EVALUATING DATABASE MANAGEMENT SYSTEMS: A FRAMEWORK AND APPLICATION TO THE VETERAN'S ADMINISTRATION HOSPITAL

by

MOHAMMAD DADASHZADEH
S.B., Massachusetts Institute of Technology
(1975)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February, 1978

Signature of Author: .....
Department of Electrical Engineering and
Computer Science, December 30, 1977

Certified by: .....
Thesis Supervisor

Accepted by: .....
Chairman, Departmental Committee on Graduate Students

EVALUATING DATABASE MANAGEMENT SYSTEMS: A FRAMEWORK AND APPLICATION TO THE VETERAN'S ADMINISTRATION HOSPITAL

by

MOHAMMAD DADASHZADEH

Submitted to the Department of Electrical Engineering and Computer Science on December 30, 1977, in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

The primary purpose of this thesis is to aid the database management system (DBMS) evaluation process by providing an example in a real-life setting - the design and implementation of a DBMS-based hospital information system.

A guideline is presented in whose context the capabilities of a specific package, Generalized Information Management (GIM-II), are examined in detail. These capabilities are then evaluated in light of the requirements of a hospital information system. The design and implementation of a medical database and such hospital applications as patient scheduling and pharmacy are investigated.

It is concluded that the GIM-II system can support a hospital information system.

Thesis Supervisor: Stuart E. Madnick

Title: Associate Professor of Management

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Stuart Madnick for his guidance and extreme patience in supervising this thesis. I am indebted to my parents for their continued support, both financial and spiritual, throughout my academic years.

TABLE OF CONTENTS

Page

ABSTRACT 2

ACKNOWLEDGEMENTS 3

TABLE OF CONTENTS 4

LIST OF FIGURES 7

1. INTRODUCTION 8

1.1 The Selection and Evaluation Process 8
1.2 Thesis Objective 11
1.3 Thesis Approach 12
1.4 Thesis Organization 13

2. MINICOMPUTER OPERATING SYSTEMS 14

2.1 What is an Operating System? 14
2.2 Components and Functions 15
2.3 System Types 16
2.4 Real-Time Multiprogramming System (RSX-11D) . 20
2.4.1 System Summary 23

3. THE MEANING AND ORGANIZATION OF DBMS . . . 26

3.1 Historical Evolution 26
3.2 The DBMS Concept 28
3.3 Requirements of a DBMS 29
3.4 DBMS Organization 31
3.5 Minicomputer DBMS 35
3.5.1 Systems Provided by a Software House . . 37
3.5.2 Systems Provided by the Hardware Vendor . 37
3.5.3 The Limiting Factors 38

4. DATABASE MANAGEMENT SYSTEM FEATURE DESCRIPTIONS 42

4.1 General Description 44

TABLE OF CONTENTS (continued)

		Page
4.1.1	History of Software	44
4.1.2	Availability and Cost of Software	45
4.1.3	Type of System	46
4.1.4	Modes of Operation	48
4.2	Computer Environment	49
4.2.1	Main Frame, Operating System, Core Requirements and Source Language	49
4.2.2	Teleprocessing and Concurrent Usage	51
4.2.3	Mass Storage	53
4.3	Database Structure	54
4.3.1	Logical File Organization	54
4.3.2	Physical File Organization	58
4.4	Data Definition	60
4.5	Updating and Data Integrity	64
4.5.1	Update Language	64
4.5.2	Batch Update	68
4.5.3	Bulk Update	69
4.5.4	System Triggered Update and Retrieval	71
4.5.5	Data Integrity	74
4.5.5.1	Data Validation	74
4.5.5.2	Concurrent Update	75
4.6	Query Capabilities	77
4.6.1	Query Language	77
4.6.2	Selection Criteria	80
4.6.3	Qualifications on Searches	83
4.6.4	Multi-File Searching	85
4.6.5	Predefined Queries	86
4.6.6	Aids on Search Formulation	87
4.7	Output Presentation and Report Generation	88
4.7.1	Default Formatting of Reports	88
4.7.2	Report Format Specification	90
4.7.3	Output Media Flexibility	92
4.7.4	Arithmetic Capability	93
4.7.5	Sorting	96

TABLE OF CONTENTS (continued)

Page

4.7.6 Data Extraction 97
4.8 Security and Error Recovery 98
4.8.1 Database Level Security 99
4.8.2 File and Field Level Security 101
4.8.3 Record Level Security 103
4.8.4 Hardware Security 104
4.8.5 Protection Against Direct Access to the Database 105
4.8.6 Backup and Recovery 106
4.9 Database Restructuring 109
4.10 Application Programming 111
4.11 System Generation 115

