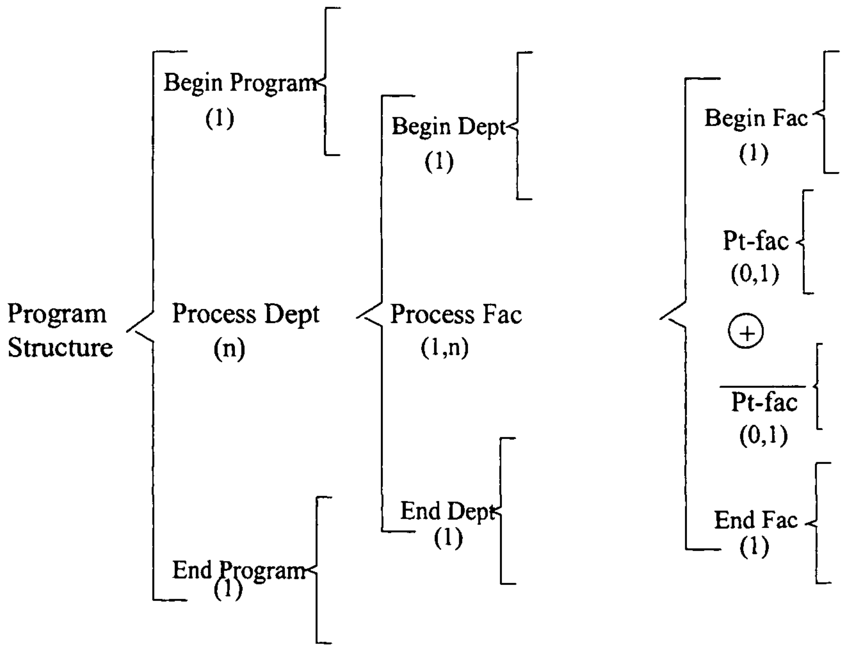


Figure (6)

Step3: Derive the program structure from the data structure of the logical input file. The program structure is shown in figure (6).

Step 4: Specify the operations needed to produce the prescribed output report.

Output Operations	Input Operations	Processing	Initialisation
Print Title	**Read first record**	**Update Total**	**Initialise Total**
Print Heading	Read next record	Update Ft-fac-total	Initialise Subtotals
Print Dept-line		Update Pt-fac-total	Initialise Ft-fac-total
Print Subtotal-line		Update Ft-Subtotal	Initialise Pt-fac-total
Print Total		Update Pt-Subtotal	

Step 5: Allocate the operations to the program structure in figure (6). This yields the structured program of figure (7).

