IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)

Application and Data Base Design

Version 1 Release 7

SH24-5022-01

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)

Application and Data Base Design

Version 1 Release 7

SH24-5022-01

Note !

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

Second Edition (December 2002)

This edition applies to Version 1 Release 7 of IBM Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS), Program Number 5746-XX1, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

You may also send your comments by FAX or via the Internet:

Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Part 3. Data Base Design

Figures

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Deutschland Informationssysteme GmbH
Department 0215
Pascal Str. 100
70569 Stuttgart
Germany
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and Service Marks

The following terms are trademarks of International Business Machines Corporation in the Unites States, or other countries, or both:

```
CICS
IBM
```

Preface

This book describes a method of performing the task of application design when DL/I DOS/VS and DL/I data bases are used. It is written for those responsible for this type of application design, and assumes some previous knowledge and experience in application design not involving data bases. Previous knowledge of DL/I DOS/VS is not required.

DL/I DOS/VS is referred to in this publication as DL/I.

This book is organized to lead you step-by-step through the process of application design for applications using DL/I. You should read it through from the beginning before you use any of the information.

The book is divided into four parts, containing an introduction, eight chapters, and two appendixes. Each chapter opens with an introduction that lists the content of the chapter, describes its purpose, and introduces the subject matter.

“Part 1. Introduction”, introduces the contents of the book to the reader.

“Part 2. Application Design”, consists of four chapters:

- “Chapter 1. The Relationship of Data Base Design to Application Design”. This chapter establishes a common base of knowledge by reviewing data base concepts and terminology, data base structures, the importance of data base design, and an application design procedure that provides an out line for the rest of the book.
- “Chapter 2. Preliminary Analysis”. This chapter briefly reviews the process of making a preliminary judgement on whether or not data bases should be used in the application under consideration.
- “Chapter 3. Obtaining Application Requirements”. This chapter describes the process of obtaining and documenting the objectives to be met by this application, as defined by the end user's requirements.
- “Chapter 4. Application Analysis”. This chapter describes the process of application analysis based on the documented application requirements. The result is a decision on the number and type of programs necessary to accomplish the goals of the application. The output will be analyzed application data to be used in the next step of creating local views.

“Part 3. Data Base Design”, consists of three chapters:

- “Chapter 5. Creating Local Views From Requirements”. This chapter describes, in detail, a process for creating local views for each of the programs that are to be implemented for the application, using the requirements data previously gathered and analyzed.
- “Chapter 6. Combining Local Views Into a System View”. This chapter describes a method of combining the local views into an integrated structure; taking into consideration data relationship, usage paths, and performance considerations.
- “Chapter 7. Converting the System View to a Physical View”. This chapter describes the DL/I data base organizations and access methods, and leads in

choosing an access method, taking into account factors that influence the choice.

"Part 4. Implementing the Application", contains a single chapter:

- "Chapter 8. Implementing the Application". This chapter gives suggestions for generating an implementation plan for the application, and for carrying out the plan.

"Appendix A. An Example of Application Design With Data Bases". This appendix is a walk-through of the design process described above, using the applications in the sample programs shipped with DL/I.

"Appendix B. A Recommended Naming Convention". This appendix describes a recommended naming convention that provides naming consistency through the entire installation.

A number of terms and phrases used in this book to describe DL/I and DL/I data bases may be new to you or have new meanings in this context. In general, they are explained the first time they are used. The most important terms are defined in the glossary at the back of the book.

Related Publications:

DL/I VSE Release Guide, SC33-6211-05

DL/I DOS/VS Release Guide, SC33-6211-04

DL/I DOS/VS Data Base Administration, SH24-5011

DL/I DOS/VS Resource Definition and Utilities, SH24-5021

DL/I DOS/VS Interactive Resource Definition and Utilities, SH24-5029.

DL/I DOS/VS Recovery/Restart Guide, SH24-5030.

Other DL/I publications:

DL/I DOS/VS General Information, GH20-1246

DL/I DOS/VS Library Guide and Master Index, GH24-5008

DL/I DOS/VS Application Programming: High Level Programming Interface, SH24-5009

DL/I DOS/VS Application Programming: CALL and RQDLI Interface, SH12-5411

DL/I DOS/VS Guide for New Users, SH24-5001

DL/I DOS/VS Messages and Codes, SH12-5414

DL/I DOS/VS Diagnostic Guide, SH24-5002

Summary of Changes

Summary of Changes
for SH24-5022-01
Version 1.7

This edition has been revised to include the titles of two new manuals that have been added to the DL/I DOS/VS library for Version 1.7 of DL/I DOS/VS. Changes also have been made to the titles of CICS/VS manuals listed in the Preface. Miscellaneous additions, improvements, and corrections also have been made throughout this manual.

Part 1. Introduction

Application design, as the term is used in this book, begins when the need for a data processing application is recognized and ends with its successful implementation. Application design includes the collection and analysis, of data requirements, program design, and implementation.

It is difficult to implement a data base application successfully without application design because of the wide ranging effect the use of data bases could have on the whole business organization.

This book can be used as an aid in the design of any application that uses DL/I data bases. It provides a method that has been used successfully. Following it as a guide will help you to develop an efficient application that satisfies the needs of your users.

Part 2. Application Design

Chapter 1. The Relationship of Data Base Design to Application Design

This chapter describes how the design of data bases is related to the design of an application as a whole. The material is divided into four major sections:

1. Data Base Concepts and Terminology
2. Data Base Structures
3. Data Base Design
4. An Application Design Procedure.

The first section reviews data base concepts and terminology. The second section compares three basic types of data base structure to DL/I data bases. If you already have data base experience, you can skip both of these sections. The third section describes the objectives and importance of good data base design, and also mentions data base design tools that you can use to supplement the design procedure described in this book. The final section outlines an application design procedure that will help you integrate data base design into the application design process. The rest of this book follows that outline.

Data Base Concepts and Terminology

A data base is a way of organizing and controlling information so that it can be used by any number of applications in an organization. It does this in ways that minimize the amount of redundant data and provide independence between the data and the way it is physically stored.

When working with a conventional file an application programmer must be fully aware of the way in which it is organized, the format of the records it contains, and of its physical characteristics (such as block size, record length, access method used, and the location of the file). Application programs using this file must be tailored to these formats and characteristics. If these characteristics are changed, the programs may also have to be changed.

On the other hand, with DL/I data bases, an application programmer needs to be aware of only those data elements needed by the particular application programs for which he or she is responsible. The programmer does not need to know the physical characteristics of the data base or where it is located. Most changes made to the physical characteristics do not affect the programmer, and the programs using the data base do not need to be altered.

The upper part of Figure 1-1 shows a physical record in a conventional file with the data elements labeled NAME, ADDRESS, and PAYROLL. The same elements (called *segments* in DL/I) might be viewed by DL/I as the data structure shown in the lower portion of the figure. The way the segments are physically stored may differ significantly from the way the data is viewed as a data structure.

Another example of a conventional physical record is shown in Figure 1-2. Again, the lower part of the figure illustrates a data structure that DL/I might make available. SKILL, NAME, EXPERIEN, and EDUCAT are the segments that make up this structure (called a *data base record* in DL/I). Although not shown in Figure 1-2, there may be multiple EXPERIEN and EDUCAT segments for each name, many names for each skill, and many skills. Figure 1-3 is an expanded example that illustrates this. Notice that, because many employees may have the same skill, multiple NAME segments exist under the first SKILL segment. Similarly, multiple EDUCAT segments exist under each NAME segment, and multiple EXPERIEN segments exist under two of the NAME segments.

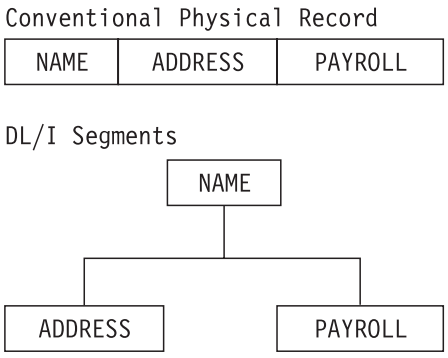


Figure 1-1. Physical Record - DL/I Segment Relationship (Example 1)

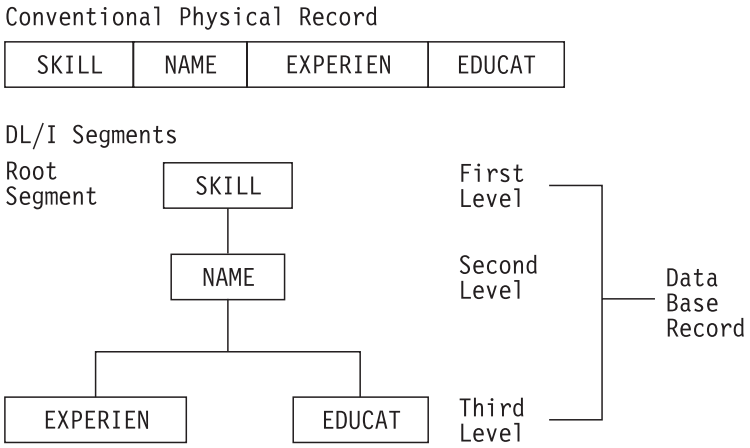


Figure 1-2. Physical Record - DL/I Segment Relationship (Example 2)

Before going on to describe ways in which data bases can be structured, we will summarize the characteristics of DL/I data bases and introduce some terms used with DL/I:

- Looking again at Figure 1-2, you can see that all of the segments shown in the data structure are subordinate to the one segment at the top of the structure. This top-most segment (the SKILL segment) is called a *root segment*.
- Segments at levels below the root segment are said to be *dependent* on those above. In Figure 1-2, NAME is dependent on the root segment SKILL. SKILL is the *cprent* of NAME. NAME is a *child* of SKILL. EXPERIEN and EDUCAT are dependent children of NAME. NAME is the parent of both of them. There can be many dependent child segments per parent.

- A *data base record* is the data structure made up of a root segment and all segments dependent on it. There can be only one root segment type in a data base record, and it can occur only once in that record. A data base record can consist of many segment types; each, other than the root segment, having any number of occurrences, as shown in Figure 1-3. A data base record can have several levels in its structure. Figure 1-2 shows three levels: SKILL is at the first level, NAME at the second, and EXPERIEN and EDUCAT at the third.
- A *data base* consists of any number of data base records, all having the same root segment type.
- The root segment normally contains a *key field*. The value in the key field controls the sequence of the data base record within the data base.
- Segments lower in level in the structure can also have key fields. These keys can be used to sequence multiple occurrences of the same segment type within a data base record.
- The segments in a DL/I data structure are always referenced by DL/I in the *hierarchical sequence* of top-to-bottom, left-to-right, as indicated by the numbers 10 through 27 in Figure 1-3.

Now we are ready to discuss the ways in which data bases can be represented as conceptual structures.

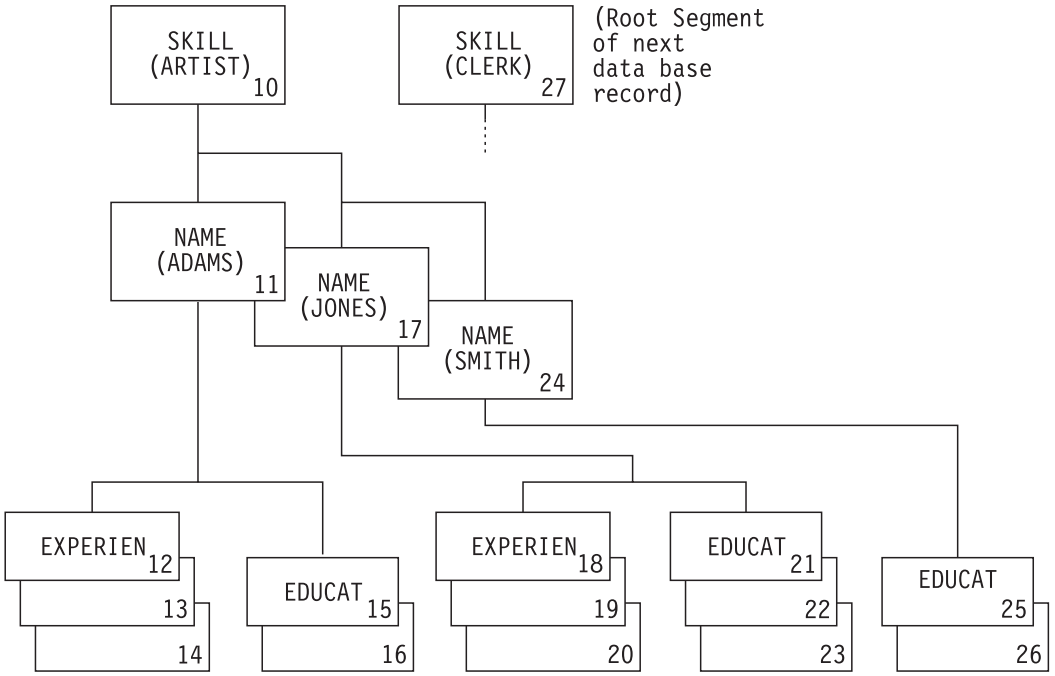


Figure 1-3. Expanded Data Base Structure

Data Base Structures

There are many ways in which a data base can be structured. Three of them are discussed in this section, as they apply to DL/I. They are called flat file, hierarchical, and network structures. A DL/I data base could be organized in any of the three ways, though the first would normally be used only as an intermediate

step in converting the conventional file organization of an existing application to a DL/I implementation.

Flat File

The method of organizing data that is most familiar is the list or table. Most of us have used them all our lives. A common name for a stored list or table is “flat file.” Figure 1-4 shows a portion of such a list, arranged in tabular form.

```
EMP-NUM  EMP-NAME          BIRTH DATE  DEPT  JOB DESCRIPTION

01432    ADAMS, W. B.       9-20-34    A10   SR. ENGINEER
49912    BATES, H. S.       5-10-45    D33   TYPIST
33961    CHARLES, A. X.    12-15-56    C02   SALESMAN
```

Figure 1-4. Flat File Structure

Each line in Figure 1-4 contains pertinent data about a particular employee and can be thought of as a *record* about that employee. All of the information on a line is related, and refers to that employee alone, but some of the items may not be unique to the particular employee. The department number, for instance, will not be unique to one person. It is possible that there may be more than one person with the same name.

In order for the information about the employee to be meaningful, it must be identified as being unique to that person. If two lines in the table contained exactly the same information, we would not be able to distinguish one line from the other. To make sure that we can tell one from the other, we have to include at least one item of data about each person that is truly unique. In our table, this is the employee number.

No two people working for a company have the same employee number. It provides a way of identifying any employee from all others, and is called a *key*. The person's birth date combined with his or her name might also be unique and could be used as a key if desired. The key that we choose to use is called the *primary key*. Any others may be used as *secondary keys*.

If we were to store the employee records on magnetic tape or other recording medium, we would most likely store them in some kind of sequential order—one record after another. The records in the table are shown in the alphabetical order of the employee's last names. If stored, we would probably store them in the order of the key—the employee number.

We could store this same information as a DL/I data base in any one of a number of different ways, the simplest of which would be as a flat file.

Hierarchical

You could, if you wished, structure many collections of data as flat files and define them to DL/I as data bases. You could use DL/I to operate on these data bases. But you would not be using DL/I as effectively as possible.

There is another familiar way of organizing data: the tree structure. An example is the corporate organization chart. Figure 1-5 shows an example of a tree structure.

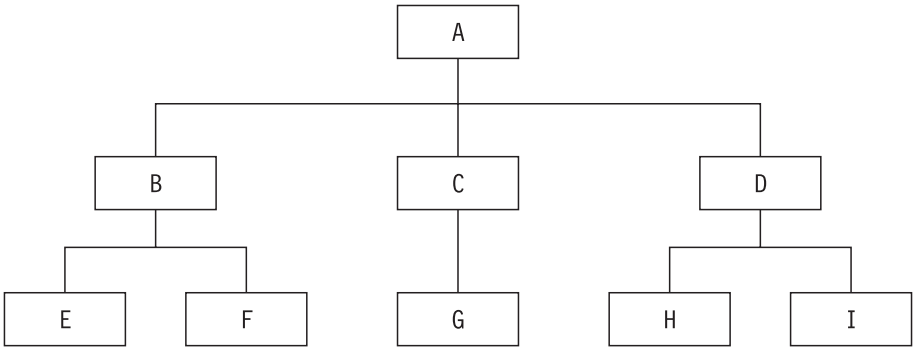


Figure 1-5. Tree Structure

A tree structure is also called a “hierarchical” structure. DL/I is designed to handle hierarchical data base structures efficiently. Figure 1-5 is hierarchical in the sense that each element except the topmost depends on the topmost or *root* element (segment). The dependent elements are arranged in *levels*. In the figure, there are three levels. Some of the elements are also dependent on other elements above them. Any element immediately below another to which it is dependent is called a *child* of that element. The element at the higher level is the *cprent* of all those immediately below it in the same path. If you trace the paths between blocks, you can see that each child can have only one parent on each level above it. This child-to-one-parent relationship is significant.

Network

The hierarchical structure just discussed can be used to represent the structure of many data bases, but the fact that there can be only one parent for each dependent element is sometimes a restriction that must be circumvented. For instance, a family tree could not be expressed as a tree structure in the sense being spoken of here, because it would require two parents at each level.

Many types of data have relationships within them that call for more than one parent for some of the elements. For instance, in a bill-of-material application, a particular product might contain part A. It might also contain sub-assemblies that also contain part A. Both the product and the sub-assembly can be considered as being parents of part A. A data base structure having this type of relationship between elements is called a “network.”

DL/I is able to process data bases having this kind of structure. It relates elements from different hierarchical data bases by using what are called *logical relationships*. Figure 1-6 shows an example of this type of structure. Note that D, G, and I each have two parents.

The next section describes the importance of good data base design in terms of design objectives.

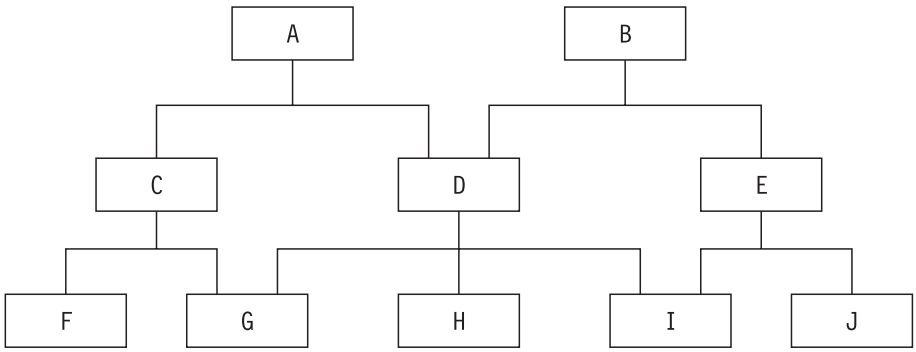


Figure 1-6. "Network" Structure

Data Base Design

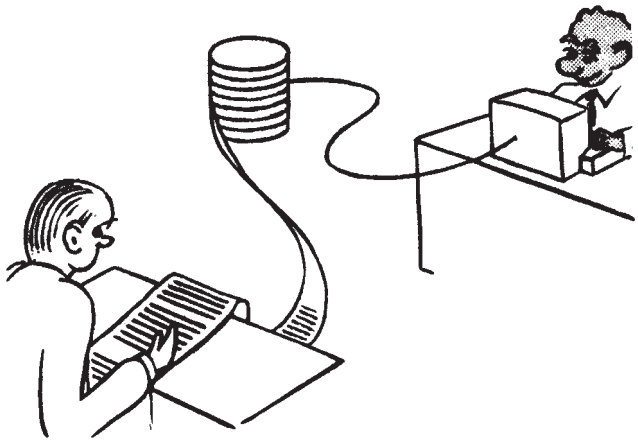
This section lists objectives that every data base design should try to meet. Benefits of good design and consequences of poor design are also listed.

Objectives

You should clearly define and document your data base design objectives. These objectives will differ from one application to another, but there are certain common objectives that every data base design should have, such as:

- To efficiently meet the requirements of the application end users
- To provide for future change and growth with a minimal disruption
- To minimize the need for redundant data
- To make the application as independent as possible from data characteristics
- To prevent unauthorized access to the data
- To protect the accuracy of the data from hardware or software failure, and from user errors.

Benefits of Good Design



Good data base design will not only meet end user's requirements, but will produce other desirable effects such as:

- Management and end users support for future data base applications
- Freeing of resources for development of other data processing applications
- Minimizing the need for tuning the data base to get better performance
- Dependable data base recovery and restart following a malfunction.

Consequences of Poor Design

Poor data base design can have harmful effects that can limit the advantages of using data bases. For example:

- A poorly designed data base may add to the cost of writing, debugging, and maintaining the programs using it.
- It may also require more hardware because of more data redundancy or less efficient use of storage.
- It may cause application programs to use more system time and resources.
- Changes and additions to the data base may require redesign.

Data Base Design Tools

The data base design section of this book provides you with a method of designing the conceptual structures to be implemented as data bases. There are other tools available that will supplement or replace that method. Among these are IBM program products such as Data Base Design Aid, DB/DC Data Dictionary, Chained File DL/I Bridge, and some industry application programs with data base design aids that you may want to investigate.

The Data Base Design Aid and DB/DC Data Dictionary are described here briefly. Sources of more detailed information are given at the end of each description.

Data Base Design Aid

This IBM program product is a collection of programs that performs a large part of the conceptual data base design process. It uses your data requirements to produce a conceptual model showing the minimum set of relationships required. It identifies inconsistencies and omissions in the requirement specifications, lists redundant data, and prints diagnostic and design analysis reports. When you're satisfied with the conceptual model, you can use it to design your physical data base.

Another benefit from using DBDA is that it standardizes gathering and recording of requirements (see Chapter 3).

For more information on DBDA (Program Number 5746-XXQ), and how to use it, see the *Data Base Design Aid Designer's Guide*.

DB/DC Data Dictionary

The DOS/VS DB/DC Data Dictionary is an IBM program product consisting of data bases that store information about your installation's data processing resources, and programs that operate on that data. These programs are used to store and modify the data definitions, produce displays and reports of the stored data, and produce definitions for the DL/I control block generation process.

The definitions stored in the dictionary data bases provide a consistent, controlled source of information for use in all of the installation's applications, but are especially valuable in the design of data bases. If you also use the Data Base Design Aid, it can give you direct input to the dictionary.

The dictionary definitions can be much more than descriptions of data base elements. You can include information about the use of data in the system, the relationships among data elements, and the relationships between data elements and business processes. These might include business entities such as personnel, departments, data processing devices, and projects.

You can use the dictionary as a data base design aid:

- To get information that defines data elements
- To get information that defines relationships between data elements or other business entities
- To get information about the number of times data elements are used
- To help eliminate duplicate data
- To control changes within data base systems.

For a description of the dictionary (Program Number 5746-XXC), see the *DOS/VS DB/DC Data Dictionary General Information Manual*. For information on how to use the dictionary, see the *DOS/VS DB/DC Data Dictionary Applications Guide*.

An Application Design Procedure

Data base design is of great importance in the successful implementation of a data processing application that uses DL/I and DL/I data bases. However, good data base design will not, by itself, ensure a successful implementation of the application. The data base design process must be part of an organized application design procedure that takes all aspects of the implementation into consideration. This section outlines a procedure that does this. It is not the only one that could be used, but it has been used successfully, and will give you a basis for developing your own procedure. The rest of this book follows this outline.

Preliminary Analysis

As you begin to consider a data processing application, you will need to make a preliminary analysis to determine whether or not you should seriously consider using data bases to implement it.

Obtaining Application Requirements

Application requirements are the results that the end users expect to receive from implementation of the application. Your first major task is to obtain and organize these requirements.