5. HOSPITAL INFORMATION SYSTEM: AN APPLICATION FOR THE DATABASE APPROACH 119

5.1 Hospital Activities and Hospital Information System . 119
5.2 Background: Hospital Computer Systems 120
5.3 The Objectives and Requirements of a Hospital Information System 124
5.4 The Architecture of a Hospital Information System . . 127
5.4.1 Approach 127
5.4.2 Architecture 128
5.5 Designing the Hospital Information System 131
5.5.1 Medical Records 131
5.5.2 Patient Scheduling and Booking 136
5.5.3 Bed Census 152
5.5.4 Pharmacy 154
5.6 Assessing the Results 160

6. SUMMARY 163

REFERENCES AND BIBLIOGRAPHY 166

LIST OF FIGURES

Figure No.		Page
1	Coupling between two files.	56
2	Coupling between three files.	57
3	GIM-II Architecture	117
5.1	Hospital Information System Architecture . . .	129
5.2	Master Patient Record Structure	134
5.3	Inpatient Record Structure	135
5.4	Unbooked Time Slots File Structure	140
5.5	Medical Procedure File Structure	141
5.6	Appointment-File Record Structure	142
5.7	Bed Census File	153
5.8	Drug File Record Structure	156
5.9	Drug Profile Record Structure	157
5.10	Inpatient Drug Profile Record Structure . . .	158
5.11	Coupling of various files.	159

CHAPTER 1

INTRODUCTION

The rationale and qualification for a database approach to provide effective management of data in an organization has been given elsewhere in the literature. In this study we are concerned with the problems of selection, evaluation, and installation of a commercially available database management system (DBMS). Any study, such as this, is highly situation-dependent. For this reason, a real-life application will be considered. The approach used will then show how to get started.

1.1 The Selection and Evaluation Process

The selection of a database management system is a major undertaking which can have prolonged effects on the organization. Not all implementations of DBMS have been successful, and some systems being marketed today may not be considered successful in several years.

An important step in the selection process is a thorough understanding of user requirements. The inability of a DBMS to satisfy even a relatively minor user need may create a severe data processing problem. At the same time, one must be careful not to confuse a need with convenience. For example, a prospective user may assert a need for online response to his query, but in fact his need is for response within two hours, or even overnight. On the other hand, user needs frequently will be in conflict with each other. For example, the need for a simple user language is in direct opposition to the need for complex data structures.

The more powerful the data structures being manipulated, the greater the complexities of user language. Similarly, a powerful online update capability may oppose data security and data integrity requirements, or even an economical checkpoint/restart facility. Therefore, tradeoff and refinement of user requirements is needed to insure that all significant conflicts have been taken into account.

Once the issues confronting the potential users of the DBMS have been thoroughly understood, an analysis of these organizational requirements in light of the capabilities provided by database management systems must be undertaken. Such an analysis would map user requirements into DBMS capabilities - typically in the form of one-to-many or many-to-one mappings - and would establish a set of system capabilities that can be weighted and used to evaluate various vendors' systems.

An important aspect of the evaluation process is the consideration that must be given to the operating environment into which the database management system will fit. The current or proposed environment for the DBMS exists through the interaction of the hardware, system software, and packaged software. An evaluation of the hardware must answer questions concerning its suitability as a vehicle for database management. For example, does the hardware have the required speed as well as sufficient capacity of both main memory and secondary storage? Can the hardware support disk storage, and online terminals? How many? Some database management systems have been tailored to operate on certain computer systems, in order to take advantage of particular hardware features available on these systems. On the other hand certain

hardware capabilities are essential. For example, memory protection hardware.

The most difficult step in the evaluation process, however, is the implications of the interface between DBMS and the operating system software. For example, is it necessary to modify the operating system in order to make it compatible with the DBMS? What modifications are advantageous to the DBMS? Which functions are handled by the DBMS and which will be passed on to the operating system?

Here, again, there is the problem of a DBMS which has been designed to run under a particular operating system. If there is less capability in the organization's computer system than the hardware/software system which was assumed in the design, there may be serious degradation in the performance of the DBMS. On the other hand, a more powerful system than the one assumed in the design may cause waste of resources. For example, if a DBMS is constructed using overlays, assuming a non-virtual machine, and then installed on a virtual machine, both the operating system and the DBMS would employ swapping between main memory and secondary storage resulting in twice the amount of work needed, if the size of the physical page is not the same as the size of the overlay.

One crucial problem in the database management system evaluation process is to answer the question: given a query on the database, what resources will the query demand?

The operating system functions called upon by the DBMS, in particular

for the execution and control of I/O operations, have a major impact on the system's response time. Identification of the CPU, memory, I/O, and other resource allocation demands of a query, in a real-time multi-programmed computer system, is perhaps the most difficult process in the evaluation of the performance of a DBMS.

1.2 Thesis Objective

As we have said, when an organization considers the implementation of a particular DBMS, it is bound to find the need for compromising some of the user requirements. This situation arises for the following reasons:

A. User needs are often in conflict with one another.

B. The general purpose capability of the package may not meet the special purpose requirements of all applications.

C. Consequences of restrictions from technical feasibility or from lack of imagination on the part of the DBMS designers.

