Program Product

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

IBM

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

Program Product

Program Number 5746-XX1

Data Language/I Disk Operating System/Virtual Storage (DL/I DOS/VS) is a data management control system developed to assist the user in implementing data base processing applications. It provides data organization methods that are conducive to the creation of, access to, and maintenance of large common data bases. DL/I DOS/VS permits the evolutionary expansion of data processing applications from a batch-only environment to a teleprocessing environment such as the Customer Information Control System DOS/VS (CICS/VS).

This manual includes a general description of the system and its various facilities and programs, listings of typical minimum configurations, and sample applications.

IBM

Preface

DL/I DOS/VS (Data Language/I Disk Operating System/Virtual Storage) is a data base management control system that improves an installation's ability to implement and maintain batch processing applications. DL/I DOS/VS permits the writing of data independent applications and provides program and data base integrity. The DL/I DOS/VS system supports application programs written in COBOL, PL/I, and Assembler language. DL/I DOS/VS executes as an application program under DOS/VS.

DL/I DOS/VS permits concurrent scheduling of multiple programs requesting DL/I DOS/VS services, thereby allowing access by more than one user to the same or different data bases at the same time. Application programs may utilize this concept in conjunction with CICS/VS (Customer Information Control System/Virtual Storage) to access DL/I DOS/VS data bases in a teleprocessing environment.

This publication is a general description of the DL/I DOS/VS (or DL/I) control system. It provides an overview of the system and its various facilities and programs, listings of typical and minimum configurations, and discussions of sample applications. The manual is intended primarily for customer executives, system administrators, system analysts, system programmers, and application programmers. The following IBM publications provide a more detailed description of DL/I DOS/VS:

DL/I DOS/VS Application Programming Reference Manual, SH12-5411

DL/I DOS/VS Utilities and Guide for the System Programmer, SH12-5412

DL/I DOS/VS System/Application Design Guide, SH12-5413

DL/I DOS/VS Operator's Reference Manual and Messages and Codes, SH12-5414

DL/I DOS/VS Logic Manual, LY12-5016

References are made in this publication to the Customer Information Control System/Virtual Storage. More information about CICS/VS can be found in the *Customer Information Control System/Virtual Storage (CICS/VS) General Information manual,* GC33-0052 for Version 1 Release 2.0, or GC33-0066 for Version 1 Release 3.0.

Because of the special nature of DL/I DOS/VS as a functional subset of IBM's Information Management System (IMS/VS), some IMS or OS specific terms are retained in DL/I DOS/VS documentation. These terms are used for ease of reference to corresponding IBM documentation and to facilitate subsequent upgrading to an upward-compatible IBM system.

All further references in this manual to DL/I DOS/VS are shortened to DL/I.

DL/I DOS/VS Enhancements

DL/I Version 1.2

This revision of DL/I provides the following functional enhancements:

Multiple Partition Support (MPS)

This capability enables application programs executing in different partitions to access the same data base concurrently. For example, online applications can now issue inquiries to a data base while a batch program updates it.

Although two programs cannot both update the same data base segment type concurrently, one or more programs can retrieve from it while another program updates the segment type.

Note: The above limitation does not apply to DL/I Version 1.3. See "Program Isolation" in the section, "DL/I Version 1.3 Enhancements."

The addition of MPS enables multiple batch and online application programs to access a data base concurrently instead of serially. MPS uses the DL/I resources and multitasking facilities of DL/I and CICS/VS. CICS/VS therefore, is a prerequisite for running MPS.

MPS follows normal DL/I online conventions.

Reload Restart

This facility is added to the HD Reload Utility. If a job is cancelled or fails, this facility enables you to restart the reloading of a data base without going back to the beginning. While running the HD Data Base Reorganization Unload Utility, a checkpoint record is written to the unloaded data base after approximately every 5000 segments.

These checkpoint records can be used to restart the reloading of the data base

During the normal reloading process, the checkpoint records are ignored except for a console message containing the checkpoint number. This message is printed each time a checkpoint is encountered. When a job is canceled or fails, the system operator should:

1. Run the Access Method Services Utility, executing the VERIFY command for each data base being referenced during the reload.

2. Resubmit the job with a unique parameter card, identifying the job as a restart of the Reload Utility.

3. Supply the last checkpoint number when prompted by the Reload Utility.

DL/I Version 1.3

This revision of DL/I provides system changes and functional enhancements that enable the user to achieve greater useability, more functional capability, improved performance in certain areas, and greater reliability, availability, and serviceability (RAS). The changes and enhancements made to DL/I in Version 1.3 and how they apply to these areas are shown in the following table.

Note: The information in this manual that applies to DL/I DOS/VS Version 1.3 is for planning purposes only until it is available.

Change/Enhancement	Useability	Functional Capabilities	Performance Improvements	RAS
Checkpoint		X		X
Program Isolation			X	X
Distributed Free Space		X	X	
Serviceability Aids				X
Performance Repack			X	
Utilities Operational Cleanup	X	X		X
Sample Problem	X			
Disk Logging	X*	X		X

*Applies only to the disk-only user.

The various line items in the table are described below in general terms and in an overview fashion to enable you to determine if and how they might fit into your DL/I system as you tailor your system to meet your installation's requirements. More detailed information can be found in the DL/I publications listed in the Preface as those publications become available for Version 1.3.

Checkpoint

Checkpoint aids the restarting of a job by allowing the user to request checkpoint records to be written on the DL/I log. In case of a job failure in a batch environment, the user can run the backout utility, which will backout to the most recently written checkpoint record. For multiple partition support and/or online tasks with CICS/VS dynamic transaction backout active, backout is performed automatically to the last checkpoint in case of failure.

Checkpointing is accomplished with a new DL/I call function, *CHKP*. The checkpoint call can be issued at appropriate times in a program as determined by the user. Programs, therefore, can be designed to issue checkpoint calls at any point the user determines a job should be restarted in the event there is a failure. Checkpoint processing generally reduces the time required to rerun a job with a long execution time, which was interrupted because of a failure.

Program Isolation

Program Isolation is an optional replacement for *intent scheduling* and consists of two functional areas -- contention management and deadlock avoidance. This feature provides a performance improvement by reducing the resource contention in a DL/I - CICS/VS environment from the segment type to that of the specific segment, thereby enhancing online response times. Program Isolation, at the segment level, supports only HDAM/HIDAM data bases. At the data base record level, Program Isolation supports only HISAM.

Contention management supplies the support for the avoidance of conflicts in data usage by making contenders for a resource wait until the resource is available and rescheduling them when it is.

Deadlock avoidance recognizes and remedies the case where two or more tasks are interlocked on the resources for which they are waiting.

Data integrity is enhanced because read-only intent on segments no longer allows a task to read data that may be backed out later because of a failure in another transaction.

Distributed Free Space

This feature is designed to improve performance for data bases where high-insert activity is anticipated. Distributed Free Space, which increases the probability that related segments will reside in close physical proximity, is available only for HDAM and HIDAM data bases.

During generation of the Data Base Descriptions (DBD), you can specify the amount of free space to be reserved at the initial loading or the reloading of the data base. This space can be specified in terms of blocks or percentage of space within the blocks. This is accomplished with a new free space (FRSPC) parameter in the DATASET statement.

Performance improvements are achieved when retrievals from and updates to the data base are made. The close proximity of related segments also reduces the need for the number of reorganizations of the data base because Distributed Free Space diminishes the number of unrelated segments at the end of the data base.

Serviceability Aids

A new tool is provided to help the user to recover from a system failure. The tool, *Log Print Utility,* enables you to print the contents of the log files. This utility can be invoked when an abnormal termination or a system failure occurs.

The utility runs under DOS/VS and, therefore, it does not issue any DL/I calls. The logs to be printed are identified through the DOS/VS job control language and input control statements.

The information contained in the logs selected for printing includes the log record ID, the task ID of the log record, the PSBname, the DMBname, the type of call, the record type, and the data in the record.

Performance Repack

The performance repack is internal to DL/I and, therefore, is transparent to the user. The repack is designed to improve performance by reducing working set requirements, thereby reducing paging activity, and by reducing path lengths through heavily used code.

Utilities Operational Improvements

Improvements have been made to certain utility operations to provide added function or to provide additional protection of the user's data bases. These improvements, evident at utility execution times, are described below.

Multilfile Log Tape Support

Multifile support enables the user to more efficiently use the installation's tape data capacity by allowing many log files to be written on a single volume instead of requiring a single volume for each batch job. This minimizes the number of tape volumes required and reduces operator intervention for mounting log tapes.

Multifile log tape support requires the normal DOS/VS tape handling procedures and the DOS/VS job control language.

Multivolume Log Tape Support

The data base recovery utilities are modified to handle multivolume log files. Multivolume log tape support requires the normal DOS/VS tape handling procedures and the DOS/VS job control language.

Unload Sequence Checking

This feature enhances data base integrity by performing a sequence check during an HD unload to alert the user to an incorrectly sequenced data base. The addition of the sequence check feature to HD unload allows the user to detect sequence errors early and to make the necessary corrections before the data base is reloaded and reorganized.

Abnormal Termination vs. Messages

This feature is designed to provide improved system response to error conditions by abnormally terminating a job instead of just a job step, thereby ensuring greater data protection. A data base can be damaged when a job terminates normally or just a job step is terminated after an error condition is detected, but an error message is overlooked. New actions associated with some utility messages have been incorporated to prevent this problem.

Sample Problem

A new sample problem is provided to demonstrate the use of secondary indexes and logical relationships to the user. The sample problem contains data bases and transactions as they might be required for online order entry, and inquiry in the retail distribution industry.

Disk Logging

Disk logging provides a DL/I logging facility for the disk-only user. It also provides an alternative for the tape and disk user. This function enables the disk-only user to take advantage of the recovery and backout features of DL/I that use the log to enhance the integrity of the data base.

VSAM is the access method for disk logging to the DL/I log file while SAM is the access method used for the CICS/VS journal.

The log records written to disk are compatible with the tape log records.

Contents

Figures

Most corporations eventually reach the stage where their data processing departments implement several stand-alone applications (for example payroll and stock control). These applications provide information service mainly at the operational levels. Each application is designed with its own dedicated files of data which bear little or no relation, in format or content, to the files designed for other applications.

As new applications are added to provide more information for management as well as operational levels (for example market research or financial planning), they too are designed as stand-alone applications with dedicated files. Also online access to the files means that the terminals are usually dedicated and that a terminal control code is written into each user program.

As data becomes an increasingly vital corporate resource, users are establishing standards for their files. They are designing compatible programs that allow one complete cycle of processing to do all file updates and to generate all the new reports needed by several levels of users.

Although such integrated application subsystems often bring rewards in terms of hardware utilization and job efficiency, they become much more complex, and program maintenance costs tend to climb.

Corporations are now evaluating computer systems, not only with regard to programming systems and hardware, but also in relation to the information needs of the total corporate environment. Demands for programming applications that interrogate and maintain large centralized information files are increasing. Along with this has evolved a concept that, on the surface at least, is fairly simple: take the individual dedicated files and put them together in one place, where they will be accessible to all present applications. Then, as new applications are designed, the new data can be added and linked to the existing data. This is known as the data base concept.

The concept of the data base cuts the close association between the application program and its files. This makes it possible, and desirable, to remove the responsibilities for common file management functions from the individual application programmer, and to package them together as a function of data base management.

The DL/I data base provides many features that facilitate implementation, change, and expansion of such applications and information files. DL/I helps to reduce data processing costs by:

- Reducing application program maintenance.

- Reducing application programmer time required to implement new applications, especially teleprocessing applications.

- Reducing the cost of converting to new hardware.

- Reducing the number of programs and/or data files required to implement applications.

- Reducing the number of files in which data is repeated.

The DL/I data base is designed to meet the needs of most corporations and institutions. Applications that might lend themselves to DL/I include:

- payroll and personnel

- manufacturing bill of material

- inventory control

- accounts receivable

- hospital records

- student records

- petroleum well records

- demand deposit accounts systems

Using DL/I, your programming staff can design applications to interface with the information files from remote terminals (using a CICS/VS interface), or in the more conventional batch mode, or in combination.

In addition, the ability to respond to frequent and anticipated high-volume information requests makes DL/I a powerful tool for the data processing user.

Why Data Bases?

In a non-data-base environment, data files are usually designed to serve individual applications, such as inventory control, payroll, accounts receivable, purchasing, etc. Each data file is specifically

designed for its own application and stored separately on tape or disk. Quite often, the data files of different applications contain common data elements. This redundant data causes an extra problem because it becomes very difficult to keep the data consistent.

Furthermore, while modifying existing applications and/or adding new applications, your programming staff has to face some of the following situations:

- Duplicate data exists on multiple files with different formats for different applications.

  As data applications become more and more complex, users need to interrelate more and more data. One method is through multiple files. However, using multiple files complicates program design and debugging procedures. So instead, many users try to simplify their problems by creating new files which are really extracts from existing files with some additional data added. This creates data redundancy and introduces the problem of maintaining the same element of data on several files. If one copy of the data changes, all other copies of the data must also be changed. Operations control must be very tight in order to keep current information in the hands of the data processing users.

- Programmers spend a significant amount of time updating existing application programs to handle changes to record layouts or I/O device characteristics. Often, program function is not affected by these changes.

- Changing applications make it desirable to move data files from one storage device to another (tape or disk), or from one access method (sequential) to another (direct).

- Programmer productivity is hindered by a limited knowledge of specific device characteristics or specific access methods.

- Batch applications have to be expanded to teleprocessing applications.

In short, every time a new series of applications is planned, your programmers have to evaluate the impact of record changes on existing programs. As a result, application programs are often in an almost perpetual state of change, adding appreciably to the overall cost of data processing.

The solution is to organize your data in a way that eliminates redundancy and to provide a method for handling the data that can readily adapt to changes without impacting your present programs. The most practical way to do this is to remove the physical characteristics of the data from the program, thus making the program *independent* of the data it uses. The data base provides this independence by removing the direct association between the application program and the physical storage of data.

What Is a Data Base?

A data base is a concept that provides for the integration, sharing, and control of common data. As an example, a company may first integrate the data for a parts application with the data for a purchase orders application (Figure 1-1). Later, data for order processing and accounts receivable may also be integrated. The data and the programs of already implemented applications need not change when the data of later applications are integrated.
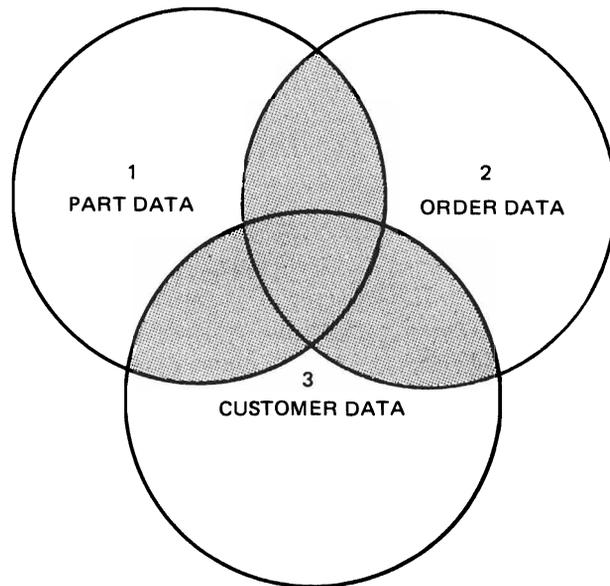
Figure 1-1. Application data integration -- Data base concepts

This is because, using the data base concept, data is stored in a hierarchical manner; that is, the most significant data resides on hierarchically higher levels while less significant but related data (dependent data) appears on subordinate levels. Through the use of a concept called *data sensitivity,* each application program views only

that data in the structure which it uses, and accesses that data through a common symbolic linkage provided by the data management portion of the data base system. For example, in Figure 1-2, assume that one application requires name and address information, and a second requires name and payroll information. The applications share their common data (name), but only the first accesses address and only the second accesses payroll. To each application, data used by other applications, other than common data, does not exist. This collection of interrelated data elements, processable by one or more applications, is called a *data base*.
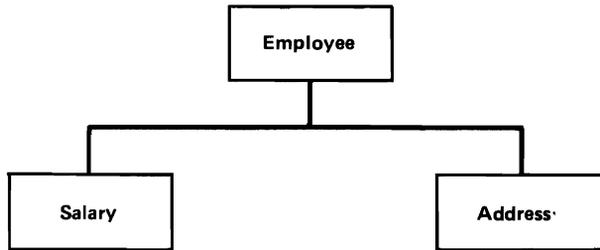
Figure 1-2. Hierarchical data base concept

What Does a Data Base Provide?

A data base provides flexibility of data organization. By removing the direct association between the application program and the physical storage of data, the data base concept allows the addition of data to an existing data base without modification of existing programs. This is called *data independence*.

The advantages of the data base are:

- Control of data redundancy and reduction of duplicate data maintenance.

- Consistency through the use of the same data by all groups in the company.

- Application program independence from physical storage organizations and access methods.

- Reduction in overall application costs.

- Data designs usable for batch and online processing.

- A system-provided focal point for the control of data.

How is the DL/I Data Base Implemented?

To build a data base operation, your programming staff reviews the data requirements of all applications, then defines the data base or data bases that serve those applications. To create a data base, the user defines to DL/I a common data structure and format that serve the applications. This definition is called a data base description (DBD). One DBD is required for each data base. The second definition required is the program specification block (PSB). The PSB defines to DL/I for each application program the data bases to be used, the data used within each data base, and the operations allowed on each data base. One PSB is required for each application program. However, the same PSB could be used by many application programs.

Both of these control blocks, the DBD and the PSB, are used to link the application data in the data bases to the application program using the data. Through DL/I's use of the DBD and PSB, application programmers can write their programs without regard to the physical structure of data. See "DL/I Data Base Structure" in Chapter 3 for a more detailed description of the DBD, PSB, and the hierarchical structure of the DL/I data base.

Chapter 2. General Description of DL/I

DL/I is a data management control system that executes as an application program in a virtual storage environment under the Disk Operating System/Virtual Storage (DOS/VS) on System/370. It is a general-purpose data management system that satisfies the diverse data base processing requirements of many companies. It simplifies your task of creating and maintaining large common data bases to be accessed by batch processing applications. Future DL/I functions may be added without affecting existing functions. DL/I also allows growth to teleprocessing applications through interface with CICS/VS.

DL/I allows application programs to be independant of access methods, physical storage organizations, and the characteristics of the devices on which the application data is stored. This independence is provided by a common symbolic program linkage and by data base descriptions external to the application program.

Much of the data used by a company has many interrelationships that can cause significant redundant storage of data when conventional organizations and access methods are used. The storage organizations and access methods of DL/I make it possible to integrate data and control the amount of data redundancy. Data can be processed in more than one sequence. All data need not be placed in a single common base. DL/I allows you to physically store the data in more than one data base while maintaining centralized control over all the data.

The concept of data *sensitivity* allows you to control the use of the data base for each application program. Each program can be limited to (that is, be sensitive to) a predetermined subset of the data. In addition, any application program can be restricted to specified types of data base requests made against sensitive data.

The DL/I system provides data base processing capabilities for System/370 DOS/VS users similar to those used on large systems. It serves two application areas: batch processing and online processing in conjunction with data communications.

In batch processing, data base transactions requested by applications are accumulated and then processed periodically against a data base. Because of this elapsed time, data in the data base is not always current. The use of batch processing should depend on how current your information must be, viewed in relation to the costs of other methods of processing data.

For data communication applications, DL/I provides an online processor with an interface to CICS/VS and employs existing user options and exits provided by CICS/VS, which is a transaction-oriented terminal management system. This type of system, as opposed to a batch system, responds to each transaction as it is requested. This method eliminates the elapsed time inherent in batch processing systems and allows you to maintain current data for your applications.

For a complete description of the CICS/VS system, refer to the *CICS/VS General Information Manual,* as listed in the Preface.

DL/I Batch System
The DL/I batch system (see Figure 2-1) contains the following functional parts:

- An initialization module
- A language interface module
- A program request handler
- The DL/I facility

Initialization Module
The initialization module allows you to input parameters to DL/I by two different methods:

- Through the normal system input device (SYSIPT)
- Through the system console (SYSLOG)

This module (step 1 in Figure 2-1) formats the parameters to the form used by DL/I and analyzes the parameters to determine:

- the program name
- whether the program is a utility or a DL/I batch program
- the program specification block name
- the amount of main storage required for the data base buffers
- whether asynchronous logging is wanted

- whether the log file is on a disk

- what trace facility, if any, is desired

The initialization module then loads the rest of the DL/I system, the user's application program, and the control blocks required to run the application program. It also formats the data base buffer pool and passes a pointer to the formatted parameter list stored within DL/I to the application program. Control is then passed from the initialization module to the application program at its program entry point (step 2).

Language Interface Module

The language interface module, which must be link edited with each application program, is entered when a data call is issued by an application program. The three programming languages supported by DL/I are COBOL, PL/I, and Assembler language. The language interface module identifies the language used for the call, translates the call to a common DL/I format and passes the call to the program request handler (step 3).

Program Request Handler

The program request handler accepts the formatted application program call from the language interface module, validates it, and passes it to the DL/I facility (step 4). Upon completion of the requested service by the DL/I facility, the program request handler passes data, if any, to the application program work area, then returns control to the application program (step 9).