Application Analysis

The next step is to analyze these requirements to determine what tasks will be done by the application and to define programs to accomplish those tasks.

Creating Local Views From Requirements

For each of the programs that you defined that will use a data base, you must develop a local view of the application requirements. A local view is a subset of all of the data elements listed in the requirements that are used by the particular program.

Combining Local Views Into a System View

The next step is to combine the local views generated for the individual programs into an integrated structure that includes them all in an optimum arrangement—the system view. This is the conceptual data base that will be implemented as a physical data base.

Converting the System View to a Physical View

The conceptual system view is now converted into the form that will be physically implemented, by taking into account such things as the choice of an access method.

Implementing the Application

Next, you should develop an implementation plan. Once the plan is in place, the final step is the actual implementation of the application, and its integration into the operation of the data processing installation.

Chapter 2. Preliminary Analysis

This chapter reviews the process of deciding whether or not data bases should be used in your data processing installation and for your particular application. There are two main topics:

1. Initial Application
2. Adding an Application.

The first section reviews the process of determining whether or not a data processing installation should install a data base system. The second section shows you how to decide whether or not your application is a candidate for implementation with data bases when your installation has DL/I already installed.

How should you go about deciding whether or not a data processing application is a candidate for using data bases? The final decision should be made after application requirements have been collected, organized, and analyzed; however, a preliminary analysis will show whether there are any factors that would make a successful data base implementation unlikely or undesirable. It is assumed, in this book, that DL/I has been installed in your installation—or is at least on order. This chapter shows how the same considerations that should have influenced that decision can influence your decision concerning your own data processing application.

Initial Application

Preliminary analysis determines whether two vital requirements can be met. Without either of them, a data base installation is unlikely to succeed.

The first of these is the *support of high-level management* personnel in your organization. It is needed because data base installation and use cut across many areas of the organization and involve many aspects of the business.

The second is *interest from the end users* of data base applications. Without this interest, the project is likely to be rejected by its users.

If both of these requirements can be met, the use of data bases is at least feasible in your organization. At this point a team should be established to carry the investigation further. After organizing themselves and defining their responsibilities, the team should:

- Identify the objectives of the data base management system
- Examine and evaluate alternatives
- Make recommendations.

Now, how can you use this procedure in making your own preliminary analysis for the application you are responsible for? If yours is the first, or one of the first, applications to be developed after DL/I has been added to the installation, the process is virtually identical, on a smaller scale.

You must also have management support. In your case, the minimum should be the management level of the involved user groups and the data processing department. Without their support you will have a very difficult task. There will be a need for management decisions and for use of management authority to ensure support from all the areas, and to resolve the conflicts that may arise. If well organized presentations do not result in this support early in the process, the project should be stopped there until it can be obtained.

You must also have the interest and support of the end user groups that will be using the application. With data base applications new to your installation, there may be reluctance on the part of some end users to leave the existing, familiar method of doing things. They may see changes in their jobs and their responsibilities because of it. They may see loss of control over what they presently control. Potential users should be shown how data bases can provide better, faster service with more accurate and consistent results. They will be interested in such things as the elimination of inconsistencies between reports and the ability to obtain up-to-date information rapidly. Again, if this interest and support is not there, the project should be delayed until it is.

If you don't feel that these two requirements can be met, you will have to enlist the support of higher management, or recommend that the application be dropped or implemented differently. Once you are satisfied that the requirements are fulfilled, you can set up a team and begin the procedure described in the next chapter. The team should include people from both the data processing and user groups. Including user personnel assures their continued interest and support.

Adding an Application

If you are responsible for adding an application in an installation with DL/I already installed, and with applications already operating, your preliminary analysis should include a similar check on the two requirements listed above. If the previous applications are successful, the support you need should be available. If the previous applications have failed, you should understand why before continuing. If management and user resistance seems too firmly imbedded, you should recommend that the application be implemented another way, delayed, or implemented in small steps.

Chapter 3. Collecting Application Requirements

This chapter describes the process of collecting the objectives to be met by your application. It is divided into five major sections:

1. Interviews
2. Existing Sources
3. Questionnaires
4. Documentation
5. Reviewing the Requirements.

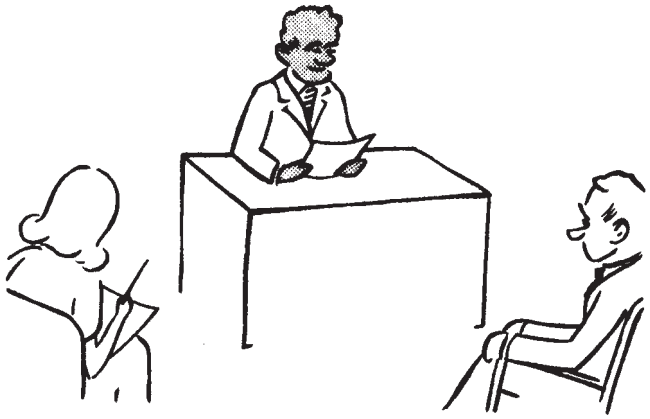
The first section discusses interviews as the main source of application requirements. The second describes existing applications as secondary sources. The third section mentions questionnaires as a supplement to interviews. The next section describes the documentation of the requirement data in a requirements listing that will serve several purposes, including generation of a data dictionary. The last section discusses the review of the documented requirements with the end users, as a check on their accuracy and completeness.

Application requirements are the results expected from your application when it becomes operational. Requirements must be collected from the end users. Depending on the size and organization of your company, the application requirements group may consist of one person, or it may be a team made up of a number of members from several areas of interest—including representatives of the end users. No matter how large the team is, there are several methods which can be used to collect the requirements. This chapter describes them and how to document the information obtained.

Interviews

Organizing the Interviews

Interviews are the most important method of obtaining application requirements. If the interviews are to be done by members of a team, each interview should be conducted by a member of the data processing department and a member from a user group (preferably not the user group being interviewed, to ensure objectivity).



Interviewing should begin with the manager of the user group or functional area. This first interview provides an opportunity to learn the organizational structure of the group and identifies the persons the manager feels should be interviewed at lower levels, it helps to gain the cooperation of those interviewed at lower levels since they know that their management is interested in their working with you, and it provides an opportunity to learn the manager's view of the group's application requirements. The interviews should then continue down through the structure, learning the requirements as they are seen at each level. The person interviewed at each level should identify the person or persons to be seen on the next lower level.

Conducting the Interviews

It is important that the first step undertaken in each interview is the establishing of a good relationship with the person being interviewed. Without this, the person is much less likely to feel involved in the purpose of the interview, with the result that important information may be missed or glossed over. The person being interviewed may even be hostile to the idea of the application being implemented with data bases involved; for fear that it may affect the job adversely, losing control of data, or because previous experience with data processing applications has been unsatisfactory. In any case, a hostile attitude may result in the omitting or “forgetting” of important information. The person's attitude can be judged by analyzing the answers given to questions about previous experience in using data processing. If past experience seems to have been good, or if there has been no past experience, the interviewee is likely to be interested and cooperative. If past experience is described in terms of dissatisfaction, an effort must be made to ease the fears that the present application will also be unsatisfactory. If this can't be done, it would be a good idea to try to interview another person from this area at a later date.

The attitude of the interviewers is also important. You should show interest in, and enthusiasm for, the successful implementation of the application but you should not seem overconfident or condescending. You should be interested in the person being interviewed, the person's job and in the effect the implementation of the application may have on that person's work. If it is true, you should assure that person that the job will be made easier and that it will not be eliminated: two items he may be concerned about.

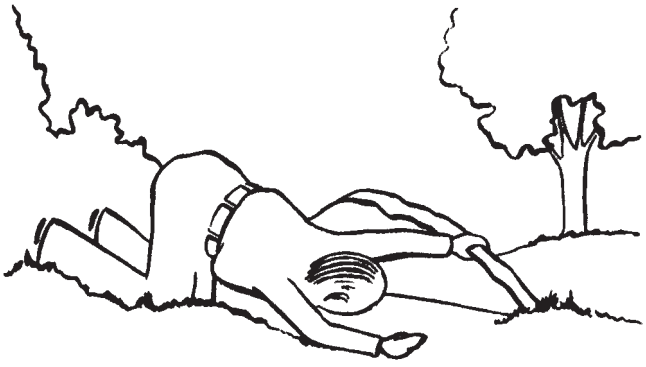
If you show a genuine interest in the person's work, he or she will almost certainly be willing to talk about it in detail, once started in the right direction. You can keep the flow coming and on the right track with the proper choice and timing of questions. Unfortunately, some people are so familiar with their jobs that they forget how much of the day-to-day operational information they keep in their heads. You will have to try to dig this out by asking for a step-by-step description of each activity.

Have the person show you, and describe, any reports presently being used—either manually generated or generated through data-processing. Discuss the items in the reports to find which ones are truly needed and which do not seem to be necessary at all. Find out if the person generates any data for existing data processing applications. Also ask if he or she maintains any job related files for personal use; either as data processing files, card files, or hand-written items. These may be important indications of user requirements, and may be useful to other users in the organization. Ask the person what information he or she would like to have that is not presently available. If there are any of these, and the application can include them, the person you are interviewing is more likely to become an enthusiastic supporter of the implementation of the application; but don't raise false hopes. Some items one person would like to have may be of no use to any other user and might be too costly to implement.

Try to avoid using technical language that the person being interviewed may not understand. It may confuse him or her and cause you to lose information or rapport. This is an area where a non-data processing member of the interview team can be of help. He or she can call a halt to jargon by asking for clarification that the person being interviewed might be embarrassed to ask for. If the interviewee uses jargon that you do not understand, be sure to get clarification before going on.

Existing Sources

A secondary source of application requirements is found in existing programs and files. Each of these should be examined after the interviews have been completed.



You should literally “leave no stone unturned.” Look carefully for data items that are used, but that were not mentioned in the interviews. These may be things that the user never sees, but that are used in the generation of a report or as input to later programs.

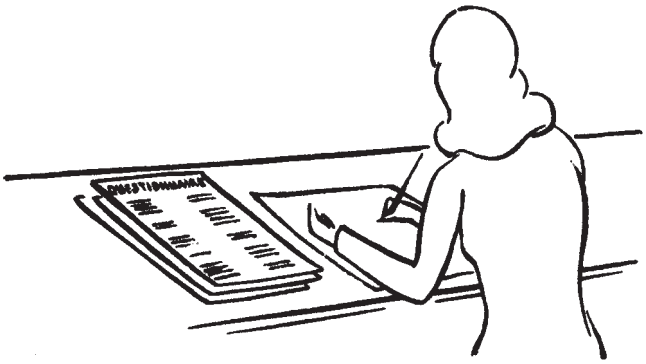
The documentation of existing data processing systems should be examined for user requirements that the users might not be aware of, even though they are involved in producing reports or processing data. Constraints and other important considerations may have been documented and later forgotten.

Questionnaires

Questionnaires can be used as supplements to, but not as replacements for, the interviews. These might be distributed to members of the user groups who were not interviewed because of time and manpower constraints. They could ask for:

- A list of reports used
- A list of data items generated as input to files or programs
- Items the user does not presently see, but would like to see
- Any job related files of any type maintained by the user.

The next step is to document the information you have obtained.



Documentation

Through the methods described above, you should be obtaining data on the requirements of each of the user groups or functional areas affected by the application. The types of data to be collected include:

- What the requirement is (a report listing certain information, for instance)
- How often the requirement must be met (once a week)
- How fast the system must respond (within eight hours)
- What actual data items are needed
- The format of the data items
- An estimate of the quantity of each data item
- Where the data will be stored (locally or on a remote system)
- Where the data will be processed (locally or remotely)
- Any special considerations or constraints.

If done thoroughly, you will accumulate a mass of data that will have to be documented in some fashion to be usable. One way to do this is to number the requirements and list each one on a separate page or pages. Assemble the pages containing the requirements of each particular user group or functional area into a numbered chapter. Assemble the chapters into a book with an index that shows the group described by each chapter, and the requirements for each group. This book is a summary of the entire application and provides a controlled, central source of the data. If you keep it up-to-date, it will be a valuable tool for years to come, in addition to being of vital importance as a source document for the rest of the application design process.

Reviewing the Requirements

Now that you have collected and documented the requirements, you should review them with the end users. This will help to ensure that the information is accurate and complete. It will provide an opportunity to make sure that you and the end users understood each other during the interview process.

You should do this review through repeat interviews, if possible. If not, copies of the appropriate sections of the requirements listing should be given to the people who were interviewed, with instructions on how you wish them to verify the information. If the review turns up inaccuracies or problems, you should interview the people directly involved to resolve the difficulty.

When the review has been completed, and both you and the end users are satisfied that you have all the available information, you are ready to begin to use your data. The next step is to perform an analysis of your application. The requirements you have collected are input to that process, as described in the next chapter.

Chapter 4. Application Analysis

This chapter shows you how to perform an application analysis, using the documented application requirements. The analysis results produces number and type of programs required by the application, and analyzed application data to be used in the next step—creating local views. There are five main topics:

1. Defining the Tasks
2. Defining Programs to Accomplish the Tasks
3. Establishing Naming Conventions
4. Creating a Data Dictionary From the Requirements Listing
5. Analyzing Application Data.

The first section describes the break-down of the application into tasks. The second section describes the definition of programs to accomplish those tasks. The third section deals with the establishment of naming conventions to minimize confusion and errors. The fourth section discusses the creation of a data dictionary to maintain and control the application data, using the information in the requirements listing. The last section deals with the analysis of the application data, and its organization.

This chapter, and those following, uses as an example a company that provides technical education to the employees of its customers. This education company has a headquarters location and several local education centers in various cities. Headquarters is responsible for developing all of the courses that the company offers. Each of the education centers is responsible for scheduling classes and enrolling students in them, at its own location.

The education company has a number of data processing applications—such as payroll—implemented, but the one that will be used as an example is that application directly involved with the courses the company offers. There may be several offerings of a particular course at different education centers. Transistor theory could be presented in New York, Chicago, and Boston. Each of these is a separate class. A class is a single offering of a particular course on a specific date at a particular education center. Circuit design in Chicago, starting on 1/23/84, for instance.

Defining the Tasks

The application requirements have been collected for the course application described above. Examination of the application requirements shows that there are several individual data processing tasks, or business processes, needed to meet those requirements. One of the requirements is for each education center to produce, weekly, a current roster for each of its classes. The current roster is to provide information about the class and the students enrolled in it. Headquarters requires that the current rosters be in the format shown in the sample in Figure 4-1.

```
CHICAGO                                                    1/06/84

    TRANSISTOR THEORY                  41837
    10 DAYS
    INSTRUCTOR(S): BENSON, R.J.        DATE: 1/16/84

  STUDENT        CUST   LOCATION             STATUS  ABSENT  GRADE
_____________________________________________________________________
 1.ADAMS, J.W.    XYZ    SOUTH BEND, IND      CONF
 2.BAKER, R.T.    ACME   BENTON HARBOR, MICH  WAIT
 3.DRAKE, R.A.    XYZ    SOUTH BEND, IND      CANC
  •
  •
  •
33.WILLIAMS, L.R. BEST   CHICAGO, ILL         CONF


  CONFIRMED = 30
  WAIT LISTED = 1
  CANCELED = 2
```

Figure 4-1. Current Roster

Another requirement is for a schedule of all the classes given each quarter. This is generated and distributed monthly by headquarters. The schedule is sorted by course code and printed in the format shown in Figure 4-2.

```
                  COURSE SCHEDULE            1/06/84

    COURSE:  TRANSISTOR THEORY     COURSE CODE:  41837
    LENGTH:  10 DAYS               PRICE:  $280


        DATE                   LOCATION
_____________________________________________________________
       APRIL 16                BOSTON
       APRIL 23                CHICAGO
       •
       •
       •
       NOVEMBER 19             LOS ANGELES
```

Figure 4-2. Schedule of Classes

Another requirement is for each of the education centers to print a quarterly report that lists the courses that each of its instructors is qualified to teach. The instructor skills report is to be in the format shown in Figure 4-3.

```
        INSTRUCTOR SKILLS REPORT              1/06/84

   INSTRUCTOR           COURSE CODE     COURSE NAME
_____________________________________________________________________
   BENSON, R. J.        41837           TRANS THEORY
   MORRIS, S. R.        41837           TRANS THEORY
                        41850           CIRCUIT DESIGN
                        41852           LOGIC THEORY
   •
   •
   •
   REYNOLDS, P. W.      41840           MICRO PROG
                        41850           CIRCUIT DESIGN
```

Figure 4-3. Instructor Skills Report

The final requirement is for a monthly report to be generated by headquarters that shows the schedules for all of the instructors in the company, in the format shown in Figure 4-4.

```
                  INSTRUCTOR SCHEDULES                    1/06/84
_____________________________________________________________________
INSTRUCTOR         COURSE         CODE     ED CENTER     DATE

BENSON, R. J.      TRANS THEORY   41837    CHICAGO       1/16/84
                   TRANS THEORY   41837    NEW YORK      3/05/84
MORRIS, S. R.      TRANS THEORY   41837    NEW YORK      1/09/84
                   CIRCUIT DES    41850    CHICAGO       1/23/84
                   LOGIC THEORY   41852    BOSTON        2/27/84
REYNOLDS, P. W.    CIRCUIT DES    41850    LOS ANGELES   3/12/84
```

Figure 4-4. Instructor Schedules

You can use the application requirements listing that you generated for your application to produce a similar break-down of the total application into tasks. They may not, of course, all turn out to be reports. There are many other types of tasks that may make up a data processing application. Once you have determined what needs to be done, and have documented those tasks, you are ready for the next step of defining the programs that are needed to accomplish them.

Defining Programs to Accomplish the Tasks

An application like that of the education company described above could be implemented as one huge, complicated program. But it is apparent, once the tasks have been defined, that such a program would be inefficient, costly to produce and maintain, and wasteful of data processing resources. The various reports need to be prepared at different times and places. For instance, one of the reports needs to be prepared weekly, two monthly, and one quarterly. Also, two of the reports are produced by headquarters and two are produced by the education centers. Obviously, individual programs are needed. This section describes a method of defining the programs to be used to accomplish the tasks of the application.

Requirements and Considerations

There are many requirements and considerations that will influence your decision on the number and type of programs needed to implement the application tasks you have defined.

Input Requirements

The requirements listing is the source of information about input requirements. For instance, Figure 4-1 shows that the following data will be needed as input to produce the current roster report:

- The location of the education center
- The name of the course
- The course number
- The length of the course
- The name(s) of the instructor(s)
- The starting date of the course
- The names of the students enrolled in the class
- The name of the company each student works for
- The location of the company
- The student's enrollment status
- The number of times the student has been absent
- The student's final grade.

The summary information in the lower left corner is generated by the program itself. The date in the upper right corner is the date the program was run, and is generated by a program function.

Output Requirements

The output requirements are also determined from the requirements listing. If the output is a report, note items that appear on the report that are not included in the input items; such as the totals in Figure 4-1, or the date that the report is produced. Note whether the output is a printed report or whether it is a display screen, a tape, punched cards, or a disk file. Some tasks may require several outputs on different mediums.

Frequency of Use

It is important to know how often the program will be used. This can influence the type of program and how much attention should be given to its efficiency. A program that is run once a year would probably be a batch program and would not have to be optimized for fast execution. A program that is run every day should be as efficient as possible. In the education company application the current roster must be produced each week, the class schedule monthly, the instructor skills report quarterly, and the instructor schedules monthly. Your application may have a requirement for a real-time, online display or for a report that is produced once a year for an annual summary.

Type of Access

What type of access does each task require: sequential or direct? If the data used by the task is always, or usually, accessed sequentially depending on the value of a particular data item (such as a name field sorted in alphabetical order), that data will be stored and the program using it will be structured much differently than in the case where individual data items are always accessed directly. For instance, the education company course schedule makes use of information that is sorted in order by course code. The program that produces this report might access the

data sequentially by course code. On the other hand, there might be a requirement to be able to produce, on a display screen, the schedule of a particular instructor. This program would need direct access to the information about the individual instructors and their schedules.

Type of Processing

Also important in determining the number and type of programs needed is information about the type of processing the task requires. Will the program read the input data, but never add to or update it? Will the program only be involved with updating the data? Must the program read the data, make decisions, and then update the data depending on those decisions? Will the program be used to initially load the data onto its storage medium, or will it add new data to that already existing? Must the program delete data previously stored? Perhaps the task requires that data be added in real-time, to keep it current. To do this, an online program will be required. Perhaps data can be accumulated and added once a month, with read-only access between updates. In this case, a batch program to do the updating, and another program to do the reading would probably be needed.

Online vs Batch

You must also decide whether programs should be online or batch. Some of the considerations have been mentioned above. In general, the following are some of the reasons for implementing a task with a batch program:

- Low frequency of use
- Large amount of output
- Input data seldom changes
- Task has low priority
- Data does not have to be completely current
- Updating can be done in batches whenever convenient.

Some reasons for using an online program are:

- High frequency of use
- Small amount of output required at a given time
- Data changes frequently
- Data must be current
- Many end users must have access to data in real-time.

In the education company application, the instructor skills report could certainly qualify for a batch program since it is produced quarterly and the data would change infrequently. On the other hand, the program that updates the information for the current roster would be a candidate for an online program since the student data would change daily. An online program could also be used to extract information on the schedule of individual instructors.

Distributed vs Local Processing

Where will the programs be run? If the company has more than one processor installed, must the programs be capable of running on all of them? If the capability for distributed data processing is available, which jobs should be run locally and which remotely? If the education company has the facilities, the current rosters could be run as local jobs at the individual education centers. The monthly course schedule could be run at headquarters and distributed to the education centers. Different types of data could be stored at different locations, but accessed from any location.

Resource Requirements

Also to be considered are any changes in resource requirements that may occur because of the application. If these requirements are too great to be met within budget and time restrictions, changes may have to be made in the implementation of the application—using different types of programs or different ways of storing data. Some of these areas are mentioned below.

Processor Capability

- Does the installed processor(s) on which the programs will be run have sufficient capacity to give the turn-around time desired?
- Is sufficient processing capability available in every location where the program is to be run?

Real Storage

- Is there sufficient installed real storage to meet the requirements with adequate performance?

Data Communications

- If programs are to be run online, has CICS/VS been installed, or is it planned?

Hardware

- Will additional disk storage be required?
- Because of response time requirements, will faster auxiliary storage devices be required?
- Will the amount of printed output require additional or faster printers?
- Will additional unit record equipment be needed?
- Are more, or different, tape drives needed?
- Will additional terminals be needed?

Data Management Support Requirements

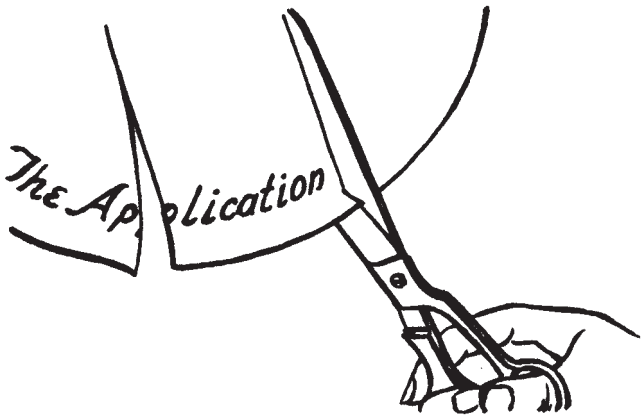
- Will the installed data management support handle the requirements of this application?
- If data bases are to be used, is DL/I installed or planned?

Personnel

- Will additional personnel be required in the programming or operational areas?
- Will existing personnel require additional training?
- Will fewer people be required in some area or areas because of the implementation of the application?