The refinement of conflicts and conformation to the generalized nature of the package, while significant, are relatively easy to attain. However, appreciating the ramifications of the last point, which are directly responsible for the successful implementation and acceptability of the system, does need a detailed technical study.

The objective of this thesis is to aid in the DBMS selection and evaluation process by:

1. Providing a framework for evaluating DBMS capabilities.
2. Providing detailed evaluation of the capabilities of a specific DBMS in light of the guideline presented.
3. Providing the management with sufficient detail to appreciate the fact that some difficult technical problems underlie the features of a DBMS.
4. Providing an example of the DBMS application to a hospital information system.

1.3 Thesis Approach

The selection and evaluation of database management systems has received much attention in the literature. There have been numerous surveys of database packages, covering from six to as many as 154 systems.[1-8] However, we feel that most studies in their treatment have been analyses of systems isolated from any real-world application. Our thrust here is to evaluate a certain package in light of a real-life application - a hospital information system. Our approach is to concentrate on examining the capabilities of the DBMS and employ the results in order to determine how a hospital information system can be designed and implemented using this DBMS. As such, we will not be concerned with the selection process. However, we shall address the issues involved and will present a guideline of DBMS capabilities that can be used in a selection process.

We have chosen as the vehicle for this study a specific package - TRW's Generalized Information Management (GIM-II)[9-12] and ASSIST communica-

tion software operating on a PDP 11/70 under the operating system RSX-11D. The choice coincides with that of the Veteran's Administration Hospital. The V.A. is contemplating the implementation of a minicomputer-based information system that employs copies of GIM-II as the common DBMS. The initial efforts are envisioned to be directed towards the establishment of a medical database and the automation of such functions as patient scheduling and pharmacy. In this thesis we present the issues regarding the suitability of the package, based on the installation's ultimate needs.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents a brief comparative study of minicomputer operating systems and summarizes the specific operating system environment which shall be utilized. In Chapter 3, we describe the overall DBMS organization and some of the implementation decisions involved. In addition, we present a discussion of minicomputer database software, and the limitations imposed due to the architecture of minis. Then in Chapter 4, we present our guideline for evaluating DBMS capabilities: the questions that need to be asked, the extent to which current technology supports each feature, and the influence of design decisions on the performance of the system. In light of the guideline presented, GIM-II's capabilities are examined in detail. Chapter 5 considers the design and implementation of a DBMS-based hospital information system. The chapter concludes with recommendations as to the appropriateness of the GIM-II system in view of the installation's ultimate needs. Finally, Chapter 6 summarizes the general issues.

CHAPTER 2

MINICOMPUTER OPERATING SYSTEMS

A database management system is intended to be utilized in an operating system (OS) environment that performs scheduling of jobs, processes input/output operations, manages auxiliary storage, and provides conventional data management facilities. The DBMS exists as a set of operational programs analogous to the control programs of an operating system. The overall operation of the DBMS, its scheduling and relation to other jobs running on the system, is usually under the supervision of the operating system. Many of the operations that need to be carried out for the functioning of a DBMS (scheduling of application programs working with the database, database access) could be done either by the OS, or by the DBMS itself. There are several methods by which a division of responsibility can be accomplished.[3,8] In the GIM-II system, only the operation of the overall DBMS is under the control of the OS; it runs as a normal application program under the host OS, it translates user requests and decides which application program should process it, it controls programs without any involvement of the OS, and it uses only the lower level facilities of the OS to access the contents of the database. This chapter is intended to provide the necessary background on operating systems, especially those available on minicomputers. Digital Equipment Corporation's RSX-11D is presented in more detail. The objective is to establish a point of reference for subsequent chapters.

2.1 What is an Operating System?

An operating system refers to a comprehensive group of routines that control the operation of a computer installation. Its principal objectives are to enable a number of users to share the computer system efficiently

ane reliably; to facilitate convenient and efficient running of programs on the computer; to minimize operator intervention and CPU delays and maximize throughput. To do so, the operating system is given a hierarchy of components which provide the level of management necessary to handle each task assigned.

2.2 Components and Functions

The software constituting an operating system consists of an executive (or monitor) routine and a number of system utilities that are run under the control of the executive. The executive acts as the primary interface between the hardware and a program running on the computer, and between the hardware and the people who use the system. When an executive is loaded into memory and started, its first duty is to interface with the operator running the system. The executive waits for the operator to request some service, and then performs that service. These services typically comprise loading and starting programs, controlling program execution, and system maintenance.

Most minicomputer executives perform job scheduling and monitoring; storage allocation and memory management; control over the activities of I/O devices; interrupt handling and error processing; and handling communication between users and operating system as well as that between the system and the operator.

Because of the complexity of tasks involved, the coding for an executive program can occupy a significant portion of the computer's memory.