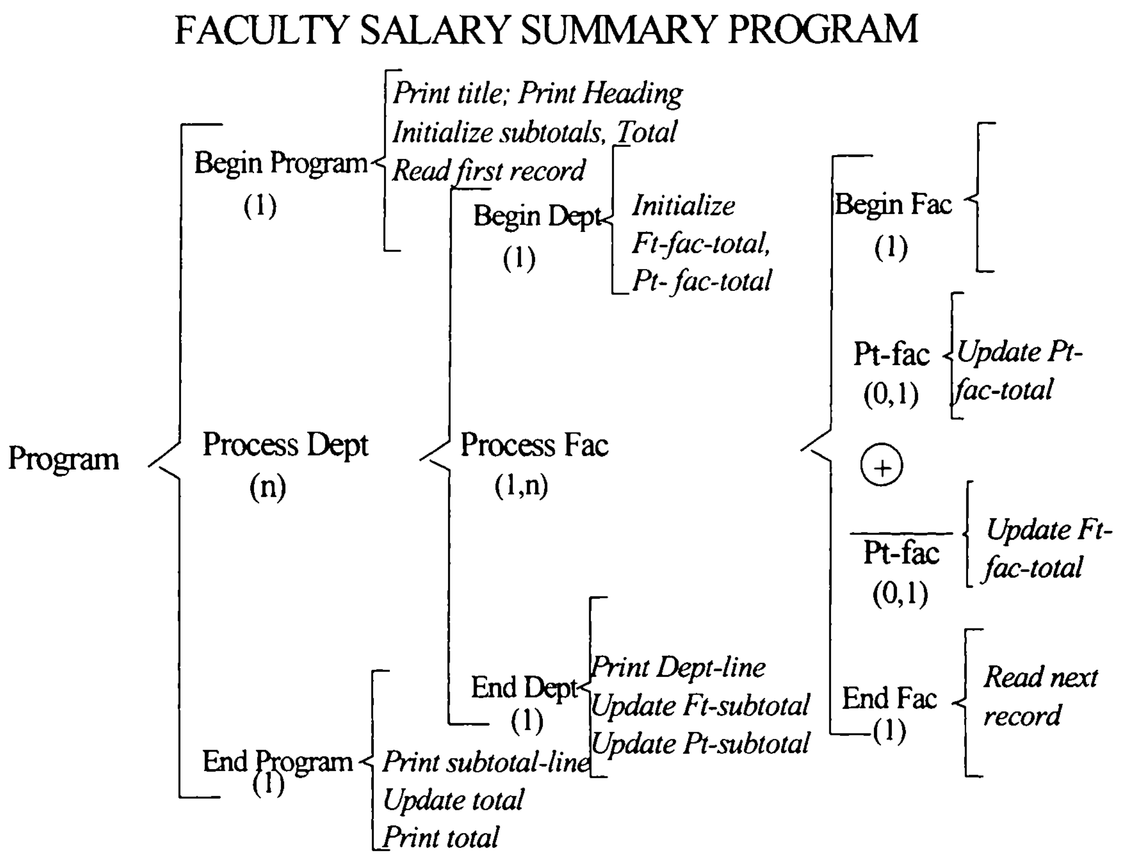


Figure 7

The above illustration of Warnier method reveals both its strength and weakness. The strength is that it systematizes the program composition process to a point where most designers using this method will produce almost identical solutions. The criticisms are:

1. It is not readily applicable to applications with multiple input and output files.
2. It does not address situations where the logical input file structure does not match the structure of the physical input file.
3. It provides no guidance for database design.

The first criticism is addressed by partitioning the application program into a set of one-output programming problems. If n is the number of output reports of the application, the application design is split into n one-output programming problems. Treating every output report as a programming problem and applying Warnier method to it will yield n logical input files. These n logical input files form the database for the application. In summary, this design approach of partitioning of the original application produces a modular solution consisting of

- n one-output programs,
- a database of n logical input files, and
- programs that map the given physical inputs of the application to the n logical input files of the database.

The second criticism is that the method does not address the case when the logical input file is not compatible with the physical input file. This non-compatibility of the physical and logical files is essentially the structure-clash problem, recognized and addressed in Jackson Structured Program (JSP) methodology (1975). Jackson showed that when the logical file structure is not compatible with the physical file structure, then the problem should not be handled in a single program. He recommends that the problem should be partitioned into two sub-problems. One sub-problem is generating the specified output from the logical file structure, and the second sub-problem is converting the physical file structure to the logical file structure.

The first sub-problem is directly addressed by the Warnier approach illustrated above. This second sub-problem can be addressed by breaking the physical data structure into its most elementary parts and then rebuilding the logical structure. Jackson suggested that the same result can be achieved more efficiently by breaking the physical file structure into the largest common data structure that is common to both the physical and logical data structures.

The third criticism of lacking guidelines for database design is a short-coming of the original Warnier method, and it is addressed next. It turns out that the approach of partitioning the application design into one-output programs provides a basis for addressing this third criticism. A common business application is used in Section 3 to illustrate the integration of a database design component in Warnier method.

INCORPORATION OF A DATABASE DESIGN STEP IN WARNIER METHOD

A database design step is added to the Warnier method after the data structure of the compatible input file is modelled in a Warnier diagram. In the database design step, the information in the Warnier diagram is transformed into a set of normalized relations. Thus, the initial two steps of Warnier method are revised as follows.

Step1: Sketch the layout of each output report and analyse each report layout to design its compatible input file. Model the data structure of each compatible input file in a Warnier diagram.

Step2: Apply the entity identification, key selection and key synchronization rules to transform the compatible input file information into a set of normalized relations. These rules are introduced and illustrated below.