Data Base vs Conventional File-processing

Should the application, or some part of it, be implemented with programs that access data stored in data bases, rather than conventional files? It may be that the application does not lend itself to implementation with data bases. In that case, conventional file-processing programs will be used. If data bases are feasible and desirable, and DL/I is installed or planned, then the programs are candidates for data base processing.



Defining Application Programs to Meet the Requirements

Now that you have reviewed and considered all the requirements and considerations that might affect the number and type of programs needed to implement your application, you are ready to separate it into manageable pieces. The education company application might be divided as follows:

- The course schedule report will be a batch program run monthly at headquarters.
- The instructor schedules report will be a batch program run monthly at headquarters. There will also be an online program that can produce the schedule of one or more individual instructors.
- The current roster report will be an online program run weekly at the individual education centers to produce printed output. It will also produce output at terminals at any time when requested. Input, such as updating and additions, can be made through a terminal at any time.
- The instructor skills report will be a batch program run by each of the education centers quarterly.
- All of these programs make use of data bases.
- There will also be batch programs to initially load the data bases.

Naming Conventions

It is important, as you proceed, to adhere to the naming conventions established in your installation. If there are no naming conventions in place, you should have them established. Any convention that is logical and consistent can be used. The *Standards Manual for DOS/VSE* describes one that should be considered. It is summarized in Appendix B. The example of application design with data bases in Appendix A follows the naming convention used with the sample programs shipped with DL/I. It is described there. Whatever naming convention is used in your installation, it should cover all aspects of programming from job names to the names of the smallest data items and should be strictly adhered to. Use of a data dictionary will help to assure this compliance.

You will make use of the naming convention when you create your data dictionary. The next section tells you how to do that.

Creating a Data Dictionary From the Requirements Listing

At this point, you should begin to create a data dictionary, if you have not already done so.

A data dictionary is a single, consistent, controlled source of information about the data used in a data processing installation. The material stored in the data dictionary can be of many different kinds, such as:

- Characteristics of each item of data including its name, size, position, and content.
- Relationships between data items including hierarchical location, pointers, and indexing.
- "Where used" information about what programs and transactions use the data.
- "How used" information, indicating how the data can be changed or accessed and any access limitations.
- Any other information desired about the data or the programs that use it.

You can create your own data dictionary or use an existing data dictionary product. IBM offers a program product, DOS/VS DB/DC Data Dictionary, program number 5746-XXC, that provides many features, such as: a data definition interface, storage of data definitions, displays and reports of the defined data, and production of definitions for the DL/I control block generation process.

You should start the data dictionary by entering in it the data from the requirements listing. Every data item in the requirements listing should have an entry in the data dictionary. Use the naming convention discussed above, so that the names will be consistent and meaningful.

An example of a simple, user-created data dictionary is shown in Appendix A, in the section "Creating a Data Dictionary From the Requirements Listing."

The data dictionary is the final tool you need for the task of analyzing application data.

Analyzing Application Data

You are now ready to begin an analysis of the data you have collected and organized. Generate an individual view of the data for each of the programs you have defined. All programs need to have their inputs and outputs defined, and their layouts specified. Any special considerations or processes need to be documented. The final, analyzed and documented, data will be used by the application programmer as input when creating the program.

Programs that make use of data stored in data bases need additional steps in the analysis process. These are described in the following section, "Data Base Design."

Part 3. Data Base Design

Chapter 5. Creating Local Views From Requirements

This chapter shows you how to create a local view of the application data for each process in the application. There are two main sections:

1. The Three Step Process
2. Building the Local Views.

The first section describes a three step process that will generate an optimized organization of the particular portion of the application data that applies to this business process. The second section describes a procedure for building the local views, which includes use of the three step process.

It is here that application design for an application making use of DL/I and DL/I data bases diverges from the procedures of traditional applications. At this point, it is necessary to perform the design of the conceptual data structure or structures that will be implemented as a data base or bases for your application. This section of the book covers the whole process in several steps. The first step is the creation of local views of the application data for each of the business processes in the application, and is the subject of this chapter.

Some of the terms you will need to know to understand this material are described here.

- A *local view* is a description of the data that an individual business process requires. It includes a list of the data elements, a conceptual data structure that shows how you've grouped data elements by the entities that they describe, and the relationships between each of the groups of data elements.
- A group of data elements that describes a particular entity is called a *data aggregate*. For example, from the education company application described in the last chapter, the data elements STUSEQ#, STUNAME, CUST, LOCTN, STATUS, ABSENCE, and GRADE all apply to a student. This group of data elements is called the student data aggregate. STUSEQ#, STUNAME, CUST, LOCTN, STATUS, ABSENCE, and GRADE are the names of data elements.

  Data elements have values as well. For the student data elements the values are a particular student's sequence number, name, company, company location, status in the class, absences, and grade. The values of the data elements in the data aggregate are not necessarily unique; all the students in the class are described in the same terms. The combined values, however, of a data aggregate occurrence are unique. No two students can have the same values in all of these data elements.
- As you group data elements into data aggregates, you will look at the data elements that make up each group and choose a data element that uniquely identifies that group. Sometimes you have to use more than one data element to uniquely identify an aggregate. The data element or group of data elements in the aggregate that uniquely identifies the aggregate is called a *key*. When the key is made up of two or more data elements it is called a *concatenated key*.

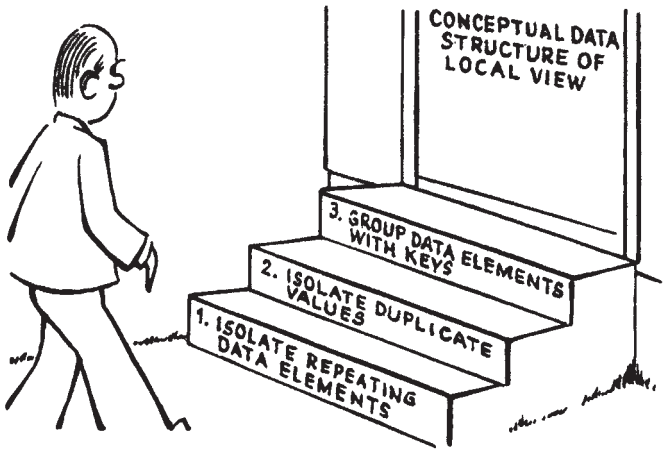
- Once you have grouped data elements by the entity they describe, you can determine the relationships between the aggregates. These relationships are called *mappings*. Based on the mappings, you can design a conceptual data structure for the business process. This data structure is a way of analyzing the relationships between the data elements required by the business process. It is not a data base. It will later be used as input for designing a data base.

The Three Step Process

By following the three steps introduced in this section you will develop a conceptual data structure for a business process's data. This is not the only method that can be used, but it is one that is effective. After this conceptual structure has been used later in the design of the physical data base, it may have to be modified to better suit the overall requirements of the application, while still meeting the requirements of the business process it describes.

The three steps to the conceptual data structure of a local view are:

1. Isolate repeating data elements that appear in a single occurrence of the data aggregate.
2. Isolate duplicate values that appear in multiple occurrences of the data aggregate.
3. Group data elements with their controlling keys.



1. Isolating Repeating Data Elements

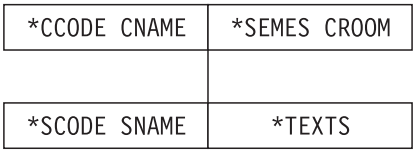
The first step requires that you:

- Make a sample of a single occurrence of each data aggregate you are concerned with.
- List every data element that you associate with each of those aggregates.
- Assign a value to each of the data elements.
- If a data element can have multiple values in a single occurrence of the data aggregate, note it as a multiple.
- Choose the data element or elements in each aggregate that best qualify as keys. Identify them with asterisks.

To introduce the process, we will use a college class roster as a simple example. For purposes of this example, assume there are four data aggregates as shown below. The list for a single occurrence would look like this:

Data Element		**Value**
*Course number	(CCODE)	1001
Course name	(CNAME)	English literature
*Semester	(SEMES)	1983-4
Classroom	(CROOM)	332
*Student number	(SCODE)	multiple
Student name	(SNAME)	multiple
*Texts	(TEXTS)	multiple

The data elements that contain multiple values must be isolated from the other data elements. Do this by showing each data aggregate in a box, and by putting the boxes containing the repeating data elements below the other boxes, like this:



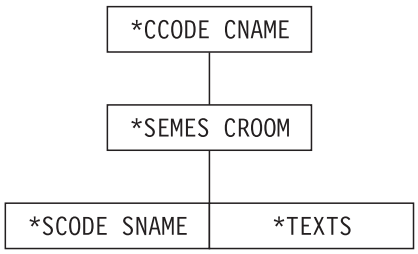
Be sure to keep data elements with their controlling keys.

2. Isolating Duplicate Values

The second step requires that you make a sample of multiple occurrences of the data aggregate, with values assigned, as shown here:

Data Element	**Value**	**Value**
*CCODE	1001	1001
CNAME	Eng lit	Eng lit
*SEMES	1983-4	1984-5
CROOM	332	235
*SCODE	multiple	multiple
SNAME	multiple	multiple
*TEXTS	multiple	multiple

Now examine the list for data elements that have the same value in both occurrences. These data elements must be isolated. This is done by moving their boxes to a higher level, as shown here:



You should choose identifying keys for any data elements that are left without keys because of this shifting.

3. Grouping Data Elements With Their Keys

In each of the first two steps, you should have kept each data element in the same group as its identifying key. The third step is a check that you have done that. Check each data aggregate to make sure that each element is with the aggregate that contains its controlling key. If not, move the element into that aggregate. Check to see that all data elements that are identified by a concatenated key are identified by every part of that concatenated key. If any are not identified by the whole key, move them and the portions of the key that they are identified by out into a separate group. For instance, consider a data aggregate containing data elements CCODE, SEMES, SCODE, SNAME, and TEXTS. Elements CCODE, SEMES, and SCODE are a concatenated key. Element SNAME is identified by the whole concatenated key. Element TEXTS is identified by CCODE, but not by SEMES or SCODE. Element TEXTS should be moved into a box of its own as a new data aggregate, with TEXTS as the key. Where this block should be placed in the data structure depends on a re-evaluation of steps one and two. An example of this condition is shown later in this chapter, in the section called “Using the Three Step Process.”

You are now ready to implement the process we have described, and build a local view of the data for each business process.

Building the Local Views

The three step process described above is a tool that will help you in the design of a conceptual data structure, but this is only part of a procedure that you can use to accomplish the total design of a local view. This section gives you details about that procedure, and shows you how the three step process fits into the procedure.

Analyzing Associations

The first thing you need to be concerned with in the design of a local view of a business process is the analysis of associations between the data elements that are listed in the requirements for that process. We will use, as an example, the class roster report requirement of the education company described in Chapter 4.

Let's analyze the associations within the class roster data aggregate. Two things are required:

- Knowledge about the data elements
- A definition of identifier.

The knowledge about the data elements comes from the interviews that were conducted with the end users, and from the requirements listing and data dictionary.

The definition of identifier is:

> Within a data aggregate, a data element A is the identifier of another data element B if every value of A never has more than one value of B associated with it.

A identifies B. For a given value of A, there can be only one possible value of B.

These are the data elements needed to fill the class roster report requirements:

EDCNTR
DATE
CRSNAME
CRSCODE
LENGTH
INSTR
STUSEQ#
STUNAME
CUST
LOCTN
STATUS
ABSENCE
GRADE

Applying the definition and our knowledge of the class data aggregate, we see that

- Assuming that all courses can be given at all education centers, then EDCNTR is not identified by any other data element.
- Although many classes can start on the same date, the starting date (DATE) of a particular class is identified by either CRSNAME or CRSCODE.
- The name of the course (CRSNAME) is identified by CRSCODE.
- The course code (CRSCODE) is identified by CRSNAME.
- The length of the course (LENGTH) is identified by either CRSCODE or CRSNAME.
- The name(s) of the instructor(s) (INSTR) is related to a class, but to no single data element.
- Though assigned by the education center, the student sequence number (STUSEQ#) isn't identified by EDCNTR, or any other data element, except STUNAME.
- The name of the student (STUNAME) is identified by the STUSEQ#.
- The name and location of the student's company (CUST and LOCTN) are related to the student, but not identified by any data element in the aggregate.
- The three remaining data elements (STATUS, ABSENCE, and GRADE) are all identified by either STUNAME or STUSEQ#.

A data element may be identified by a combination of other data elements, rather than, or in addition to, a single one. For instance, STUSEQ# is identified by

STUNAME, as we just saw; but, for a particular class, it is also identified by the combination of EDCNTR, DATE, and either CRSCODE or CRSNAME. INSTR is also identified by this same combination, since it identifies a particular class the instructor(s) will teach.

Identifying Keys

As was mentioned above, a key is a data element or group of data elements that uniquely identifies a data aggregate. These can be called primary keys. We can also have keys that are used, not for unique identification, but to identify certain properties. These are called secondary keys. The two types are discussed separately below. Either type of key can be made up of a combination of data elements to form a concatenated key.

Primary Keys

The primary key is the data element or combination of data elements within a data aggregate that is used to uniquely identify an occurrence of that data aggregate. Its value is the value that is searched for when a request is made for a particular occurrence of the data aggregate. Without the use of a unique key it would be impossible to be sure that you had located the individual occurrence you had requested.

Any data element that identifies at least one other data element is a candidate for primary key. Turning again to the class data aggregate we see that EDCNTR, DATE, CRSNAME, CRSCODE, STUSEQ#, and STUNAME are all possible candidates. Remember that data elements can also be identified by combinations of other data elements. Thus, such combinations as EDCNTR, DATE, and CRSNAME or CRSCODE are also candidates. The choice of the primary key from among the candidates is largely a matter of common sense applied to knowledge of the business process. For instance, both STUNAME and STUSEQ# are candidates, but STUNAME would not be a good choice because it is possible for more than one student to have the same name. STUSEQ# is designed to be a unique number for each student, so it would be the better choice.

The primary keys of the class roster data aggregates are indicated here by asterisks, with the individual aggregates shown separately:

*EDCNTR	*STUSEQ#	*INSTR
*DATE	STUNAME	
*CRSCODE	CUST	
CRSNAME	LOCTN	
LENGTH	STATUS	
	ABSENCE	
	GRADE	

Secondary Keys

A secondary key is not a key that uniquely identifies an occurrence of a data aggregate. Instead, it identifies those occurrences that have a property named by the key. A secondary key can be used to locate those occurrences. For instance, if there was a requirement for a listing of all courses of a particular length, establishing LENGTH as a secondary key would make it possible to address those occurrences directly, without searching through every occurrence of the data aggregate. Further information about the use of secondary keys will be found in the section “Identifying Alternate Processing Sequences (Secondary Indexing Candidates),” later in this chapter.

Now you are ready to make use of the three step process described earlier in this chapter.

Using the Three Step Process

The three step process is used to develop a conceptual data structure for your local view. The current roster requirement of the education company continues to serve as the example that is used to illustrate the process.

1. Isolating Repeating Data Elements

The first step is to look at a single occurrence of the data aggregate for data elements that repeat. Table 5-1 shows an occurrence of the class aggregate.

Table 5-1. Single Occurrence of Class Aggregate

Data Element	Class Aggregate Occurrence
EDCNTR	CHICAGO
DATE	1/16/84
CRSNAME	TRANSISTOR THEORY
CRSCODE	41837
LENGTH	10 DAYS
INSTR	multiple
STUSEQ#	multiple
STUNAME	multiple
CUST	multiple
LOCTN	multiple
STATUS	multiple
ABSENCE	multiple
GRADE	multiple

The data elements that are listed as multiple are the data elements that repeat. Separate these repeating data elements by moving them to a lower level, keeping data elements with their controlling keys.

The repeating data elements for a single class are STUSEQ#, STUNAME, CUST, LOCTN, STATUS, ABSENCE, and GRADE. INSTR is also a repeating data element because some classes require two instructors, although this class requires only one.

When you isolate repeating data elements, you arrive at the structure shown in Figure 5-1.

The asterisks in Figure 5-1 signify the data elements that make up the key. For the class aggregate, it takes three data elements to make a key that will uniquely identify the course.

After you have shifted repeating data elements, make sure that each element is in the same group as its key. INSTR is separated from the group of data elements describing students because the information about instructors is unrelated to the information about students. For instance, the student sequence number does not control who the instructor is.

The key of the topmost aggregate is complete within itself. The complete key of an aggregate at a lower level consists of its own key concatenated with the keys of the aggregates associated with it at higher levels. This is because a key at a lower level means little if you don't know the keys of the higher aggregates. For

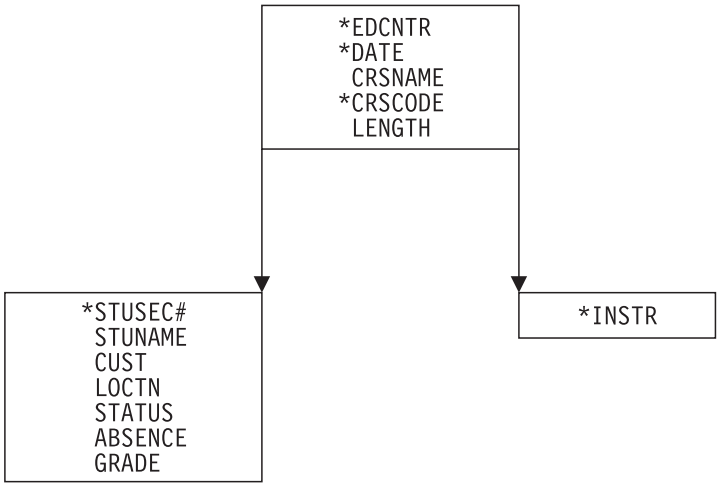


Figure 5-1. Current Roster Step 1

example, if you knew that a student's sequence number was 4, some of the information you would be able to find about the student associated with that number would be meaningless if it were not associated with a particular class. But, when the key for the student aggregate is concatenated with that for the course aggregate (EDCNTR, DATE, and CRSCODE), the student information is uniquely identified.

Figure 5-1 shows these aggregates with their corresponding keys:

- Course aggregate: EDCNTR, DATE, CRSCODE
- Student aggregate: EDCNTR, DATE, CRSCODE, STUSEQ#
- Instructor aggregate: EDCNTR, DATE, CRSCODE, INSTR.

2. Isolating Duplicate Values

The second step is to consider multiple occurrences of the aggregate, looking for duplicate values. To choose the occurrences correctly, you need to think about what information is actually contained in the data aggregate. The class aggregate describes one offering of a particular course. We call this a class. There can be several classes for one course. Each one is a separate occurrence of the aggregate. Another course would have separate occurrences for each of its classes. If there are any duplicate values, they would be most likely to show up in occurrences of the same course. If we had to look at the course information in a slightly different way, our choice might be different. For instance, if we had an application that required the listing of all the classes associated with each course, we might have a data aggregate made up of CRSCODE, CRSNAME, DATE, and EDCNTR. In this case, each occurrence of the aggregate would be a different course, so we could choose any two occurrences to look at. In our case, we will look at two occurrences of course number 41837. Table 5-2 shows these occurrences. There are several duplicate values in the figure.

Table 5-2. Multiple Occurrences of Class Aggregate

Data Element List	Occurrence #1	Occurrence #2
EDCNTR	CHICAGO	NEW YORK
DATE	1/16/84	3/12/84
CRSNAME	TRANS THEORY	TRANS THEORY
CRSCODE	41837	41837
LENGTH	10 DAYS	10 DAYS
INSTR	multiple	multiple
STUSEQ#	multiple	multiple
STUNAME	multiple	multiple
CUST	multiple	multiple
LOCTN	multiple	multiple
STATUS	multiple	multiple
ABSENCE	multiple	multiple
GRADE	multiple	multiple

The data elements that are listed as multiple are the data elements that repeat. The values in these elements will not be the same. The aggregate will always be unique for a particular class.

CRSCODE, CRSNAME, and LENGTH are the data elements that can have duplicate values. Student status and grade could have duplicate values, but they should not be separated because they are not meaningful values by themselves. These values would not be used to identify a particular student. This becomes clear when you remember to keep data elements with their controlling keys.

When you have compared the multiple occurrences and identified the data elements with duplicate values, shift those elements to a higher level. If you need to, choose a key for aggregates that do not then have keys.

When you isolate duplicate values, you arrive at the structure shown in Figure 5-2.

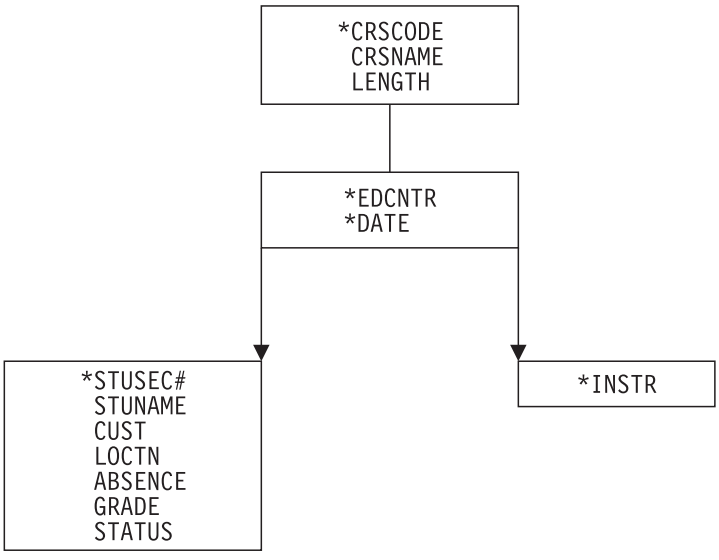


Figure 5-2. Current Roster Step 2

3. Grouping Data Elements With Their Keys

The third step is often a check on the first two steps. They may already have done what this step requires.

This step is to make sure that each data element is in the group that contains its controlling key, and that the data element is identified by the full key. If the data element is identified by only part of the key, or by a data element that is not a key, separate the data element and place with it the full key that does identify it.

In this example, CUST and LOCTN are not identified by the STUSEQ#. They are related to the student, but they aren't identified by it. They identify the company and company address of the student.

CUST and LOCTN are not identified by the course, the education center, or the date. They are separate from all of these things. Since a student is only associated with one CUST and LOCTN, but a CUST and LOCTN can have many students attending classes, the CUST and LOCTN aggregate should be separated from the STUDENT aggregate. Figure 5-3 shows what the structure looks like when you separate CUST and LOCTN.

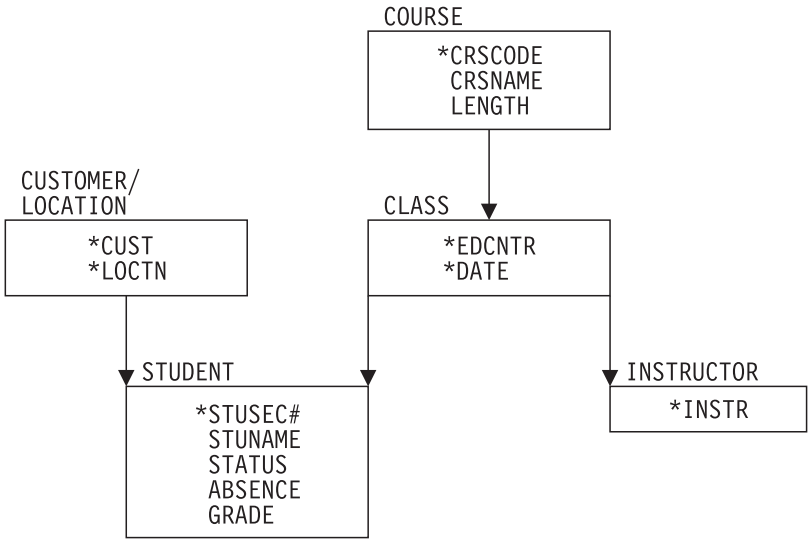


Figure 5-3. Current Roster Step 3

Where the customer and location information will be physically located is part of data base design. Data structuring should only be concerned with separating any inconsistent data elements from the rest of the data elements.