With the concern for space in small systems, minicomputer executives are usually divided into two distinct parts: a permanently resident portion and a transient one. The transient portion includes the loader, the dump routines, and the operator communication routines. Once the executive initiates another program's execution, the transient portion can be over-written or swapped out, while the resident portion remains available to act on requests from the program.

System utilities enhance the capabilities of an operating system by providing processing support services in the following areas: program development, file management, and system management. Program development utilities include text editors, assemblers and compilers, linkers, program libraries, timing services, and testing and debugging aids. File management utilities provide support for system files as entities rather than for the individual records within the files, and include file copy, transfer, and deletion programs. System management is concerned with system generation, updating the operating system in response to the changing needs of the installation, system status interrogation routines, and logging and accounting programs.

2.3 System Types

This section surveys the range of minicomputer operating systems from the viewpoint of resource management.[13-15]

1. Single-user operating systems manage their resources based on the demands from a single individual. Such cases are found in the

application of minicomputers to scientific or process control program development. The operating system is primarily designed to run on a minimum configuration system, providing only the essentials for controlling the operation of a minicomputer. The advantage of using such a system over no operating system lies mainly in the fact that once the executive is in memory, all further interaction with the system can be performed from the operator's console instead of having to work with the minicomputer front panel switches.

2. Single-job operating systems execute a program until either it is completed or it is interrupted by the operator. Only one program is active at any time, and as such it has exclusive use of all the system's resources.

3. Foreground/background operating systems provide the sharing of the system's resources between two independent programs that are co-resident in memory. One of these programs (foreground program) is serviced with a higher priority than the other (background program). Foreground program operation is usually bound by the speed of a terminal device (e.g., a card reader) or the activity of a device through which data is only available sporadically (e.g., a communications terminal). When the foreground program becomes limited by the unavailability of its limiting device, it relinquishes control to the background program. The background program is allowed to execute until the foreground program again requests services.

4. Multiprogramming operating systems are an extension of the foreground/background concept in which many jobs, instead of only two, compete for the system's resources. While one job has been temporarily interrupted (generally to service an I/O request), another task can have control of the CPU. There are basically two approaches to multiprogramming: priority processing and parallel processing. In the former approach, the processing and I/O demands of a hierarchy of several programs are being serviced strictly on a priority basis. Within each program two or more levels of processing priority may be defined (e.g., real-time I/O, non-real-time I/O, processing). All peripheral devices in the system issue interrupt signals that are intercepted by the interrupt-handler routine which determines, on the basis of a priority-ordered queue of programs, which I/O or processing demand is to be fulfilled next. The executive also assumes control when a processing task is unable to proceed; in this case, the job is momentarily suspended and control is transferred to the next higher priority task. Furthermore, a high-priority task can interrupt a lower-priority task if it requires immediate service.

   In the parallel processing approach, the I/O and processing demands of a collection of programs are being serviced strictly on an equal opportunity basis. Peripheral interrupt signals are controlled in the same fashion as in the priority processing approach; however, the algorithm for fulfillment of demands is completely different. Input/output requests are serviced in a manner to optimize the utilization of various hardware facilities (e.g., minimize arm move-

ment on disk), and to dquate service distribution between concurrent programs. CPU demands are likewise fulfilled bo insure optimal balancing between programs. This approach is especially suited for larger installations in which many system facilities can be utilized simultaneously.

5. Time-sharing operating systems allow multiprogramming with the parallel processing approach in an interactive environment. The objectives are the optimization of both hardware and user productivity. Each job is given a certain amount of CPU time (called a time slice), at the conclusion of which it is interrupted and control is given to the next waiting job. Because of its high internal speed, the processor can service a number of programs on a time scale which seems instantaneous to each user. The system may assign priority levels to jobs, depending on whether they are computer-bound or I/O bound. Furthermore, the user may be allowed to specify the minimum guaranteed time for his job, as well as modifying its priority. When the aggregate user requests are not sufficient to saturate system capacity, the excess time can be devoted to the processing of a batch operation in the background mode.

6. Real-time operating systems are found in applications where the response of the system must keep step with external events; for example, laboratory automation or process control. Typically, various peripherals (such as paper-tape readers, teletypewriters, remote terminals, analog to digital converters, touch-wires) sporadically enter data pertinent to some supervisory control program.

The interrupt servicing mechanism interprets the interrupts, bringing into memory the appropriate program for execution. When two interrupts occur virtually concomitantly, some priority structure is needed to decide which is to be executed first. This interrupt priority is not necessarily related to program priority which refers to the sharing of CPU time between a number of programs according to some scheduling algorithm. Program priority in real-time systems is usually established by associating a next activation time with each program; when a program is found that is due to run, it is brought into memory for execution. The real-time executives also allow the user to develop application software (in a background mode) concurrent with the real-time operation. Overall, such systems often resemble a minimal time-sharing system.

2.4 Real-Time Multiprogramming System (RSX-11D)[15, 16]

RSX-11D is a multipurpose operating system designed to support multiple requests for services while maintaining real-time response to each demand.