DL/I Facility

The DL/I facility is the data management portion of the DL/I system. Through this facility, the application program inserts, retrieves, deletes, or replaces the data in the data bases used by the application program (steps 5 through 8). As these operations are performed, the DL/I facility performs all the data maintenance tasks required on the data bases. Optionally, all changes to the data base can be recorded on the DL/I system log. The Virtual Storage Access Method (VSAM) performs data management services using these DL/I access methods:

- Simple Hierarchical Indexed Sequential Access Method (Simple HISAM)

- Hierarchical Indexed Sequential Access Method (HISAM)

- Hierarchical Indexed Direct Access Method (HIDAM)

- Hierarchical Direct Access Method (HDAM)

The Sequential Access Method (SAM) performs data management services for data bases using these access methods:

- Simple Hierarchical Sequential Access Method (Simple HSAM)

- Hierarchical Sequential Access Method (HSAM)

For a description of the above mentioned DL/I access methods, refer to "Data Base Organization and Access Methods" in Chapter 3.
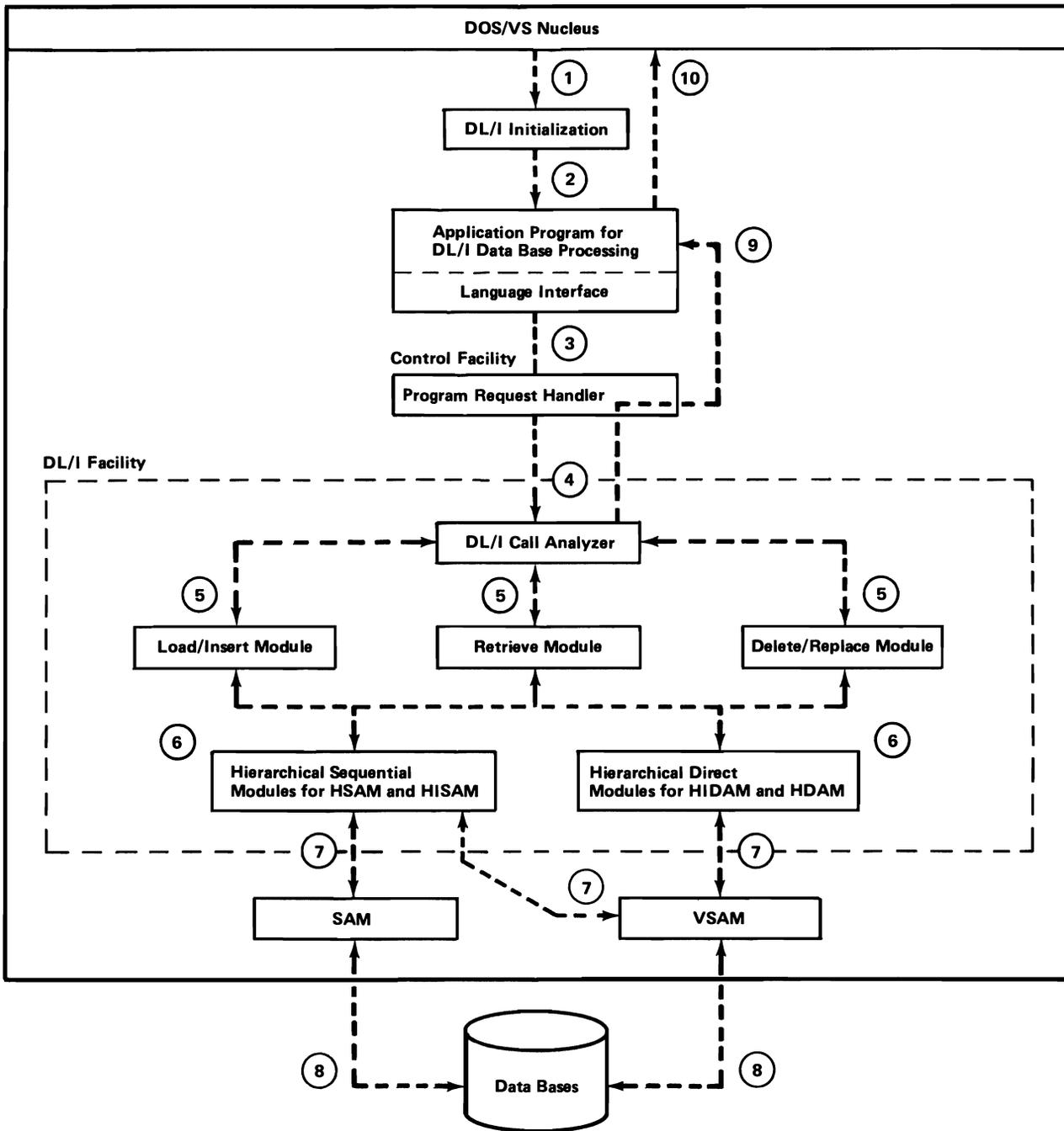
Figure 2-1. DL/I batch system

DL/I Online Processor

The DL/I system operating in a teleprocessing environment under CICS/VS contains all the functional parts listed for the batch system, plus a set of service routines called the DL/I online processor. These routines establish a connection between DL/I and the CICS/VS - DL/I interface.

In an online environment (see Figure 2-2), the DL/I system executes within the CICS/VS partition. CICS/VS provides exit interfaces to DL/I for the following:

- DL/I system initialization during CICS/VS initialization (step 1).

- DL/I system termination during CICS/VS termination (step 12).

- DL/I user task scheduling of DL/I resources before an application program accesses DL/I (step 3).

- DL/I user task completion and return of DL/I resources after the application program has completed DL/I processing (step 11).

| CICS/VS Partition |
| Online DL/I Processor |
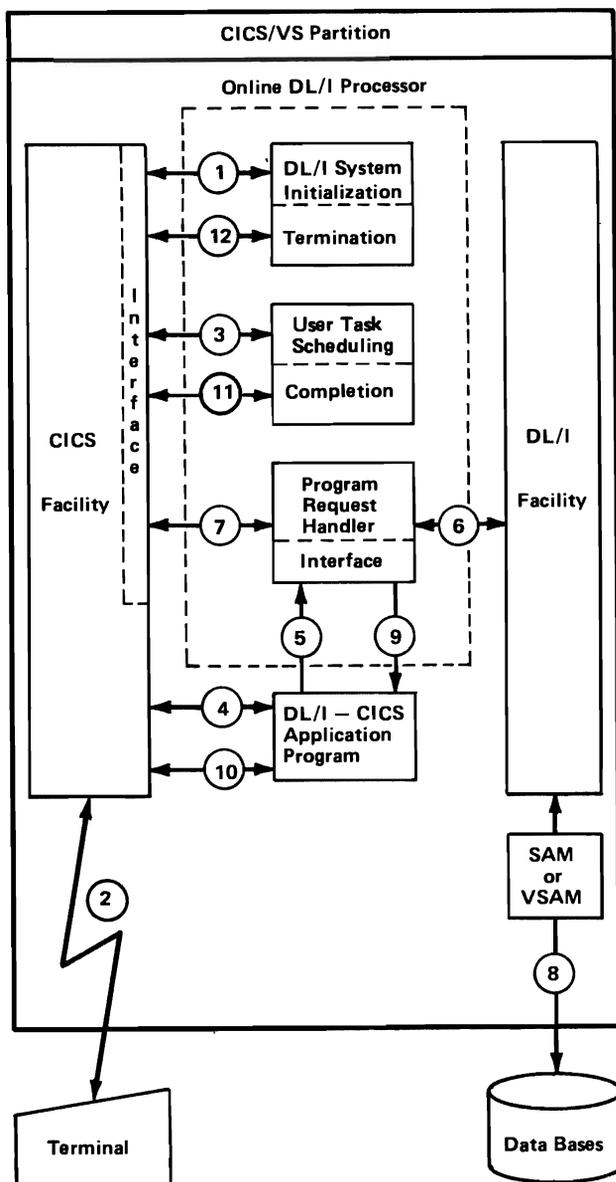
Figure 2-2. DL/I - CICS/VS data communication system

When the user application program issues a DL/I call, control passes to the online language interface module (step 5) and the program request handler. The program request handler validates the call and passes it to the DL/I facility (step 6).

The DL/I facility invokes CICS/VS services through the online interface for such functions as transaction and storage management. On completion of the DL/I call, the DL/I facility returns control to the CICS/VS application program via the program request handler (step 7). The program request handler also interfaces with CICS/VS for any functions performed external to DL/I.

Multiple Partition Support (MPS)

DL/I has the capability to enable application programs executing in different partitions to access the same data base concurrently. This capability, multiple partition support (MPS), permits, for example, online applications to issue inquiries to a data base while a batch program updates it.

Although two programs cannot both update the same data base segment type concurrently, one or more programs can retrieve from it while another program updates the segment type.

Note: The above limitation does not apply to DL/I Version 1.3. See "Program Isolation" in the section, "DL/I Version 1.3 Enhancements."

The addition of MPS enables multiple batch and online application programs to access a data base concurrently instead of serially. MPS uses the DL/I resources and the multitasking facilities of DL/I and CICS/VS. CICS/VS therefore, is a prerequisite for running MPS. MPS follows normal DL/I online conventions.

Note: MPS does not invalidate the restrictions that normally apply to DOS DASD sharing between CPUs.

Utility Programs and Procedures

The following utility support programs and procedures are provided with DL/I.

Program Specification Block (PSB) Generation

The program specification block generation procedure creates the control blocks that define to DL/I the data bases used, the data within each data base, and the operations allowed on each data base by a particular application program.

Data Base Description (DBD) Generation

The data base description generation procedure creates the control blocks that define to DL/I for each data base the data base name, its data structure, its data format, and the DL/I access method used.

Application Control Blocks Creation and Maintenance Utility

Before the program and data base descriptions created by the PSB and DBD generation procedures can be used, they must be merged and expanded to an internal format. Instruction execution and direct access wait time is minimized by prebuilding the required application control blocks by means of the application control blocks creation and maintenance utility. Then, when an application program is to be run, its control blocks are read in directly (if they are not already in main storage), and control is passed to the application program.

Data Base Reorganization Unload and Reload

When the data in a data base is updated, the physical structure of the data may change, increasing access time. Also the space occupied by obsolete data is not in all cases reclaimed and reused. The data base reorganization unload and reload utilities may be used to unload, reorganize, then reload simple HISAM, HISAM, HIDAM, and HDAM data bases to eliminate these problems.

In addition, the HD Reorganization Reload Utility also provides a reload restart facility. If a job is canceled or fails, this facility enables you to restart the reloading of a data base without going back to the beginning. While running the HD Data Base Reorganization Unload Utility, a checkpoint record is written to the unloaded data base after approximately every 5000 segments. These checkpoint records can be used to restart the reloading of the data base.

During the normal reloading process, the checkpoint records are ignored except for a console message containing the checkpoint number. This message is printed each time a checkpoint is encountered. When a job is canceled or fails, the system operator should:

1. Run the Access Method Services Utility, executing the VERIFY command for each data base being referenced during the reload.

2. Resubmit the job with a unique parameter card, identifying the job as a restart of the Reload Utility.

3. Supply the last checkpoint number when prompted by the Reload Utility.

Data Base Recovery

The data base recovery system comprises five utility programs and is designed to provide a rapid, accurate, and easy-to-employ means of restoring the contents of a physical data base after destruction.

Data Base Data Set Image Copy Utility

The data base data set image copy to utility dumps individual files of a data base to tape or disk in a format suitable for use by the data base data set recovery utility.

Data Base Change Accumulation Utility

The data base change accumulation utility sorts records from the data base log tape and combines all records that update the same segment. The result is a sequential file that contains a condensed description of all changes to the data bases.

Data Base Data Set Recovery Utility

The data base data set recovery utility reconstructs individual files of a data base by first obtaining from the data base image copy file an image of the file at a time at which it is known to be valid and then merging the accumulated application data from the data base change accumulation utility. Finally, any DL/I system log tapes that were not included in the accumulated change input are applied until the file contains the desired data.

Data Base Backout Utility

The data base backout utility reads the DL/I system log tape backward and removes (backs out) changes made to the data base from the point at which the DL/I system abnormally terminated to the most recently written checkpoint (Version 1.3), or to the point at which the program started processing. After backout, the status of the data base is the same as if the job or transaction were never executed. This utility also creates a log tape that reflects the backed out changes.

Log Print Utility (Version 1.3)

The log print utility enables you to print the contents of the log files. This utility can be invoked when an abnormal termination or system failure occurs.

The information contained in the logs selected for printing includes the log record ID, the task ID of the log record, the PSBname, the DMBname, the type of call, the record type, and the data in the record. After backout, the status of the data base is the same as if the job or transaction were never executed.

Application Support Program

Low-Level Code and Continuity Check

DL/I includes an application support program to initially generate and update the 'low-level code'. This code specifies hierarchical levels within a structure of root segments connected to each other by logical relationships. This function facilitates processing bills of material in a manufacturing environment and is known in this industry as 'low-level code'. Low-level coding is completed by a continuity check that prevents an item from being contained in its own structure.

Techniques of low-level coding are also useful to solve many non-manufacturing problems in cases where the problems can be expressed as networks or as directed graphs.

The support program is a callable subroutine that becomes a part of a user-written application program.

More information about low-level code and continuity check can be found in the publication, *IBM System/370 Low-Level Code/Continuity Check in Data Language/I DOS/VS, Program Reference and Operations Manual,* SH20-9046.

Chapter 3. DL/I System Concepts

While a general description of DL/I has been given, there are numerous technical considerations that require additional discussion. This chapter discusses those considerations that are of particular interest to personnel responsible for planning the use of DL/I.

DL/I operates as an application program in virtual storage under DOS/VS. DL/I consists of the control facility and the data base processing portion of the system, referred to earlier as the DL/I facility. The DL/I system also provides the interfaces necessary to permit DL/I to operate as part of the CICS/VS teleprocessing environment.

DL/I Data Base Structure

The DL/I data base concept allows user's data and programs to be independent from the access methods and storage organizations chosen by the data base administrator. The function of the data base administrator is to provide a single source for data and program requirements.

Note: The duties of the data base administrator are defined later in this chapter.

The application program interface to the data in the data base is a common symbolic language. The application program is unaware of the particular storage organization, storage device, and access method chosen for any data base. Nor is the program aware of any pointers that might be used in the physical storage organization.

One way to illustrate the structure of a DL/I data base is to take a look at a typical programming application and compare the methods used to handle it (that is, data base vs. non-data-base). Figure 3-1 shows a sequentially organized employee master file on disk storage. This file, containing information about each employee, is used to process a payroll application of 30 programs.

Its contents are typical. The control information includes employee number, name, department, and date hired. The address can be up to four lines. Payroll information includes salary and year-to-date tax data, with provisions made for five deductions.

In a non-data-base DOS/VS installation, each of the 30 programs accessing that file contains a description of its physical attributes, and a description of the data within the record. In the DL/I

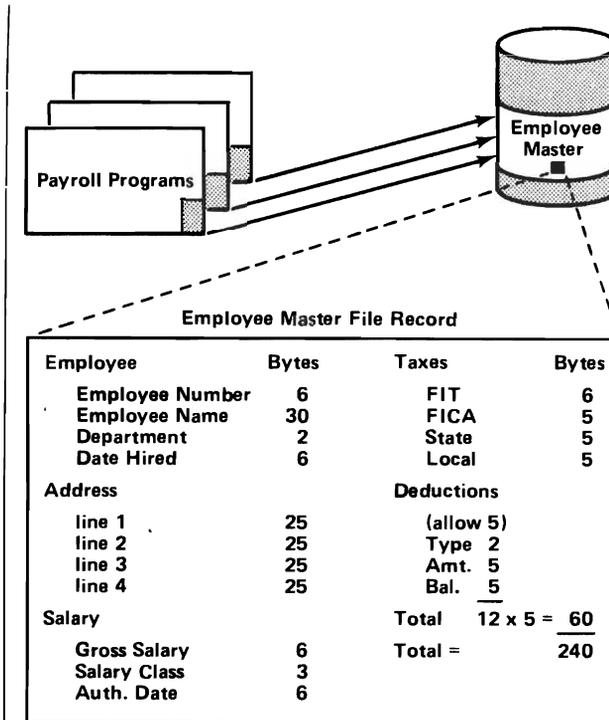| Employee Master File Record			
Employee	Bytes	Taxes	Bytes
Employee Number	6	FIT	6
Employee Name	30	FICA	5
Department	2	State	5
Date Hired	6	Local	5
Address		Deductions	
line 1	25	(allow 5)	
line 2	25	Type 2	
line 3	25	Amt. 5	
line 4	25	Bal. 5	
Salary		Total 12 x 5 = 60	
Gross Salary	6	Total = 240	
Salary Class	3		
Auth. Date	6		

Figure 3-1. Sample payroll application

DOS/VS installation, the data is defined independently of your application programs, thus providing the basis for data independence. The section, "Data Base Definition," in this chapter explains this concept in detail.

Hierarchical Data Structure

In the DL/I data base structure, data is represented as a hierarchical organizational structure, where certain information in a data base is related to other subordinate types of information in a hierarchical manner. This hierarchical data structure is called a *logical data structure*.

Application programs written to use DL/I deal with logical data structures. *Logical* refers to the manner in which the application program *sees* the data. A logical data structure is always a hierarchical structure of data elements called segments. Programs written to process logical data structures can be independent of the physical data structure. *Physical* refers to the manner in which data is stored on a tape or a direct access device. An application program in DL/I never deals directly with a physical data structure. Most data process-

ing information, regardless of industry, can be viewed as a logical data structure. Figure 3-2 shows the employee data base as it could be described in DL/I.
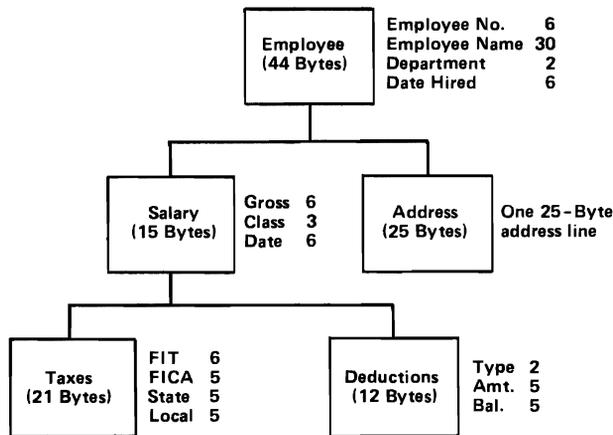

Figure 3-2. DL/I logical data base structure (hierarchical data structure)

Segments

Each box shown in Figure 3-2 is known in DL/I terms as a *segment*. In this case, the structure depicts five different types of segments (Employee, Salary, Address, Taxes, and Deductions). Each segment consists of one or more data fields. The Employee segment, for example, has the employee number, name, department number, and date of hire. In this example, the Employee segment contains the identifying information for the data structure (name and number), and it is called the *root segment. There is only one root segment in a data hierarchy.* All others are dependent segments. The Salary and Address segments relate to the Employee segment, and depend on the Employee segment for their full meaning (Whose salary? Whose address?).

Each type of segment can vary in length (Employee is 44 bytes, Salary is 15 bytes, etc.), and segments of the same type may also vary in length. In this example all segments of a specific type in the data base, such as Employee, are the same length. Figure 3-2 shows one of the four occurrences of the address segment; one 25-byte segment for each line of the address.

There can also be many levels of dependency. For example, the Tax and Deduction segments are subordinate to and depend on the Salary segment. The Salary segment, in turn, depends on Employee, the root segment.

Segment Relationships

The basic building element of a hierarchical data structure is the *parent/child relationship* between segments of data. In Figure 3-3, the root segment (Employee) is also the *parent* of all the segments (that is, Salary and Address) that depend on it. The dependent segments, Salary and Address, are called *children* of the parent segment (Employee).

The same relationship exists down the structure (Salary is the parent of Taxes and Deductions; Taxes and Deductions are children of Salary).

Each *occurrence* (or instance) of a *parent segment* has associated with it any number of (0, 1, 2, ...) *occurrences of a child segment* type. Note the distinction between a *segment type* (the kind of segment), and the *segment occurrence* (the segment and its particular contents and location). In Figure 3-3, the parent segment, Employee, has four occurrences of the child segment type, Address. Each child segment type has associated with it 1 occurrence of a parent segment.

As shown in Figure 3-3, a parent (Employee) can have several child segment types (Salary and Address). Also, a child segment (Salary) can at the same time, be a parent segment, that is, have children itself (Taxes and Deductions). The segment with no parent (Employee), the one at the top, is called the *root segment*.

All the parent/child occurrences for a given root segment are grouped together to form a *DL/I data base record.* In this example, although the logical data structure contains 5 *segment types,* the logical data base record actually contains 8 *segments* because of the multiple occurrences of the address segment. Data base records for a given logical data structure may vary in size according to the number of occurrences of a given segment. The collection of all these like-data-base-records is a *DL/I data base.*

Since each dependent segment in the hierarchy has only one parent (immediate superior segment) the hierarchical data structure is sometimes called a *tree* structure. Each branch of the tree is called a *hierarchical path.* A hierarchical path to a segment contains all consecutive segments from the top of the structure down to that segment.

DL/I allows a wide variety of hierarchical data structures. The maximum number of different segment types is 255 per hierarchical data structure. A maximum of 15 segments levels can be defined in a hierarchical data structure. There is no restriction on the number of occurrences of

each segment type, except as imposed by the physical access method limits of your DOS/VS system.

Segment Sensitivity

The significance of the hierarchical structure is important to application programmers. In DL/I, they no longer see a physical record. They see a series of segments in a parent/child relationship. They request segments of data, rather than physical records. The concept of segment *sensitivity* allows a program to be restricted to *seeing* only those segments of information that are relevent to the processing being performed. For example, an application program could be written to see only the Employee and Address segments of the data base record shown in Figure 3-2. The program need not be aware of the existence of the Salary segment and its children.