Identifying Relationships (Mapping)

Once you have arranged the data aggregates in a conceptual data structure, document the mappings between the aggregates. A mapping between two data aggregates is the quantitative relationship between them. The reason for recording mappings is that they reflect the relationships between aggregates in the data structure that you've developed. A data base structure can be constructed later that will satisfy all of the local views, based on the mappings. In determining mappings, it's easier to refer to the data aggregates by their keys, rather than by their collected data elements.

For our purposes, there are three possible relationships between any two data aggregates:

- One-to-one

  For each data aggregate A, there is only one occurrence of data aggregate B. For example, for each instructor, there would be only one salary data aggregate if that had been included in our application. Mapping notation shows this as:

  ```
  Instructor <--------> Salary
  ```

- One-to-many

  For each data aggregate A, there are two or more occurrences of data aggregate B. For example, for each class, there are multiple students. Mapping notation shows this as:

  ```
  Class <-------->> Student
  ```

- Many-to-many

  Data aggregate B has many A data aggregates associated with it and data aggregate A has many B data aggregates associated with it.

  In a hierarchical data structure, a parent can have one or more children, but each child can be associated with only one parent. The many-to-many association does not fit into a hierarchy, because each child can be associated with more than one parent. Many-to-many relationships occur between data aggregates in separate business processes. A many-to-many relationship indicates a conflict in the way those business processes need to process those data aggregates. This is covered in the section “Identifying Intersecting Attributes (Logical Relationship Candidates),” immediately following.

The mappings for the current roster are:

- `Course <-------->> Class`

  For each course, there may be several classes scheduled, but a class is associated with only one course.

- `Class <-------->> Student`

  A class has many students enrolled in it, but a student may be in only one class offering of this course.

- `Class <-------->> Instructor`

  A class may have more than one instructor, but an instructor only teaches one class at a time.

- `Customer/location <-------->> Student`

  A customer may have several students attending a particular class, but each student is only associated with one customer and location.

Identifying Intersecting Attributes (Logical Relationship Candidates)

When a business process needs to associate data aggregates from different structures, logical relationships can make that association possible.

Defining logical relationships lets you create a structure that doesn't exist as a physical data base—but can be processed as though it does. When you relate data aggregates in separate structures the data structure that's created from these

logical relationships is called a logical structure. You do this by storing the data in the path where it's accessed more frequently, and storing a pointer to the data in the path through which it will be accessed less frequently.

If we draw the structure we have developed so far in a slightly different way, we can identify two problem areas in it. In the re-structured view shown in Figure 5-4, the STUDENT data aggregate is involved in relationships with two other aggregates: CLASS and CUST. STUDENT can be identified by its association with either CLASS (`Class <--->> Student`) or CUST (`Customer <--->> Student`).

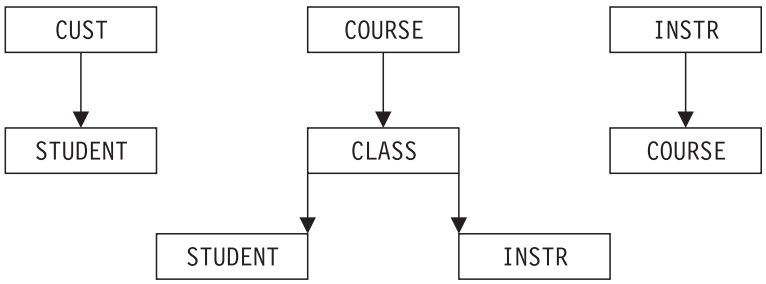


Figure 5-4. Education Data Structures

Suppose the CUST aggregate is part of an existing data base; if so, you could define a logical relationship between the CUST and STUDENT aggregates. You'd then have the structures shown in Figure 5-5. The CUST/STUDENT structure would be a logical structure.

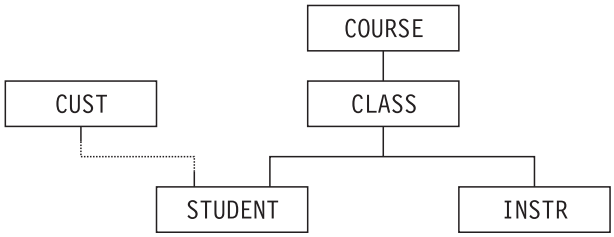


Figure 5-5. Unidirectional Logical Relationship

This kind of logical relationship is called unidirectional, because the relationship is “one-way.”

The other conflict you can see in Figure 5-4 is the one between COURSE and INSTR. For one course there are several classes, and for one class there may be several instructors,

```
COURSE<-------->>CLASS<-------->>INSTR
```

but each instructor is qualified to teach several courses.

```
INSTR<-------->>COURSE
```

You can resolve this conflict by using a bidirectional logical relationship. You could store the INSTR aggregate in a separate structure, with a pointer to it stored in the INSTR aggregate in the course structure; and you could store the COURSE aggregate in the course structure, with a pointer to it in the COURSE aggregate in the INSTR structure. This would give you the two structures shown in Figure 5-6.

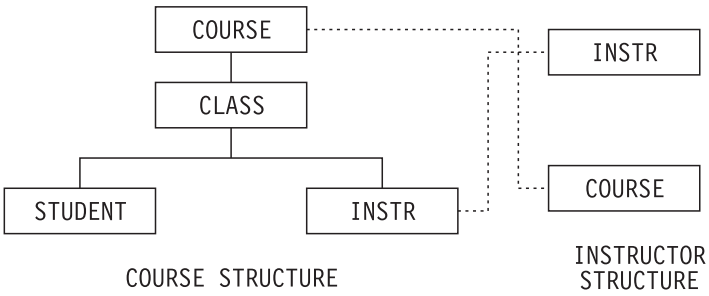


Figure 5-6. Bidirectional Logical Relationships

Identifying Alternate Processing Sequences (Secondary Indexing Candidates)

Secondary indexing makes it possible for a business process to identify a data aggregate through the value of a data element other than the aggregate's primary key. For example, suppose you are employed by the education company and need to find out, from a terminal, whether a particular student is enrolled in a class. If you aren't sure the student is enrolled in the class, you probably don't know the student's sequence number; but the key for the STUDENT aggregate is STUSEQ#. If you issue a request for an occurrence of the STUDENT aggregate, and identify the occurrence you want by the student's name (STUNAME), instead of the student's sequence number (STUSEQ#), DL/I might have to search through all STUDENT aggregate occurrences to find that particular one. Assuming that the STUDENT occurrences are stored in order of student sequence numbers, there's no way for DL/I to know where the STUDENT occurrence you want is just by having the STUNAME.

Using a secondary index in this example would be like making STUNAME the primary key of the STUDENT aggregate for this business process. Other business processes would still access this aggregate with STUSEQ# as the key.

To do this, you'd index the STUDENT aggregate on STUNAME in a secondary index. When you do, and you indicate to DL/I that you're using a secondary index for that aggregate, DL/I processes accesses of the aggregate as though the indexed data element is the key.

Considerations That Might Alter a Local View

The local view that you create by using the procedure described in this chapter represents the inherent structure of that view. It doesn't take into account any considerations that might arise in the actual implementation of the business process. Such considerations must be taken into account when the local view is combined with other local views into a system view and then implemented as a data base. The local view, as implemented, may differ from the one you have created, in order to accommodate these considerations. In re-structuring your local view for this purpose it may be necessary to create a structure that doesn't fit the conditions of the three step process. This must be done with great care. Some of the considerations you should be aware of are mentioned in the following sections.

Retrieval

Considerations involving the retrieval of data from the data base that might affect the structuring of your local view include the following:

- For performance reasons, it might be necessary to include a data element in more than one data aggregate.
- For security reasons, it might be necessary to move one or more data elements into separate data aggregates.
- If certain data elements are always retrieved in a sequential order in relation to each other, it may be expedient to change their position within the structure.

Addition

Adding new data elements or data aggregates at any stage may require re-structuring of your local view. You should try to anticipate this possibility and design your views so that changes can be accommodated with the least amount of re-structuring.

When you do add new data elements or data aggregates, you should repeat the procedure described in this chapter to be sure that you don't inadvertently create any of the problems it is designed to prevent.

Replacement

It may be necessary to move data elements with information that requires frequent replacement into positions in the structure that make them more accessible. This will give better performance.

Be sure that data elements that require frequent replacement are not included in more than one data aggregate. This eliminates making redundant replacements.

Deletion

DL/I requires that when a data aggregate at a higher level in a hierarchical structure is deleted, any aggregates associated with it at lower levels are also deleted. To prevent loss of data, it may be necessary to modify your local view by restructuring the position of certain data elements, or duplicating them in other aggregates.

We will now re-enforce what we have been discussing by giving more examples of the creation of local views.

Further Local View Examples

This section goes through three more examples of designing a local view, based on the education company's application. These examples will not be discussed in detail.

Schedule of Courses

Headquarters keeps a schedule of all courses given each quarter and distributes it monthly. The schedule is sorted by course code and printed in the format shown in Figure 5-7.

Note that the price of the course has been included.

```
                    COURSE SCHEDULE              1/06/84

     COURSE:  TRANSISTOR THEORY     COURSE CODE:  41837
     LENGTH:  10 DAYS               PRICE:  $280


     DATE                           LOCATION
_____________________________________________________________________
     APRIL 16                       BOSTON
     APRIL 23                       CHICAGO
     •
     •
     •
     NOVEMBER 19                    LOS ANGELES
```

Figure 5-7. Schedule of Classes

Table 5-3 lists the data elements and two occurrences of the data aggregate.

Table 5-3. Class Schedule Data Elements

Data Elements	Occurrence 1	Occurrence 2
CRSNAME	TRANS THEORY	MICRO PROG
CRSCODE	41837	41840
LENGTH	10 DAYS	5 DAYS
PRICE	$280	$150
DATE	multiple	multiple
EDCNTR	multiple	multiple

Using one occurrence of the data aggregate, we can isolate repeating data elements as shown in Figure 5-8.

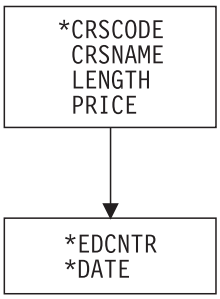


Figure 5-8. Class Schedule Conceptual Structure

Next, looking at two occurrences of the data aggregate we see that there are no duplicate values, so the second step (isolating duplicate values) is not necessary.

All data elements are already grouped with their controlling keys, so the third step is not needed either.

Now that we've developed a conceptual data structure we can determine the mappings for the data aggregate.

The mapping for this local view is:

```
Course <--------->> Class
```

Instructor Skills Report

Each of the education centers prints a report showing the courses that each of its instructors is qualified to teach. The report is in the format shown in Figure 5-9.

```
        INSTRUCTOR SKILLS REPORT              1/06/84

   INSTRUCTOR        COURSE CODE        COURSE NAME
_____________________________________________________________
   BENSON, R. J.     41837              TRANS THEORY
   MORRIS, S. R.     41837              TRANS THEORY
                     41850              CIRCUIT DESIGN
                     41852              LOGIC THEORY
   •
   •
   •
   REYNOLDS, P. W.   41840              MICRO PROG
                     41850              CIRCUIT DESIGN
```

Figure 5-9. Instructor Skills Report

Table 5-4 shows the data elements and two occurrences of the data aggregate.

Table 5-4. Instructor Skills Data Elements

Data Elements	Occurrence 1	Occurrence 2
INSTR	BENSON, R. J.	MORRIS, S. R.
CRSCODE	multiple	multiple
CRSNAME	multiple	multiple

The result of isolating repeating data elements in one occurrence of the data aggregate is shown in Figure 5-10.

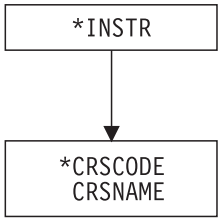


Figure 5-10. Instructor Skills Conceptual Structure

There are no duplicate values in two occurrences of the data aggregate, and all data elements are with their controlling keys. Steps two and three are not necessary.

The mapping for this local view is:

```
Instructor <--------->>  Course
```

Instructor Schedules

Headquarters produces a report showing the schedules for all the instructors. Figure 5-11 illustrates its format.

```
                    INSTRUCTOR SCHEDULES                    1/06/84

INSTRUCTOR          COURSE          CODE     ED CENTER      DATE
_____________________________________________________________________
BENSON, R. J.       TRANS THEORY    41837    CHICAGO        1/16/84
                    TRANS THEORY    41837    NEW YORK       3/05/84
MORRIS, S. R.       TRANS THEORY    41837    NEW YORK       1/09/84
                    CIRCUIT DES     41850    CHICAGO        1/23/84
                    LOGIC THEORY    41852    BOSTON         2/27/84
REYNOLDS, P. W.     CIRCUIT DES     41850    LOS ANGELES    3/12/84
```

Figure 5-11. Instructor Schedules

Table 5-5 lists the data elements and two occurrences of the data aggregate.

Table 5-5. Instructor Schedules Data Elements

Data Elements	Occurrence 1	Occurrence 2
INSTR	BENSON, R. J.	MORRIS, S. R.
CRSNAME	multiple	multiple
CRSCODE	multiple	multiple
EDCNTR	multiple	multiple
DATE	multiple	multiple

We can isolate repeating data elements in one occurrence of the data aggregate as shown in Figure 5-12.

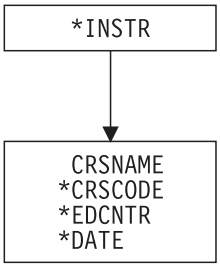


Figure 5-12. Instructor Schedules Conceptual Structure, Step 1

In this example, CRSNAME and CRSCODE can be duplicated for one instructor or for many instructors, for example, 41837 for Benson and 41850 for Morris and Reynolds, so the second step is necessary to isolate duplicate values as shown in Figure 5-13.

All data elements are grouped with their controlling keys in the current data structure, so the third step is not needed.

The mappings for this local view are:

```
Instructor <--------->> Course
Course <--------->> Class
```

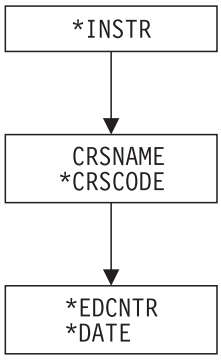


Figure 5-13. Instructor Schedules Conceptual Structure, Step 2

Combining the requirements of the four examples presented in this chapter and designing a structure for a system view of the application based on these requirements is covered in the next chapter.

Chapter 6. Combining Local Views Into a System View

This chapter shows you how to combine the local views of the business processes making up the application into an integrated structure called a system view. There are two main sections:

1. Generating a System View
2. Defining Implementation Requirements.

The first section suggests a procedure for generating an optimum system view from the local views. The second section shows how to define the requirements that must be considered in the implementation of the system view as an actual data base.

Having generated and documented a local view for each of the business processes in your application, you must combine these conceptual structures into what is called a system view. A system view is a conceptual data structure that integrates the individual structures of the local views into an optimum arrangement for physical implementation as a data base. The system view is used to define the structure of the physical data base or bases required by the application, and the ways in which they inter-relate. You will also generate a list of requirements that will influence the decisions that must be made in the physical design.

Generating a System View

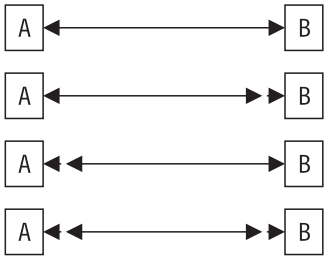
This part of the chapter describes a procedure for combining the local views into an optimum system view structure. However, there are several operations involved in the procedure that you should become familiar with first. These operations are covered in the following sections.

The procedure for generating the system view involves drawing the individual data elements of the local views as separate boxes, with the relationships between them shown as arrows. The direction and number of the arrowheads define the relationships. This is an extension of the method of showing the mapping between data aggregates described in Chapter 5.

A single arrowhead

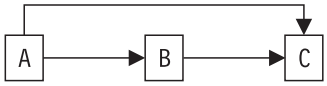
indicates a relationship in which a given value of A can have only one value of B associated with it. If you know a value of A, you also know the corresponding value of B. A identifies B. Doubling the arrowheads

indicates a relationship in which a given value of A can have zero, one, or many values of B associated with it. If you know a value of A, you do not know that one of the values of B corresponds to it. A does not identify B. By showing the relationship in each direction, we can represent one-to-one, one-to-many, many-to-one, and many-to-many relationships.

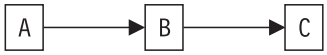


Eliminating Redundancies

At each stage of combining the local view drawings, you should look for and eliminate redundancies between data elements with single-arrow relationships. For instance, if you see a configuration like this:



since A identifies B and B identifies C, this implies that A identifies C, and the relationship between A and C is redundant. It can be removed, like this:



However, you must be careful that any relationships you remove are truly redundant. There are situations where seeming redundancy is actually a different problem. For instance, C might actually be two different data elements that had erroneously been assigned the same name.

Identifying Keys

As you combine the local views, you must identify, and pay close attention to, the keys in the system view. They and the relationships between them are very important in generating a system view that is an optimum view and will continue to be such as the data base changes and grows in the future.

Primary Keys

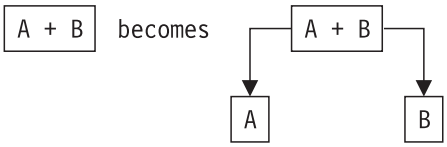
A primary key is a data element that uniquely identifies one or more other data elements. Any data element in your system view with one or more single arrows leaving it is a candidate for primary key. If there is more than one candidate, you will have to choose one as the primary key. For convenience in later translating your drawing into a hierarchical structure, draw keys of parent data aggregates above the keys of dependent data aggregates. This will ensure that single arrows between keys point up as often as possible. It will also ensure that the uppermost key in the final drawing is the root key. A root key is a primary key with no single arrows leaving it, going to another primary key. Make the primary keys and the single arrow links between keys as obvious as possible—possibly by using color.

Secondary Keys

A secondary key does not uniquely identify another data element. It identifies occurrences of data elements having a certain property. Data elements that are potential secondary keys are easy to identify in your system view. They will be boxes with double arrows leaving them.

Concatenated Keys

Concatenated keys are keys that are made up of two or more separate data elements. When you come across them in drawing your system view, show all of the data elements making up the key within one box. Then put each of the component parts in separate boxes with single-arrow links from the box containing the whole key, like this:



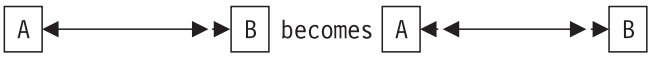
This ensures that each data element is identified by only one key data element. Be sure that every data element associated with a key is identified by the whole key.

Removing Undesirable Associations

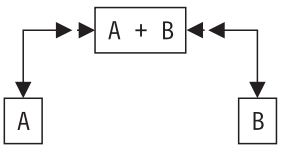
As you build your system view, check at each stage for undesirable associations between data elements and keys. This is equivalent to the third step of the three step process for generating local views (see Chapter 5). It consists of making sure that every data element is grouped with its key and that it is identified by the full key. If a data element is identified by only part of the key, or by a data element that is not a key, the data element must be moved to a separate group and the key that does identify it added there.

Mapping Between Keys

In Chapter 5 we introduced the concept of mapping to show the quantitative relationships between *data aggregates*. Now, in the process of combining local views into a system view, we need to consider the mappings between individual *data elements*. The concept is the same and the mapping notation is similar. However, there is one special case that you should take care to recognize and avoid: there should not be any many-to-many relationships between data elements that are primary keys. You should try to design your system view so that such relationships won't appear in the future as the data base changes and grows. Many-to-many mappings will be avoided by using this procedure: when a mapping appears between keys in one direction, add the equivalent relationship in the opposite direction.



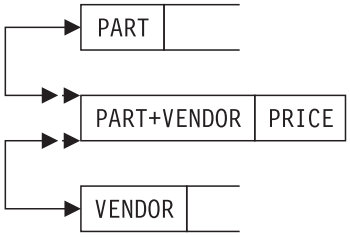
If this creates a many-to-many mapping as it does here, and there is any possibility that the added relationship might be used, introduce a concatenated key into the system view to replace it, like this:



It is possible that data elements making up concatenated keys will not remain as keys themselves in the final system view. In this case, they and the links to them can be eliminated at that time.

Intersection Data

Occasionally you will find situations where data is related to the association between other data elements. For example, a company might buy parts from several other companies. Each vendor might charge a different price for the parts it supplies. If you have a data element called PRICE, that data element isn't associated with PART only or VENDOR only. You need both PART and VENDOR to determine PRICE. PRICE is data that only has meaning in relation to the association between PART and VENDOR. This type of information is called *intersection data*. You can handle intersection data the same way you handle many-to-many mapping between keys: create a new data aggregate containing the intersection data, with the data elements it is associated with as its concatenated key. Our example would look like this:



Intersecting Attributes

Intersecting attributes may appear as you develop the system view. They are non-key data elements with arrows pointing to them from more than one key. They are candidates for logical relationships as described in the last chapter.

Isolated Attributes

Isolated attributes are data elements not identified by any primary key. They will appear as boxes with double-arrow links, but with no single arrows entering or leaving. These may appear as a result of misinterpreting user's requirements, so they should be evaluated carefully. If they are legitimate, they can be made a repeating data element in a variable length aggregate, or can be made a solitary key—a single data element aggregate.

Local Views as Input

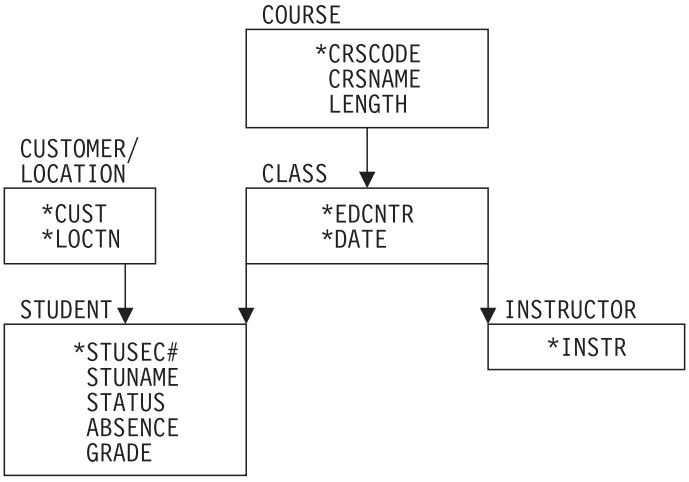
The local views that you created for the business processes in your application are the input to the procedure for generating a system view. In the education company example of Chapter 5, there are four local views to be combined. Each of these is summarized for reference.