To the user and to computer operations personnel, a unit of work is the job. To the executive, the basic unit of work is called a task. More specifically, a task consists of one or more programs written in a source language (such as MACRO or FORTRAN), assembled or compiled into an object format, and then built into a task image by a control program called the Task Builder. In addition to the normal linkage functions of combining object modules or creating overlays, the Task Builder sets up

the task control information that determines the task's resource requirements and relation to other tasks in the system. The significant task attributes are:

- Partition - the contiguous area of memory where the task will reside when it executes.
- Priority - the task's relationship to other tasks, used to resolve contention for system's resources.
- Checkpointability - the task's ability to be swapped out of memory when a higher priority task requests the partition in which it is active.

Once a task is built, it can be installed in the system and executed. Task installation refers to registering its control information with the system; the task is neither in memory nor in competition for system's resources. A task is activated by the operator or another active task in the system.

When an installed task is activated, the system allocates the necessary resources, brings it into memory for execution, and places it in competition with other active tasks. In this manner, when a task is needed to service a real-time activity, it can be introduced into the system quickly since its basic parameters are already known.

RSX-11D uses the variable-sized partition concept for storage allocation. Each task is assigned a partition in which it is constrained to operate,

and all partitions can operate in parallel. There are two types of partitions: user controlled and system controlled. A user controlled partition is allocated to only one task and the user assumes memory management in this type of partition. The Task Builder makes it possible for the user to build overlaid tasks and call these overlays from disk.

A system controlled partition is intended for the execution of tasks where the user wishes the system to handle the allocation of memory. The executive controls and allocates the partition dynamically to contain as many tasks as possibly will fit. Tasks are brought in from the disk on a priority basis and are loaded into the first available contiguous memory area in the partition. At the termination of a task, the memory space it occupied becomes available again. Core fragmentation is handled by automatic memory compaction; tasks are moved to obtain a large enough area in the partition to load another task. Dynamic expansion of a particular task beyond its originally specified region can also be accomplished upon specific request.

Central processor control is assigned to active tasks (a task is considered active from the time it execution starts until the time it has exited) according to their priorities. A task's default priority (ranging from a low of 1 to a high of 250) is set when the task is built. Once a task is activated, it runs to completion or until a significant event occurs; typical events might be input/output completion or an external interrupt. When a significant event is declared, the executive interrupts the executing task and searches for a task capable of executing. The highest

priority active task that has all the resources it needs, and can make use of them, gains control of the CPU.

2.4.1 System Summary[15]

System type	Large, multi-user, general-purpose system for concurrent real-time applications, program development and general data processing.
CPU's supported	PDP-11/40 with Extended Instruction Set and Memory Management PDP-11/45 with Memory Management PDP-11/70
Memory ranges	Minimums: 48K words for little or no program development 56K words for simultaneous applications execution and program development Maximum: 124K words on PDP-11/40 and PDP-11/45 1024K words on a PDP-11/70
Additional CPU hardware supported	PDP-11/40 Floating Point Unit PDP-11/45 Floating Point Processor PDP-11/70 Floating Point Processor
Minimum peripherals	Console terminal with two disk drives

Additional peripherals	Fixed-head disk system
	DECtape system
	Cassette system
	Line printer
	Card reader
	Paper tape reader/punch
	Laboratory Peripheral System
	Industrial Control System
	Analog/Digital Converter
	Terminal/Line Interfaces
	Programmable Clock
System utilities	Line Text Editor
	Source Language Input Editor
	Task Builder with Global Cross Reference
	Library Management Utility
	User and Executive On-line Debugger
	Program Patch Utility
	Core Dump Analyzer
	Peripheral Interchange Utility
	Media Backup Utility
	File Exchange Utility
	File Dump Utility
	File Verification Utility
	File and Index Sort Utility (option)
	User and Executive Trace Utility
	Task Accounting and Reporting Package

	On-Line Device Error Logging and Analysis Package
	Multiplexed I/O Spooling Package
Languages	MACRO
	FORTRAN IV
	FORTRAN IV-PLUS
	COBOL
Batch facilities	Single-stream batch processor

CHAPTER 3

THE MEANING AND ORGANIZATION OF DBMS

The number of software packages on the market that are advertised as database management systems is well over one hundred; and the choice continues to increase. Nevertheless, the number of software packages that deserve to be called generalized database management systems is much less.

In this chapter, we describe the meaning and organization of database management systems. The emphasis is placed on the overall DBMS organization, so that some topics are covered lightly. A case in point is that of database system capabilities. It is an important area and has become a subject in its own right. For this reason, we have left the detailed discussion of DBMS capabilities for the next chapter.

As it is pertinent to our study to understand the inherent difficulties in putting a database management system on a minicomputer, we have included, in this chapter, a discussion of minicomputer database software.