The inputs to an expense-accounting system [Chand & Yadav (1980)] are checks written to pay for expenses. Each check contains following information:

- check number
- date of check
- payee name
- amount of check
- expense note (this occurs 1-20 times per check and contains expense description, expense amount, and accounting code consisting of product line number, cost center number, and expense account number)

EXPENSE ACCOUNT REPORT

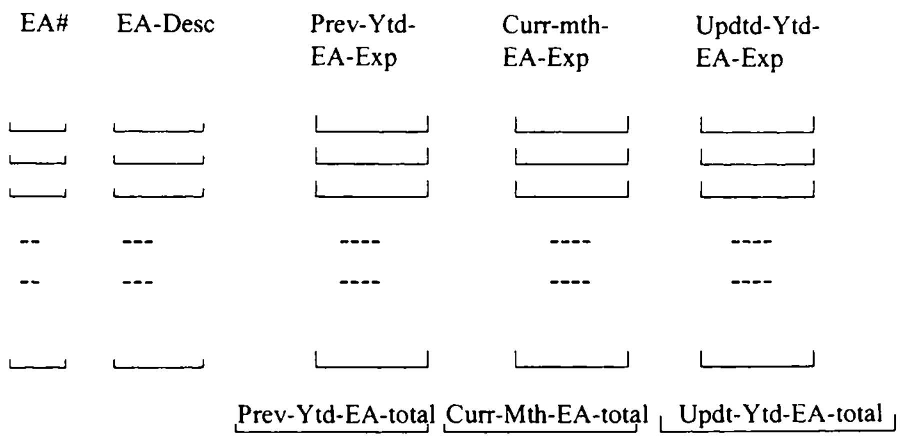


Figure (8)

COST CENTER EXPENSE ACCOUNT REPORT

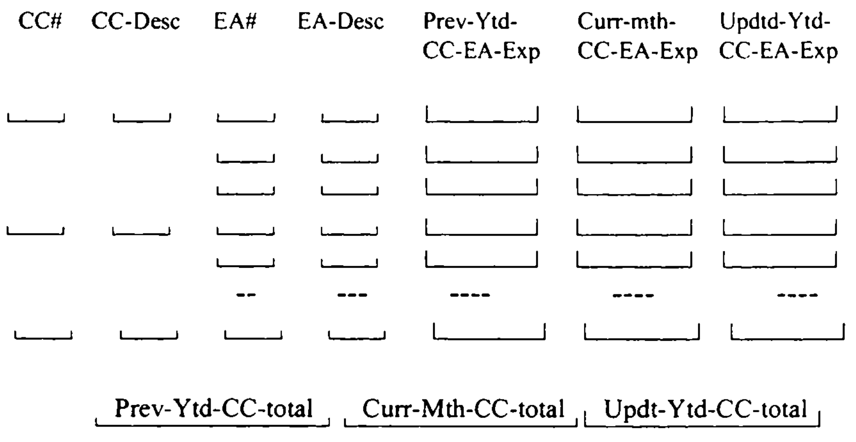


Figure (9)

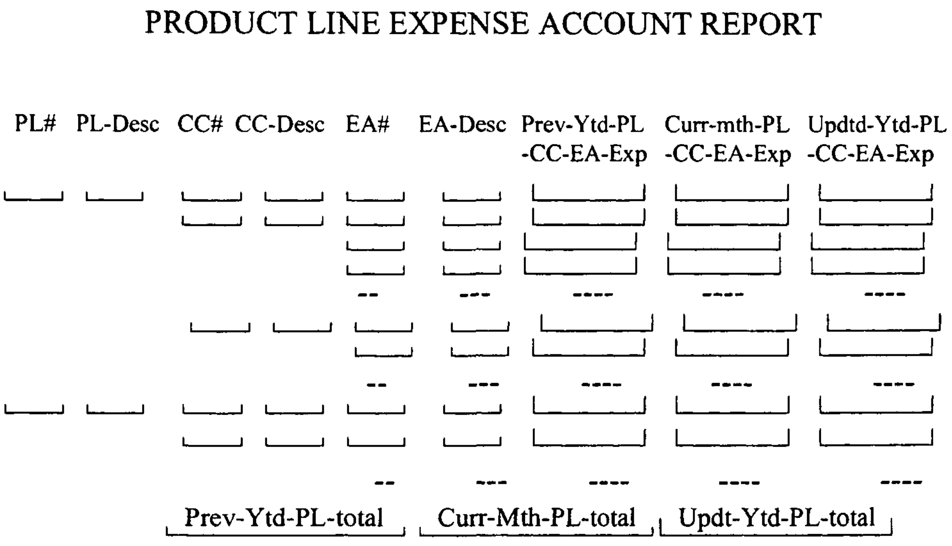


Figure (10)