Local View 1. Current Roster

Current Roster Data Elements:

CRSNAME	Course name
CRSCODE	Course code
LENGTH	Length of the course
EDCNTR	Education center giving the class
DATE	Date when the class starts
CUST	Customer that employs the student
LOCTN	Customer's location
INSTR	Instructors for the class
STUSEQ#	Student's sequence number
STUNAME	Student's name
STATUS	Student's enrollment status
ABSENCE	Student's absences
GRADE	Student's final course grade

Current Roster Conceptual Data Structure:



Current Roster Mappings:

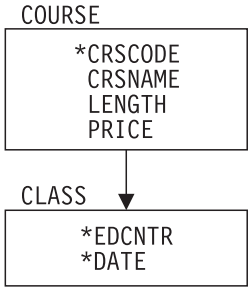
```
Course <-------->> Class
Class <-------->> Student
Class <-------->> Instructor
Customer/location <-------->> Student
```

Local View 2. Schedule of Classes

Schedule of Classes Data Elements:

CRSCODE	Course code
CRSNAME	Course name
LENGTH	Length of the course
PRICE	Charge for the course
EDCNTR	Education center giving the class
DATE	Date when the class starts

Schedule of Classes Conceptual Data Structure:



Schedule of Classes Mappings:

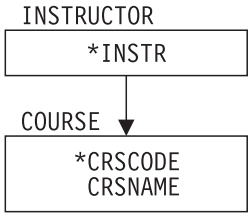
```
Course <-------->> Class
```

Local View 3. Instructor Skills Report

Instructor Skills Report Data Elements:

INSTR	Instructor
CRSCODE	Course code
CRSNAME	Course name

Instructor Skills Report Conceptual Data Structure:



Instructor Skills Report Mappings:

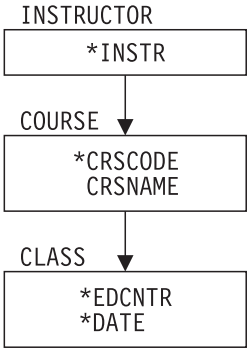
```
Instructor <-------->> Course
```

Local View 4. Instructor Schedules

Instructor Schedules Data Elements:

INSTR	Instructor
CRSNAME	Course name
CRSCODE	Course code
EDCNTR	Education center giving the class
DATE	Date when the class starts

Instructor Schedules Conceptual Data Structure:



Instructor Schedules Mappings:

```
Instructor <-------->> Course
Course <-------->> Class
```

Combining Local Views

Procedure

Now you are ready to take the local views and combine them, using the following procedure:

1. Draw the first local view, using boxes and connecting arrows, as described above. Draw concatenated keys as directed in the section on concatenated keys, and make sure that all single-arrow links from them are to data elements that are identified by the whole key. Draw only the relationships that are required for this local view.
2. Distinguish the primary keys in some way.
3. Look at the relationships between primary keys and add the inverse relationships where they are not already present. If this creates many-to-many relationships between keys, decide whether the inverse relationship would be used under any circumstances. If it would, create a new concatenated key made up of the keys that are of concern. This concatenated key should be drawn in the same way as any other concatenated key.
4. Look for redundant relationships and eliminate those that are truly redundant.
5. Merge another local view, handling it the same as the first, then repeat the previous steps until all the local views have been merged into your drawing.
6. Locate the root keys (primary keys with no single arrows to other keys). Redraw the system view with the root key on top, and dependent keys in hierarchical order at lower levels. This should result in most single-arrow links between keys pointing up. Mark the links between primary keys so that they are obvious.
7. Look for isolated attributes and handle as described above in the section “Isolated Attributes.”
8. Identify intersecting attributes (logical relationship candidates).
9. Redraw the diagram with each primary key and the data elements it identifies as one large box.

10. Identify the secondary keys. These are data elements with one or more double-arrow links leaving them. Show these secondary key links on your final drawing.
11. Add performance considerations by doing the following:
    a. Identify all paths that involve high usage, need fast response time, or are used in online programs.
    b. Estimate the number of times per month each path will be used. Total the number of times each individual relationship will be used (both directions if indicated).
    c. Estimate the size of each data aggregate as it appears in the final drawing.
    d. Estimate the number of occurrences for each relationship indicated by double arrowheads (one-to-many).

    Evaluating this information may make restructuring of your system view necessary.
12. When you are satisfied with the system view, check to see that each local view is still handled correctly.

Now, we will illustrate the use of the procedure in an example.

Example

The process is illustrated by combining the four local views of the education company example. In the figures on the fold-out sheet at the back of the book, asterisks and green color indicate primary keys. Relationships between primary keys are indicated by solid green lines. Other relationships are shown as dotted blue lines. Changes resulting from the combining of a local view with the preceding diagram are shown in red.

Figure 6-1 on the fold-out sheet shows the current roster local view drawn as described in step 1 above.

Figure 6-2 shows the result of merging local views 1 and 2 (schedule of classes). The only change necessary is the addition of the PRICE data element to the course aggregate.

Figure 6-3 shows the result of merging local view 3 (instructor skills) with the previous drawing. A path from the instructor aggregate to the course aggregate has been added.

There is no change when local view 4 (instructor schedules) is added. This view makes use of existing paths.

CRSCODE is the root key since there are no single arrows from it to other primary keys. It is not necessary to re-draw the diagram because the root is already on top and most of the single arrows between keys point up. There are no isolated attributes. There are no non-key intersecting attributes. Figure 6-4 shows the final diagram re-drawn with each primary key and the data elements it identifies as one large box. The double arrow link from INSTR to CRSCODE indicates that INSTR is a secondary index candidate as described in Chapter 5. The two logical relationship candidates are those discussed in Chapter 5. They are identified by the separate arrows leading to different keys. Figure 6-5 shows how Figure 6-4 can be re-drawn as a DL/I hierarchical structure in a form that can be used later in

the task of physical implementation, described in the next chapter. But we have one more thing to do before we go to that.

Defining Implementation Requirements

Now that you have combined the local views into a system view, you are almost ready to pass the results of your work to data base administration. Data base administration (DBA) is a function unique to data processing installations using a data base management system such as DL/I. All tasks involved in the implementation, administration, and control of DL/I and DL/I data bases are DBA functions.

There is one more thing to do before data base administration takes over. In addition to access to the requirements listing, the data dictionary, and your system view, DBA will need one more item from you. This is a list of implementation requirements to be considered in making the decisions for implementing the system view as a data base. The following sections describe the major areas to cover.

Performance Considerations

You should provide DBA with your best estimate of how many occurrences of each data element will be stored, and how often each element will be accessed. This will guide DBA in locating the high usage paths. It may be possible to design the data base so that the performance in accessing these paths is maximized. Also, an important performance consideration is the relative frequency of retrieving, updating, inserting, and deleting the data elements. Online query programs will need to be able to access certain data very rapidly. Special consideration should be given to situations where certain information has a great deal of update or change activity.

Data Access Requirements

DBA will want to know how each program will access root segments and how it will access segments within data base records. The choice of an access method for the data base is strongly influenced by the type of access (sequential or direct) used for each of these types of segment.

Structural Considerations

Be sure that DBA is aware of conditions in the system view that require the use of logical relationships or secondary indexing. Plans to use variable length segments or a segment edit/compression routine (see Chapter 7) must be made known. Any of these will force DBA to make use of one of the direct access data base organizations. Call attention to any other structural details in the system view that were included for a specific, non-obvious purpose.

Security Requirements

If there are any data elements that must be made secure from unauthorized access, DBA must be made aware of these. Either data elements (fields) or data aggregates (segments) can be made available to only those programs that are authorized to access them.

Processing authority can also be controlled. DBA should be made aware of which programs are to be allowed to process data by retrieving, inserting, updating, and

deleting; and which are to be allowed to process data by only one or a combination of these functions.

DBA can also protect data from being accessed by non-DL/I programs through encryption of the data.

Recovery Requirements

There may be special requirements regarding the planning and scheduling of data base back-up, reorganization, and recovery that should be discussed with DBA at this point. Important data that has a great deal of update activity is an example of the type that requires special consideration in this area.

Chapter 7. Converting the System View to a Physical View

This chapter introduces you to the considerations and decisions involved in the physical implementation of the system view of the application. It is a link between application design and data base administration. There are two main sections:

1. DL/I Data Base Organization and Access Methods
2. Factors in the Choice of an Access Method.

The first section describes the sequential and direct data base organizations and the DL/I access methods associated with each of them. The second section describes the factors that influence the selection of one of the access methods for the physical implementation of the data base.

At this point you should have completed the design of the structure or structures that make up the system view of the application. This is a conceptual structure and represents the inherent properties of the data with little consideration for the way in which it will be physically implemented as a data base. From this point on, the physical design of the data base will be handled as a data base administration task. Your system view and the implementation requirements that were discussed in the last chapter will be used as input to that task.

This chapter provides the link between application design and data base administration. The information in this chapter briefly describes the decisions that must be made in performing the physical design, and the requirements and considerations that influence those decisions. This information will help you to understand how the design of the system view may influence the decisions, and how the decisions may make it necessary for you to modify your system view. For instance, if your system view involves two or more inter-related structures that require the use of logical relationships for physical implementation, DBA has no choice but to use a direct access method. On the other hand, if DBA chooses to use a sequential access method for implementing your system view, it may be necessary, for performance reasons, for you to change the position of certain data aggregates in your system view.

Because this is the point at which DBA becomes directly involved in the physical implementation of the structure you have designed, we will use specific DL/I terms from now on, rather than the more generic terms we have used in connection with your tasks. In DL/I:

- A data aggregate is called a *segment*.
- A data element is called a *field*.
- A physical implementation of a logical or conceptual structure is called a *data base*.

DL/I Data Base Organization and Access Methods

The major decisions that must be made in the physical design of a data base are the selection of the DL/I data base organization and corresponding access method to be used in implementing the design. There are two DL/I data base organizations and four basic types of DL/I access methods, plus two access methods that are used in special cases. Data base administration makes these choices based on the way in which the majority of programs that use the data base will access the data.

In making these decisions, DBA will need information from you that will include answers to questions like the following:

- How will each program access root segments?
  - Direct access of individual segments
  - Sequential access of segments
  - Both.
- If both, in what proportion?
- How will each program access the segments within each data base record?
  - Direct access of individual segments
  - Sequential access of segments
  - Both.
- If both, in what proportion?

Note: The segments within the data base record are the dependents of the root segment.

- Will the program be capable of updating the data base, rather than simply retrieving data?
- If yes, in which of the following ways:
  - By adding new data base records?
  - By deleting data base records?
  - By adding new segments to existing data base records?
  - By deleting segments from existing data base records?
  - By modifying existing segments?
- What is the estimated frequency and amount of change in each of the applicable cases in the list above?

It's important to note the distinction between accessing a data base record and accessing segments within the record. A program might access data base records sequentially, but once within a record, the program might access the segments directly; or vice versa. These are two different requirements, and can influence the choice of access method. The same distinction applies to updating a data base record as opposed to updating a segment within the data base record.

Sequential Organization (HS)

The sequential organization is exactly what the name implies: the segments in the data base are stored in hierarchical sequence, one after another. There is no need for pointers in a sequential data base.

DL/I provides two sequential access methods, each of which also has a variation having certain restrictions. The major difference between the two methods is that one makes use of an index while the other doesn't:

- The Hierarchical Sequential Access Method, HSAM, can only process segments (both root and dependent) sequentially.
- The Hierarchical Indexed Sequential Access Method, HISAM, processes dependent segments sequentially but has an index to access root segments (records) directly.

Some of the advantages of the sequential organization and access methods are:

- Fast sequential processing
- Direct processing of data base records with HISAM
- Low DL/I storage overhead because the sequential methods relate segments by adjacency rather than with pointers
- HSAM provides support for data bases on tape.

Some disadvantages of the sequential organization and access methods are:

- Slower access to the rightmost segments in the hierarchy, because HSAM and HISAM have to read through all of the other segments to reach them.
- HISAM requires frequent reorganization of the data base to reclaim space formerly occupied by deleted segments, and to keep the logical records of a data base record physically adjacent.
- HSAM data bases can't be updated; the old data base has to be read and the changes interleaved as a new data base is created.

HSAM Access Method

HSAM is a hierarchical access method that can only handle sequential processing. You can retrieve data from HSAM data bases, but you can't update the data. Updating can only be done by interleaving modifications as a new data base is created from the old.

HSAM is a useful access method when you are:

- Storing historical data
- Using the data base to collect data or statistics that will not need to be updated
- Always processing the data sequentially.

HSAM stores data base records in the sequence in which you submit them. They and the segments they contain can only be processed sequentially—in the order in which they were loaded. HSAM stores dependent segments in hierarchical sequence.

HSAM data bases are very basic data bases. Since the data is stored in hierarchical sequence, there is no need for pointers or indexes.

Simple HSAM Access Method

Simple HSAM is similar to HSAM except that it contains only root segments. A Simple HSAM data base cannot have a hierarchical structure. This access method is used chiefly as a conversion aid, permitting existing conventional tape and direct access storage device files to be used as data bases.

HISAM Access Method

HISAM is an access method that stores segments in hierarchical sequence with an index to locate root segments. Segments are stored in a logical record until the end of the logical record is reached. If there are still segments remaining in the data base record, they are stored in an overflow data set.

HISAM is well-suited for:

- Sequential access of records
- Sequential access of dependent segments.

Even though your processing has some of the characteristics above, HISAM is not necessarily a good choice if:

- You need to access dependents directly.
- There will be a high volume of insertions and deletions.
- A lot of the data base records exceed average size and have to use the overflow data set. This is because the segments that go into the overflow data set require additional I/O.

For data base records, HISAM data bases:

- Store records in key sequence
- Can locate a particular record with a key value by using the index.

For dependent segments, HISAM data bases:

- Start each HISAM data base record in a new logical record in the primary data set
- Store any remaining segments from one or more logical records in the overflow data set if the data base records won't fit in the primary data set.

Unlike the direct access methods, which are described below, HISAM doesn't make space that is vacated during deletions available for automatic reuse. The data base must be reorganized to make use of this space. The insertion of new segment occurrences may cause existing segments to be moved in order to keep the hierarchical sequence within the record. The space that is vacated when segments are moved to the overflow data set during this process is reusable by subsequent inserts, but space in the primary data set can only be reclaimed during reorganization.

Simple HISAM Access Method

Simple HISAM is similar to HISAM except that it contains only root segments. A Simple HISAM data base cannot have a hierarchical structure. This access method is used chiefly as a conversion aid, permitting existing key sequenced data sets (KSDS) to be accessed as data bases. Simple HISAM does not use an overflow data set.

Direct Organization (HD)

Direct organization provides for the locating of any data base record in the data base directly, without searching sequentially through the records from the beginning. Direct access in DL/I is accomplished by using either a randomizing routine or an index. DL/I can find any data base record that you want, independent of the sequence of data base records in the data base. Direct access can give good results with either direct or sequential processing.

Direct access uses pointers to maintain the hierarchical relationships between segments of a data base record. By following pointers, DL/I can access a path of segments without first passing through all segments in the preceding paths. In direct access, pointers and addresses are maintained internally.

Some of the requirements that direct accessing satisfies are:

- Fast direct processing of roots using an index or a randomizing routine
- Good sequential processing of data base records using the index
- Fast access to a path of segments via pointers.

In addition, when you delete data from a direct access data base, the new space is made available almost immediately. This provides efficient space utilization, and means that you don't have to reorganize the data base often to take advantage of unused space.

The larger DL/I overhead caused by the pointers is a disadvantage. However, if direct access answers your data access requirements, it is more efficient than using a sequential access method.

Randomized Access

DL/I HD randomized access uses a randomizing routine to locate its root segments, then chains dependent segments together in their hierarchical paths. HD randomized access is efficient for a data base that will primarily use direct access.

The requirements that randomized access satisfies are:

- Direct access of root segments by root keys through a randomizing routine
- Direct access of paths of dependents
- Even distribution of data base records in storage
- New data base records and new segments are added by putting the new data into the nearest available space
- The space created by the deletion of data base records and segments is automatically reusable for other records or segments.

Randomized Access Characteristics: For root segments in a randomized access data base, DL/I:

- Stores them at the location determined by the randomizing routine, rather than in key sequence.
- Uses a randomizing routine to locate the root segments.
- Returns root segments in physical sequence, not key sequence, when root segments are retrieved sequentially.

With randomized access, dependent segments:

- Are stored anywhere, as close together as possible
- Are chained together with pointers within a data base record.

An Overview of How Randomized Access Works: When a data base record is stored in an HD randomized data base, one or more direct address root anchor points (RAPs) are kept at the beginning of each physical block. The RAP points to a chain of root segments. This chain is made up of all root segments that the randomizing routine assigns to this block and RAP. They are called *synonyms*. HD randomized also keeps a pointer at the beginning of each physical block that points to any free space in that block. When you insert a segment, DL/I uses this pointer to locate free space in the physical block. To locate a root segment in a randomized access data base, you provide DL/I with the root key. The randomizing routine uses it to generate the relative physical block number and the RAP that points to the chain of root segments. The RAP value specifies the location of the first root within a physical block.

Although HD randomized can place roots and dependents anywhere in the space allocated to the data base, it's good policy to choose HD options that keep roots and dependents close together.

Randomized access performance can be very good. How good depends largely on the randomizing routine you use. Performance also depends on other implementation factors such as:

- The block size you use
- The number of RAPs per block
- The pattern for chaining together different segments through pointers
- The distribution of key values.

For sequential access of data base records by root key you must use a randomizing routine that stores roots in physical key sequence, or a secondary index.

Indexed Access

DL/I HD indexed access is the method that is most efficient for an approximately equal amount of direct and sequential processing. It is least efficient in the sequential access of dependents. It uses a separate index data base to locate the root segments of the data base records. Some specific requirements that it satisfies are:

- Direct and sequential access of records by their root keys.
- Direct access of paths of dependents.
- New data base records and new segments are added by putting the new data into the best available space, as determined by DL/I.
- The space created by the deletion of data base records and segments is automatically reusable for other records or segments.

HD randomized access should be the first choice, but if not, indexed access can answer most processing requirements that involve direct processing or an approximately even mixture of direct and sequential processing.

Indexed Access Characteristics: With indexed access, root segments:

- Are initially loaded in key sequence
- Are stored wherever space is available after initial loading
- Are identified through the index, by the root key value that you supply.

With indexed access, dependent segments:

- Are stored anywhere, as close together as possible
- Are chained together with pointers within a data base record.

An Overview of How Indexed Access Works: Indexed access uses two data bases: one, the primary data base, holds the data; the other is the index data base. The index data base contains entries for all of the root segments in order of their key fields. For each key entry, the index data base contains the address of that root segment in the primary data base.

When you access a root, you supply the key of the root. HD indexed locates the key in the index to find the address of the root, then goes to the primary data base to locate the segment.

HD indexed chains dependent segments together so that when you access a dependent segment, a pointer in each higher level segment locates the next segment below it in the hierarchy.

When you process data base records directly, HD indexed locates the root through the index, then locates the dependent segments of the root by using the pointers.

If you are going to process data base records sequentially, you can specify special pointers in the DBD so that DL/I doesn't have to go to the index each time to locate the next root segment. These pointers chain the roots together. If you don't chain roots together, HD indexed always goes to the index to locate a root segment. When you process data base records sequentially, HD indexed accesses roots in key sequence in the index. This only applies to sequential processing. When you access a root segment directly, HD indexed uses the index, instead of pointers, to find the root segment you've requested.

Which access method should be used for a particular data base? We will discuss some factors that influence that choice.

Factors in the Choice of an Access Method

As stated in the introduction to this chapter, the system view of the application reflects the inherent properties of the data, and may need to be modified to reflect the needs of the physical implementation of the data base. It was also stated that the requirements of the application could influence physical design.

This section discusses some of those factors that influence the choice of an access method. In Chapter 5, you were shown how to identify situations that require logical relationships or secondary indexing for implementation. In Chapter 6, you were shown how to define other implementation requirements. All of these, plus other special direct access considerations will be discussed further, in relation to their influence on the choice of access method. You will then be shown a method DBA can use to select an access method that meets all of the requirements.

Special Direct Access Considerations

DL/I provides several functions that are available for use only with the hierarchical direct organization. If the design of your data base makes it desirable or necessary to use one or more of these functions, DBA's choice of organization must be hierarchical direct. These functions are described briefly in the following sections. For complete details on each of them, see *DL/I DOS/VS Data Base Administration*.

Logical Relationships

You were shown, in Chapter 5, how to recognize conditions that might arise in the creation of local views that would be candidates for logical relationships. In DL/I, logical relationships are a method of creating relationships between segments in different hierarchies. Since these relationships don't physically exist, they are called logical relationships.

Logical relationships:

1. Permit two application programs to process the same segment through different hierarchical paths
2. Permit a segment's parent in one application's view to act as that same segment's child in another application's view.

Accessing a Segment Through Different Paths: As an example of the first use, we will use a program in a health care application for a medical clinic. This application program processes data in a purchasing data base, but also requires access to a segment in a patient data base.

- Another program, program A, processes information in the patient data base about the patients at the clinic: the patients' illnesses and their treatments.
- Our program, program B, is an inventory program that processes information in the purchasing data base about the medications that the clinic uses: the item, the vendor, information about each shipment, and information about when and under what circumstances each medication was given.

Figure 7-1 shows the hierarchies that Program A and Program B require for their processing. They both need access to information contained in the TREATMNT segment in the patient data base:

- The date that a particular medication was given
- The name of the medication
- The quantity given
- The doctor who prescribed the medication.

To Program B this isn't information about a patient's treatment; it's information about the disbursement of a medication. To the purchasing data base, this is the disbursement segment (DISBURSE). In Figure 7-1 the TREATMNT segment and the DISBURSE segment contain the same information.

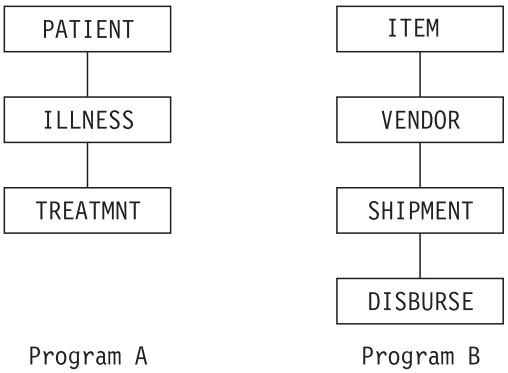


Figure 7-1. Medical and Purchasing Hierarchies

Instead of storing this information in both hierarchies, DBA can use a logical relationship. A logical relationship solves the problem by storing a pointer from where the segment is needed in one hierarchy to where the segment exists in the other hierarchy. In this case, you can have a pointer in the DISBURSE segment to the TREATMNT segment in the medical hierarchy. There are a number of considerations involved in deciding which segment contains the pointer. Performance is one. When DL/I receives a request for information in a DISBURSE segment in the purchasing hierarchy, DL/I retrieves the TREATMNT segment in the medical hierarchy pointed to by the DISBURSE segment. Figure 7-2 shows the physical hierarchy that Program A would process and the logical hierarchy that Program B would process. DISBURSE is a pointer segment to the TREATMNT segment in Program A's hierarchy.

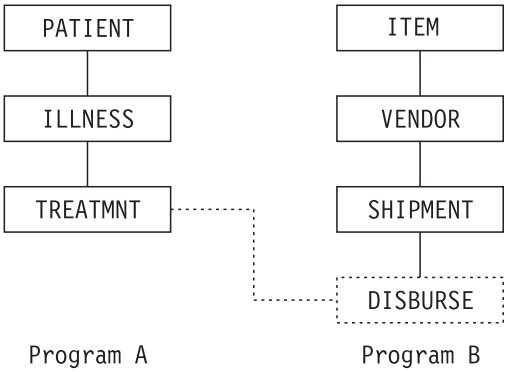


Figure 7-2. Logical Relationships Example

To define a logical relationship between segments in different hierarchies, DBA creates a logical data base definition. A logical DBD defines a hierarchy that does not physically exist, but can be processed as though it does. Program B would use the logical structure shown in Figure 7-2 as though it were a physical structure.

Inverting a Parent/Child Relationship: As an example of the second use of logical relationships (a segment's parent in one application program acts as that segment's child in another program), consider the following situation:

- The inventory program (Program B above) processes information about medications, using the medication (ITEM) as the root segment.
- A purchasing application program, Program C, processes information about which vendors have sold which medications. Program C processes this information using VENDOR as the root segment.

Figure 7-3 shows the hierarchies for each of these application programs.