Basic Segment Types in a Hierarchical Data Structure

The following describes the several segment types and their interrelations within a hierarchical data structure. Refer to Figure 3-4:
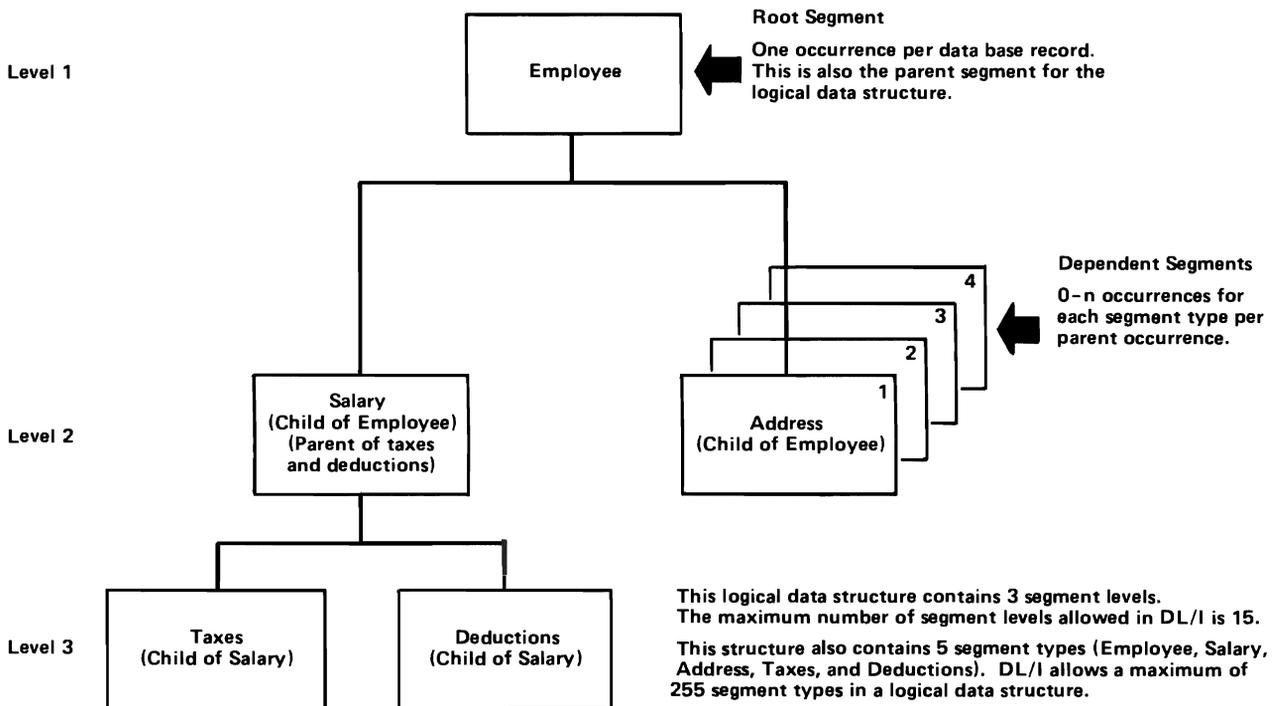
- The segment on top of the structure is the *root segment*. Each root segment normally has a *key field* (also called the sequence field) which serves as the unique identifier of that root segment, and as such, of that particular data base record. The key field in this case is the employee name.

- A *dependent segment* relies on some higher level segment for its full meaning and identification.

- A *parent/child* relationship exists between a segment and its immediate dependents.

- Different occurrences of a particular segment type under the same parent are called *twins*.

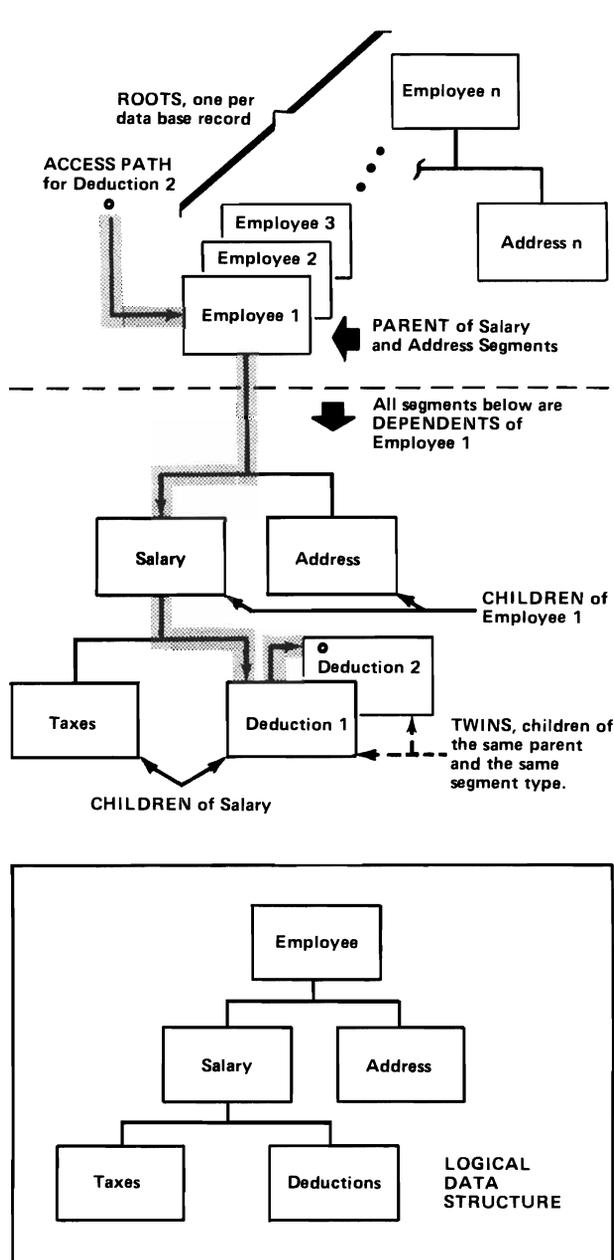Figure 3-3. Logical data structure -- The programmer's view

Figure 3-4. Segment types and their relationships in a hierarchical data structure

Definitions

Based on the previous discussion of the structure of the DL/I data base, following definitions apply:

- Segment. A data element containing one or more logically related data fields. A segment is the basic data element that interfaces between the application program and DL/I and upon which the user defines sensitivity. A segment may be fixed length or, under certain conditions, variable length.

- Sensitivity. A means by which the user defines which subset of the data within the data base can be accessed by an application program and what operations may be made against that subset of the data base. This defines a logical view of a data base.

- Logical data base record. A set of hierarchically related segments of one or more segment types. Each segment type may have a unique length and format. As viewed by the application program, the logical data base record is always a hierarchical tree structure of segments.

- Logical data base. The major unit of data storage under DL/I - a set of logical data base records stored in the DL/I organization and for any one of the DL/I access methods.

Logical Data Structures/Physical Data Structures

Application programs written to use DL/I deal with *logical data structures*. This refers to the manner in which the application programs *sees* or *views* the data. A DL/I logical data structure consists of one or more *hierarchical data structures(s)*. Programs written to process these data structures can be independent of the *physical data structure*. *Physical* refers to the manner in which the data is stored on a direct access storage device or tape. A DL/I application program never deals directly with a physical data structure.

The data described by the DL/I data base structure in Figure 3-2 is physically stored as shown in Figure 3-5. Employee, the root segment, points to its children (Salary and Address). Salary also points to its children (Taxes and Deductions). The multiple occurrences of segments are also linked. It is through these pointers that DL/I retrieves segments of data. The application programmer, however, need not be aware of the physical storage.

Physical storage is accomplished through the use of two unique DL/I storage organizations: hierarchical sequential and hierarchical direct. Six DL/I access methods -- simple HSAM, HSAM, simple HISAM, HISAM, HIDAM, and HDAM, as well as a multiple indexing facility for HIDAM and HDAM -- are provided to allow access to these organizations.

These storage organizations and access methods are discussed under "Data Base Organization and

Access Methods" in this chapter. The application program interface with these organization types and access methods is totally symbolic.
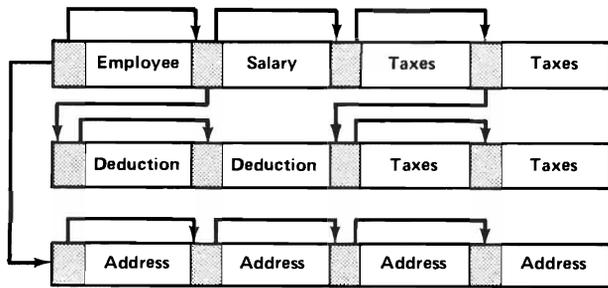

Figure 3-5. Physical data structure

Sequence Fields and Access Paths

To identify and to provide access to a particular data base record and its segments, DL/I uses *sequence fields*. Each segment normally has one of its fields denoted as the sequence field. However, not every segment type need have a sequence field defined. Particularly important is the sequence field for the root segment, since it serves as the identification for the data base record. Normally, DL/I provides a fast, direct *access path* to the root segment of the data base record based on this sequence field. This direct access is extended to lower level segments if the sequence fields of the segments along the hierarchical path are specified too.

Note: The sequence field is often referred to as the *key field*, or simply, *key*.

Figure 3-4 shows as a shaded line, an example of an access path. It must always start with the root segment. This is the access path as used by DL/I. The application program, however, can directly request a particular Deduction segment of a given Salary of a given Employee in one DL/I request, by specifying a sequence field for all three segment levels.

Data Base Definition

The entire hierarchical structure (names of segments, sizes of segments, hierarchy) as well as its physical attributes (the fact that it is on disk, organized sequentially, blocked 5) is kept in two tables external to the programs in the DL/I system itself. These tables are created and maintained by a specific individual in your installation called the data base administrator.

Note: See "Data Base Administration" in this chapter for more information about the data base administrator.

Because both tables are created and maintained independently of your application program(s), they provide the basis for data independence.

DBD (Data Base Description)

The first table is called the DBD (data base description). It describes most of the file characteristics you must put into every non-data base DOS/VS program. Each DBD is created from statements you provide. The statements define the hierarchical data structure and physical organization of the data base. These statements are assembled as the data base description generation procedure.

The DBD contains a description of the contents of the data base, the names of the segments, their hierarchical relationship, and the physical organization and characteristics of the file. You can think of the DBD as the master description of everything that is in the data base.

The DBD provides DL/I with the mapping from the application data structure of the data base used in the application program to the physical organization of the data used by DOS/VS. The data structure can be remapped into a different physical organization without application program modification. Other application data can also be added to this data base and not require a change to the original application programs. The concept of the DBD reduces application program maintenance caused by changes in the data requirements of the application. The three types of DBDs are:

- The *physical DBD,* which provides the definition of a single hierarchical structure. It can be used, in this form, by application programs. If logical relationships exist, the physical DBD contains a definition of these relationships with the other hierarchical structure. These relationships can be within the same DBD or with another DBD. Multiple logical relationships can exist within a single physical DBD.

- The *logical DBD,* which provides the redefinition of two or more related hierarchical structures into a new hierarchical structure. These hierarchical structures can be from the same or different DBDs. The logical DBD relies on the logical relationships that were defined in the physical DBD(s).

Note: More information on logical relationships is included in this chapter under the heading, "Logical Relationships".

• The *secondary index DBD,* which allows the definition of a secondary access path into a physical or. logical DBD.

Note: Secondary indexing is described later in this chapter.

The process of generating a DBD is called *data base description generation* DBDGEN).

PSB (Program Specification Block)

The other table is called the PSB (program specification block). The PSB defines the application data structure for each application program. It is created from statements you provide for each of your application programs. The PSB defines which segments of the data base a specific program requires (the application data structure required by that application program). A PSB contains one or more *PCBs* (program communication blocks), one for each hierarchical data structure the program intends to use. Each PCB defines the hierarchical (sub)structure the program *sees* from the physical or logical data base. It specifies for each segment the kind of access allowed by the program (read only, update, insert, load, and delete). There is at least one PSB for every program that uses the data; more than one program may use the same PSB. You can think of the PSB as describing the logical data needed for the program (usually a subset of the entire data base). The process of generating a PSB is called *program specification block generation* PSBGEN).

DL/I Control Blocks

Before the program and data base descriptions created by the PSB and DBD generation procedures can be used, they must be merged and expanded to an internal format.

DL/I provides a utility that creates a DMB (data management control block) for each related DBD CSECT and an expanded PSB for each related PSB CSECT. When DL/I is initialized, the DMBs and PSBs for the applications program are loaded into storage and control is passed to the application program.

DL/I, the Application Program Interface

Figure 3-6 shows that at execution time, DL/I uses the combination of the DBD describing the file, and the PSB describing the data needs of a specific program to satisfy the requests of the application programs. DL/I acts as an intermediary between the application program and the data itself.
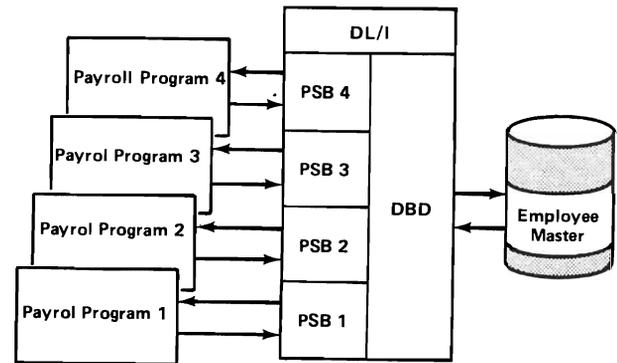

Figure 3-6. DL/I is the intermediary between the application program and the master file.

When a program needs a segment of information, it issues instructions to DL/I using a standard request format. DL/I, using the PSB and DMB, interprets the request. When necessary, DL/I issues the appropriate I/O command to obtain data from the data base and return the requested segment to the program.

Some benefits of this concept for typical situations that might arise in your installation are:

File expansion. Assume the personnel department wants to maintain information on the military service of each active employee. This information must be added to the existing master record for each employee, because the original design of the record did not include this data.

In our example of a non-data base DOS/VS environment, this would require modification to the record description in each of the 30 payroll programs, and a recompilation and retest of each program.

However, with DL/I, the military information can simply be added as additional segments to the employee data base without affecting existing payroll programs at all (Figure 3-7). The payroll programs continue to be supplied with the segments they need and are not aware of the expansion of the data base. Thus, DL/I reduces the maintenance impact of record changes in your installation and simplifies the expansion of files.
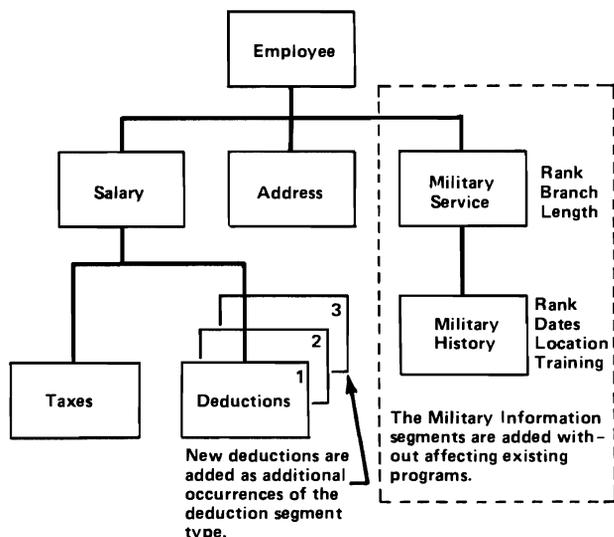
Figure 3-7. DL/I simplifies file expansion

Elimination of variable length records. Suppose the payroll department finds that it needs to allow more than five deductions for one employee. Again in our example of a non-data-base DOS/VS environment, you have to expand the size of each employee's record to allow more deductions, or you have to change the record format to variable length. Either way requires that you modify, recompile, and retest the application programs.

With DL/I, no application program changes are necessary. The new deductions are added to the employee data base as additional occurrences of the deduction segment type (Figure 3-7). The application program simply processes the Deduction segments until there aren't any more. This leaves the variability open-ended, and simplifies the programming associated with this variability.

DL/I, therefore, reduces the complex systems design and programming that exist today because variable length records are no longer required.

Data security. With DL/I, confidential information in the data base can be restricted to only the programs that require it. The segments that a specific application program is permitted to access are defined in the PSB, external to the program itself. If the segment is not named in a program's PSB, it cannot be requested. This allows centralized control over who can access which portion of the data base. This concept is called segment *sensitivity*. Each application program is sensitive to only the segments it needs.

In addition to restricting application programs to the segments they can access, the processing options of each application program can also be restricted. This prevents erroneous destruction of the data base by improper updating. The updating can be limited to a program or two, and the remaining programs may be permitted only to retrieve information. As you can see in Figure 3-8, the programs PSB defines which functions it is permitted to perform on the data base. Programs 2 and 5 may update the data base: all others may only retrieve information.

Because DL/I allows your installation to control who can access segments, and what processing they are permitted to do, your data base is protected against erroneous updating, and confidential data is restricted to those with a need to know.
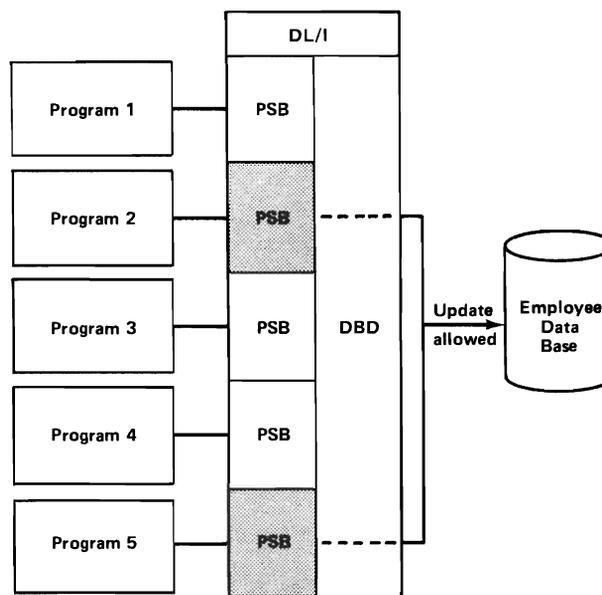

Figure 3-8. The application program's PSB defines program processing options. In this example, programs 2 and 5 have update capabilities.

Device independence. Suppose your installation wants to change to a new disk device. For example, a new disk becomes available that has the capacity to greatly expand your capability to maintain data online. Since several files in your installation have a need for online availability, you decide to convert to these new disks.

In the non-data-base environment, this conversion will again probably require a modification, recompilation, and retest of every program using the data just to describe the new device and change blocking factors.

With DL/I, since the physical characteristics of the data base are maintained external to the application programs in the DBD, all programs are shielded from these changes and become device-independent. Changes from one disk device to another, or changes to the way data is physically stored, do not affect the application program. Figure 3-9 illustrates that with DL/I, all changes required for new device support are made through the use of DL/I utility programs and alterations to the DBD.
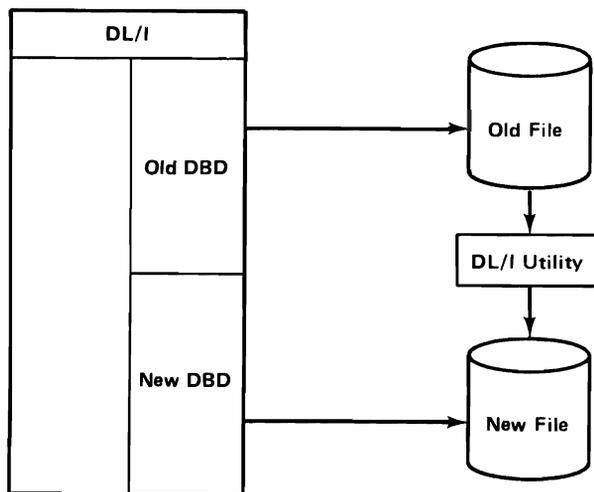

Figure 3-9. DL/I provides device independence. To change your files to a new disk device, simply change the DBD and use the DL/I utility programs to load the new disk.

In addition to changing the physical device, you can change access methods as well. Assume the payroll programs were originally written for batch-only operations, using sequential access methods. However, a need develops to inquire randomly into this data base, and this requires a change in access methods.

This change can be made in DL/I in the same way as changing a device. Simply alter the DBD and use the DL/I utility programs to unload and reload the file. The application programs will not be aware that the access method has changed.

By providing this device independence, DL/I gives you the added flexibility to take advantage of new I/O devices without impacting your existing program investment. You can also change access methods to suit the processing needs of the application.

Reduced data redundancy.   What if your installation wants to develop a new system involving job incentives for manufacturing employees? In the non-data-base environment you must either merge the new information required in the existing employee master file (and force a modification of all programs using it), or you must create another file. You already know the problems involved in expanding the file, so let's look at the alternative. You decide to construct a separate file for the job incentives system, using pertinent control information from the employee master file. This is probably a better alternative than changing a lot of application programs. The problems arise, however, as changes, additions, and deletions occur. You now have two files to be updated with the possibility of developing inconsistent data (one file is not updated, while the other one is). Tight controls must be instituted in data maintenance, and you will probably have a time lag between updates. Some changes may not reach both files.

With DL/I, you no longer need to create separate files. The file may be expanded as your needs grow. The important benefit of this approach is that once a change is made to a data base record, it is effective for all programs using the data base. By providing one data source for all programs, DL/I assists you in reducing or eliminating data redundancy and keeping data consistent.

Increased programmer productivity.   Your installation probably has at least one application (similar to Figure 3-10) where it was necessary to create four or five interrelated files and a large complex program to handle the interactions necessary to manipulate them.

With DL/I, this is no longer necessary. You can simply expand the data base to allow a wider range of information processing to be accomplished. Figure 3-11 shows that no matter how complex the data base may become, to application programmers it remains a simple, logical series of segments, because they are dealing with only that portion of the full data base needed for the application. Very few application programmers see a data base this large. They actually see only the portion of the data base they need for their particular application.