The data structure of the logical files compatible to the three reports in figures (8), (9), and (10) are respectively presented in figures (11), (12), and (13).

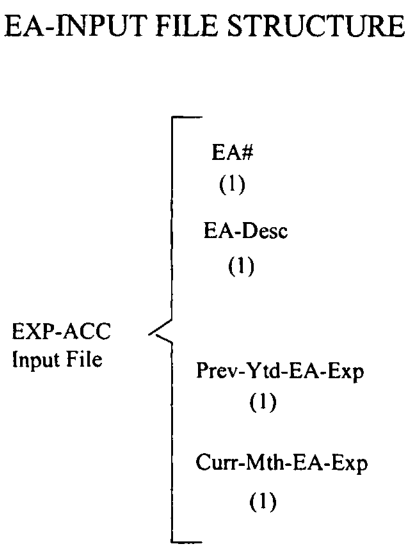


Figure (11)

CC-INPUT FILE STRUCTURE

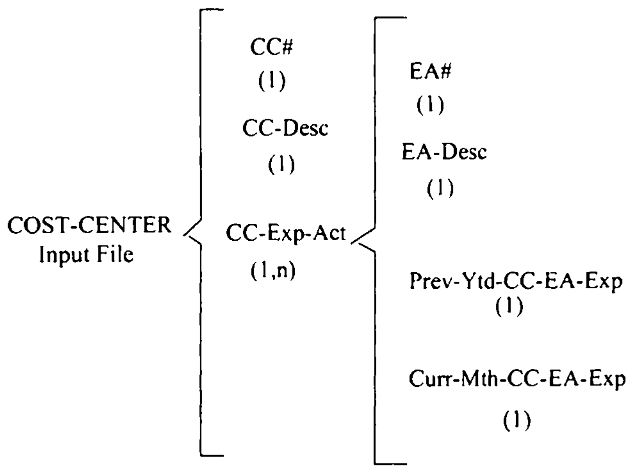


Figure (12)

PL-INPUT FILE STRUCTURE

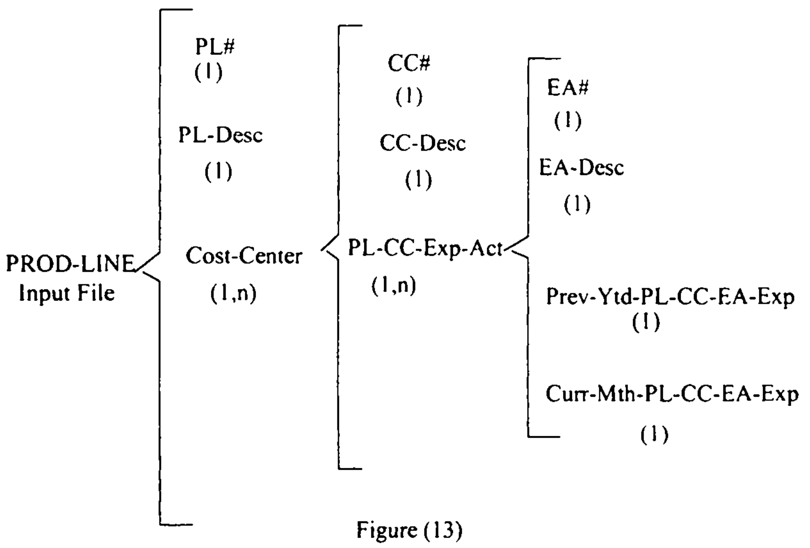


Figure (13)

Database Design Step

This section presents and illustrates rules for transforming the information in the compatible input-file structures into a set of normalized relations. The rules aid in identifying database entities, specifying primary keys, and synchronizing entity keys across the file structure hierarchy.