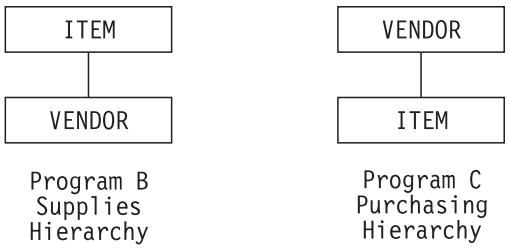


Figure 7-3. Supplies and Purchasing Hierarchies

DBA can use logical relationships to handle these requirements. In this example, the ITEM segment in the purchasing hierarchy would contain a pointer to the actual data stored in the ITEM segment in the supplies hierarchy. The VENDOR segment, on the other hand, would actually be stored in the purchasing hierarchy. The VENDOR segment in the supplies hierarchy would point to the VENDOR segment stored in the purchasing hierarchy.

Figure 7-4 shows the modified hierarchies.

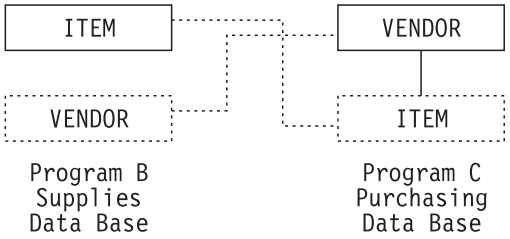


Figure 7-4. Program B and Program C Hierarchies

If you didn't use logical relationships in this situation, you would have to store the same data in both paths. This redundant data would have to be updated in both places each time it changed. In addition to the additional space required for storage, there would be the possibility of inconsistent data during the period of updating the two data bases.

Secondary Indexes

Secondary indexing was mentioned in Chapter 5 as a means of solving the problem of identifying a segment through the value of a data element other than the primary key. Actually, there are two situations in which secondary indexing is useful:

1. When an application program needs to retrieve a segment in a sequence other than the one that has been defined by the segment's key field
2. When an application program needs to retrieve a segment based on a condition that is found in a dependent of that segment.

Examples to illustrate these conditions use two application programs that process the patient hierarchy shown in Figure 7-5. There are three segment types in this hierarchy:

- PATIENT contains three fields: the patient's identification number, the patient's name, and the patient's address. The patient number field is the key field.

- ILLNESS contains two fields: the date of the illness and the name of the illness. The date of the illness is the key field.
- TREATMNT contains four fields: the date the medication was given, the name of the medication, the quantity of the medication that was given, and the name of the doctor who prescribed the medication. The date that the medication was given is the key field.

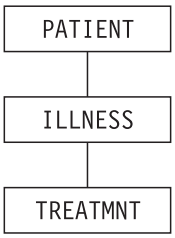


Figure 7-5. Patient Hierarchy

Using a Different Key: As an example of the first case, suppose you have an application program that processes requests to determine whether or not an individual has ever been treated in the clinic. Obviously, you won't be able to supply an identification number for that person, even though the patient's identification number is the key field of the PATIENT segment.

If you issue a request for a PATIENT segment and identify the segment you want by the patient's name instead of the patient's identification number, DL/I might have to search through all of the PATIENT segments to find the PATIENT segment you've requested. DL/I doesn't know where a particular PATIENT segment is just by having the patient's name.

There is a way to make it possible for this application program to retrieve PATIENT segments in the sequence of patients' names (rather than in the sequence of patients' identification numbers). DBA can create a separate data base called a secondary index. This data base is really an index to the patient name fields in the PATIENT segments. The PATIENT index entries are in order by the patient names. Using the index entries in the secondary index data base, DL/I can easily locate a PATIENT segment when you supply the patient's name. DL/I goes directly to the secondary index and locates the PATIENT index entry for the name you've supplied. The index entry contains a pointer to the PATIENT segment in the patient hierarchy. DL/I can determine whether or not a PATIENT segment for the name you've supplied exists, and return the segment to the application program if it does exist. If the requested segment doesn't exist, DL/I indicates this to the application program by returning a “not-found.”

There are three terms involved in secondary indexing that you should know.

- The ***pointer segment*** is the index entry segment in the secondary data base. DL/I uses the pointer segment to find the segment that you've requested in your original hierarchy. In the example above, the pointer segment is the index entry in the secondary index data base that points to the PATIENT segment in the patient hierarchy.
- The ***source segment*** is the segment in your original hierarchy that contains the field you're indexing. In the example above, the source segment is the PATIENT segment in the patient hierarchy, since you're indexing on the name field in that segment.

- The ***target segment*** is the segment in your original hierarchy that the secondary index points to; it's the segment that you want to retrieve.

In the example above, the target segment and the source segment happen to be the same segment—the PATIENT segment in the patient hierarchy. The source segment and the target segment can also be different segments. For instance, if the field in the ILLNESS segment that contains the name of the illness was used as the field to be indexed, then the ILLNESS segment would be the source segment. You could create a list of all patients who had had each illness by using the index. Figure 7-6 shows the relationship of the segments in this case.

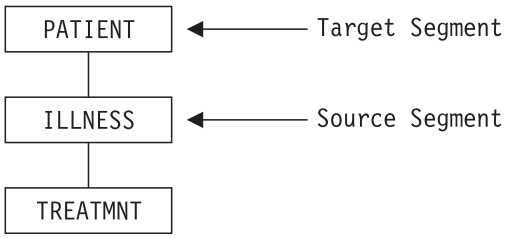


Figure 7-6. Indexing a Root Segment

Dependent segments can be used as target segments just as root segments can. If you do use a dependent segment as a target segment, you create what is called an *inverted structure*. You have, in effect, created a new hierarchy. The segment you index becomes the root segment, and its parent becomes a dependent. For example, suppose you index the ILLNESS segment on a field in the TREATMNT segment. In this case, the ILLNESS segment is the target segment, and the TREATMNT segment is the source segment. Figure 7-7 shows the original hierarchy and the apparent hierarchy created by the secondary index.

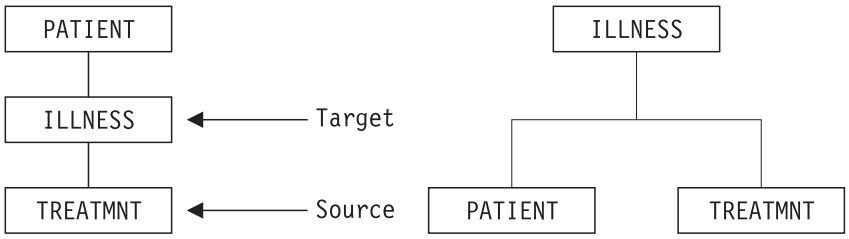


Figure 7-7. Indexing a Dependent Segment

Retrieving Segments Based on a Dependent's Qualification: As an example of the second use of secondary indexing (retrieving a segment based on a condition in a dependent of that segment), suppose that the medical clinic wants to print a monthly report on the patients who have visited the clinic during that month. If the application program that processes this request doesn't use a secondary index, the program has to retrieve a PATIENT segment, then retrieve the ILLNESS segments for that PATIENT segment. The program tests the date in the ILLNESS segments to determine whether or not the patient has visited the clinic during the current month, and prints the patient's name if the answer is yes. The program continues retrieving PATIENT segments and ILLNESS segments until it has retrieved all the PATIENT segments.

But with a secondary index, you can make that program's processing much simpler. To do this, DBA indexes the PATIENT segment on the date field in the ILLNESS segment. DL/I recognizes the name of the field that you're indexing the PATIENT

segment on, and the name of the segment that contains the index field. The application program can then issue a request to DL/I for a PATIENT segment and qualify the request with the date in the ILLNESS segment.

In this example, the PATIENT segment is the target segment; it's the segment that you want to retrieve. The ILLNESS segment is the source segment; it contains the information that you want to use to qualify your request for PATIENT segments. The index segment in the secondary data base is the pointer segment. It points to the PATIENT segment.

Variable Length Segments

Variable length segments are used by application programs to process variable length text or descriptive data. They can also be used, in some cases, to make more efficient use of secondary storage space.

Variable length segments are defined as such by DBA when the segment type is defined in the DBD generation procedure. This definition causes a two-byte size field to be included as the first field in the data portion of the segment. It is the responsibility of the application program to determine the size of each occurrence of the segment, and load that value into the size field.

Since this function of DL/I is only available with the hierarchical direct organization, your use of variable length segments forces DBA to choose one of the HD access methods for implementing the data base.

You would use a variable length segment when the data to be included in a particular field may vary with each occurrence of the segment. For instance, if a segment had been included in the health care application to list the symptoms the doctor had found for the patient's illness, it would be a likely candidate for a variable length segment. This is shown with a couple of examples of the contents of this field:

```
SEVERE HEADACHE WITH NAUSEA.

COMPOUND FRACTURE, LEFT ULNA.  LACERATIONS OF
LEFT HAND AND FOREARM.
```

Segment Edit/Compression

The segment edit/compression exit facility of DL/I makes it possible for the user to supply a program routine that will edit a variable length segment as it comes from or goes to the data base. The routine can be made to encode data for security purposes, to format data for application programs, and to compress the data to remove redundant characters. You will not be directly involved with segment edit/compression in your tasks, other than to suggest its use when you have specified variable length segments.

Implementation Requirements

It is the responsibility of DBA to create a data base or bases that will best serve the interests of the corporation. This sometimes involves compromises between the needs of individual applications. The requirements of your application may be completely separate from those of other applications, but it is likely that DBA must consider your requirements in relation to existing or possible future applications.

Before making final decisions regarding the implementation of your system view as a data base, DBA will consider the list of implementation requirements you defined

in Chapter 5. Some of the ways in which implementation requirements can influence these decisions are listed here:

- Performance considerations
  - DBA will locate the high usage paths and try to optimize the performance of programs using them. To do this, it may be necessary to rearrange the hierarchy.
  - Online query programs require fast response. This may require a rearrangement of the hierarchy, and may force the choice of access method.
  - Data bases with a great deal of inserting or deleting may force the use of one of the direct access methods to improve performance and to eliminate the need for frequent data base reorganization to recover unused space.
  - The type of processing that is most frequent may force the choice of access method for performance reasons.
- Data access requirements
  - If access of both root and dependent segments is to be sequential, DBA may choose the HSAM access method.
  - If access of roots is largely direct, but dependents are accessed sequentially, DBA may choose HISAM.
  - If access of both root and dependent segments is primarily direct, DBA may choose the randomized access direct method.
  - If there are approximately equal amounts of direct and sequential processing, DBA may choose the indexed access direct method.
  - If online access is required, HSAM or SHSAM may not be chosen.
- Structural considerations

  Use of logical relationships, secondary indexing, variable length segments, or the segment edit/compression exit will force DBA to choose one of the direct access methods.
- Security requirements
  - To prevent unauthorized access to data, DBA must use the DL/I segment or field level sensitivity facilities. It might be decided that it would be better to move a field or segment to a different location in the hierarchy to facilitate this.
  - DBA will assign processing authority to each individual application program depending on the type of processing you have specified for that program: retrieving, inserting, updating, deleting, or some combination of these.
  - Encryption can be used to prevent non-DL/I programs from making use of information stored in DL/I data bases. Encryption can be done through the use of a segment edit/compression exit routine. This facility is not available for the sequential access methods, so they could not be used if encryption was required.
- Recovery requirements

  Conditions, such as frequent or voluminous updating, may require special procedures for back-up and recovery.

Choosing an Access Method

DBA will choose an organization and access method for implementing the data base or bases that your application needs by putting to use:

- The information on requirements that you have supplied
- Knowledge of the requirements of other applications
- Knowledge of the requirements, restrictions, and functions of DL/I.

The decision tree shown in Figure 7-8 illustrates a method DBA can use in making this choice. The tree is an aid, but the final decision must be influenced by a study of all the factors listed above.

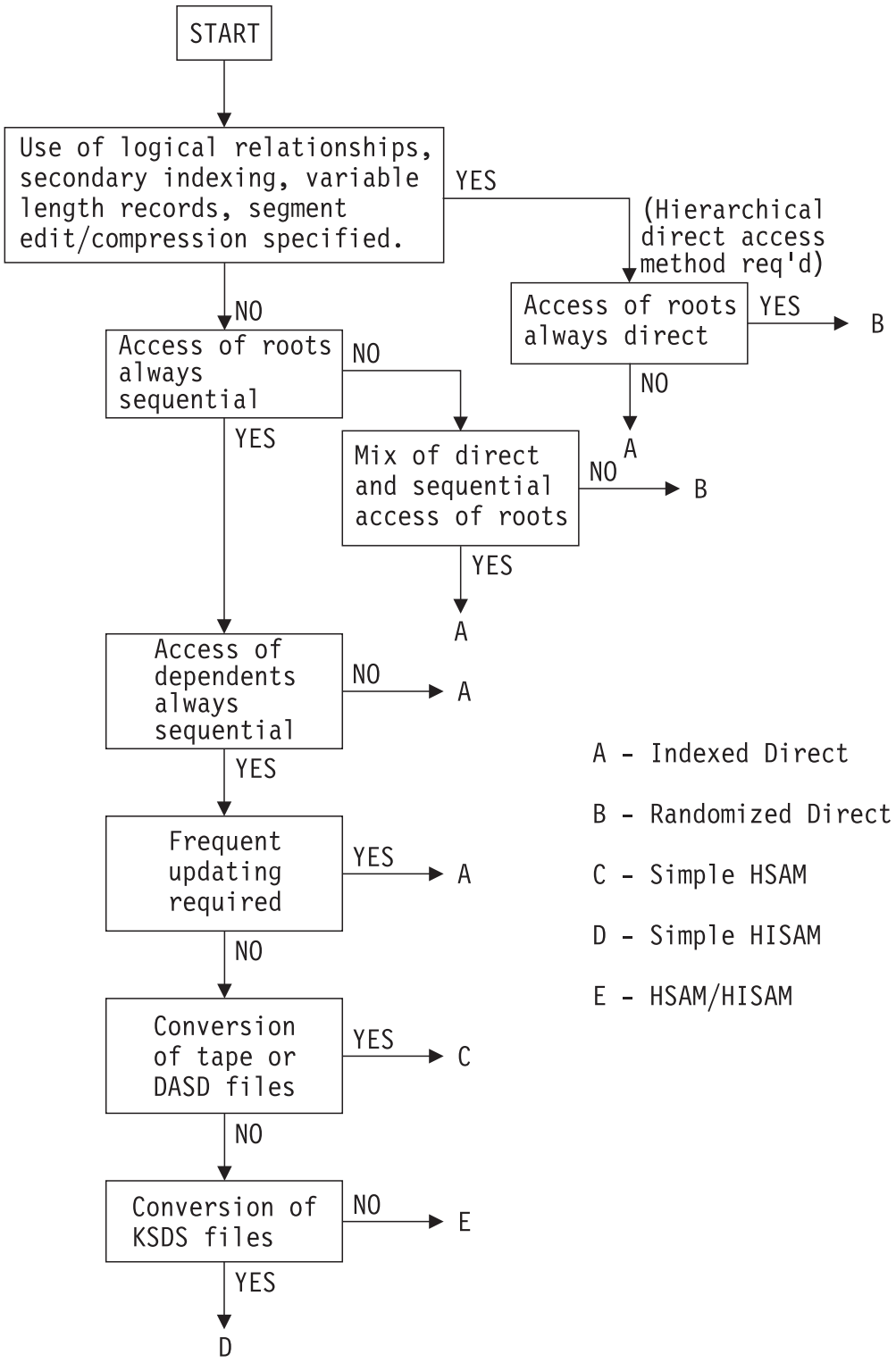


Figure 7-8. Access Method Decision Tree

Chapter 8. Implementing the Application

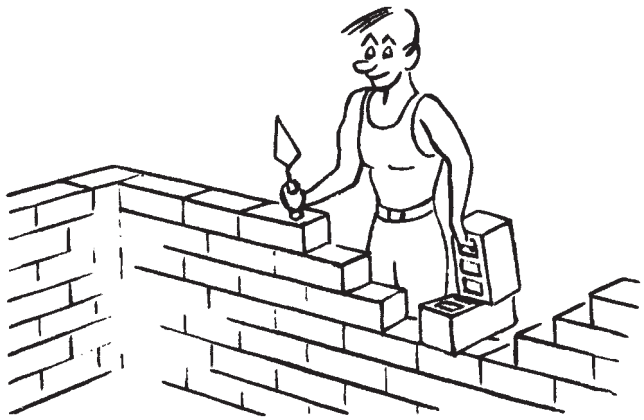
This chapter gives you suggestions on how to generate an implementation plan that will provide an orderly phasing in of an application. It also gives suggestions for carrying out the plan. There are two main sections:

1. Implementation Plan
2. Implementation.

The first section gives suggestions for generating an implementation plan for the application, addressing such items as which programs need to be available together, the phases in implementation that will provide the smoothest and most successful introduction of the application, and contingency plans. The second section gives suggestions for carrying out the plan, and some pitfalls to be aware of.

Implementation Plan

If this is a new installation, you should devise an implementation plan for the application while the design of the data base is being completed and DL/I is being installed. If the application is a large or complicated one, implementation should be done in phases. On the other hand, the phases you choose should be large enough so that each one accomplishes a useful portion of the whole application. The process is like building a wall one brick at a time.



The first step in developing the plan is to decide which requirements are closely enough related to make it necessary to implement them together. The relationship could be one of function or it could be through use of common data elements. The requirements listing that you created after collecting application requirements will be your primary source of this information. You included in one chapter of that listing all of the requirements of a particular user group or functional area. This means that all of the requirements in a chapter are candidates for grouping into a module to be implemented in a phase. Examining the listing for a particular chapter, you can pick those that should be implemented together, but you may find some that

can be left for a later phase, or that can be implemented with requirements from other functional areas because of common data element usage.

To identify those requirements from different functional areas that should be considered as candidates for implementation in the same phase, use the data dictionary. Look at the where-used information for each entry in the data dictionary. All requirements that appear in that field for a particular data element can be considered for implementation in the same phase.

At this point, the detailed design of the data base(s) should be complete, program design should be firm, and planning for any special data collection or conversion activities should be done.

The last step is to decide which modules will be implemented in each phase, and to make an outline of the order and timing of the implementation.

It is vitally important that the phase you choose to implement first be a success. The users will base their judgment of the application on how this first segment of it turns out. If it goes well, and fulfills their requirements and expectations, they will be enthusiastic and cooperative when it comes time for the next phase. If this first phase fails to meet requirements and expectations, the word will be spread, and all users will be skeptical and uncooperative toward implementation of the rest of the application.

To help ensure the success of the first phase, it should be easy to implement and should be useful. By “easy to implement” we mean:

- All the requirements have been accurately documented
- The technical approach is known and presents no problems
- Trained personnel are available
- The required data already exists
- Installation of new hardware isn't needed.

To appear useful to the user, the phase must reduce the workload, provide the information faster than previous methods, or provide previously unavailable information or functions.

If the first phase is a success, and if the results of it are visible to other user groups and management, the rest of the implementation of the application will be accomplished much more easily; however, you should, as part of your implementation plan, decide on contingency plans that will allow you to be flexible in the implementation, and to recover if something goes wrong.

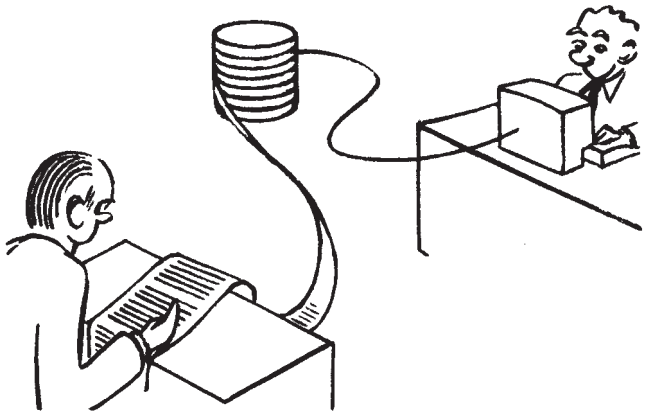
Implementation

Even though you have a complete, well thought-out plan for implementing your application, it is not likely that you will be able to follow it exactly. There are equipment and human resource problems to contend with. Time estimates may have been wrong. Data that was scheduled to be available at a certain time may not be available. New hardware that is required may be delayed in delivery or installation. This is where your contingency plans come into play. It is vital that you do not begin implementation of the application until you have covered every possible difficulty.

Your implementation schedule should have some built-in slack to help absorb unexpected difficulties or misjudgments of time necessary for certain steps. Do not allow yourself to be rushed during the implementation of the first phase if this is at all possible. If the implementation is completely new, you should install it and test it thoroughly before transferring it to production status. If the application is to replace an existing application, you must run in parallel during the testing period. This may be expensive and complicated, but is extremely important, especially if the application involves a vital business function.

You should work closely with the end users as the testing proceeds and you approach production status. Be sure that they are satisfied with the results and performance before you go to production. You should track the application after it is in production for a long enough period to make sure that it is working smoothly. Set up procedures and lines of communication to ensure that problems and difficulties are reported quickly and accurately.

Once the first phase is running smoothly, the rest can be implemented step-by-step according to your plan.



You have completed the application and data base design process. The users can now reap the benefits of that design in their daily work.

Appendix A. Appendix A: An Example of Application Design With Data Bases

This appendix is an example of the application design process described in the body of the book. The application implemented in the sample programs shipped with DL/I is used in this walk-through of the procedure. The data base design procedure described in Part 2 is used to design the necessary data bases.

The sample application documented in this appendix is for a fictitious company (a wholesale distribution firm) that offers a wide variety of electronic components. The components are purchased from various vendors and sold to customers. Most customer orders arrive by telephone. Because of this and the growth in the number of orders and variety of items, an upgrade of the existing inventory control and customer order applications was necessary. It was decided to build a new system which integrated these applications utilizing the DL/I data base approach.

Some objectives for the new application were:

- Implement:
  - Inventory control with its associated purchase order processing
  - Customer order processing
- Provide central control of inventory, purchase orders, and customer orders
- Provide accurate status information on items in stock, on order, and delivered
- Provide accurate entry of both purchase orders and customer orders with respect to items in stock
- Provide a base for online processing of orders and inquiries.

The implementation of this system will be the common thread throughout the examples used in this appendix.

Inventory Data Base

Information about items in stock is managed by the inventory control department. All data will be stored in the Inventory data base. This data base consists of one record for each item the company stocks. Each record identifies:

- Standard information for all items
- Stock location information for those items that are in stock
- Purchase information for those items that need restocking.

Customer Data Base

Information about customer orders is managed by the sales department. All order data will be stored in the Customer data base. It consists of one record for each customer order. Each record identifies:

- Standard information for each order and customer
- Order detail information for each ordered item
- Shipment information for this order.

A link is required to the Inventory data base because it is necessary to know which parts are on order by each customer and vice versa.

Obtaining Application Requirements

The process of obtaining application requirements was completed and the following documentation was included in the requirements listing:

Customer Listing

Requirement: Listing of all data for every customer, in approximately the format shown below.