Data base coding is standardized and simplified because application programmers no longer have to spend time describing the data and the environment to the operating system as in the non-data-base DOS/VS environment.
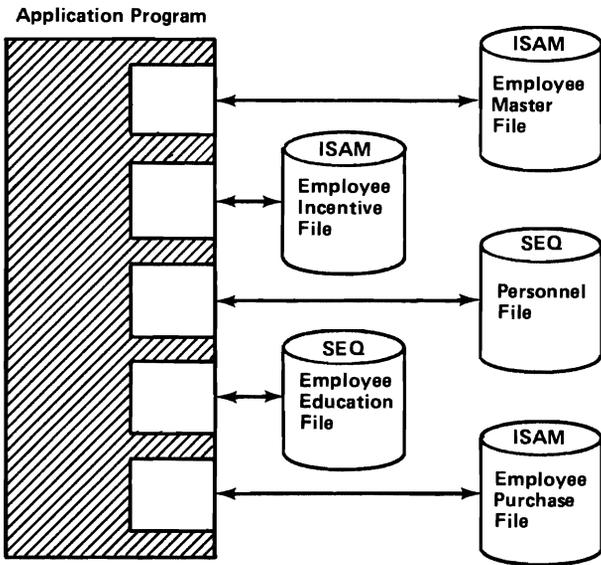
Application Program

Figure 3-10. In DL/I, large complex application programs used to process the data of four or five interrelated files are not needed.
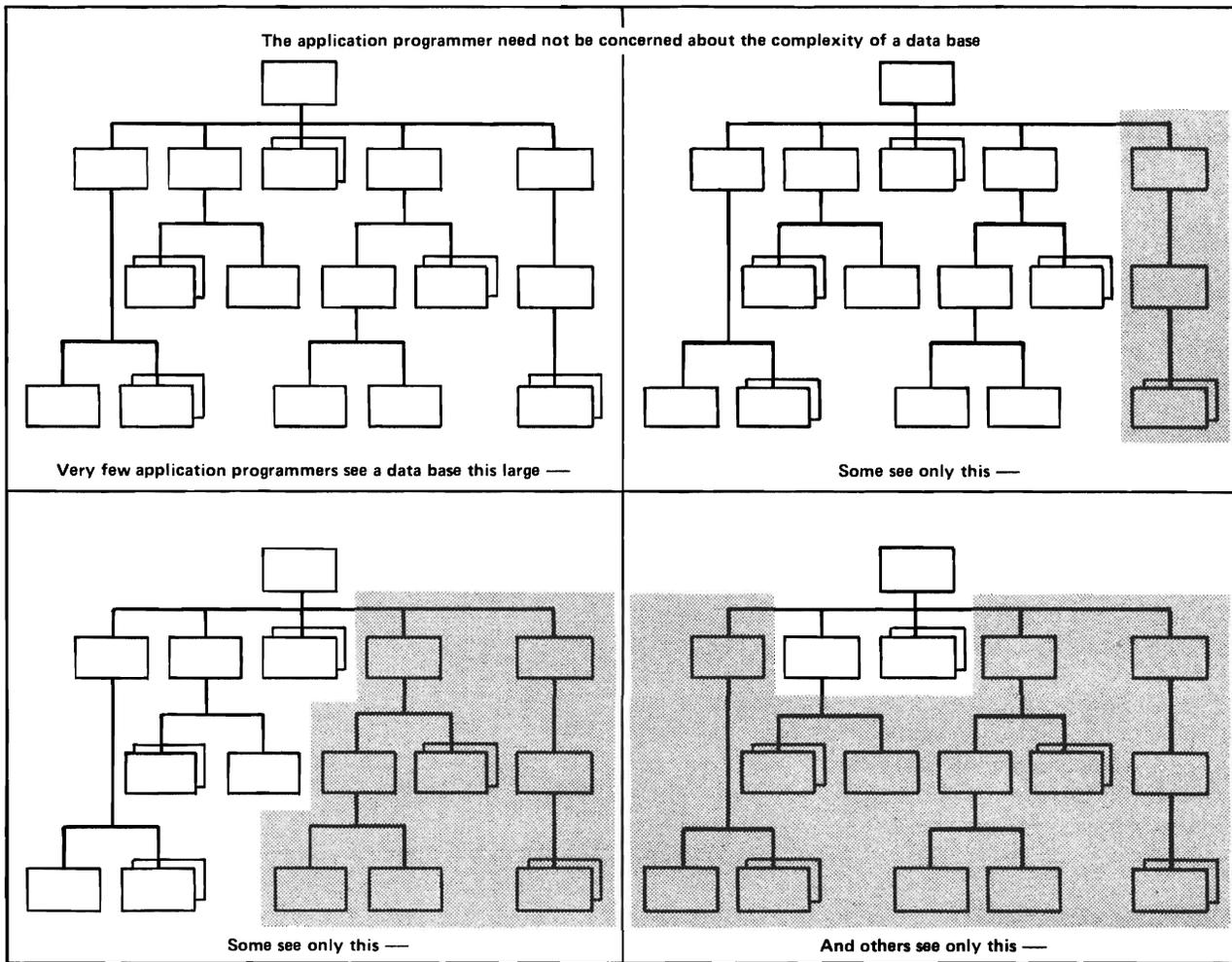

The application programmer need not be concerned about the complexity of a data base

Very few application programmers see a data base this large —

Some see only this —

Some see only this —

And others see only this —

Figure 3-11. The application programmer need not be concerned with the complexity of a data base.

Logical Relationships

In addition to the basic DL/I facilities discussed so far, DL/I provides a facility for HDAM and HIDAM data bases to interrelate segments from different hierarchies. For example, suppose that one of the requirements of the job incentives system described previously is to get information on the skills of employees (including a description of each skill), and job-standards data. This portion of our employee data base example is shown in Figure 3-12. Let's also assume that another application, Skills Inventory, also exists whose data base is as shown in Figure 3-13. As you can see in Figure 3-13, the data that is needed by the Job Incentive System is contained in three segments of the Skill data base: Skill, Description, and Standards. But how do you get it?
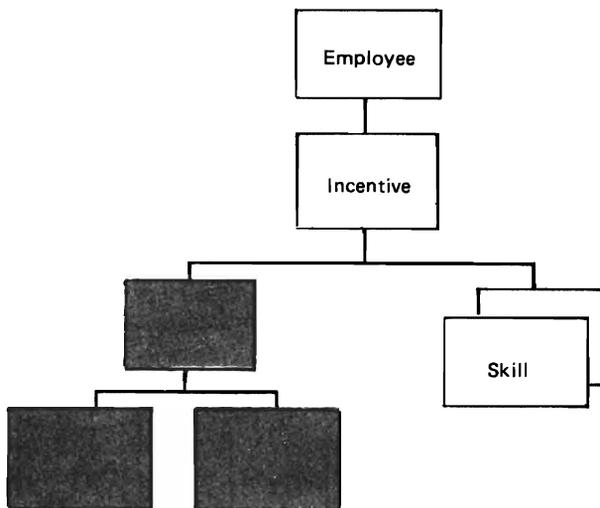
Figure 3-12. Job incentive system as included in the Employee data base

One way might be to copy the Skill data into the record of each employee who has that skill.
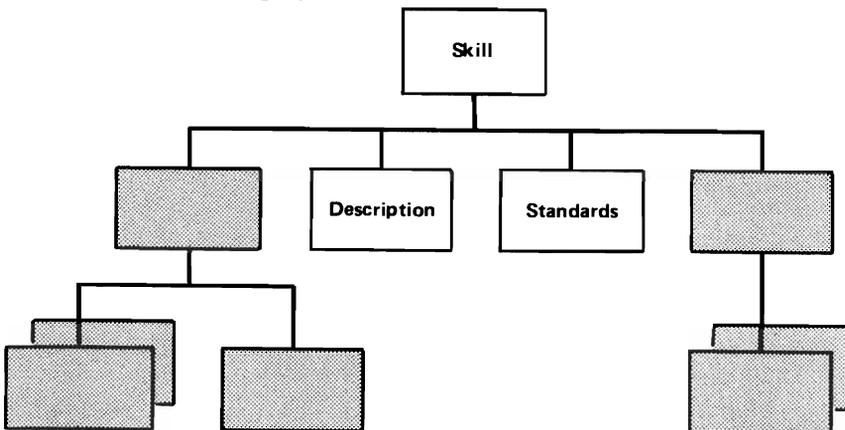
This does away with the need to interrelate the Employee and Skill data bases, but it creates redundant data. For example, several employees could have the same skill, and this data would be repeated over and over again.

Another approach might be to replace the Skill segment in the Employee data base with a pointer segment that directly addresses the Skill segment. Let's call this segment, Skill Number. Now when the information is needed, the programmer accesses the Skill data base using the pointer in the Skill Number segment. Under this arrangement, the Skill Number segment is called the pointer segment and the Skill segment is called the target segment. Of course the data in the two data bases and the relationship between the two must be kept up to date.

With DL/I, the programmer doesn't have to worry about maintaining this relationship. This is because, once the logical connections between segments have been specified (in the DBD), DL/I maintains the relationship between the two data bases automatically. This capability, known as *logical relationship,s* provides direct pointers as shown in Figure 3-14. As changes are made to the data bases, the pointers are automatically updated. The logical relationship between the two data bases is actually viewed by the programmer as one logical data structure as shown in Figure 3-15. So, when a programmer asks for a Skill segment for an employee, DL/I retrieves the right segment from the Skill data base and the application programmer doesn't even have to be aware that this data base exists.

Because the pointer-target segment relationship shown in Figure 3-14 is in one direction, this is called a unidirectional logical relationship.
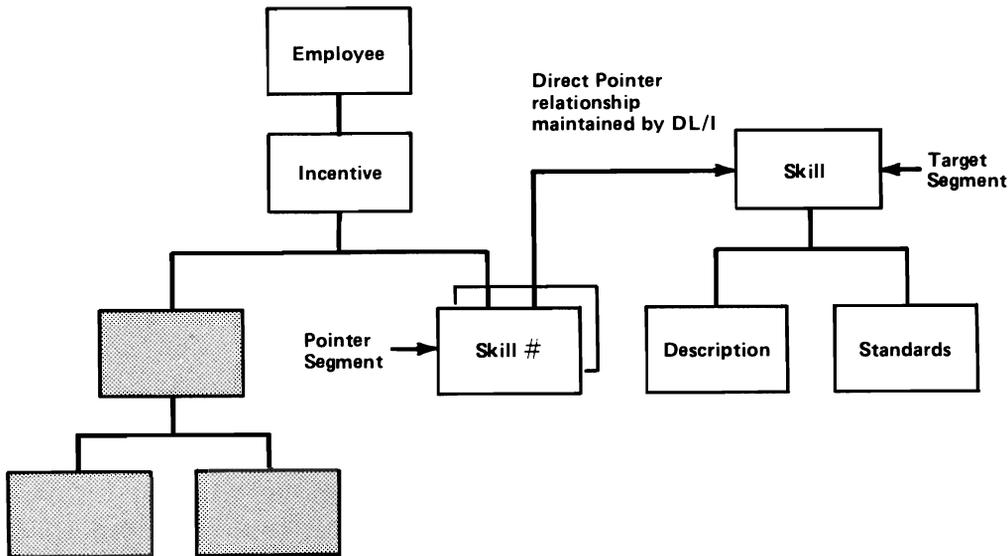

Figure 3-13. Skills Inventory data base

Figure 3-14. Through the capability of logical relationships, DL/I automatically maintains the relationship between the Employee data base and the Skills Inventory data base.
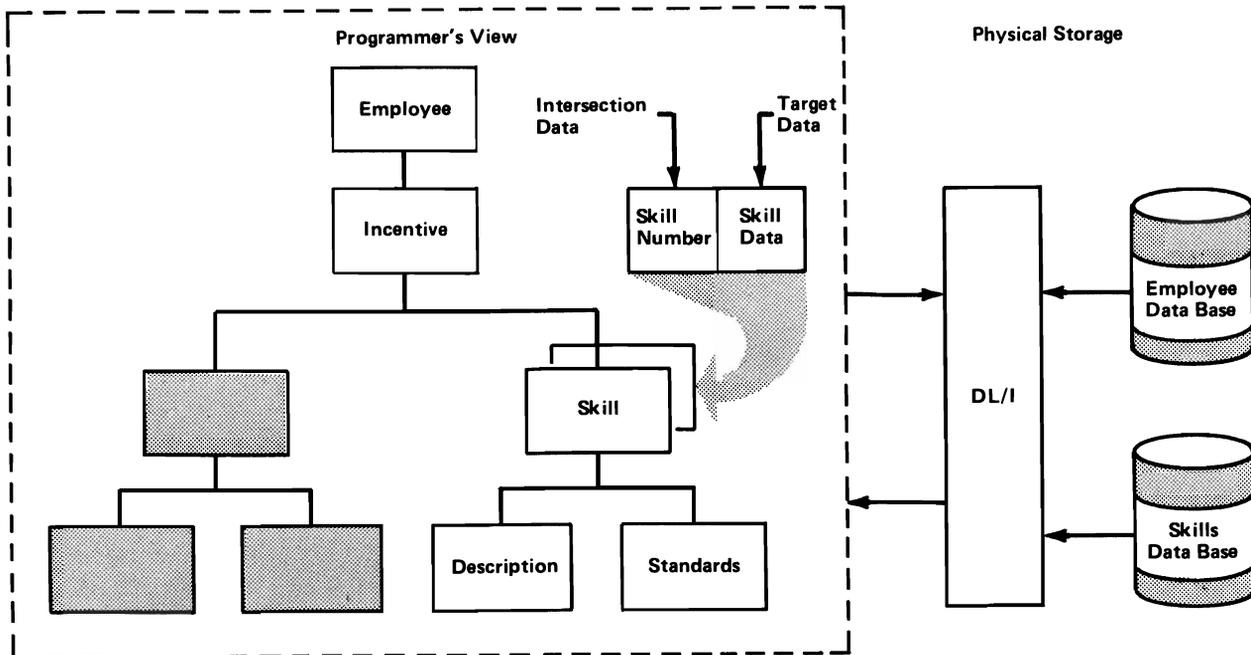

Figure 3-15. The logical relationship between two data bases is seen as one logical data structure.

Bidirectional logical relationships are also possible in DL/I. Suppose you wanted to analyze the performance of employees who shared a specific skill? By allowing the target segment (Skill) of Figure 3-14 to point backwards through the pointer segment (Skill Number), you can use the same logical relationship to give you a second different logical data structure as shown in Figure 3-16. Note that the Employee segment - Skill segment relationship, as shown in Figure 3-15, has now been inverted to allow access to a specific Skill segment, and from there, reaching all employee names related to that skill. This is called an *inverted data structure*.
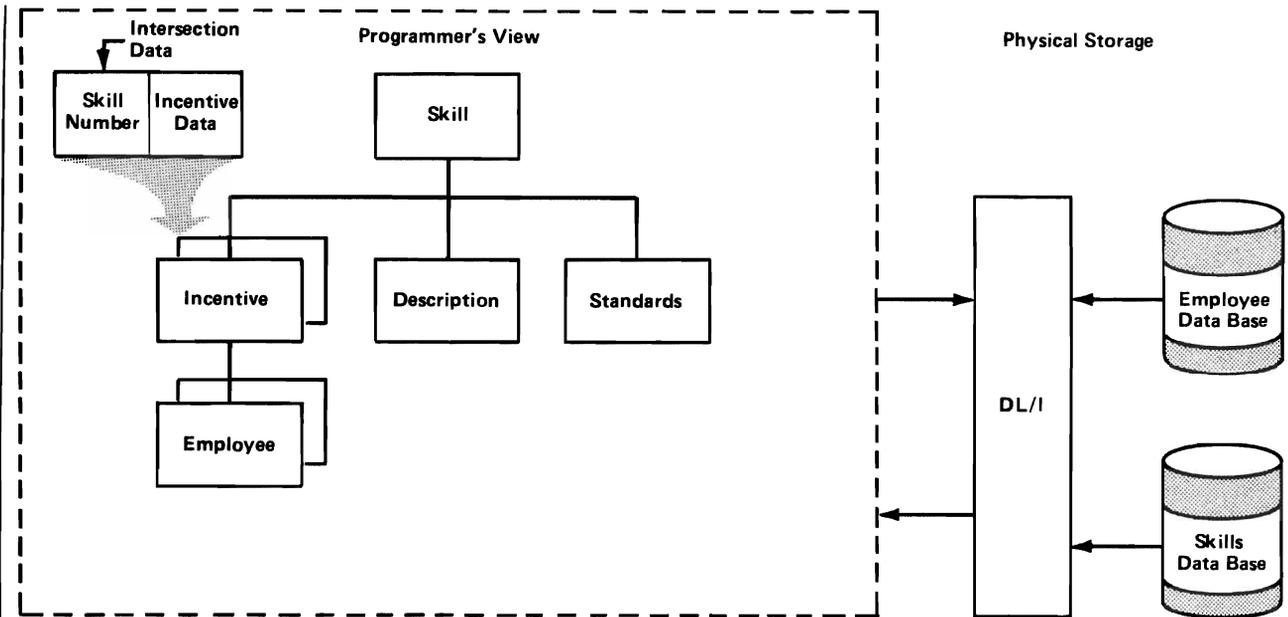
Figure 3-16. The same logical relationship developed for the logical data structure in Figure 3-15 can be used to design different logical data structures for other applications.

This feature gives you flexibility in application design, because you can look at data in a way that is natural and logical to the application program regardless of where the data actually resides.

Intersection Data

In addition to having a pointer, a pointer segment may also contain user data. For example, the user data in the Skill Number segment of Figure 3-14 is a specific identifier for the target segment (Skill). However, it could have been data describing when the employee was last called upon to use that skill, or simply the name of the skill. This data, which is unique to the relationship between a specific pointer segment and a target segment is called *intersection data*.

Because of the pointer-target segment relationship, the Skill segment of Figure 3-15 may be thought of as the concatenation of the intersection data in the pointer segment (Skill Number) and the data in the target segment (Skill). This is shown in Figure 3-15 as the contents of the Skill segment for that logical data structure.

The data content of the pointer and target segments may be retrieved and modified independently, or the pointer-target segment relationship may be retrieved and modified as one concatenated segment.

Figure 3-16 shows that for an inverted data structure, the intersection data in the pointer segment (Skill Number) is concatenated with the data in the Incentive segment.

Secondary Index Data Bases

As shown in Figure 3-17, another feature of DL/I is the ability to access an HDAM or HIDAM data base through secondary indexes.

For example, suppose it isn't enough to access the Employee data base by employee number alone? Suppose you also had to access it by name (for alphabetic retrieval), or by department (for a budget application)?

With DL/I you simply specify the fields to be indexed. These indexes are then created and maintained by DL/I. The application programmer need not be aware of them.

Secondary indexing allows application programmers to work with data base structures that are simplest and easiest for their particular needs. The result is increased productivity in coding and testing new programs.
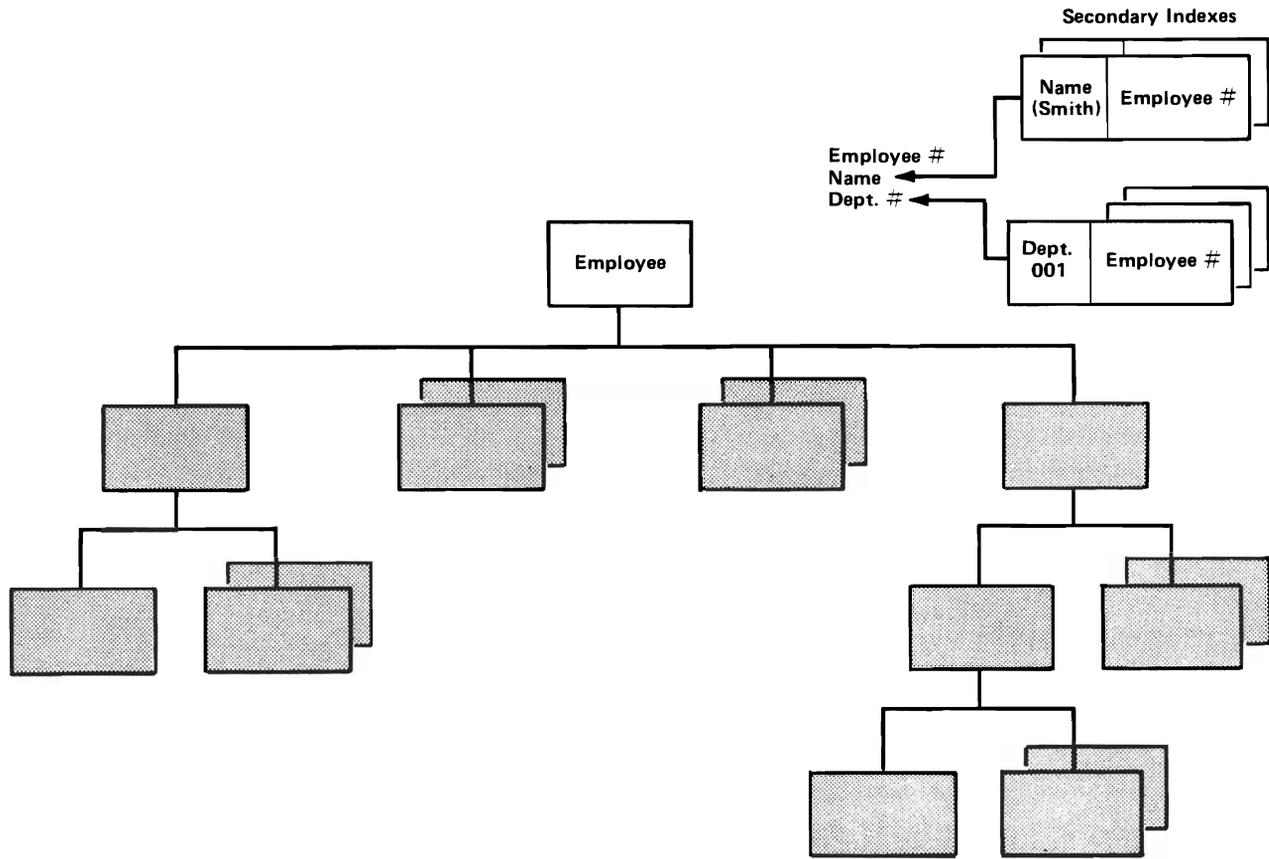
Figure 3-17. Secondary indexing allows the application programmer to work with a data base structure that is best suited to the particular needs of the application.

Additional Definitions

In addition to the definitions given previously, the following definitions apply within the DL/I environment:

- Logical data structure. A set of hierarchically related segments that serve as a prototype of a logical data base record. Only the segment types and not the number of occurrences of each segment type are illustrated.

- Logical data base record. A family of related segments described in the logical data structure but containing all related occurrences of each segment type. A logical data base record may exist as contiguous sets of segments within related physical blocks. In this case, the logical data base record is represented as one physical data base record. Alternatively, the logical data base record may be composed of segments from several physical data base records. The relationships between the physical data base records that represent the logical data base records are

accomplished by direct address techniques and indexes.