Entity Identification Rule:

Scan the Warnier diagram sequentially from the right side. Collect all the data items in the right most hierarchy into an entity. Remove the rightmost hierarchy and the corresponding data item in its adjacent hierarchy from the Warnier diagram. Repeat this process until there are no more hierarchies in the Warnier diagram.

The application of the entity identification rule to the file structure in figure (11) yields one entity, which is shown below with its attributes.

Exp-Acc (EA#, EA-Desc, Prev-Ytd-EA-Exp, Curr-Mth-EA-Exp)

The entity identification rule, applied twice to the file structure in figure (12), yields the following two entities.

CC-Exp-Acc(EA#, EA-Desc, Prev-Ytd-CC-EA-Exp, Curr-Mth-CC-EA-Exp)
Cost-Center(CC#, CC-Desc)

The entity identification rule is applied three times to the file structure in figure (13) resulting in the following three entities.

PL-CC-Exp-Acc(EA#, EA-Desc, Prev-Ytd-PL-CC-EA-Exp, Curr-Mth-PL-CC-EA-Exp)
Cost-Center(CC#, CC-Desc)
Prod-Line(PL#, PL-Desc)

Key Selection Rule:
Identify the candidate keys in each entity definition and select as primary key among the candidate keys the one that is most meaningful to the user.

The application of the key-selection rule updates the above entity definitions as follows.
Figure (11) Entity:
Exp-Acc (EA#, EA-Desc, Prev-Ytd-EA-Exp, Curr-Mth-EA-Exp)
Figure (12) Entities.
CC-Exp-Acc(EA#, EA-Desc, Prev-Ytd-CC-EA-Exp, Curr-Mth-CC-EA-Exp)
Cost-Center(CC#, CC-Desc)
Figure (13) Entities.
PL-CC-Exp-Acc(EA#, EA-Desc, Prev-Ytd-PL-CC-EA-Exp, Curr-Mth-PL-CC-EA-Exp)
Cost-Center(CC#, CC-Desc)
Prod-Line(PL#, PL-Desc)

Key Synchronization Rule:
Scan the hierarchy of the Warnier diagram sequentially from the left. Define the key of the outer most hierarchy as the nest key. Add the nest key to the entity corresponding to the next adjacent hierarchy. If there is a many-to-many relationship between these two entities, form a composite key by adjoining the nest key to the key of the adjacent hierarchy. Otherwise, the nest key is added as a foreign key to the entity corresponding to the inner hierarchy. Remove the outermost hierarchy, and repeat the process until there is no hierarchy [Batra (1997)].

The key-synchronization rule does not apply to the hierarchy in Figure (11). Therefore, there is no change in the following relation.

Exp-Acc (EA#, EA-Desc, Prev-Ytd-EA-Exp, Curr-Mth-EA-Exp)

The key-synchronization rule is applied once to the hierarchy in Figure (12). Since there is many-to-many relationship between Cost-Center and CC-Exp-Acc entities, the key-synchronization rule yields the following updated relations.

Cost-Center(CC#, CC-Desc)
CC-Exp-Acc(CC#, EA#, EA-Desc, Prev-Ytd-CC-EA-Exp, Curr-Mth-CC-EA-Exp)

The key-synchronization rule is applied twice to the hierarchy in Figure (13). Since there is many-to-many relationship between Prod-Line and Cost-Center entities, the nest key, PL#, is concatenated with the Cost-Center key, CC#. The product line hierarchy is removed. The nest key is now the composite key, PL# CC#. Once again there is a many-to-many relationship between Cost-Center and PL-CC-Exp-Acc entities, the nest key is concatenated to the PL-CC-Exp-Acc entity.key. The application of the synchronization rule yields the following updated relations.

Prod-Line(PL#, PL-Desc)
Cost-Center(PL#, CC#, CC-Desc)
PL-CC-Exp-Acc(PL#, CC#, EA#, EA-Desc, Prev-Ytd-PL-CC-EA-Exp, Curr-Mth-PL-CC-EA-Exp)

Eliminating the traditional partial and transitive dependencies, and renaming one of the entities, results in the following set of third normal form relations.