Input Data:

Customer number
Customer name
Customer contact
Customer address
Customer locations
    Name
    Number
    Address
    Contact
Customer orders
    Order date
    Order number
    Reference data
    Item count
    Total order amount
    Inventory item number
    Line item number
    Quantity ordered
    Quantity shipped
    Quantity back ordered
    Item amount
Customer order history
    Order date
    Order number
    Reference data
    Item count
    Order amount
    Order status
Credit limit
Credit balance

Report Format:

```
                LIST OF CUSTOMER DATA BASE

NAME: COMPANY X, INC.           NUMBER: 000001
CONTACT: MR. JOHN SMITH
STREET: 10 MAIN STREET          CITY: NEW YORK, NY  10010
REGION: EASTERN REGION          CONTACT: MR. JOHN DOE
STREET: 69 BROAD STREET         CITY: PHILADELPHIA, PA  11020
-------------------------------------------------------------
01/29/80  ORDER NO.: 100500  DESC.: FIRST 1980 ORDER
ITEM NO.  DESCRIPTION              ORDERED  SHIPPED  BACK ORD.
000100    INTEGRATED CIRCUIT       000040   000040   000000
DOLLAR AMOUNT: $       400.00

REGION: WESTERN REGION          CONTACT: MR. JAMES SMITH
STREET: 5296 BATTLESHIP BLVD.   CITY: SAN DIEGO, CA 93210
-------------------------------------------------------------
05/10/80  ORDER NO.: 102050  DESC.: SECOND 1980 ORDER
ITEM NO.  DESCRIPTION              ORDERED  SHIPPED  BACK ORD.
000200    TRANSISTOR               000018   000008   000010
000300    RESISTOR                 000017   000017   000000
DOLLAR AMOUNT: $        88.00

CREDIT BALANCE: $     1000.00  AMOUNT LAST ORDER: $    14000.00
LAST ORDER DATA: ORDER SHIPPED COMPLETE AND ON TIME
```

Frequency: Daily

Response: Overnight

Inventory Listing

Requirement: Listing of complete inventory data, in approximately the format shown below.

Input Data:

- Item number
- Item description
- Quantity on hand
- Quantity on order
- Quantity reserved
- Unit price
- Unit of issue
- Vendor information
  - Number
  - Name
  - Address
  - Contact
- Substitute item number
- Inventory location number

Report Format:

```
                LIST OF INVENTORY DATA BASE

ITEM NO.: 000100  DESCRIPTION: INTEGRATED CIRCUIT
UNIT COST: $    10.00  QTY ON HAND: 000025  QTY ON ORDER: 002000
OPEN ORDERS: DATE      ORDER NO.  ORDERED  SHIPPED
           05/10/80    102150     000100   000050
VENDOR NAME: COMPANY A, INC.       CONTACT: MR. JOHN ROE
STREET: 207 SOUTH STREET           CITY: ANYTOWN, NY  13520
WAREHOUSE LOCATION:   12-2  SUBSTITUTE ITEM NO.: 000500
```

Frequency: Daily

Response: Overnight

Interactive Order Entry and Query

Requirement: An interactive order entry and order query system with the following requirements:

1. Must provide order entry for new customer and new location, existing customer and new location, and existing customer and existing location.
2. Customer can have one or many locations placing orders. Each location can place zero or many orders. The entry operator assigns order number, order date, and order description. Order consists of item numbers and quantities. Maximum of five items per order. Maximum quantity per item is 9,999.
3. Display complete order information in format similar to customer listing above.
4. Order entry causes automatic update of inventory and notes need for back order.
5. Query can be based on order date, customer name or number, or location name or number.
6. Query can be for complete order, status of items, customer and location information for specific order, orders associated with specific customer location, and locations associated with a specific customer.

Input Data:

- Customer number
- Customer name
- Customer contact
- Customer address
- Customer locations
  - Name
  - Number
  - Address
  - Contact
- Customer orders
  - Order date
  - Order number
  - Reference data
  - Item count
  - Total order amount
  - Inventory item number
  - Line item number
  - Quantity ordered
  - Quantity shipped

Quantity back ordered
Item amount
Customer order history
Order date
Order number
Reference data
Item count
Order amount
Order status
Credit limit
Credit balance
Item number
Item description
Quantity on hand
Quantity on order
Quantity reserved
Unit price
Unit of issue
Vendor information
Number
Name
Address
Contact
Substitute item number
Inventory location number

Report Format: To be determined

Frequency: Online

Response: As fast as possible

Application Analysis

Defining the Tasks

From the requirements listing, we see that the following tasks must be implemented:

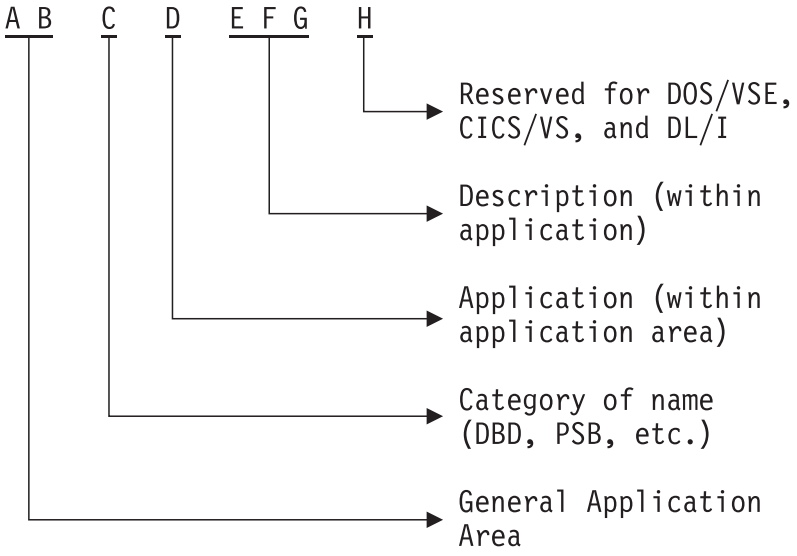
1. A listing of all data for every customer, in a format similar to that shown in the listing.
2. A listing of all inventory data, in a format similar to that shown in the listing.
3. An interactive order entry and inquiry system to fulfill the listed requirements.

Defining Programs to Accomplish the Tasks

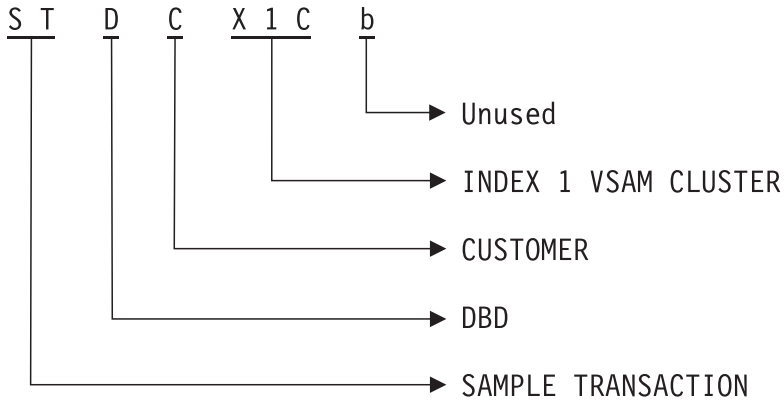
After defining and analyzing the requirements and considerations as described in Chapter 4, it was decided that the customer and inventory data listing tasks could be accomplished by one program. The interactive order entry and inquiry system would be a single program.

Naming Conventions Used in the Sample Application

The naming conventions used in the sample application observe the following format:



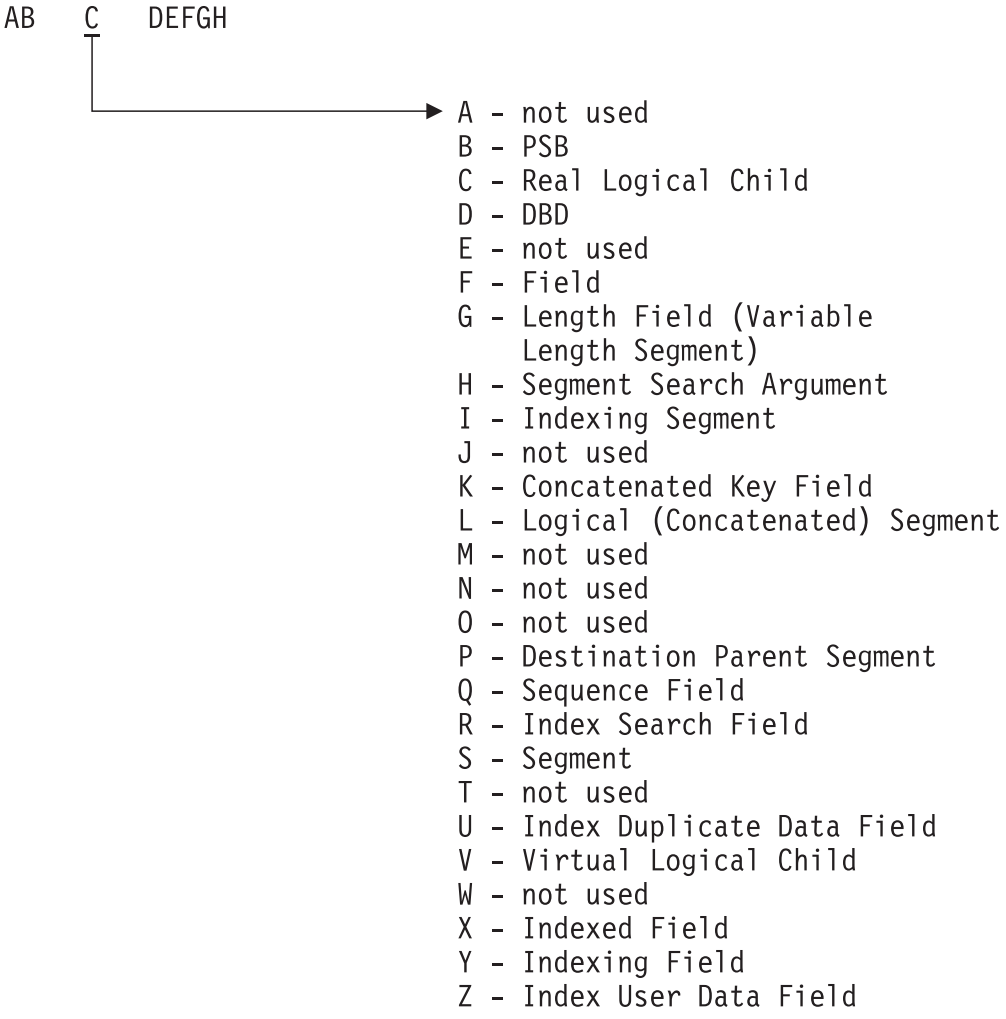
Example:



Thus: The name STDCX1C represents the VSAM cluster definition for Index 1 of the CUSTOMER data base within the Sample Transaction set of applications.

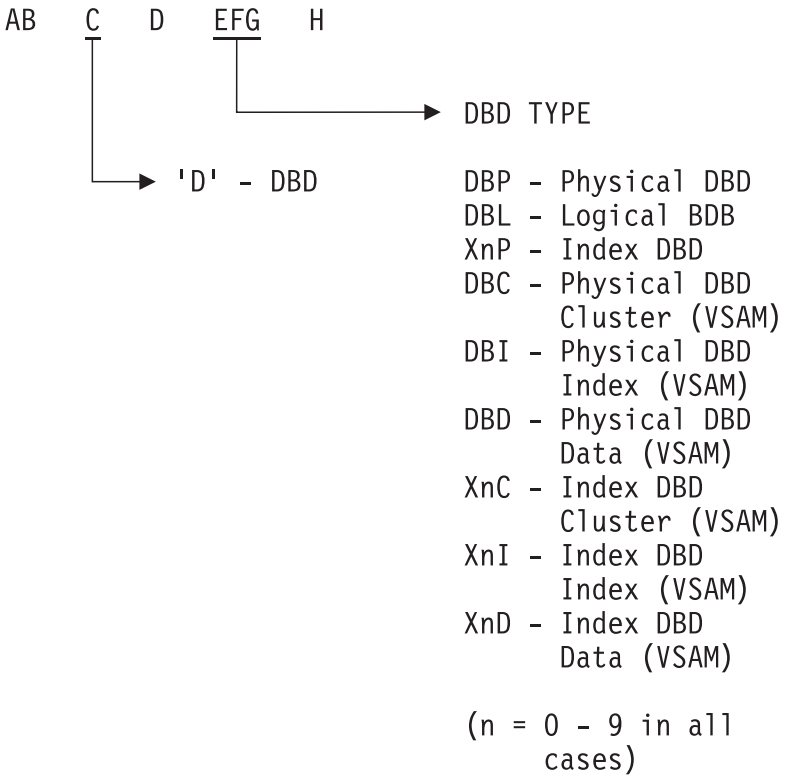
Naming Conventions - Application Area

Naming conventions - Categories



Naming Conventions - Applications

Naming Conventions - DBD



Note: The names of the segments and data elements shown in all data base examples are as they will be used in the final online application. Therefore, some names used in the early examples are not consistent with the naming conventions described.

Creating a Data Dictionary From the Requirements Listing

Notice that the third position of the names in the following data dictionary listing has not been filled in. This is because, at this stage of the design, we don't know for sure what category each will finally fall into as the design progresses. When the design is complete, the correct category designation from the list in the naming convention can be added.

Name	Description	Length (bytes)
ST_IINO	Item Number	6 (key)
ST_IIDS	Description	25
ST_IIQH	Quantity on hand	6
ST_IIQO	Quantity on order	6
ST_IIQR	Quantity reserved	6
ST_IIPR	Unit price	6 (3 dec. places)
ST_IIUN	Unit of issue	1
ST_IVNO	Vendor Number	6 (key)
ST_IVNM	Vendor Name	25
ST_IVA1	Loc. Address Line 1	25
ST_IVA2	Loc. Address Line 2	25
ST_IVA3	Loc. Address Line 3	25
ST_ISNO	Sub. Item Number	6 (key)
ST_ILNO	Inventory Loc. No.	6 (key)
ST_ILQT	Quantity	6
ST_CCNO	Customer Number	6 (key)
ST_CCNM	Customer Name	25
ST_CCA1	Cust. Address Line 1	25
ST_CCA2	Cust. Address Line 2	25
ST_CCA3	Cust. Address Line 3	25
ST_CLNO	Location Number	6
ST_CLNM	Location Name	25
ST_CLA1	Loc. Address Line 1	25
ST_CLA2	Loc. Address Line 2	25
ST_CLA3	Loc. Address Line 3	25
ST_CODN	Order Date (yr-mo-day) and Order Number	12
ST_CORF	Order Reference Data	25
ST_COIC	Order Item Count	2
ST_COAM	Order Amount	12
ST_CIIN	Inventory Item Number	6
ST_CILI	Line Item Number	2
ST_CIQO	Quantity Ordered	6
ST_CIQS	Quantity Shipped	6
ST_CIQB	Quantity Back Ordered	6
ST_CIAM	Item Amount	12
ST_CSCL	Credit Limit	12
ST_CSBL	Credit Balance	12
ST_CHDN	Order Date (yr-mo-day) and Order Number	12
ST_CHRF	Order Reference Data	25
ST_CHIC	Order Item Count	2
ST_CHAM	Order Amount	12
ST_CHOS	Order Status	77

Creating Local Views From Requirements

We are now ready to create local views using the data from the requirements listing and data dictionary. For simplicity, only the last four characters of the data element names will be used.

First, we will divide the data elements into data aggregates, name each aggregate according to the naming convention, and select a key for each aggregate (the key may change or be eliminated as the design proceeds).

- From the inventory data we have four aggregates:
  - Inventory Item (___IITM), made up of

    IINO (key)
    IIDS
    IIQH
    IIQO
    IIQR
    IIPR
    IIUN

  - Vendor (___IVND), made up of

    IVNO (key)
    IVNM
    IVA1
    IVA2
    IVA3

  - Substitute Item (___ISUB), made up of

    ISNO (key)

  - Inventory Location (___ILOC), made up of

    ILNO (key)
    ILQT

- From the customer data we have six aggregates:
  - Customer Name and Address (___CCST), made up of

    CCNO (key)
    CCNM
    CCA1
    CCA2
    CCA3

  - Customer Location (___CLOC), made up of

    CLNO (key)
    CLNM
    CLA1
    CLA2
    CLA3

– Customer Order(___CORD), made up of

  CODN (key)
  CORF
  COIC
  COAM

– Order Item (___CITM), made up of

  CILI (key)
  CIIN
  CIQO
  CIQS
  CIQB
  CIAM

– Customer Status (___CSTA), made up of

  CSCL (key)
  CSBL

– Customer History (___CHIS), made up of

  CHDN (key)
  CHRF
  CHIC
  CHAM
  CHOS

Note: For simplicity, in most of the examples in the rest of this appendix, the names of the aggregates will be used instead of listing all of the data elements.

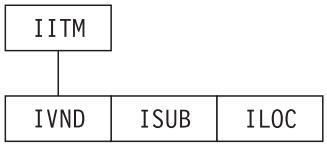
Now we can lump together all of the data that applies to the inventory application as though it were one aggregate, like this:

IITM	IVND	ISUB	ILOC

Using the format of the printout in the requirements listing and the data dictionary, we can tabulate occurrences of the inventory aggregate as shown here:

Data Element List	Occurrence #1	Occurrence #2
IINO	00100	00200
IIDS	INTEGRATED CIRCUIT	TRANSISTOR
IIQH	002500	003000
IIQO	002000	005000
IIQR	000025	000500
IIPR	010000	006750
IIUN	1	1
IVNO	multiple	multiple
IVNM	multiple	multiple
IVA1	multiple	multiple
IVA2	multiple	multiple
IVA3	multiple	multiple
ISNO	multiple	multiple
ILNO	multiple	multiple
ILQT	multiple	multiple

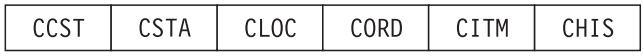
Performing the first step of the three step procedure, isolating repeating data elements, shows that since the IVND, ISUB, and ILOC aggregates can all have possible repeating data elements, we must move them to a lower level than the IITM aggregate.



Looking at the two occurrences of the inventory aggregate shows that there are no duplicate values between the occurrences, so the second step is not applicable.

Checking the keys we previously assigned shows that the data elements are all with their identifying keys, so the third step is also not necessary.

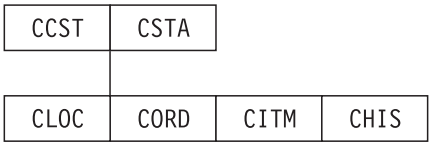
Putting all of the customer data into one aggregate, we get:



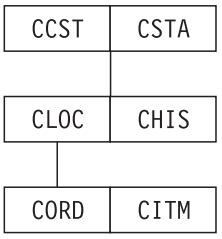
Tabulating two occurrences of the customer aggregate gives us this:

Data Element List	Occurrence #1	Occurrence #2
CCNO	000001	000002
CCNM	COMPANY X, INC.	COMPANY Y, INC.
CCA1	MR. JOHN SMITH	MR. HENRY ADAMS
CCA2	10 MAIN STREET	104 ELM STREET
CCA3	NEW YORK, NY 10010	SOMECITY, NY 13990
CSCL	000000015000	000000025000
CSBL	000000001000	000000014980
CLNO	000010	000002
CLNM	multiple	multiple
CLA1	multiple	multiple
CLA2	multiple	multiple
CLA3	multiple	multiple
CODN	multiple	multiple
CORF	multiple	multiple
COIC	multiple	multiple
COAM	multiple	multiple
CIIN	multiple	multiple
CILI	multiple	multiple
CIQO	multiple	multiple
CIQS	multiple	multiple
CIQB	multiple	multiple
CIAM	multiple	multiple
CHDN	multiple	multiple
CHRF	multiple	multiple
CHIC	multiple	multiple
CHAM	multiple	multiple
CHOS	multiple	multiple

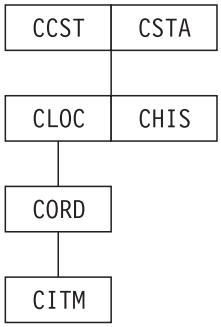
The CLOC, CORD, CITM, and CHIS aggregates can all have repeating values in their data elements so, in the first step, they must be moved down.



A little thought shows that the CORD and CITM aggregates can have repeating values under one occurrence of the CLOC aggregate, so they must be moved to a lower level under the CLOC aggregate.



Since there can be many items under a single order, the CITM aggregate must be moved to a level under the CORD aggregate, like this:



Looking at the two occurrences of the customer data aggregate shows that there are no duplicate values, so the second step is unnecessary.

The data elements are all with their keys, so the last step is also unnecessary.

Combining Local Views Into a System View

We will combine the two local views into a system view by using the procedure described in Chapter 6.

Note: For the sake of simplicity, in the illustrations that follow, the data elements identified by a given primary key are shown in a box with that key. The single arrow links from the key to those data elements are assumed.

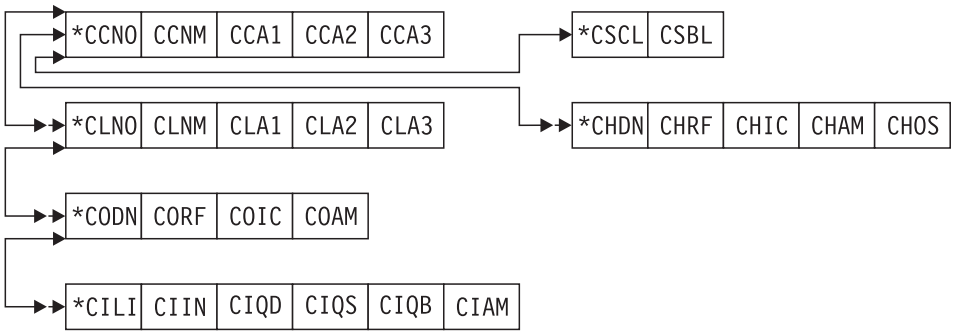


Figure A-1. Customer Data Local View

Figure A-1 shows the customer data local view drawn according to the procedure. This view is straightforward. The adding of the inventory view is also straightforward except for the fact that the inventory item aggregate in the inventory view must have a path to the order item aggregate in the customer data view.

The combined local views are shown in Figure A-2.

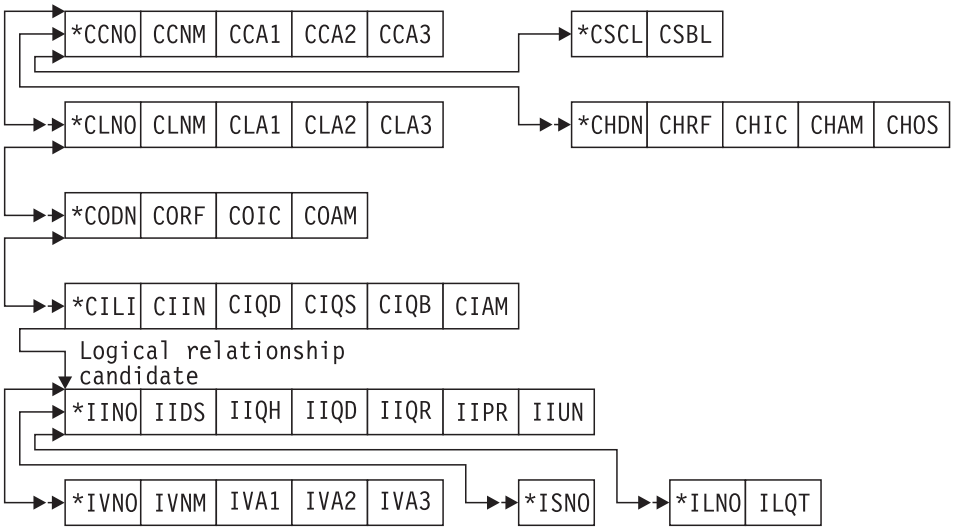


Figure A-2. Customer Data and Inventory Local Views Combined

The system view will be implemented as two separate data bases with the logical relationship link between them. The two root keys will be CCNO and IINO. There are no isolated attributes. The only logical relationship candidate is the one between the views. There are no secondary key candidates. Figure A-3 shows the system view re-drawn as a hierarchical structure as DBA might look at it.