3.1 Historical Evolution

In the early days of business data processing, the responsibility for organizing and maintaining a collection of data rested entirely with the programmers who wished to make use of them. They had to be aware of its physical structure, and in order to access data items of interest to them, they had to write programs that explicitly manipulated these physical structures. As organizations grew and the number of files and

programs increased, this responsibility proved to be an enormous burden on the programmers.

Generalized file management systems were introduced in an attempt to alleviate the programming burden for their users in the processing of standard files. Typical of these systems is the following philosophy: data is accessed on an application basis. Therefore, given the nature of the data associated with an application, and the retrieval operations required of the system, the internal data structures of the files are designed so as to achieve optimal program performance. These, however, tend to embody pre-defined, static relations between the data. Application programs also become frozen to the data structures chosen, and a small change in the organization of one file may create major upheavals in the whole system. Furthermore, the data in one file and for one set of application programs is generally not available to other programs. This, of course, does not mean that data which exists, perhaps in distributed form, on other files is not usable, but that it is usually necessary to process such data into another form or format for use in another application. For convenience, or perhaps the lack of knowledge needed to use a file created by another programmer, each application often started from its own data. Consequently, a high degree of data redundancy resulted in traditional file management systems.

The need for sharing of common data and convenient access to it by its ultimate end-users, compounded with the increasing information requirements of management, marked a radical departure from the existing file systems and gave rise to the notion of integrated databases with retrieval

programs or managers.

3.2 The DBMS Concept

An integrated database may be defined as the repository of all information among which an organization wants to maintain relationships. The DBMS is the software system responsible for organizing, maintaining, and accessing databases. It acts as an intermediary between the application programs or end-users that wish access to a database, and the actual physical data.

Using a DBMS, the data would be stored once for all the applications that need the information. This helps solve the problems of file proliferation and data redundancy. The profits to be gained from the elimination of unnecessary redundancy are reduced storage and updating costs. In addition, it improves the consistency of the database, since the possibility of having different copies of data in different stages of updating is removed.

Effective utilization of integrated databases depends upon two important concepts: data independence and non-procedural access. Data independence means that a DBMS must shield the user from the mundane details of the internal physical representation of the data, instead allowing him to concentrate on his conceptual logical view of the data. This means that application programs interacting with the database need not be concerned with the physical organization of the data, and the database structure can be reorganized as needs dictate without rewriting the

applications.

Non-procedural access enables a user to identify and select the desired data in terms of the properties it possesses, rather than by means of an explicit search through the database. Such an access language reduces the complexity and cost of writing application programs, and makes the database easy to use by non-expert programmers. In addition, it makes it much easier to respond to ad hoc and unanticipated information retrieval requests from top management for planning purposes.

3.3 Requirements of a DBMS

One of the perennial problems that arises when discussing DBMS software is that the term has come to represent a variety of concepts. As a result of marketing strategies, a DBMS can mean anything from an enhanced searching facility to a complete full-blown system. On the other hand, a system which goes all the way to a sophisticated report formatting capability may actually have very crude underlying database concepts. It is therefore desirable to define a database management system in terms of the general objective it could be expected to provide.

Palmer[7] lists the following as the requirements of a database management system:

1. The controlled integration of data, so as to avoid the inefficiency and inconsistency of duplicated data.

2. The separation of physical data storage from the logic of the applications using the data, to aid flexibility and ease of change in a dynamic environment.

3. A single control of all data, permitting controlled concurrent use by a number of independent on-line users.

4. Provision for complex file structures and access paths, such that relevant relationships between data units can be readily expressed and data can be retrieved most efficiently for a variety of applications.

5. Generalized facilities for the rapid storage, modification, reorganization, analysis and retrieval of data, so that the use of a database system imposes no restrictions upon the user.

6. Privacy controls to prevent unauthorized access to specific units of data, types of data or combinations of data.

7. Integrity controls to prevent misuse or corruption of stored data, and facilities to provide complete reconstruction in the event of hardware or software failure.

8. Performance, both in a batch mode and on-line, that is consistent, measurable, and capable of being optimized.

9. Compatibility with major programming language, existing source programs, a variety of hardware systems and operating systems, and data external to the database.

Contemporary database management systems provide only limited versions of some of these desirable objectives. Furthermore, the implementation methods can be significantly different, resulting in differences in features and performance characteristics.

3.4 DBMS Organization

In its totality, a database management system may be regarded as a combination of staff, software and hardware functions responsible for the tasks associated with maintaining and accessing databases. The following major organizational units, aimed at facilitating the sharing of data among users in addition to lessening the programming burden, can be recognized in any database management system:

1. The Data Description Language (DDL) is used by the individual responsible for establishing the database (called the database administrator, DBA) to provide a centralized definition of the logical data structure and physical storage structure of the database. This definition of the database is called its schema.