Exp-Acc (EA#, EA-Desc, Prev-Ytd-EA-Exp, Curr-Mth-EA-Exp)
Cost-Center(CC#, CC-Desc)
CC-Exp-Acc(CC#, EA#, Prev-Ytd-CC-EA-Exp, Curr-Mth-CC-EA-Exp)
Prod-Line(PL#, PL-Desc)
Cost-Center(PL#, CC#)
PL-CC-Exp-Acc(PL#, CC#, EA#, Prev-Ytd-PL-CC-EA-Exp, Curr-Mth-PL-CC-EA-Exp)

This completes the illustration of a database design step in Warnier method.

IMPLICATIONS

The incorporation of a database design step frees Warnier method from the stigma and perception that it is not suitable for designing solutions of large business applications and systems. As illustrated above, a large business application can be partitioned into a set of one-output programs and the schema for these one-output programs can be merged to define the database.

The extended Warnier approach presented here can serve as an effective bottom-up approach of developing a normalized database for an application. In addition, it can complement the traditional entity-relationship (E-R) approach. The entity-relationship approach is a popular and very successful approach for building data models during the analysis phase of the systems development life cycle. It has been found that the entity-relationship approach forces the analyst to acquire a deeper understanding of the business entities and business rules and, consequently, uncover requirement errors earlier in the system development life cycle. However, the conversion of the entity-relationship data model into a normalized relational schema does not take into account the processing requirements. Since the Warnier approach forces the designer to explicitly address each user output requirement in the development of a logical database, it can be used as a means of validating whether the entity-relationship model has missed any attributes or relationships needed by the user.

There are other benefits of this approach. First, it provides a problem structuring approach where the problem is partitioned into program modules and each module produces a specific user output. Second, since the logic and code of the program is derived from explicitly exploiting the data structures of the outputs using a standard self-documenting process, the approach synchronizes the mental and physical efforts of a team of software developers. In our judgement, this approach is an excellent approach for teaching and learning programming and bottom-up data base design.

Since the Warnier method was invented in the 1970s, it is important to ask whether it has any role to play in the modern computing environment of electronic commerce, distributed computing and object orientation. We have reflected on this issue and we present our views as follows.

First, it is important to keep in mind that there are three components of software design, namely process design, data design, and interface design. The advancements in the development and execution technologies and the changes in application domains often thrusts one or the other component of software design as being more prominent in terms of an area of study leading to the development of better theories and methods of practice. For example, today's focus on electronic commerce applications has made the interface design component more prominent. But this does not mean that process and data design have faded or become less important. In our opinion, process, data and interface designs are the three legs of the software stool and, all contributions to further the theory or practice in any one of these three areas are equally important. The extended Warnier approach presented here furthers the understanding and practice of the data design component of software development.

Second, object-orientation is the dominant approach for developing systems software and commercial software products and it is also diffusing, although slowly, in the business applications arena. At present, in the information systems world the primary use of object technology is in the area of interface design, whereas the back-end technology is still process-based and is dominated by relational database technology. Clearly, the extended Warnier approach presented here is directly applicable to the current back-end technology. The question is what role Warnier approach will play in the object world. At this time we can only say that just as this paper extends the original Warnier approach to address the database design problem, we are hopeful that future enhancements of Warnier approach with accommodate object solutions.

REFERENCES

Batra, D. (1997), "A Method for Easing Normalization of User Views," **J. Mgt. Info. Sys.** Vol.14, No.1, pp. 215-233.

Chand, D. R. & Yadav, S.B.(1980) , "Logical Construction of Software," **Comm. of ACM**, Vol. 23, No. 10, pp. 546-555.

Jackson, M. A.(1975), **Principles of Program Design**, Academic Press, London,

McClure, Carma & Martin, James (1995), **Structured Techniques**, Prentice Hall

Warnier, J. D.(1974), **Logical Construction of Programs**, Van Nostrand Reinhold, N.Y.

Warnier, J. D.(1978), **Program Modification**, Martinus Nijhoff

Warnier, J. D.(1981), **Logical Construction of Systems**, Van Nostrand Reinhold, N.Y.