Data Base User Interface

A common symbolic program linkage and data base description allow the application program to request DL/I to:

- Retrieve a unique segment (GET UNIQUE)

- Retrieve the next sequential segment (GET NEXT)

- Retrieve the next sequential segment within the same parent (GET NEXT WITHIN PARENT)

- Replace the data in an existing segment (REPLACE)

- Delete the data in an existing segment (DELETE)

- Insert a new segment (INSERT).

The common symbolic program linkage handles the following languages: COBOL, PL/I, and Assembler language. The external data base descriptions describe the logical data structure and physical data organization of the data base to DL/I. Using these techniques, it is possible to physically reorganize established data bases in a timely manner without modification to application programs.

In the COBOL language, the common symbolic program linkage enables use of the CALL verb to perform the input/output functions listed above. Application programs written in PL/I or Assembler language use similar statements to reference DL/I. Because of this approach to data reference, input/output operations and associated system control blocks are not compiled into the application program. This prevents dependence upon currently available access methods and physical storage organizations.

Each data base description is created from user-provided statements that define the logical data structure and physical organization of each data base. These statements are input to a procedure of DL/I, the result of which is the creation and storage of a data base description. This data base description provides DL/I with a "mapping" from the logical structure of the data base used in the application program to the physical organization of the data used by DOS/VS data management. Other application data can be added to this data base without necessitating changes to application programs that use the data. The concept of the data base description reduces application program maintenance caused by changes in the data requirements of the application.

Data Base Organization and Access Methods

DL/I supports two basic physical storage organizations. The first organization, *hierarchical sequential* (HS), provides the basis for both the hierarchical sequential access method (HSAM) and the hierarchical indexed sequential access method (HISAM). Two additional access methods are provided for HS organized data bases. They are called simple hierarchical sequential access method (simple HSAM) and simple hierarchical indexed sequential access method (simple HISAM). The primary difference between these access methods and the two previously mentioned is that the simple HISAM and simple HSAM data bases must be defined as containing only root segments. The user may define, during DBD generation, any DOS/VS SAM fixed

length record file as a simple HSAM data base and any VSAM fixed length record key sequenced file (KSDS) as a simple HISAM data base whether or not the file was loaded by a DL/I application.

The second organization, *hierarchical direct* (HD), provides the basis for two more accessing techniques. The hierarchical direct organization is made available through the hierarchical direct access method (HDAM) and the hierarchical indexed direct access method (HIDAM). HDAM uses an addressing algorithm for direct access support of the hierarchical direct organization. An algorithm must be supplied by the user installation for each HDAM application. DL/I provides three sample randomizing routines that the user may reference when writing algorithms. (See the publication *DL/I DOS/VS Utilities and Guide for the System Programmer*.) HIDAM is an indexed access support of the hierarchical direct organization.

To allow faster and/or differently sequenced access, for HIDAM or HDAM data bases, one or more additional index data bases can be affiliated with a data base. These secondary indexes may be based on the values of any field and/or field combination in most segment types of the referenced data base. They may also carry system-controlled replications of data from that data base as well as user-controlled information.

Secondary index data bases have a hierarchical sequential structure (root segment only) and are comparable to a HISAM data base. Apart from serving as alternate access paths to a data base, they may be processed as data bases themselves. In the hierarchical direct organization, segments can be of variable length.

The primary differences between the hierarchical direct and the hierarchical sequential organizations are the manner by which segments are related and the techniques of data access. Segments in the hierarchical sequential organization that represent one data base record (that is, a physical hierarchical tree structure) are related by being physically adjacent. This requires that segments which represent one data base record be contained in a variable number of storage blocks unique to that data base record. Figure 3-18 shows the hierarchical sequential physical storage of the logical data structure for a Skills Inventory Data Base.

Physical blocks (a variable number) required to contain a particular data base record are related either by physical juxtaposition (HSAM) or by direct address relationships (HISAM).
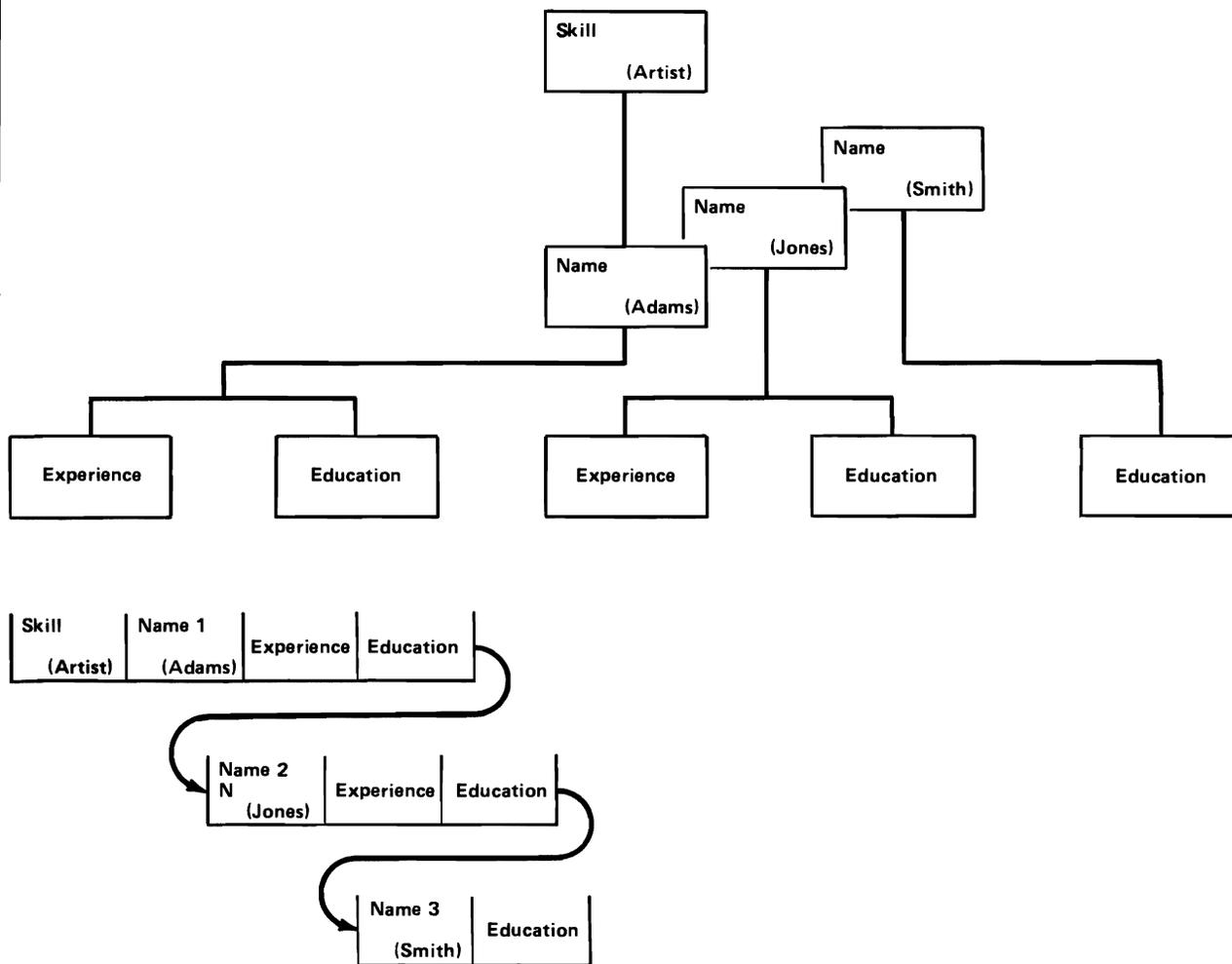
Figure 3-18. Hierarchical sequential physical storage of the logical data structure for a Skills Inventory data base.

Simple HSAM and HSAM are used for sequential storage and access on tape or direct access storage. The DOS/VS Sequential Access Method (SAM) provides the data management services.

Simple HISAM and HISAM are used for indexed sequential access to the hierarchical sequential organization. The DOS/VS Virtual Storage Access Method (VSAM) provides the data management services. A HISAM data base is comprised of one VSAM key sequenced file (KSDS) and one entry sequenced file (ESDS). Each data base record starts with a root segment and is stored in a VSAM key sequenced file (KSDS). As many dependent segments of that root segment as can be accommodated are placed in the KSDS physical record. If required, additional space is obtained from the VSAM entry sequenced file (ESDS), and overflow dependent segments for that root segment are stored in one or more ESDS physical records. Direct addresses relate the KSDS physical record and

all ESDS physical records for one HISAM data base record. A simple HISAM data base contains only root segments stored in one VSAM key sequenced file (KSDS).

Segments in the hierarchical direct organization that represent one data base record (that is, a physical hierarchical tree structure) are stored in one or more physical blocks. However, all segments in that data base record, rather than physical blocks containing the data base record, are related by direct addresses. Each segment in a data base record relates to segments of the same type as well as to adjacent segment types through direct addressing. Physical blocks that contain segments of a data base record are not related by direct addressing.

Figure 3-19 represents the segment direct address relationships in the physical storage of the logical structure from Figure 3-18 when the hier-

archical direct organization is used. Within a data base record, occurrences of a particular segment type are related by direct addressing to other occurrences of the same segment type (physical twins) under a given parent. In addition, the segment type immediately above (physical parent) and the first and last occurrence of each segment type immediately subordinate (physical children) are related by direct addressing.
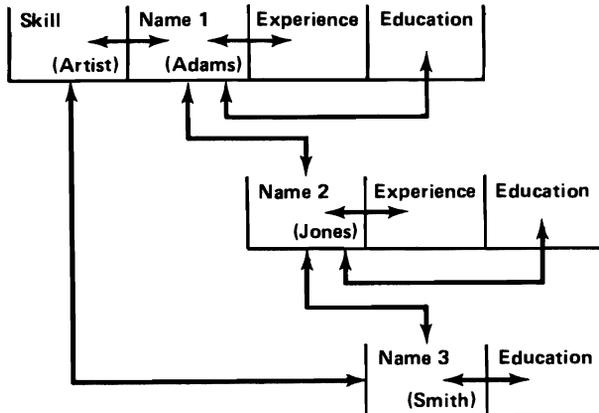
Figure 3-19. Hierarchical direct physical storage of Figure 3-18 logical structure

In the hierarchical direct organization, the space requirement for each segment is increased from that required in the hierarchical sequential organization. This space is required to accommodate direct addresses. However, the following advantages are gained:

- More rapid direct access to segments within a data base record.

- The ability to share space in a direct access storage block across multiple data base records. One physical block may contain segments from different data base records. This may result in considerable saving in data base storage space.

- The ability to reuse space occupied by deleted segments through the maintenance of free-space addresses.

- The possibility, through user provided exit routines, to edit or compress segments before they are physically stored in the data base, and to reconstruct or expand them after retrieval from the data base. This feature can save additional direct access space and/or increase data security.

HDAM is used for algorithmic addressability to records in the hierarchical direct organization. VSAM provides the data management services for HDAM.

HIDAM is used for indexed access to the hierarchical direct organization. VSAM provides the data management services for HIDAM. The index for HIDAM represents a key sequenced file (KSDS) and is referred to by DL/I as a HIDAM INDEX data base. Each KSDS logical record in the HIDAM INDEX contains the key of a root segment and a direct address pointer to the root segment. The root segment and all dependent segments of a data base record are placed in physical blocks in the HIDAM data base. Because the data base for index storage is separated from the data base for segment or physical data base record storage, reorganization of the index separate from data is facilitated.

Segment Definition and Format

The foregoing discussion has introduced some of the concepts used in the physical storage of data. In both storage organizations discussed, the term "data base record" is used. The data base record is represented as a simple hierarchical tree structure of related segments. The hierarchical tree structure represents one sort sequence of segments. This chapter has explained how logical data base records are constructed from one or more physical data bases. This is how other sort sequences of segments are obtained. The foregoing discussion also indicated that segments in the hierarchical direct organization are related by direct addresses. With the exception of two special cases, each segment, independent of data base organization, is composed of two parts - the *prefix* and the *data*. The prefix contains:

- Segment type

- Segment deletion indicator

- Direct addresses that establish intersegment relationships.

The format of the prefix for any segment type is unique. It is determined by the data base organization and segment interrelationships. The segment format, which includes both prefix and data portions, is specified in the data base description. The use of the segment prefix is controlled entirely by DL/I. An application program need not be concerned about the presence or the format of the segment prefix as only the data portion of a segment is passed between an application program

and DL/I. The data portion of a segment is composed of one or more user-supplied data fields. One of these fields may control the physical sequence of occurrences of that segment type.

The two exceptions mentioned above occur with simple HSAM and simple HISAM data bases, which consist of root segments only. In these cases, the root segments contain only data and no prefix is present. HSAM and HISAM data bases of this simple structure appear like conventional fixed blocked data files. Therefore, it is possible to treat conventional fixed blocked data files as simple data bases with no intervening load operations. These data files may have been created by non-DL/I programs and can still be read by these programs, while DL/I calls may be used to access and process these files as DL/I data bases.

Interrelated Data Base Records

Even though logical data base records can be described in the form of simple hierarchical trees, their physical representation may be considerably more complex. A logical data base record might be composed of segments from only one physical data base record. Alternatively, a logical data base record might be composed of segments from several physical data base records. These physical data base records could be contained in one data base or in multiple data bases. Here the term data base means a family of physical data base records in which all have a common hierarchical segment structure.

Figure 3-20 indicates the relationships that can be established between physical data base records in two different data bases. These relationships are again described through the parent-child-twin terminology. However, the relationships are considered logical rather than physical. For the name segment of Adams, the *logical child* segments are the pointer segments under skill for artist and skill for mechanical engineer. All pointer segments under skill segments that point to the name segment for Adams are considered *logical twins*. The name segment for Adams is considered the *logical parent* for all pointer segments that point to it.

Since logical data base records are constructed from one or more physical data base records and their intersegment relationships, you should understand the terms logical data base and physical data base. A physical data base is composed of physical data base records. The logical data base is composed of one or more physical data bases. The physical data base records are interrelated by logi-
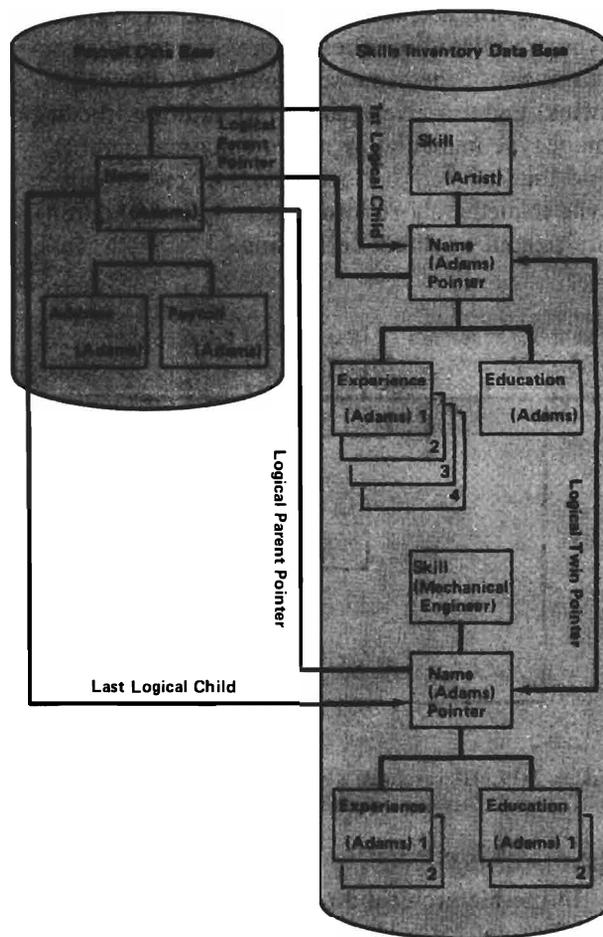
Figure 3-20. Relationship between physical data base records (two data bases)

cal parent-child-twin and physical parent-child-twin relationships.

Indexed and Indexing Data Base Records

Primary indexing as used with HISAM and HIDAM is considered to be part of the physical storage organization of a data base. Secondary indexing of a HDAM or HIDAM data base is accomplished by additional data bases, using a root-only structure comparable to a HISAM data base.

A secondary index data base record points to the indexed segment through a direct address pointer. The relation between an index and its indexed data base is similar to what was earlier described as a logical relationship between two physical data bases; with the indexed segment being considered to be the target segment.

The data upon which a secondary index is based, that is, the data which will comprise the KSDS key in the index data base, may exist within the indexed segment or in any segment which is hierarchically below the indexed segment. This data may come from several noncontiguous fields within a segment, however, it must be unique; optionally, system-maintained replications of other fields within this segment as well as user-controlled information can be carried as data in the index data base.

User options and exit routines are provided to suppress creation of an index record for any data base record.

Data Base Administration

The centralization of data and control of access to this data is essential to a data base management system. One of the advantages of this centralization is the availability of consistent data for more than one application. This dictates a tighter control of that data and its usage. Responsibility for an accurate implementation of control lies with the data base administration function. Because the actual implementation of this function is largely dependent on a company's organization this discussion is limited to the general characteristics of data base administration. Although the data base administration function is usually performed by a person called the data base administrator, quite often this function is actually performed by a group of individuals with experience in both application and system programming. The duties of the data base administrator are to:

- provide standards for and control the administration of the data bases and their use.

- provide guidance, review and approval of data base design.

- determine the rules of access to the data bases and monitor their security.

- control the data base integrity and availability, monitoring the necessary activities for reorganization and back-up recovery.

- enforce procedures for accurate, complete, and timely updates of the data bases.

- approve the operation of new programs with existing production data bases, based on results of testing with test data bases.

Online Processing Capability

DL/I functions may be extended to the telecommunication environment through an especially modified teleprocessing program. The following describes highlights of the Customer Information Control System/Virtual Storage (CICS/VS), which provides the interfaces necessary to permit DL/I to operate in an online environment. For more detailed information, see the *CICS/VS General Information Manual,* listed in the Preface.

CICS/VS is a general-purpose data base/data communication interface between DL/I DOS/VS and user-written application programs (either COBOL, PL/I, or Assembler language). The user is given the facilities to generate a CICS/VS system configuration applicable to his needs and to define the environment in which the system is to execute. User exits are provided for optional processing as required for specific system operation. Also provided is a macro facility to communicate application program service requests.

Functions necessary to support DL/I and those required to support other standard terminal applications are provided by CICS/VS through the following management facilities:

Task Management: Provides the dynamic multitasking facilities necessary for effective, concurrent transaction processing. Functions associated with this facility include priority scheduling, transaction synchronization, and control of serially reusable resources.

Storage Management: Controls main storage allocated to CICS/VS. Storage acquisition, disposition, initialization, and request queuing are among the services and functions performed by this component of CICS/VS.

Program Management: Provides a multiprogramming capability through dynamic program management while offering a real-time program fetch capability.

System Recovery Management: Intercepts program interrupts and causes the individual CICS/VS transaction in which the interrupt occurred to be abended, rather than the whole of the CICS/VS partition. This function also intercepts the DOS/VS partition abends, permitting CICS/VS to terminate in a controlled manner. With CICS/VS Version 1.3, this capability is extended to permit CICS/VS to recover from such a partition abend and continue running normally.

Time Management: Provides control of various optional task functions (system stall detection, runaway task control, task synchronization, etc.) based on specified intervals of time or time of day.

Dump Management: Provides a facility to assist in analysis of programs and transactions undergoing development or modification. Specified areas of main storage are dumped onto a sequential data set, either tape or disk, for subsequent offline formatting and printing using a CICS/VS utility program.

Terminal Management: Provides polling according to user-specified line traffic control as well as user-requested reading and writing. This facility supports automatic task initiation to process new transactions. The testing of application programs is accommodated by the simulation of terminals through devices such as card readers, line printers, tape units, and disk devices.

File Management: Provides a data base facility using direct access and indexed sequential data management. This function supports updates, additions, random retrieval, and selective retrieval (browsing) of logical data on the data base. CICS/VS provides access to the DL/I facilities of DL/I DOS/VS.

Transient Data Management: Provides the optional queuing facility for the management of data in transit to and from user-defined destinations. This function has been included to facilitate message switching, data collection, and logging.

Temporary Storage Management: Provides the optional general-purpose "scratch pad" facility. This facility is intended for video display paging, broadcasting, data collection suspension, conservation of main storage, retention of control information, etc.

In addition to the management functions described, CICS/VS provides the following system service programs:

Sign On/Sign Off: Provides terminal operator identification (security).

Master Terminal Function: Provides dynamic user control of the system. The master terminal operator can change the status and values of par-

ameters used by CICS/VS and thereby alter the operation of the system.

Supervisory Terminal Function: Provides a terminal-oriented subset of the services available to the master terminal operator to individuals defined as having supervisory authorization. The supervisory subset can be used at any terminal signed on under a given supervisor's jurisdiction.

System Statistics: Provides the capability to dynamically log system statistics.

Abnormal Condition: Intercepts abnormal conditions (except those associated with a terminal) not handled directly by the operating system.

Terminal Abnormal Condition: Intercepts terminal abnormal conditions not handled directly by the operating system.

System Termination: Allows the user to terminate operation of CICS/VS by gathering summary statistics, closing data sets, and returning control to the operating system.

Trace: Provides a program debugging facility that reflects the execution of CICS/VS macro instructions by CICS/VS management programs and user-written application programs.

Dynamic Open/Close: Allows the user to dynamically OPEN/CLOSE his data sets during the real-time execution of CICS/VS.

CICS/VS is a modular program with system generation capabilities. The user selects those modules which meet his unique requirements during the system generation process. Some of the selected modules are resident in main storage; others are loaded into storage when required during execution of the user-written application program. Communication between the user-written application program and CICS/VS is through CICS/VS macro instructions.

Activity within the DL/I system is initiated by data from a terminal on the system. All application program requests for terminal and file read-write are processed by DOS/VS through CICS/VS. The DOS/VS supervisor and data management services are used by CICS/VS when they are functionally applicable. Features are incorporated which assist in the serviceability of components of the system to provide maximum system availability.