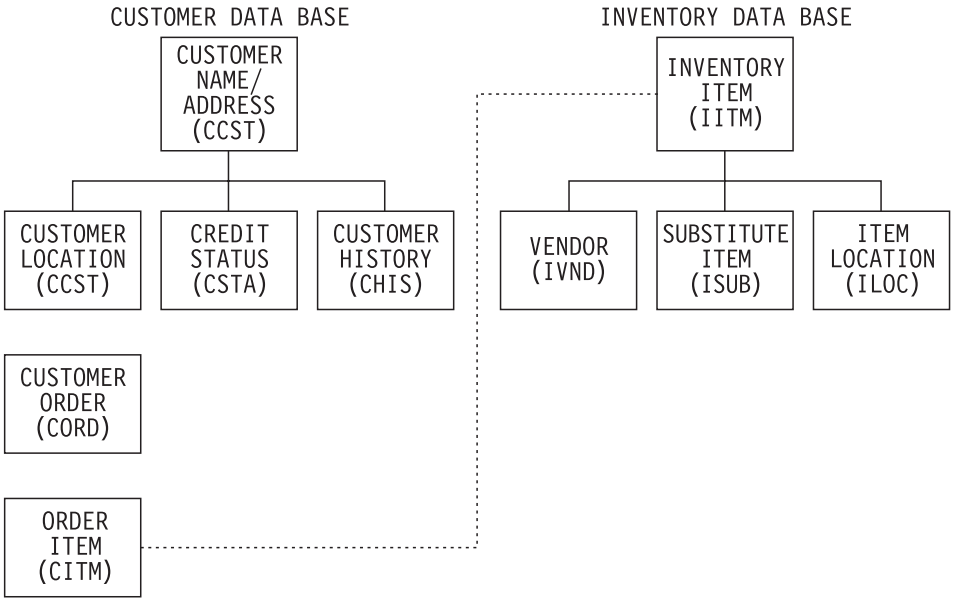


Figure A-3. System View as a Hierarchical Structure

Converting the System View to a Physical View

The details of the decisions regarding the choice of organization and access method, and of the physical implementation of the system view, are too extensive to be included here. That phase of the design is a DBA function and is described in *DL/I DOS/VS Data Base Administration*.

Appendix B. Appendix B: A Recommended Naming Convention

This appendix summarizes a naming convention recommended in the *Standards Manual for DOS/VSE*.

The *Standards Manual for DOS/VSE* describes and recommends an installation-wide naming convention that covers all possible situations in the DOS/VSE environment. Those parts of the convention that are most significant to you in your work are summarized here.

Acronym	Description	Valid Contents
S	General system	Alpha
PP	Project code	Alpha
f	Frequency of run	Numeric 0-daily 1-weekly 2-monthly 3-quarterly 4-yearly 5-as needed 6-one time job
J	Job	Alpha-assigned in sequence A-Z. If more than 26 jobs/project, divide into two projects.
n	Program number	Numeric-in sequence within job 0-9. If only one program/job, then it is always 0.
mm	Phase number	Numeric-in sequence 00-99.
r	Filename sequence number	Numeric-in sequence 0-9.
	or	
	Procedure component sequence number	Numeric-in sequence 0-9.
d...d	Descriptive file name	Alphameric - 1-8 character description of job.

Job Name

```
Job                         SPPfJ

EXAMPLE

Description                 Acronym     Content

General system              S           P
Project code                PP          CK
Frequency of run            f           1
Job                         J           A
```

Putting this all together, the job name is PCK1A.

Program Name

The program name is the phase name used at link-edit time to catalog the phase into a core image library. It is also the term generally used when referencing this program for any reason.

```
Program                     SPPfJn00

EXAMPLE

Description                 Acronym     Content

General System              S           P
Project Code                PP          CK
Frequency of run            f           1
Job                         J           A
Program                     n           1
Constant                    00          00
```

Putting this all together, the program name is PCK1A100.

Phase Name

The phase name is the same as the program name unless there are multiple phases in a program. In that case, the last two digits are used to designate the phase number. The root or first phase is always the same as the program name. It always ends in “00.”

```
Phase                       SPPfJnmm

EXAMPLE

Description                 Acronym     Content

General System              S           P
Project Code                PP          CK
Frequency of run            f           1
Job                         J           A
Program                     n           1
Phase number                mm          01
```

Putting this all together, the phase name is PCK1A101. Since the last two digits are not “00,” you know that this is not the root phase of this program. The root phase would be named PCK1A100.

Additional Conventions for DL/I

All data base jobs and programs will utilize the convention described, but a modification is required to define names that identify data base related components such as DBDs, PSBs, segments, fields, filenames, and file-IDs. (DL/I terms such as DBD and PSB are defined in the glossary.)

The first three positions of these names (ZBB) will identify them as data base components. The “Z” is a constant that is only used for data bases. The “BB” is unique for each data base and will identify all elements addressing that data base.

The fourth position of the names (ZBB*C*) is a category code to identify the specific element being named. The contents of this position are as follows:

C Real logical child segment
D DBD or filename or file-ID
F Field
P PSB
Q Sequence (key) field
S Segment
U Duplicate data field in index segment
V Virtual logical child segment
W Destination parent segment
Z User data field in index segment.

The fifth thru eighth positions of a name (ZBBCXXXX) will vary depending on the type of element. The values for these positions are specified in the following sub-sections.

DBD

- ZBBDttt -

  where
  ttt=
  DBP Physical DBD
  DBL Logical DBD
  XnP Index DBD (n is 0 thru 9)

Example: Physical DBD for the Personnel/Payroll data base in the Administration System - ZPRDDBP.

Note: The DBD name can be one to seven alphameric characters. The at-sign (@) must not be used.

Filename

- ZBBDvvv -

  where
  vvv=
  DBC Cluster for physical DBD
  DBI Index for physical DBD
  DBD Data for physical DBD
  XnC Cluster for index DBD (n is 0 thru 9)
  XnI Index for index DBD (n is 0 thru 9)
  XnD Data for index DBD (n is 0 thru 9)

Example: Filename for the VSAM cluster of a physical DBD - ZPRDDBC.

Note: The filename can be one to seven alphameric characters.

File-Id

- ZBBDvvv0.d...d -

  where
  vvv=
  Same as filename value
  and
  d...d=
  Alphabetic description of file content

Example: VSAM cluster name for physical DBD - ZPRDDBC0.HIDAM.

PSB

- ZBBPnnn -

  where
  nnn=
  Numeric sequence within a data base

Example: ZPRP003.

Note: The PSB name can be one to seven alphameric characters. The at-sign (@) must not be used.

Segment

- ZBBSGddd -

  where
  Gddd=
  G—Alphanumeric segment identifier for this segment that is unique within a data base.
  ddd—Alphabetic abbreviation describing contents of this segment type. This is project/content oriented rather than dependent on the physical structure.

Example: The tax segment of the Personnel/Payroll data base - ZPRSTTAX.

Note: The segment name can be one to eight alphameric characters.

Field

- ZBBFGddd -

  where
  Gddd=
  G—Same segment identifier as segment in which field occurs.
  ddd—Alphabetic abbreviation describing content.

Example: The number-of-exemptions field in the tax segment - ZPRFTEXS.

Note: The field name can be one to eight alphameric characters.

Glossary

ACB. Application control block.

ACBGEN. Application control block generation.

ACT. Application control table.

aggregate. See *data aggregate*.

application control blocks. The control blocks created from the output of DBDGEN and PSBGEN, e.g., a DMB of an internal PSB created by the ACB utility program.

application control block generation (ACBGEN). The process by which application control blocks are created.

application control table (ACT). A DL/I online table describing those CICS/VS application programs that utilize DL/I.

attribute. A property of an entity. It contains a value. Synonymous with *field*.

batch processing. A processing environment in which data base transactions requested by applications are accumulated and then processed periodically against a data base.

business process. A defined function of a business enterprise, usually interrelated through information requirements with other business processes. Example: Personnel management is the business process responsible for employee welfare from pre-hire to retirement. Related to the accounting business process through payroll.

child. Synonymous with *child segment*.

child segment. A segment one or more levels below the segment which is its parent, but with a direct path back up to the parent. Depending on the structure of the data base, a parent may have many children; however, a child has only one parent segment. Referring to Figure G-1:

- all the B, C, D, E, and F segments are children of A-001.
- C-5 and C-7 are children of B-01 (and A-001), but not children of the other B segments.
- B-02 has no children.

See also *logical child* and *physical child*.

concatenated key. The key constructed to access a particular segment. It consists of the key fields, including that of the root segment and successive children down to the accessed segment.

data aggregate. A group of data elements that describe a particular entity. Synonymous with *segment*. See also *data element*.

data base (DB). (1) (ISO)[1] A set of data, part of the whole of another set of data, and consisting of at least one file, that is sufficient for a given purpose or for a given data processing system. (2) A collection of data records comprised of one or more data sets. (3) A collection of interrelated or independent data items stored together without unnecessary redundancy to serve one or more applications. See *physical data base* and *logical data base*.

data base administration (DBA). The tasks associated with defining the rules by which data is accessed and stored. The typical tasks of data base administration are outlined in the *DL/I DOS/VS Data Base Administration*, SH24-5011.

data base administrator (DBA). A person in an installation who has the responsibility (full or part time) for technically supporting the use of DL/I.

data base definition (DBD). A description of the physical characteristics of a DL/I data base. One DBD is generated and cataloged in a core image library for each data base that is used in the installation. It defines the structure, segment keys, physical organization, names, access method, devices, etc., of the data base.

data base integrity. The protection of data items in a data base while they are available to any application program. This includes the isolation of the effects of concurrent updates to a data base by two or more application programs.

data base organization. The physical arrangement of related data on a storage device. DL/I data base organizations are hierarchical direct (HD) and hierarchical sequential (HS). See *hierarchical direct organization* and *hierarchical sequential organization*.

data base record. A collection of DL/I data elements called segments hierarchically related to a single root segment.

[1] International Organization for Standardization, Technical Committee 97/Subcommittee 1

data dictionary. (1) A centralized repository of information about data, such as: its meaning, relationship to other data, usage, and format. (2) A program to assist in effectively planning, controlling, and evaluating the collection, storage, and use of data. For example, DOS/VS DB/DC Data Dictionary.

data element. The smallest unit of data that can be referred to. Synonymous with *field*. See also *data aggregate*.

data field. Synonymous with *field*.

data independence. (1) The concept of separating the definitions of logical and physical data such that application programs do not depend on where or how physical units of data are stored. (2) The reduction of application program modification in data storage structure and access strategy.

data management block (DMB). The data management block is created from a DBD by the application control blocks creation and maintenance utility, link edited, and cataloged in a core image library. The DMB describes all physical characteristics of a data base. Before an application program using DL/I facilities can be run, one DMB for each data base accessed, plus a PSB for the program itself, must be cataloged in a core image library. The DMBs and the associated PSB are automatically loaded into main storage from the core image library at the beginning of the application program execution (their loading is controlled by the parameter information supplied to DL/I at the beginning of program execution).

data set. A named organized collection of logically related records. They may be organized sequentially, as in the case of DOS/VSE SAM, or in key entry sequence, as in the case of VSE/VSAM. Synonymous with *file*.

DB. Data base.

DBA. (1) Data base administration (2) Data base administrator.

DBD. Data base description.

DBDGEN. Data base definition generation—the process by which a DBD is created.

DB/DC. Data base/data communication.

DC. Data communication.

dependent segment. A DL/I segment that relies on at least the root segment (and other dependent segments) for its full hierarchical meaning. Synonymous with *child segment*.

destination parent. The physical or logical parent segment reached by the logical child path.

device independence. The concept of writing application programs such that they do not depend on the physical characteristics of the device on which data is stored.

direct access. The retrieval or storage of a VSAM data record independent of the record's location relative to the previously retrieved or stored record.

DMB. Data management block.

entity. An item about which information is stored. It has properties that can be recorded. Information about an entity is a record.

entry sequenced data set (ESDS). A VSAM data set whose records are physically in the same order as they were put in the data set. It is processed by addressed direct access or addressed sequential access and has no index. New records are added at the end of the data set.

ESDS. Entry sequenced data set.

field. (1) a unique or nonunique area (as defined during DBDGEN) within a segment that is the smallest unit of data that can be referred to. See also *key field*. (2) Any designated portion of a segment.

field level sensitivity. The ability of an application program to access data at the field level. See *sensitivity*.

forward. Movement in a direction from the beginning of the data base to the end of the data base, accessing each record in ascending root key sequence, and accessing the dependent segments of each root segment from top to bottom and from left to right. Referring to Figure G-1, forward accessing of all segments shown would be in the following sequence:

A-001, B-01, C-5, C-7, B-02, B-03, C-2, D, E, F, F, F, A-002.

HD. Hierarchical direct.

HDAM. Hierarchical direct access method.

HIDAM. Hierarchical indexed direct access method.

HIDAM index. A data base that consists of logical DL/I records each containing an image of the key field of a HIDAM root segment. A HIDAM index data base consists of one VSAM KSDS (keyed sequenced data set).

hierarchical direct access method (HDAM). Provides for direct access to a DL/I data base in the HD

organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a randomizing routine. An HDAM data base consists of one VSAM entry sequenced data set (ESDS).

hierarchical direct organization. An organization of DL/I segments of a data base that are related by direct addresses and may be accessed through an HD randomizing routine or an index.

hierarchical indexed direct access method (HIDAM). Provides for indexed access to a DL/I data base in the HD organization. Segments are stored in VSAM control intervals and are referenced by a relative byte address. Root segments are accessed through a HIDAM index data base. A HIDAM data base consists of one VSAM Entry Sequenced Data Set (ESDS).

hierarchical indexed sequential access method (HISAM). Provides for indexed access to a DL/I data base. A HISAM data base consists of one VSAM key sequenced data set (KSDS) and one VSAM entry sequenced data set (ESDS).

hierarchic sequence. The sequence of segment occurrences in a data base record defined by traversing the hierarchy from top to bottom, front to back, and left to right.

hierarchical sequential access method (HSAM). The segments of a DL/I HSAM physical data base record are arranged in sequential order with the root segments followed by the dependent segments. HSAM data bases are accessed by the DOS/VSE sequential access method (SAM).

hierarchical sequential organization. An organization of DL/I segments of a data base that are related by physical adjacency.

hierarchy. (1) An arrangement of data segments beginning with the root segment and proceeding downward to dependent segments. (2) A "tree" structure.

HISAM. Hierarchical indexed sequential access method.

HS. Hierarchical sequential.

HSAM. Hierarchical sequential access method.

index data set. An ordered collection of DL/I index entries consisting of a key and a pointer used by VSAM to sequence and locate the records of a key sequenced data set (KSDS). Organized as a balanced tree of levels of index.

index pointer segment. The segment that contains the data and pointers used to index the index target segments.

index record. A system-created collection of VSAM index entries that are in collating sequence by the key in each of the entries.

index segment. The segment in the index data base that contains a pointer to the segment containing data (the indexed segment). Synonymous with *index pointer segment*.

index source segment. The segment containing the data from which the indexing segment is built.

index target segment. The segment pointed to by a secondary index entry, that is, by an index pointer segment.

indexed segment. A segment that is located by an index. Synonymous with *index target segment*.

intersection data. Any user data in a logical child segment that does not include the logical parent's concatenated key.

key. (1) The field in a segment used to store segment occurrences in sequential order. (2) A field used to search for a segment. (3) Synonymous with *key field* and *sequence field*.

Note: A segment may or may not have a key, that is, a sequence field. All root segments, except for HSAM and simple HSAM data bases, must have keys. DL/I ensures that multiple segments of the same type that have keys are maintained in strict ascending sequence by key. The key may be located anywhere within a segment; it must be in the same location in all segments of the same type within a data base. The maximum sizes for keys are 236 alphameric characters for root segments, and 255 for all dependent segments. Keys provide a convenient way to retrieve a specific occurrence of a segment type, maintains the uniqueness and sequential integrity of multiples of the same segment type, and determine under which segment of a group of multiples new dependent segments are to be inserted. Keys should normally be prescribed for all segment types; the exception being if there will never be multiples of a particular type or if a particular segment type will never have dependents.

level. Level is the depth in the hierarchical structure at which a segment is located. Roots are always the highest level and the segments at the bottom of the structure are the lowest level. The maximum number of levels in a a DL/I data base is 15. For purposes of documentation and reference, the levels are numbered from 1 to 15, with the root segments being level number 1. Referring to Figure G-1:

- Three levels are shown.
- The A segments (roots) are at the highest level (level 1).
- The C and E segments are at the lowest level (level 3).

local view. A description of the data that an individual business process requires. See *system view*.

logical. When used in reference to DL/I components, logical means that the component is treated according to the rules of DL/I rather than physically as it may exist, or as it may be organized, on a physical storage device. For example, a logical DL/I record (a root segment and all of its dependent segments grouped) might be contained on several physical records or blocks on a storage device; and because of prior insertions and deletions, the segments might be in different physical sequence than that by which they are retrieved logically for the application program by DL/I.

logical child. A pointer segment that establishes an access path between its physical parent and its logical parent. It is a physical child of its physical parent; it is a logical child of its logical parent. See also *logical parent* and *logical relationship*.

logical data base. A data base composed of one or more physical data bases representing a hierarchical structure derived from relationships between data segments that can be different from the physical structure.

logical data base record. (1) A set of hierarchically related segments of one or more segment types. As viewed by the application program, the logical data base record is always a hierarchic tree structure of segments. (2) All of the segments that exist hierarchically dependent on a given root segment, and that root segment.

logical data structure. A hierarchic structure of segments that is not based solely on the physical relationship of the segments. See also *logical relationships*.

logical parent. The segment a logical child points to. A logical parent segment can also be a physical parent. See also *logical child* and *logical relationship*.

logical relationship. A user defined path between two segments; that is, between logical parent and logical child, which is independent of any physical path. Logical relationships can be defined between segments in the same physical data base hierarchy or in different hierarchies.

logical twin. All occurrences of one type of logical child with a common logical parent. Contrast with *physical twin*. See also *twin segment*.

parent. Synonymous with *parent segment*.

parent segment. (1) A segment that has one or more dependent segments. Contrast with *child*. (2) A parent is the opposite of a child, or dependent segment, in that dependent segments exist directly beneath it at lower levels. A parent may also itself be a child. Referring to Figure G-1:

- A-001 is the parent of all B, C, D, E, and F segments.
- D is a parent of E; yet a child of A.
- B-02 is not a parent.
- None of the level-3 segments are parents.

path. The chain of segments within a record that leads to the currently retrieved segment. The formal path contains only one segment occurrence from each level from the root down to the segment for which the path exists.

PCB. Program communication block.

physical child. A segment type that is dependent on a segment type defined at the next higher level in the data base hierarchy. All segment types, except the root segment, are physical children because each is dependent on at least the root segment. See also *child segment*.

physical data base. An ordered set of physical data base records.

physical data base record. A physical set of hierarchically related segments of one or more segment types.

physical data structure. A hierarchy representing the arrangement of segment types in a physical data base.

physical parent. A segment that has a dependent segment type at the next lower level in the physical data base hierarchy. See also *parent*.

physical segment. The smallest unit of accessible data.

physical twins. All occurrences of a single physical child segment type that have the same (single occurrence) physical parent segment type. Contrast with *logical twins*. See also *twin segment*.

pointer. A physical or symbolic identifier of a unique target.

primary key. The data element or combination of data elements within a data aggregate that uniquely identifies an occurrence of that data aggregate. See *secondary key*.

program communication block (PCB). Every data base accessed in an application program has a PCB associated with it. The PCB actually exists in DL/I and its fields are accessed by the application program by defining their names within the application program.

program specification block (PSB). A PSB is generated for each application program that uses DL/I facilities. The PSB is associated with the application program for which it was generated and contains a PCB for each data base that is to be accessed by the program. Once it is generated, the PSB is cataloged in a core image library, and subsequently processed by a utility along with the associated DBDs to produce the updated PSB and DMBs; all of these are cataloged in a core image library for subsequent use by the application program during execution.

PSB. Program specification block.

PSBGEN. PSB generation—the process by which a program specification block is created.

RAP. Root anchor point.

record. A data base record is made up of at least a unique root segment, and all of its dependent segments. See also *data base record*.

Referring to Figure G-1: A-001, B-01,C-5, C-7, B-02, B-03, C-2, D, E, F,F,F constitute a data base record.

root anchor point (RAP). A DL/I pointer in an HDAM control interval pointing to a (chain of) root segment(s).

root segment. The highest level (level 1) segment in a record. A root segment must have a key unless the organization is HSAM or simple HSAM. The sequence of the root segments constitutes the fundamental sequence of the data base. There can be only one root segment per record. Dependent segments cannot exist without a parent root segment; but a root segment can exist without any dependent segments.

secondary index. Secondary indexes can be used to establish alternate entries to physical or logical data bases for application programs. They can also be processed as data bases themselves. See also *secondary index data base*.

secondary index data base. An index used to establish accessibility to a physical or logical data base by a path different from the one provided by the data base definition. It contains index pointer segments.

secondary key. A data element or combination of data elements within a data aggregate that identifies those occurrences of the aggregate that have a property named by the key. Used to locate those occurrences. See *primary key*.

segment. A segment is a group of similar or related data that can be accessed by the application program with one I/O function call. There may be a number of segments of the same type within a record.

segment name. A segment name is assigned to each segment type. Segment names for the different segment types must be unique within a data base. Synonymous with *segment type*.

segment occurrence. One instance of a set of similar segments.

segment type. Different segment types may have different lengths, but within each single type, all segments must be the same length (unless variable length segments have been specified by DBA).

Referring to Figure G-1, there are six different types of segments: A through F. Synonymous with *segment name*.

sensitivity. (1) A DL/I capability that ensures that only data segments or fields predefined as "sensitive" are available for use by a particular application program. The sensitivity concept also provides a degree of control over data security, inasmuch as users can be prevented from accessing particular segments or fields from a logical data base. (2) Sensitivity to the various segments and fields that constitute a data base is controlled, on a program-by-program basis, when the PSB for each program is generated. For example, a program is said to be sensitive to a segment type when it can access that segment type. When a program is not sensitive to a particular segment type, it appears to the program as if that segment type does not exist at all in the data base. Segment sensitivity applies to types of segments, not to specific segments within a type, and to all segment types in the path to the lowest level sensitive segment type.

sequence field. Synonymous with *key field*.

sequential processing. Processing or searching through the segments in a data base in a forward direction. See also *forward*.

simple HISAM. A hierarchical indexed sequential access method data base containing only one segment type.

source segment. A segment containing the data used to construct the secondary index pointer segment. See also *secondary index data base*.

system view. A conceptual data structure that integrates the individual structures of local views into an optimum arrangement for physical implementation as a data base. See *local view*.

transaction. A specific set of input data that triggers the execution of a specific process or job.

twin segments. All child segments of the same segment type that have a particular instance of the same parent type. See also *physical twins* and *logical twins*.

twins. Synonymous with *twin segments*.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

Communicating Your Comments to IBM

IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Application and Data Base Design
Version 1 Release 7

Publication No. SH24-5022-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of the book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF form and either send it postage-paid in the United States, or directly to:

  IBM Deutschland Entwicklung GmbH
  Department 3248
  Schoenaicher Strasse 220
  D-71032 Boeblingen
  Federal Republic of Germany

- If you prefer to send comments by FAX, use this number:
  - (Germany): 07031-16-3456
  - (Other countries): (+49)+7031-16-3456

- If you prefer to send comments electronically, use this network ID:

  INTERNET: s390id@de.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

**IBM Data Language/I Disk Operating System/
Virtual Storage (DL/I DOS/VS)
Application and Data Base Design
Version 1 Release 7**

Publication No. SH24-5022-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH24-5022-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

File Number: S370/S390-50
Program Number: 5746-XX1

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH24-5022-01