   The logical data structure, the user's conceptual view of the data, provides the user with the facility to perceive data in terms suitable for his processing needs. The data structures are the composition of the data items in a record (the records in a file, and so on) as it is viewed by the user without regard to the way they are actually stored within the computer. The DBMS translates from the user's abstract view of data to the detailed physical level.

The most important contemporary data structures are: the hierarchical model,[17] in which the data takes on a tree-structured form bearing relation to its presumed logical structure and/or access pattern; the network model,[7,18] where records are connected in an arbitrarily complicated list structure; and the relational model,[18,19] in which information is arranged in a collection of named two-dimensional tables (with a fixed number of named columns and a variable number of unnamed rows called tubles) called "relations".

The storage structure is the physical representation of a particular data structure on some storage medium. It is the system's physical view of data and, as such, determines the method of placement and access to data. (Storage Structure and Access techniques are discussed in 3, 7, and 8.)

The DDL is thus used to define the structure and format of data in the database, the logical view of the database, and the methods of access to the data. The schema definition serves as a template for all data records that will be entered in the database, and describes both their individual structure and possible relationships among them.

The DBMS processes the schema definition in order to produce data dictionaries which are then available for processing of transactions against the database. The data dictionary is a table consisting of an entry for each data element (item, record, etc.) in the schema definition. The entry contains, generally in both source and coded form, the name, type, length and other attributes of the element,

including in particular whatever information is necessary to physically locate instances of that element.

It is important to note that the schema definition is itself data, and it is quite possible, in fact, to store the definition in exactly the same way that user data is stored, and to use system facilities to access it. This implies that there are data dictionaries for the dictionaries themselves. To avoid an infinite recursion, of course, an initial definition has to be built into the system. (This dictionary driven structure is the mechanism employed in GIM-II.)

Finally, the DDL can also be used by the programmers to describe the data and structures with which they are concerned and to furthermore describe it in the form they wish. This individual or application view of the database (called subschema) is intended to provide for multiple views of the database. In the current systems the extent of differences allowed between the user's view and that of the schema is nominal, and the schema to subschema relationship is more or less a mapping process.

2. The Data Manipulation Language (DML) is the interface used by programmers to access or modify the database. DML capabilities include the operations of selection and retrieval of data from the database, its modification, its storage and its deletion. It can also be used to establish and remove relationships among data. The DML may be a language in its own right (self-contained) or it may be used in conjunction with a programming language such as COBOL (host-

language systems). In host-language systems, communication from the program may be in the form of subroutine CALL statements, or it may be in the form of verbs added to the language. The verb form requires some modification to the host-language compiler, or the addition of a precompiler phase to process the DML statements before compiling the program. The most common approaches to implementing the DML interface in self-contained systems are interpretation, where each DML command is examined and executed by the DBMS at run time; and compilation, where either the DML programs are compiled directly into object code, or they are first translated into a high or low-level computer language and then compiled using the conventional compiler.

3. In addition to the facilities for data definition and data manipulation, a DBMS provides a number of capabilities to help control the physical and logical integrity of data. They include: semantic integrity, which is concerned with the validation and conformity of data elements to their definitions and applies to all data entering, stored in, or leaving the system; access control, which limits the extent of access allowed to a database by a particular user or application; concurrency control, which provides protection against undesirable interactions or lost updates in a shared database environment; physical integrity, which provides for the restoration of the database after a system failure.

3.5 Minicomputer DBMS

Producing software for a minicomputer is a very expensive process, compared to the cost of the hardware, and since the marketing cycle of a minicomputer is shorter than that of a large computer, these heavy costs are recurrent. It is not surprising, then, that there is much less software available for minis compared to what is available for larger mainframes. The increasing competitiveness in the minicomputer market has driven out many of the original manufacturers, leaving only those whose production in general can be measured in thousands of computers per year. This has allowed, in recent years, the production of more sophisticated software as standard by the manufacturer, without losing the cost advantage of the hardware.

Today, many minicomputer makers are turning development away from scientific and process control fields that were traditionally their mainstay, and concentrating earnestly on business applications.[20] There are a number of reasons for this apparent trend:

1. Megalyte main directly accessible storage, virtual memory, and overlapped core which were once reserved for large-scale machines are now being profitably supported by mini-computers.

2. Peripheral manufacturers have now made available large-scale mass storage (such as 300 Mb disk drives) for minis, and virtually every other high-speed, high-volume peripheral typically found on central computer installations can be

obtained for a mini.

3. Commercial application languages such as COBOL are being supported by vendors. (Actually, the COBOL on a mini is a subset of the language as it is understood on a large mainframe.)

4. Database management systems have begun to appear through independent software houses and original equipment manufacturers.