CICS/VS operates as a single task within a partition and may operate in a dedicated or multiprogramming environment. The selection of the environment is the user's responsibility, as is the selection of system options beyond those required for the operation of CICS/VS.

Chapter 4. User Installation Requirements

The DL/I system provides a generalized data base/data processing program designed to enhance the user's ability to implement data processing applications. In order to ensure successful implementation of his programs with DL/I, certain requirements are placed on the user. These fall into two categories: user installation responsibilities and a user schedule to plan for installation.

User Installation Responsibilities

The user must ensure that personnel are sufficiently trained in DL/I operations.

User data bases must be designed and created. A data base description (DBD) generation must be performed for each data base.

The user must define how each application program will access data bases and to which segments each application program will be sensitive. This is accomplished by program specification block (PSB) generations.

User application programs must be written (or modified) to access DL/I data bases using DL/I calls. If segment edit/compression is planned, appropriate exit routines must be written. In addition, if a user accesses data bases with HDAM, he must provide a randomizing routine for his application. The DL/I system provides three sample randomizing routines to assist the user.

If the user plans CICS/VS online operations, he must additionally tailor his operations for and generate a CICS/VS system. He must also train his personnel in CICS/VS operations.

User Schedule

The user should provide adequate leadtime to provide for and implement the above tasks. Estimates of the man months required to perform these tasks are as follows:

Task	Man Months
1. Train personnel	
• Machine operations	1/4
• Application programming	1/2
• System programming	1
2. Design and generate data bases (DBD)	2
3. Define application program access (PSB)	1
4. Provide randomizing routine	1/4

In addition to the above, consideration should be given to the time required to write application programs, and to processing in an online environment. If the user wishes to use CICS/VS, he should allow an additional three man months to tailor his system and train his personnel for CICS/VS operations.

The purpose of this chapter is to illustrate the effects of a paging environment upon the performance of a DL/I DOS/VS application program. The contents of this chapter are as follows:

- Description of the test environment and application program.

- Performance measurements for HDAM, HIDAM, and HISAM data bases.

Test Environment

The tests were executed on a System/370 Model 145 with 512K bytes of main storage and 2314 Direct Access Storage Facility.

- Four 2316 disk packs were used containing:

    (1) SYSRES and

    (2) Page data set on channel 1

    (3) DL/I data bases and

    (4) Core image library on channel 2

- The test data consisted of ten root segments and 262 dependent segments and was loaded for HDAM, HIDAM, and HISAM.

- VSAM control interval sizes were 512 bytes for KSDS and 1024 bytes for the ESDS.

- The application program issued 1633 segment retrieval calls and 618 segment updates against the data base.

- The application program was DOS/VS page fixed (PFIX).

- The DL/I program was executed in the background partition V=V with varying real storage allocation (ALLOCR=).

- A control program was executed in the foreground 2 partition V=R for the remaining real storage (512K - SUPV - BG = F2).

Performance Measurements

The objective of the tests was to show how performance of a DL/I application will decrease as real storage (number of pages) decreases. It must be noted that the performance tests were conducted using a relatively small data base; no other par-

tition was executing V=V; and the application program used was designed for a specific environment. Changing any one of the above could alter the individual results significantly for the three access methods. Therefore, the results of these tests should not be used to compare the relative merits of one access method against another.

Four tests were conducted for each access method with varying amounts of real storage allocated. Three elements had to be considered in calculating real storage allocation. They were:

1. The application program storage requirements

2. VSAM module and control block storage requirements

3. DL/I module and control block storage requirements.

Since the application program is a user responsibility and represents a variable, the application program used in the performance measurement was DOS/VS/PFIXed, and the number of pages used by the application program is *not* included in the following tables.

The secondary storage requirements for VSAM may be in excess of 200K bytes (including control blocks, buffers and utilities). However, less than 40% of this code is considered as main processing code. As a basis for calculating the normal VSAM storage requirements, seven pages were estimated for VSAM modules plus one page for HIDAM or HDAM buffers (KSDS) or two pages for HISAM buffers (KSDS + ESDS).

The final factor in calculating the number of pages of real storage required for the partition was the DL/I modules and control blocks. The storage requirements varied for the access method and will be discussed with the access method description.

HDAM

The performance measurements for HDAM are illustrated in Figure 5-1. For the first test sufficient main storage was allocated (200 pages) to ensure no paging. In the next three tests the total number of pages of real storage for VSAM and DL/I were 50 pages, 38 pages and 30 pages, based on the following:

- VSAM = 7 pages. No storage allocated for VSAM buffers since DL/I buffer pool was used.

- DL/I = Total DL/I module and control blocks storage requirements were 86K bytes (43 pages). Included in this amount is one 32K byte buffer pool (32 1024 byte buffers). 43 pages (100%), 31 pages (70%) and 23 pages (56%) were used.
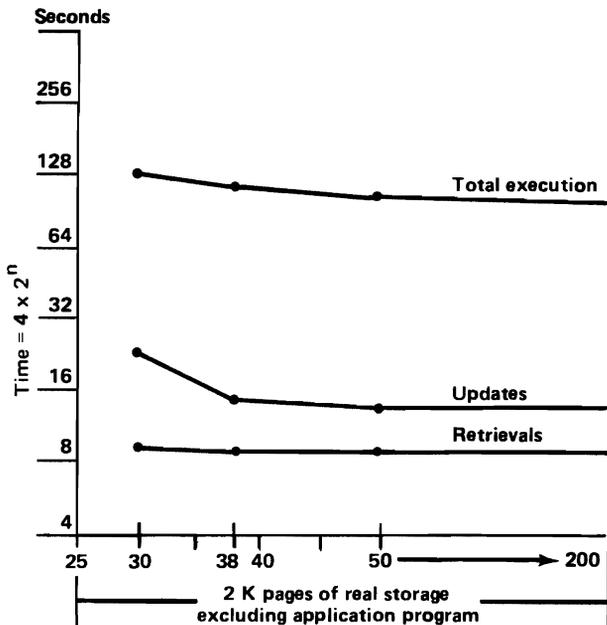
Figure 5-1. DL/I execution for HDAM data base in paging environment.

HIDAM

The performance measurements for HIDAM are illustrated in Figure 5-2. For the first test sufficient main storage was allocated (200 pages) to ensure no paging. In the next three tests the total number of pages of real storage for VSAM and DL/I were 52 pages, 39 pages and 30 pages, based on the following:

- VSAM = 8 pages, one page of which was allocated because three 512 byte buffers are required by VSAM for the KSDS.

- DL/I = Total DL/I module and control blocks storage requirements were 88K bytes (44 pages). Included in this amount is one 32K byte buffer pool (32 1024 byte buffers). 44 pages (100%), 31 pages (70%) and 22 pages (50%) were used.
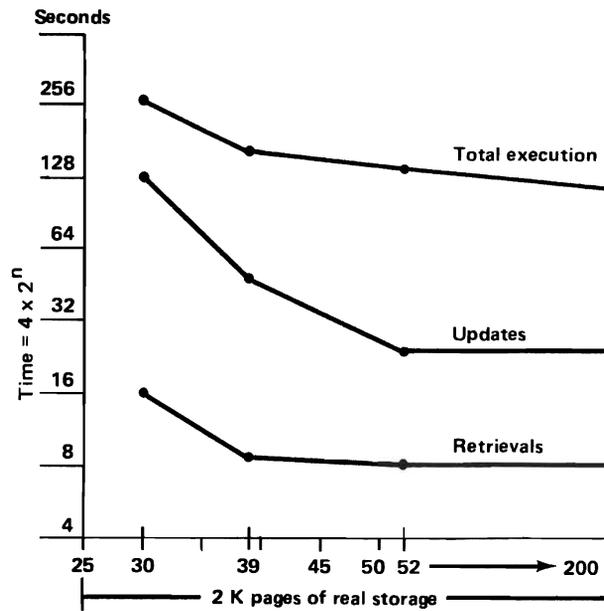
Figure 5-2. DL/I execution for HIDAM data base in paging environment.

HISAM

The performance measurements for HISAM is illustrated in Figure 5-3. For the first test sufficient main storage was allocated (200 pages) to ensure no paging. In the next three tests the total number of pages of real storage for VSAM and DL/I were 36 pages, 30 pages and 28 pages, based on the following:

- VSAM = 9 pages, two pages of which were allocated because three 512 byte buffers are required by VSAM for the KSDS, and two 1024 byte buffers are required for the ESDS.

- DL/I = Total DL/I module and control blocks storage requirements were 82K bytes (41 pages). Included in this amount was one 32K byte buffer pool (32 1024 byte buffers). However, no more than three DL/I buffers will ever be utilized (for insert call processing only) since the buffer management function is performed by VSAM for HISAM data bases. Therefore, the total effective DL/I storage requirements is 54K bytes (27 pages). 27 pages (100%), 21 pages (78%) and 19 pages (70%) were used.
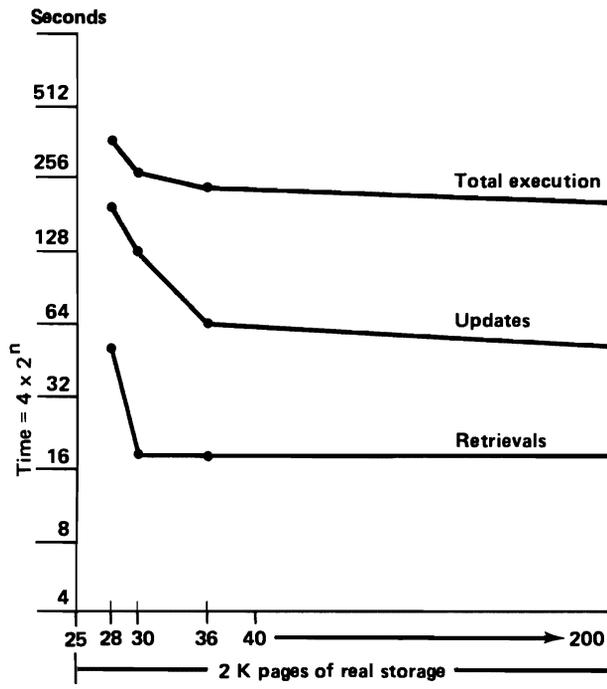
Figure 5-3. DL/I execution for HISAM data base in paging environment

Chapter 6. Machine Configurations

Minimum DL/I Configuration

The minimum requirements for executing DL/I DOS/VS are outlined below. Two environments are shown, a batch environment and an online environment in conjunction with CICS/VS. In both cases, a limited and a full set of DL/I functions are considered to show the resulting difference for the real storage requirements.

The estimates of real storage requirements are derived from the size of the modules in the DL/I system necessary to handle normal execution, that is, modules handling exceptional conditions such as errors or open/close, have not been included. Thus, if the specified real storage is available to DL/I and if no exceptional condition arises, then no paging will occur in the DL/I code. Of course, as for every system executing in a virtual storage environment, if performance is not critical, DL/I can execute in less real storage. Also, a trade-off may be considered between the available real storage in a central processing unit model and its instruction rate. For example a System/370 Model 125 may require less real storage than a System/370 Model 115 to achieve the same performance, since its internal speed can make up for the additional paging overhead.

The minimum storage estimates provided here are based on the following application profile:

- Two HISAM data bases

- One buffer subpool with three buffers of 512 bytes each

- VSAM Control Interval size of 512 bytes

- No use of advanced functions (secondary indexing, logical relationships, variable length segments)

For the online system, the profile includes:

- Two PSBs in use

- Two tasks running concurrently

System Function	Units Permitted		
Processing Unit	Any System/370 supported by DOS/VS with a minimum real storage of 128K for a batch system or 188K for an online system in conjunction with CICS/VS.		
Minimum Real Storage (Including VSAM)	The practical minimum amount of real storage is:		
	Batch System		
	• Basic retrieve operations only	60K	
	• All functions	80K	
	Online system (including minimum CICS/VS/VS requirements*)		
	• Retrieve operations only	130K	
	• All Functions	150-200K	
	Multiple Partition Support (MPS)		
	The real storage requirements are the same for a system with MPS as they are for an online system plus 4 to 10K for each active MPS batch partition.		
Minimum Virtual Storage	Virtual storage must be:		
	Batch System	512K	
	Online System	768K	
System Console	See DOS/VS.		
Tape Units**	At least two 9-track 2400 or 3400 series tape units and control.		

* For further information about the storage and CICS/VS-supported terminals and features, see the *CICS/VS General Information manual,* as listed in the Preface.

**Not required if disk logging is used (Version 1.3).

System Function	Units Permitted
Direct Access	For system libraries and working storage space, any devices supported by DOS/VS. Minimum space for system use and maintenance:
	Batch system (75 cylinders 2316 or equivalent)
	Online system with CICS/VS/DOS/VS (150 cylinders 2316 or equivalent.)
	For DL/I data base storage, within the capabilities and restrictions of DOS/VS support by the virtual storage access method and sequential access method:
	2314/2319 Direct Access Storage Facility
	3330 Disk Storage
	3340 Disk Storage.
Telecommunication Facilities Available with CICS/VS	Various terminals, terminal control units, and programmable special features are supported by CICS/VS.

Typical DL/I Configuration

A typical configuration for DL/I DOS/VS includes:

- System/370 3138 Processing Unit (512K) with:

  One block multiplexer channel with integrated storage control

  One multiplexer channel

  One selector channel

  One tape control

  Four 9-track magnetic tape units

  Four 3330 Disk Storage

  One 3215 Console Printer-Keyboard Model 1

  One 2821 Control Unit Model 5

  One 2540 Card Read Punch

  Two 1403 Printers Model N1.

The organization of these components is shown schematically in Figure 6-1.

Note: The maximum number of input/output devices, including communication lines, that may be attached to the system is in accord with the capabilities of DOS/VS.

Typical DL/I Real Storage Requirements

The following is an example of the real storage requirements for a typical DL/I system that could serve as a usable production system. The figures are based on the following application profile:
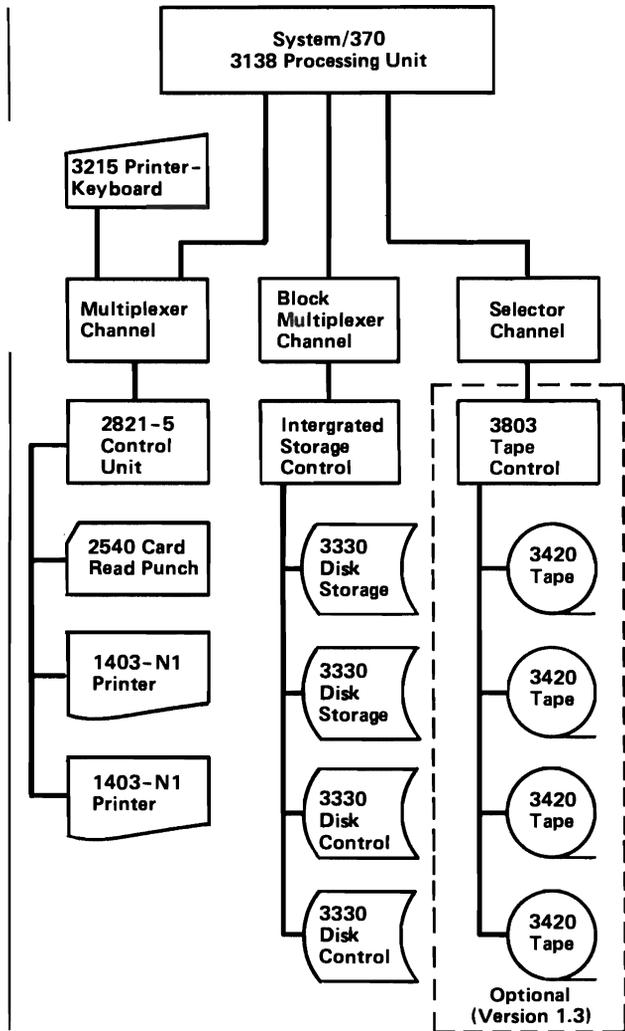
Figure 6-1. Typical configuration—DL/I system

- Five HDAM data bases
- One HIDAM data base
- Use of advanced functions
- Use of data base change logging
- One buffer subpool with 12 buffers of 2K each

For the online system, the profile covers:

10 PSBs in use
5   tasks running concurrently

The real storage requirements are:

Real Storage	The typical amount of real storage is:	
	Batch System	
	• Retrieve operations only	100K
	• All functions	140K
	Online system (including typical CICS/VS/VS requirements)	
	• Retrieve operations only	220K
	• All functions	240-260K
Virtual Storage	Virtual storage (including the real storage) is:	
	Batch System	600K
	Online System	1000K

For a more detailed discussion on storage considerations and how to calculate your own virtual and real storage requirements, see *DL/I DOS/VS System/Application Design Guide,* listed in the Preface.

Chapter 7. Programming Requirements

DL/I will execute exclusively on System/370 using the Disk Operating System/Virtual Storage (DOS/VS). DL/I is written in Assembler language and uses the Virtual Storage Access Method (VSAM) and Sequential Access Method (SAM) data management facilities.

The following components of DOS/VS are required:

- Control and service programs
  - IPL and buffer load, 5745-SC-IPL
  - Supervisor, 5745-SC-SUP
  - Job control, 5745-SC-JCL
  - Linkage editor, 5745-SC-LNK
  - Librarian - maintenance & service, 5745-SC-LBR

- Assembler, 5745-SC-ASM

- Data management
  - Sequential disk IOCS, 5745-SC-DSK
  - Sequential tape IOCS, 5745-SC-TAP
  - Virtual storage access method (VSAM), 5745-SV-VSM.

One of the following program products is required:

- DOS/VS Sort/Merge, 5746-SM1

- DOS Sort/Merge, 5743-SM1.

In addition to the above DOS/VS components, the user may require the following:

- DOS/VS COBOL Compiler and Library, 5746-CB1 or Library only, 5746-LM4

- Full ANS COBOL V3 Compiler, 5736-CB2, and Full ANS COBOL Library, 5736-LM2

- ANS Subset COBOL, 5736-CB1

- ANS COBOL, 360N-CB-482/370N-CB-482

- PL/I Optimizing Compiler and Libraries, 5736-PL3

- PL/I Optimizing Compiler, 5736-PL1

- PL/I Resident Library, 5736-LM4

- PL/I Transient Library, 5736-LM5.

- Customer Information Control System/Virtual Storage (CICS/VS), 5746-XX3, Version 1.2, including an update for module DFHTBP which is required if multiple partition support is to be implemented, or CICS/VS Version 1.3 (required for DL/I DOS/VS Version 1.3).

Chapter 8. Sample Applications

This chapter presents several data base applications which are examples of the types of terminal inquiry and update transactions that a user might employ. In addition, the data base record structures and organizations necessary in providing efficient inquiry and update processing are described. Examples are taken from several industries, but these are by no means to be considered all-inclusive. The reader may wish to work out other examples from his own industry. The examples are described by considering:

- The information that a terminal operator might want or that a report might contain

- The logical data structure necessary to supply the desired information

- The most suitable data organization and access methods

The DL/I data base concept allows for the correction of a data base, using the traditional manner of system design printed above. However, DL/I provides the ability to add new segment types into existing data bases as well as the ability to create new data bases with minimal impact to the system user. This data base approach to system design promotes evolutionary system development.

Manufacturing Industry

The following list of questions represents typical requests for information in the manufacturing industry:

What parts represent the component parts of an assembly, and in what quantity is each component part required?

Where is a given part used in the composition of assemblies, and in what quantities is the part used?

What is the inventory status of a part where multiple inventory sites exist?

What open purchase orders exist for a given part and who is the vendor supplying the part?

What work orders exist for a given part, and what is the status of each work order?

What are all the open purchase orders and work orders?

What operations are performed in the construction of a particular part, and at what work centers are these performed?

What operations are performed at a particular work center, and what parts are affected by these operations?

The first two questions relate to the structure of a product and can be asked for a particular part, a substructure of a product composed of many parts, or an entire product. One question is really the inverse of the other. The logical data structure for answering both questions is shown in Figure 8-1.
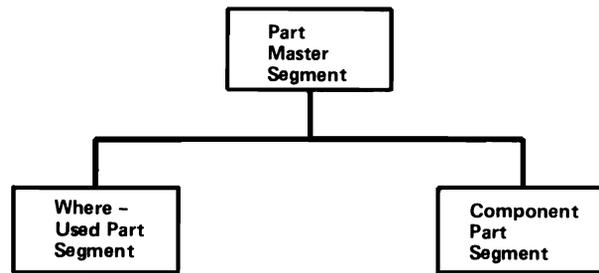
Figure 8-1. Logical data structure for part data base

However, the where-used information for one part master is the same as the component part information of the part master where it is used. Actually, the where-used information for one part is redundant with the component part information of its assemblies. The pointer segment—target segment concept introduced in Chapter 3 can therefore be employed as shown in Figure 8-2. The functions of the where-used part segment and the component part segment can be achieved by one segment type. This segment type is called the component part/where-used segment. A one-level bill of material is produced by proceeding from a part master to its component part/where-used segments by physical child and physical twin relationships. One-level where-used information is obtained by proceeding from a part master to component part/where-used segments by logical child—logical twin relationships.
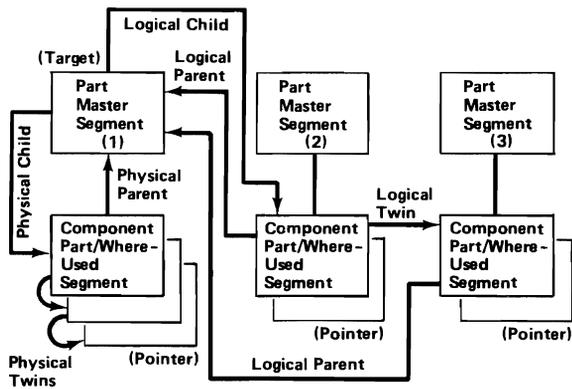
Figure 8-2. Three interrelated physical data base records for three parts

The part master segment 1 in Figure 8-2 represents a part that is used as a component for both master segment 2 and part master segment 3. Figure 8-3 illustrates a simple hierarchical structure and these relationships. The dependent segment of Figure 8-3 represents the concatenation of data from the pointer segment and target segment; the data in the pointer segment is intersection data.
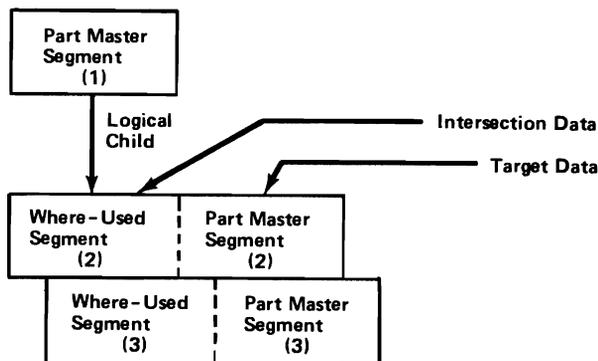
Figure 8-3. Logical data structure for usage of part 1 of Figure 8-2.

Part master segment 2 in Figure 8-2 represents a part that is composed of the part described in part master segment 1 in Figure 8-2 and, let us assume, a part master segment 4. This relationship of component part explosion can again be illustrated by means of a simple hierarchical tree. Notice the concatenation of pointer and target segment data in Figure 8-4. The physical existence of the component part/where-used segment exists under the part master for which it represents component part data. Therefore, the relationship from the part master 2 to its component part segments 1 and 4 is a physical parent-child relationship. The relationship for part master 1 to its where-used

information in the combined component part/where-used segment is by address chains. This is a logical parent-child relationship.
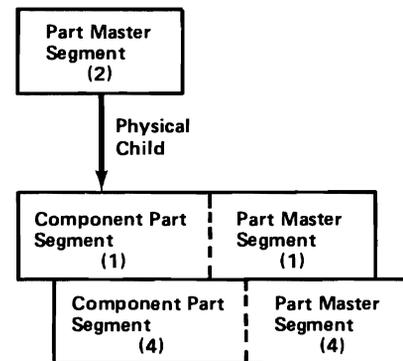
Figure 8-4. Logical data structure for component part definition of part 2 of Figure 8-2

The inventory status for a particular part could be supplied by the data structure in Figure 8-5.
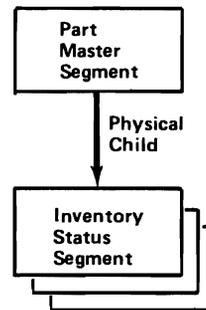
Figure 8-5. Logical data structure for part inventory status

Each inventory status segment for a particular part is a dependent segment under the part master segment. Each represents the inventory for the part at a particular location.

The open purchase orders and vendors assigned for a particular part could be supplied by the data structure in Figure 8-6.
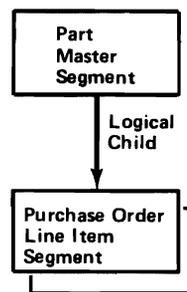
Figure 8-6. Logical data structure for part purchase order

Each purchase order line item segment represents a line item in an open purchase order pertinent to this part. Of course, the inverse question of what purchase orders have line items that affect a given part might be asked. This problem is similar to the component part/where-used situation and can again be solved with the pointer segment—target segment concept. Let us assume part master segment X in Figure 8-7 has a line item in two purchase orders, A and B.
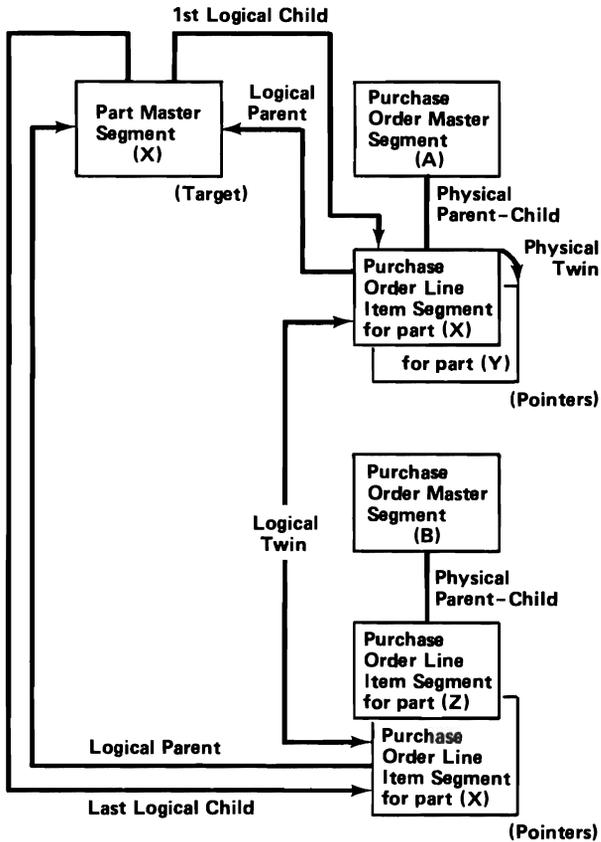
Figure 8-7. Physical data base records with pointer segment—target segment concept for part purchase orders

The next questions, regarding what operations are performed in the fabrication of a part and what operations performed at a given work center affect a given part, again involve inverses. These can be answered by the two logical data structures in Figure 8-8.
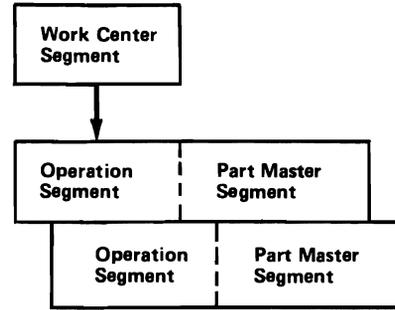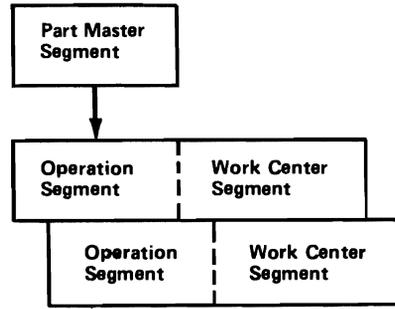
Figure 8-8. Two logical data base structures showing fabrication operations

Using the pointer segment—target segment concept, Figure 8-9 illustrates how data redundancy is removed. Let us assume that part master A has operations 1, 2, and 3 performed in its fabrication at work centers X, Y, and Z, respectively.
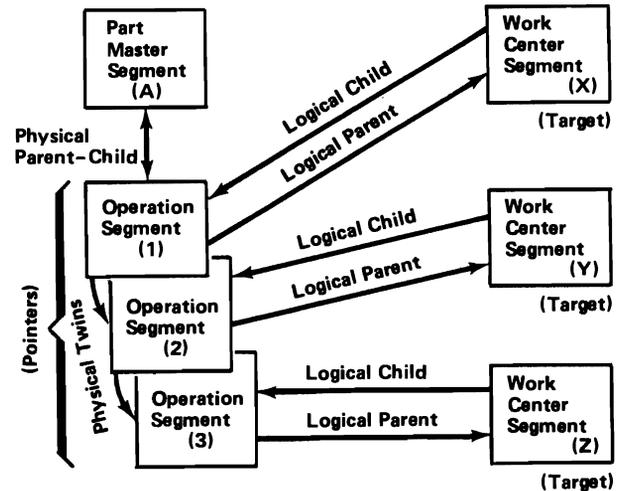
Figure 8-9. Pointer segment—target segment concept showing elimination of data redundancy

The operation segment under other part masters, where the operation is performed at work center Y, would be logical twins of the operation

segment 2 under part master A. By following the logical child relationship from work center segment Y to operation segment 2, and then following the logical twin relationship to related operation segments, the question of all parts affected by operations at a particular work center can be answered.

The following segment relationships can now be formulated into physical and logical data bases. Figure 8-10 illustrates the physical data base records.
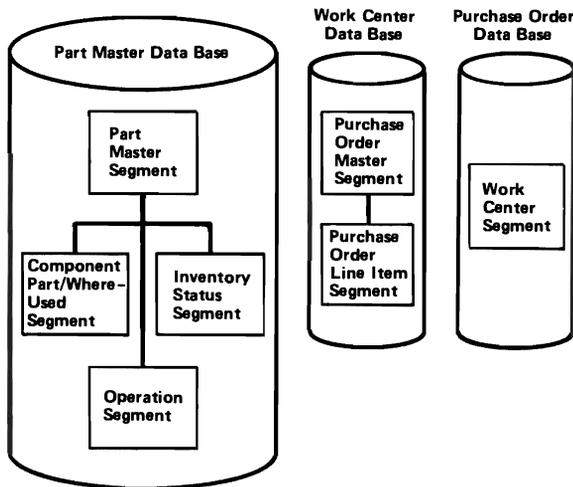
Figure 8-10. Physical data base records under pointer segment—target segment concept

These three physical data bases are interrelated by the pointer segment—target segment concept as shown in Figure 8-11. The interrelationships as well as the physical data base records allow logical data base records to be described as indicated in the foregoing discussions.

Let us assume that HIDAM is chosen for the part master and work center structures. Also assume that HDAM is chosen for the purchase order structure. Figure 8-11 illustrates the physical data bases. Logical relationships are accomplished by direct address pointers.

Financial Industry

The following list of questions represents typical requests for information in the banking environment:

What are all the accounts associated with a particular individual? This information might be desired when a customer wants to make a deposit but does not know his account number.

What is the status of loans outstanding to a particular individual? This information might be desired when a bank officer is asked to accept a loan request from a customer.

Does the checking account of a known account number contain a balance adequate to cash a check?

What is the amount of money needed to pay off the installment loan for a known loan account?
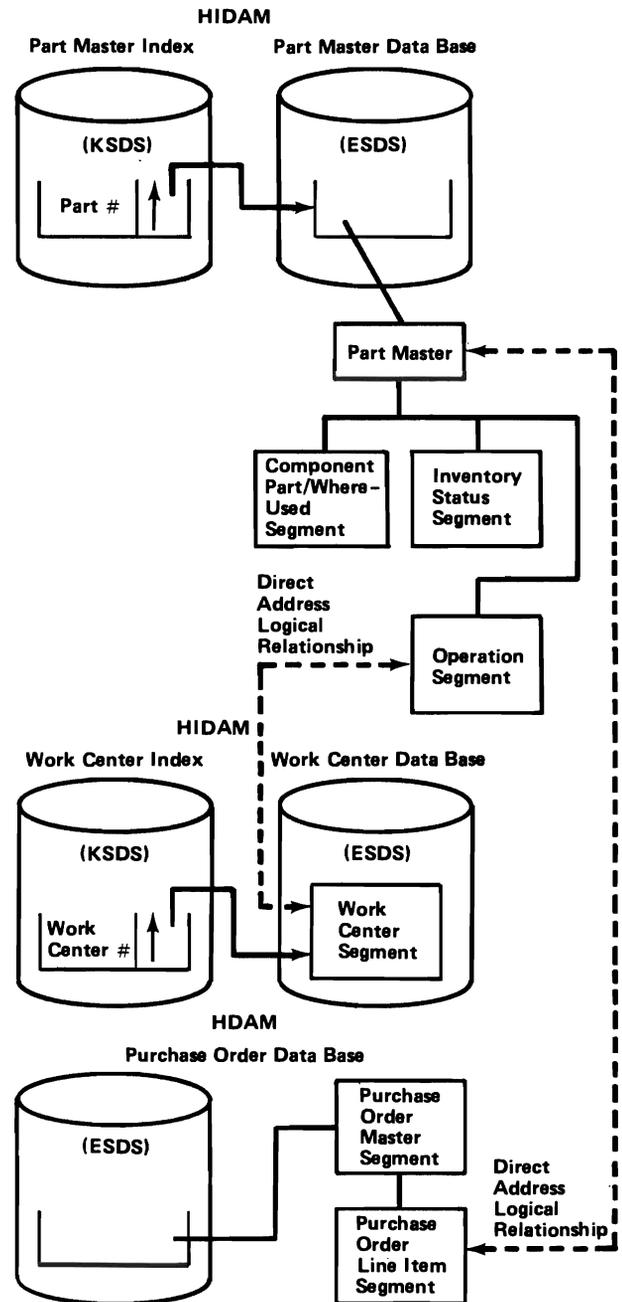
Figure 8-11. Physical data bases under HIDAM and HDAM

What is the property held and what is the par value for each property held in a trust account?

What trust accounts hold a particular property, such as a particular stock? The additional question might be asked: What is the quantity in shares held of the stock in each trust account?

These questions and the subsequent data structures should stimulate the reader to consider other questions and additional data elements and structures. Both query and update operations against the data base are possible.

The first question, concerning all accounts associated with a given individual, could be answered with the logical data structure of a customer information record in Figure 8-12.
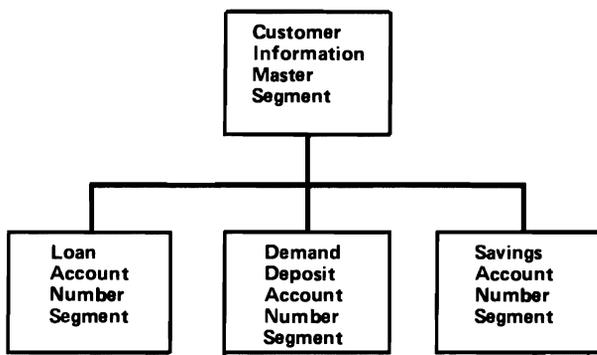
Figure 8-12. Logical data structure of a customer information record—financial

A customer information master segment would exist for each of the bank's customers and might contain name, address, dates, codes, and a customer-identifying key. The loan, demand deposit, and savings account number segments might be keyed on account number and indicate account type and relationship to individual customer.

The second question, concerning the outstanding loans for an individual, can be answered with the same data structure.

The answer to the third question, concerning the current balance in demand deposit account, can best be answered with a data structure organized by account number. However, we have already established a demand deposit account (number) segment subordinate to customer information master segment in Figure 8-12. DL/I gives the ability to enter the data base either by customer name or account number. This may be approached with the use of the pointer

segment—target segment capabilities. Figure 8-13 illustrates the necessary data structure.
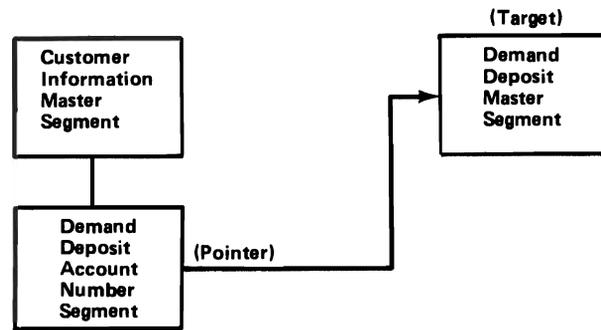
Figure 8-13. Data structure with pointer segment—target segment relationship

Referring to Figure 8-13, the pointer segment—demand deposit account number segment—is subordinate to customer information master segment. It contains only a pointer to the applicable demand deposit master segment and the data describing the relationship between the customer and the account. The demand deposit master segment with its dependent segments contains most of the information about the account including current balance. The backward relationship from target segment to its associated pointer segments allows the specification of logical data base structures illustrated in Figure 8-14.
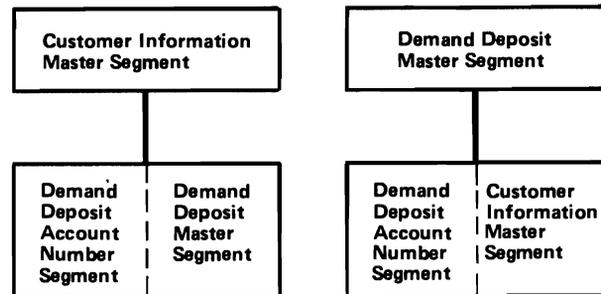
Figure 8-14. Logical data base structures showing customer information specifications

The demand deposit account number segment (pointer) is limited in data content to data unique to the relationship between a given customer and a particular demand deposit master account. This represents intersection data.

An example of the physical data base records and the logical interrelationships obtainable is depicted in Figure 8-15.

These two questions are answered with inverse data structures and present a problem similar to the customer information master—account master data structures. In addition, there is probably the requirement for intersection data, such as how much (how many shares) of a particular property is held in a particular trust account.

If we utilize the pointer segment—target segment concept, the logical data structures in Figure 8-16, can be defined as illustrated in Figure 8-17.
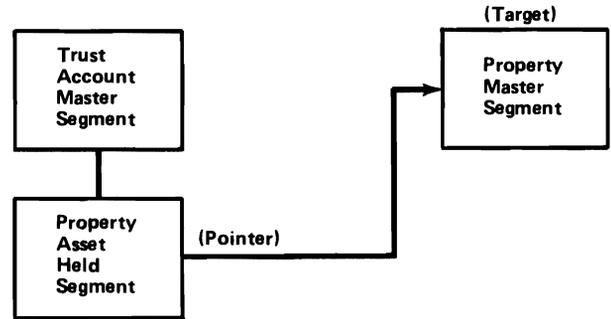
Figure 8-17. Logical data structure with pointer segment—target segment relationship

The pointer segment—property asset held segment—can contain the intersection data. The logical data base structures that can now be derived are shown in Figure 8-18.
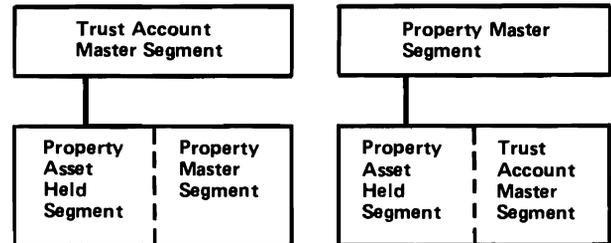
Figure 8-18. Logical data base structures with intersection data

The physical data base records and the logical relationships for the trust and property data structures are depicted in Figure 8-19.

Figure 8-15. Physical data base records and logical interrelationships

The questions concerning the properties held in a particular trust account and the trust accounts holding a particular property can be answered with the logical data structures illustrated in Figure 8-16.

Figure 8-16. Logical data structures showing properties and trust information
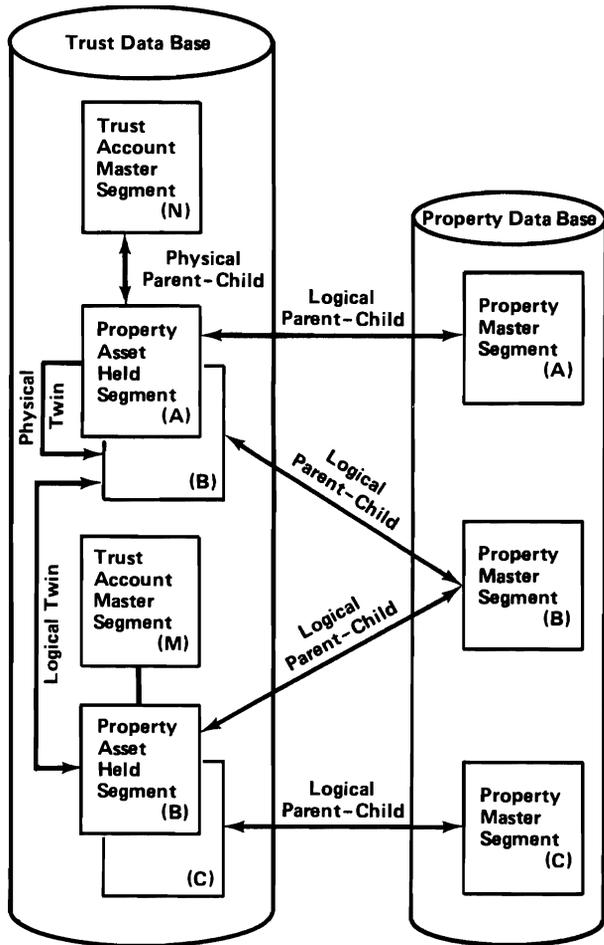
Figure 8-19. Physical data base records and logical relationships

Of course, additional segments can be added to the logical data base structures above that do not participate in pointer segment—target segment relationships. Under the property asset held segment type might exist a lot segment type. The lot segment might contain data pertinent to a particular buy or sell of the property, such as date of trade, broker, how acquired, unit price, and total value of trade. Dependent to the savings master segment type might be segments that describe deposits and withdrawals on a savings account. Dependent to the property master segment type might be segments that describe different property types, such as stocks, bonds, notes, mortgages, and contracts.

If HIDAM is used for storage of the customer information and account data base, Figure 8-20 illustrates the data base organization and access method.
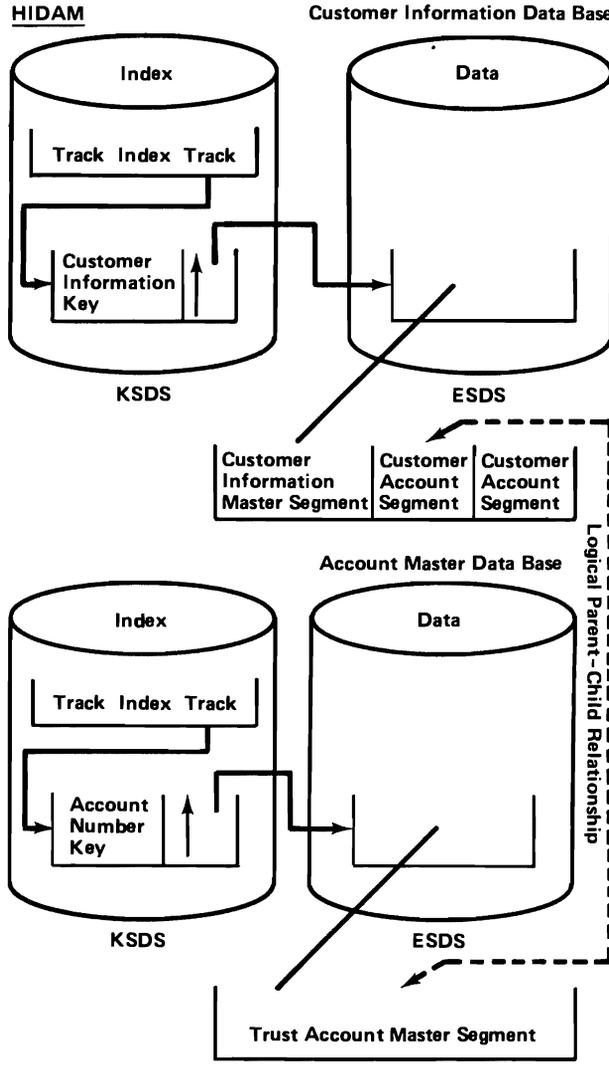
Figure 8-20. Data bases stored using HIDAM

Medical Industry

In the medical environment, a typical use of DL/I might be the storage of medical information about patients in a hospital or clinic. Associated with each patient is one data base record with the root segment containing basic information about the patient. This patient master segment may be keyed on patient identification such as social security number. In addition, it contains name, address, age, birth date, sex, and race. (See Figure 8-21.)

Patient Master Segment

Figure 8-21. Medical data base record root segment—patient master segment

For each visit to the hospital or clinic, a visit segment might be appended as a dependent segment to the patient master segment, as shown in Figure 8-22.
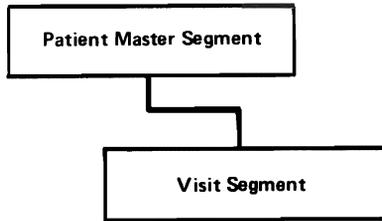
Figure 8-22. Logical data structure with one dependent segment

The visit segment might contain the date and purpose of the visit as well as the attending physician's name. Information obtained during the visit might cause the physician to make a diagnosis of the patient's problem. It is possible to consider a diagnosis segment dependent to the visit segment, as shown in Figure 8-23.
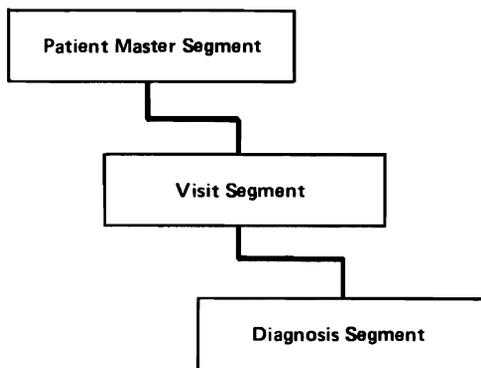
Figure 8-23. Adding second dependent segment (diagnosis segment) to medical data base record root segment

Certain visits might involve surgical operations performed on a particular site of the human body. The inclusion of site and surgery segments under visit segment may be considered. An alternative approach may be to consider the site segment directly dependent to the patient master segment and the surgery segment dependent to the site segment (see Figure 8-24).
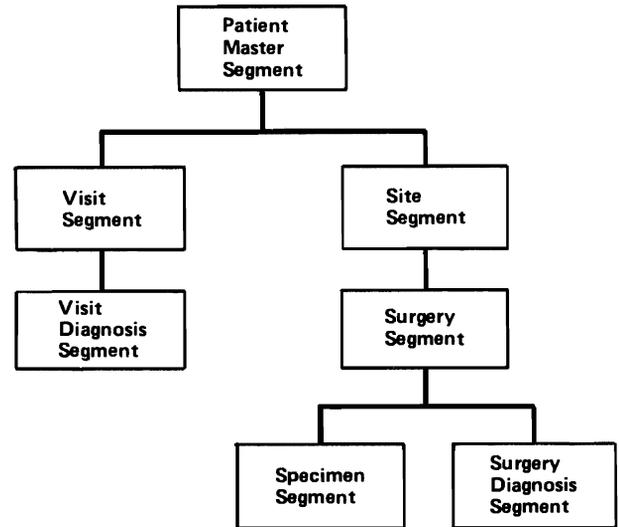
Figure 8-24. Logical data structure of medical data base record or data base

The second approach is more effective if multiple surgeries are performed on a site at different visits. The relationship between a particular visit and a particular surgery can be achieved through a field in the visit segment relating to a particular surgery.

It would be appropriate to consider specimen and diagnosis segments under the surgery segment. These would be pertinent to a particular surgery.

If the visit to a clinic or doctor does not involve surgery but involves X-ray or radioactive treatment, additional segment types may be considered as dependent from the site segment. If a visit involves only the application of, or prescription for, a drug or medicine, a drug segment type might be considered dependent to a visit segment.

Questions that might be asked of information contained within a data base of the structure depicted in Figure 8-24 can be of a simple or complex nature. The simple type of question to answer would be a request for information about a particular patient. The answers to questions of this simple type are facilitated because the data base is structured on patient identification sequence. Online terminal inquiry and update are quite practical.

A complex question might involve listing patients or information about patients who received a particular drug, contracted a particular disease, or satisfy a combination of such query criteria. The use of pointer-target segment relationships can assist in answering these questions. Additional

tree structures can be created that interrelate with the patient tree structure. These tree structures could be associated with drugs, diseases, or other search arguments.

Process Industry

Online Order Entry/Production Control System
A total order entry/production control system has computer control of production from acceptance of orders through manufacturing, shipment, and delivery of the order. An implementation plan for these applications can be developed from the functional relationship that exist between these functions. The plan allows for a logical growth pattern through the implementation of online order entry, in-process inventory control, and plant balancing.

One of the major requirements in the installation of such a large-scale system is the ability to implement the individual program in modular increments. Each increment should gradually increase the functional scope of the system without the necessity of reprogramming previously written programs that utilize the same data base. The existing data bases should grow to service the new and additional applications.

A good beginning for a total system is an online order entry application that includes all of the processing operations necessary to accept orders from customers and provide the necessary follow-up until the entire order has been shipped. The initial phase of online order entry would include acceptance of orders for stocked items. A following expansion would place additional information in the data bases that would permit acceptance of orders for items that necessitated initiation of a mill order to produce the item or items requested. When this application is added, it is necessary to add programs that check in-process inventory, operation routing, and facility loading information. It would also be necessary to create mill order plan records. The organization of these required records is shown in the examples.

In addition to the functions performed by the online order entry, the in-process inventory control programs would provide basic material control and order status. Plant balancing programs that would balance long-range scheduling objectives with short-range sequencing objectives in order to optimize production in relation to customer orders could subsequently be added to the application.

If the data bases illustrated and the programs described were implemented, the following types of questions could be answered on communication terminals: What is the availability of a stocked product requested by a customer? What ship date can be promised for an item that requires manufacturing? What is the credit limit of a given customer and what is the total amount of unpaid invoices? What is the status of an existing customer order?

In addition, many changes to the information contained in the data bases can be entered from the communication terminals. This would include the following types of transactions:

- Enter receipt of new stock

- Provide notification of low limits of inventory

- Add new items to an existing order

- Change the quantity previously entered in a customer order

- Change the ship-to location of an existing order

The following logical data structure (Figure 8-25) can be used as a base to install the described applications, permit the entry of data as listed, and answer the types of inquiries described.
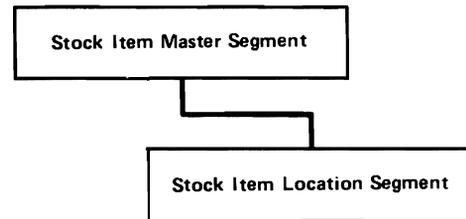
Figure 8-25. Logical data structure for stock item data base

Figure 8-25 illustrates a logical data structure in which a stock item master segment exists in a data base for each stocked item. Subordinate to a stock item master segment, one or more stock item location segments exist. These segments describe inventory locations where a given quantity of the stocked items exists and is available.

The question concerning the ability to fill an order from stock can be answered with the data structure in Figure 8-25 by making inquiries against the described stock item records.
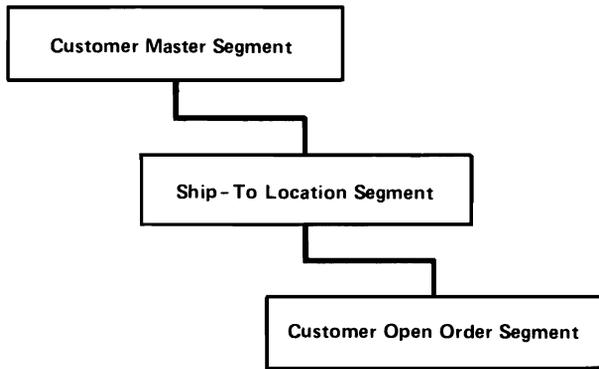
Figure 8-26. Logical data structure for customer master data base

The ability to do a credit check on a particular customer requires availability of information on each customer. Let us consider the logical data structure in Figure 8-26. The customer master segment is keyed upon a unique customer identifier and contains information such as credit clearance level, name and address, and total amount of unpaid invoices. This segment provides the answer to the credit check. In addition, this customer may have one or more locations to which he wishes the orders to be shipped. The ship-to location segments provide this information. The customer open order segments indicate all open orders for a particular customer and a particular location. These segments provide the pointers to the details of each open order. This open order data structure describing each open order, Figure 8-27, is required in addition to the stock item and customer information data structures.
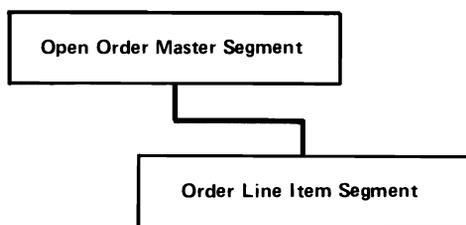
Figure 8-27. Logical data structure for open order data base

This logical data structure would contain an open order master segment, which is keyed upon order number. Subordinate to the open order master segment are one or more segments describing each line item in an order.

The open order master segment contains the status of the order, the due date, the customer name, and the customer's ship-to location for the order. Using this logical data structure and the stock item data structure previously discussed, order status inquiries can be answered.

Since the customer master segment and ship-to location segment data structure represents stable data, the user may select the hierarchical sequential organization and HISAM.

The stock item data structure is more volatile with the updating or inserting and deleting of stock item location segments. The user may select the hierarchical direct organization and HIDAM. One of the reasons for this selection is reuse of deleted segment space with a hierarchical direct organization data base.

The open order data structure is quite volatile. All segments associated with an order exist only for the life of the order. If an order is modified, one or more segments may be updated, inserted, or deleted. In addition, frequent inquiry against the structure may be required for determining order status. Here the user may consider the hierarchical direct organization and HDAM. Figure 8-28 depicts the three logical data structures stored in the suggested organizations with the indicated access techniques.

If this application is expanded to include in-process inventory items as well as stocked items for handling orders, the additional questions might be asked:

What is the status of an open order that required initiation of a mill order to produce the item?

What is the routing of operations performed in the processing of a given product?

What is the workload on a given plant facility? If a particular plant facility breakdown occurs, can another facility be used to complete the order?

If a plant facility breakdown occurs, what is the effect on an order's status?

In addition to answering questions such as those stated above, the following data structures can allow for data base update processing to assure maximum plant facility usage and minimal time until customer order availability.

Figure 8-29 describes the mill order plan and routing relationship. The mill order plan segment is keyed on mill order item number and includes a plan of manufacture and time schedule. The in-

HISAM — Customer Master Data Base

Figure 8-28. Physical data bases under HISAM, HDAM, and HIDAM

process inventory segment includes the status of a mill order. Each operation routing segment to be under an in-process inventory segment describes
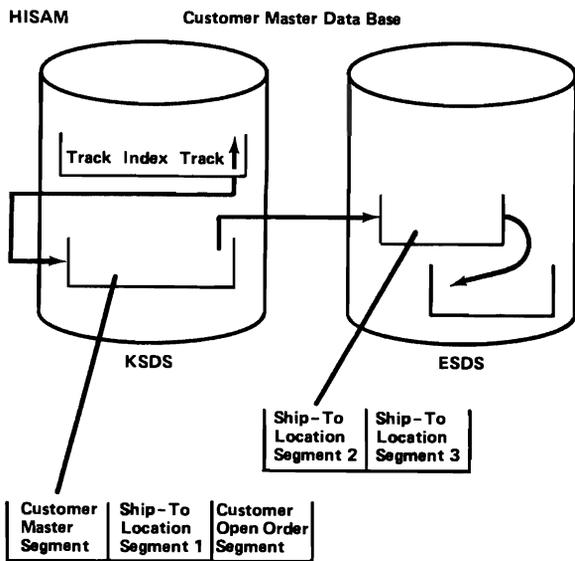
an operation performed in the process of producing the mill order item and the facility at which the operation is to be performed.
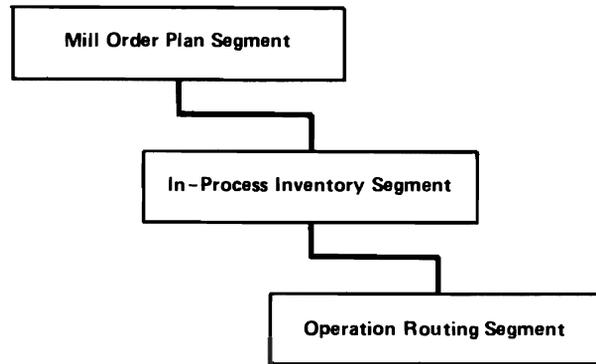
Figure 8-29. Logical data structure for mill order planning
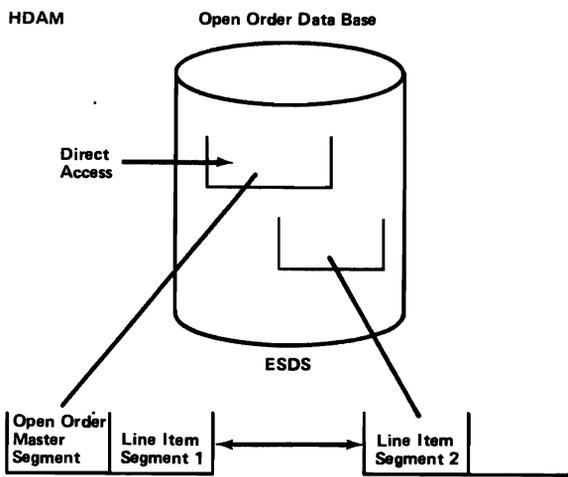
Figure 8-30 describes plant facility segment, operation routing segment, in-process inventory segment, and mill order plan segment.
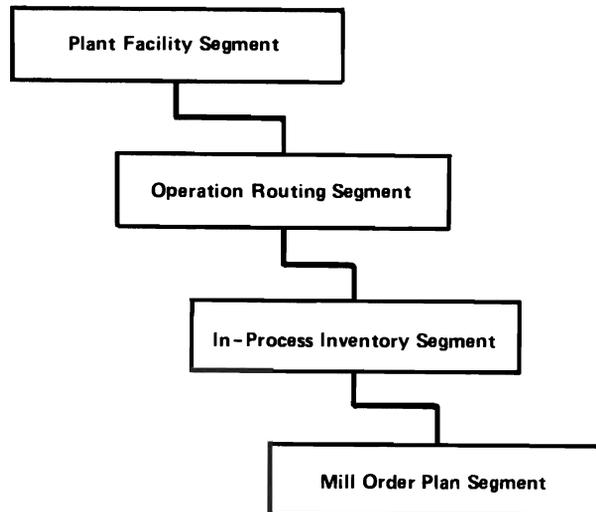
Figure 8-30. Logical data structure for plant facility

The plant facility segment is keyed on facility number and may contain facility loading data, such as total time scheduled, etc. The operation routing segment contains the individual operations performed on a given mill order item at that facility. Figures 8-29 and 8-30 are actually inverse data structures of each other and can be considered for physical storage through the pointer segment—target segment concept. Figure 8-31 restates Figures 8-29 and 8-30 using the pointer—target concept.
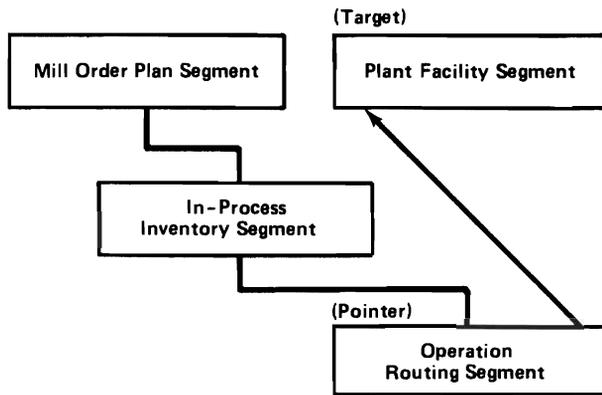
Figure 8-31. Logical data structure for mill order plan and plant facility using pointer—target concept (Figures 8-29 and 8-30)

Referring to Figure 8-32, consider the mill orders A and B with operations performed at plant facilities X and Y. Figure 8-29 depicts the physical data base records and logical relationships.

In addition to these data structures, a relationship must be stated from the open order data base to the mill order plan segments. This can be achieved with a new segment in the open order data base indicating the pertinent mill order plan segment.
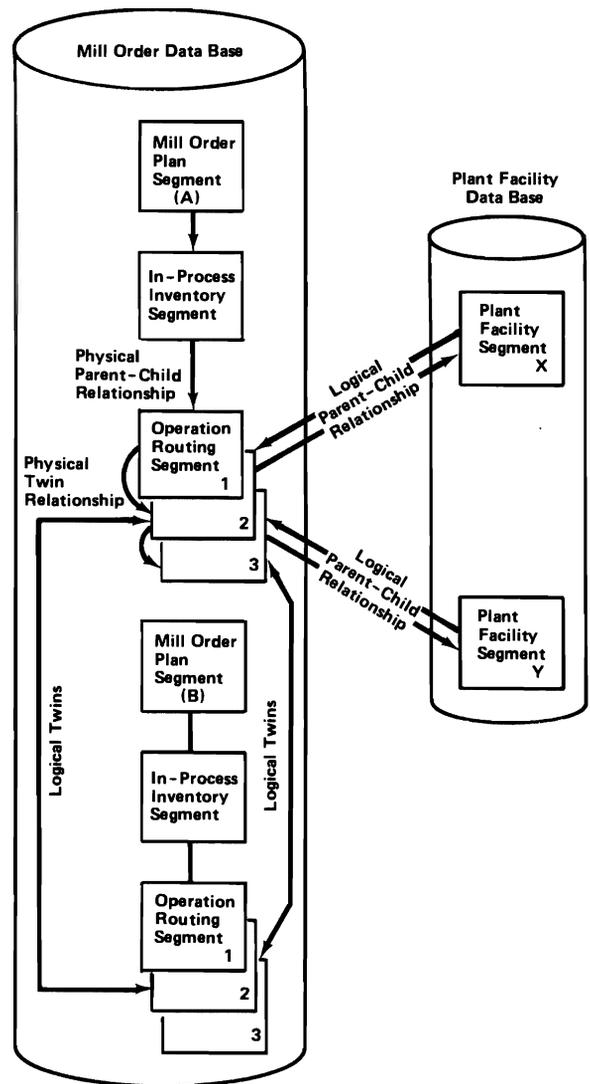
Figure 8-32. Physical data base records and logical relationships—mill order plan and plant facility

Appendix A: Comparison to IMS/VS and DL/I-Entry DOS/VS

The following chart illustrates which major features of IMS/VS are also contained in the smaller, upward compatible systems DL/I-Entry DOS/VS and DL/I DOS/VS Versions 1.1, 1.2, and 1.3.

Feature	E	1.1	1.2	1.3	IMS
Batch processing partition (region)	x	x	x	x	x
Telecommunication facility					
(Integrated)					x
(CICS/VS Interface)	x	x	x	x	x
Message scheduling facility					x
Data base recovery facility		x	x	x	x
Checkpoint/restart facility (Checkpoint for DL/I)				x	x
Data base description (DBD) generation	x	x	x	x	x
Program specification block (PSB) generation	x	x	x	x	x
System generation (CICS/VS online only)		x	x	x	x
Preformatted control blocks		x	x	x	x
Multiple Partition Support			x	x	x
Program Isolation				x	x

Data Base Support

Feature	E	1.1	1.2	1.3	IMS
Simple hierarchical sequential access method	x	x	x	x	x
Hierarchical sequential access method (HSAM)	x	x	x	x	x
Simple hierarchical indexed sequential access method	x	x	x	x	x
Hierarchical indexed sequential access method (HISAM)	x	x	x	x	x
Hierarchical direct access method (HDAM)	x	x	x	x	x
Hierarchical indexed direct access method (HIDAM)		x	x	x	x

Data Management Support

Feature	E	1.1	1.2	1.3	IMS
VSAM	x	x	x	x	x
ISAM					x
OSAM					x
BSAM					x
QSAM					x
SAM	x	x	x	x	x

Application Program Language Support

Feature	E	1.1	1.2	1.3	IMS
PL/I	x	x	x	x	x
COBOL/VS	x	x	x	x	x
Assembler	x	x	x	x	x

Data Base Structures and Processing

Feature	E	1.1	1.2	1.3	IMS
Physical data bases	x	x	x	x	x
Logical data bases	x	x	x	x	x
Fixed-length segments	x	x	x	x	x
Variable-length segments		x	x	x	x
Data compression of segments		x	x	x	x
Twin pointing for HDAM and HIDAM		x	x	x	x
Hierarchical pointing for HDAM and HIDAM					x
Secondary indexing	x	x	x	x	x
Multiple segments in hierarchic path with a single call		x	x	x	x
Multiple data set groups					x
Distributed free space for HDAM and HIDAM				x	x
Boolean logic in SSAs					x
Command codes		x	x	x	x

Index

GH20-1246-4

IBM

®

DL/I DOS/VS
General Information
GH20-1246-4

**READER'S
COMMENT
FORM**

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? _____

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)
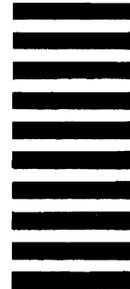
GH20-1246-4

Reader's Comment Form

Cut or Fold Along Line

DL/I DOS/VS General Information   Printed in U.S.A.   GH20-1246-4

GH20-1246-4

DL/I DOS/VS General Information   Printed in U.S.A.   GH20-1246-4