It is inevitable that a large minicomputer, with hardware and software like that, would be able to compete for office and business processing jobs along with the traditional large-scale computer.[21,22]

The importance of database software is well appreciated by minicomputer manufacturers; some are releasing software of their own, while others have linked up with independent vendors. VARIAN, for example, is formally aligned with TOTAL, and it is probable that other manufacturers, particularly smaller ones, will make arrangements with other suppliers.[23]

3.5.1 Systems Provided by a Software House

The systems built for sale or rental by software houses have a lot to recommend them. They are reasonably mature products that have solved many different kinds of database problems. They are frequently designed to be machine independent, able to move to a different computer, even to a different computer vendor. Furthermore, independent vendors know that they have to provide a better, more cost-effective system in order to stay in competition. Support and maintenance, however, represent the usual drawbacks of dealing with an independent supplier.

3.5.2 Systems Provided by the Hardware Vendor

Theoretically, at least, there are several advantages to using manufacturer-supplied software. First, the DBMS should "fit" with other system software (operating system, compilers, input/output routines) better than a system developed by another organization. Secondly, any problem with the system can be directed to a single source, without argument as to whether the DBMS, the operating system, or the hardware caused it.

The major disadvantage of using a manufacturer-supplied system is that it may commit the buyer to large investments in nonstandard hardware and software. In most cases, the new system is available only on the latest, largest, and most expensive machine a manufacturer makes. This is a consequence of the high cost of software - only on large system sales there is enough profit to pay for software development. Another problem is that vendor software, being relatively new, is unstable,

causing crashes and inexplicable bugs which are extremely costly. Furthermore, the pace of business in the minicomputer field is so hectic that there is often no time to tailor complete, fully operational software systems to go with each new hardware announcement. As a result, even if a new DBMS really should have a better operating system, it is often released before the operating system is completed. And when the operating system is finally announced, the user finds out that while speeding up the DBMS as promised, it also takes 128K more core than they have, or only runs on an updated or upgraded hardware configuration. Unfortunately for the user this implies still more investment.

3.5.3 The Limiting Factors

While it is definitely true that database packages are available for minicomputers, it is also true that none of them is as powerful as functionally similar ones found on a typical large-scale system.[24] This disparity arises because of several minicomputer characteristics that make the task of installing a DBMS on a mini more difficult.[25]

1. Minicomputers are of course small, while DBMS tend to make use of large amounts of memory. IBM's IMS, for example, takes about 128K bytes, and that is often all the space available on a mini. In addition to the DBMS, space is required for operating system and program development. Hence, the need to avoid making software larger than absolutely necessary becomes critical on a minicomputer.

One solution to this problem is to divide the DBMS into overlays. During execution the overlay segments are fetched from secondary storage on demand, i.e., when they are called and are not already in primary storage. With this approach the disk access time becomes a bottleneck. To alleviate the problem, the code generated should be reentrant, so that the overlay segments need not be written back to the overlay file, and the sharing of code by several processes becomes possible.

Of course, with each new approach the amount of code grows quickly and the result can be a system where the programs themselves are a significant percentage of the total system storage. Unfortunately, the problem is not simply storage space; it is that better systems mean more user facilities, and user facilities mean greater user utilization which implies system loading leading to poor response times.

2. Another characteristic of the minis is their small word size - 16 bits. This means that their memories, memory buses, and I/O buses are all 16 bits wide. To perform a 32-bit operation or the equivalent amount of work, they must do two loads, two stores, and use two CPU registers. This clearly drags down their effective throughput. Multi-user environments, as is the case with database systems, dictate extensive use of indirect addressing, and this creates double work instructions for most 16-bit minis. The next difficulty for 16-bit minis is their small address space within a program, i.e., their absolute core addressing capability. If one has 16 bits for

addressing, one can only address 64K bytes of memory. This does not affect how large the total memory can be, but it does affect how large any one program can be. Problems come up well after the system is installed and has proved successful. As more work is put into the system certain internal tables have to expand, but chances are that some of the programs are already close to the maximum program size, and as a result something is bound to go wrong. To go above 64K bytes, manufacturers have had to introduce "memory management" type devices.

3. The next area showing itself to limitation when DBMS's are installed on minis is that of the single bus architecture. While elegant in concept and primarily responsible for keeping down the cost of new device interfacing, these single bus machines are being limited by data traffic saturation on the bus itself. New faster disks are the prime cause of this. Yet, another limitation is the load on CPU caused by interrupt servicing of a large number of terminals.

4. Probably the most important characteristic of minis is that they tend to be found far removed from computer centers and trained staffs. If anything goes wrong, the first people who will try to set things right are likely not to know very much about computers. Recovery procedures have to be overdesigned and the on-site documentation has to be written to a higher standard. Minicomputer DBMS's should, to as large an extent as possible, keep everything that could involve an operator, especially exceptional things like restart procedures, fallback procedures, and reorganizing the system,

as simple as possible.

It is clear that minicomputer technology can support comprehensive database management systems. Almost every manufacturer is heading in that direction. It is also clear, however, that there are limits to system capabilities that would probably not be encountered with larger computer systems. Many of these limitations will undoubtedly disappear in the next few years. Others will persist simply because of inherent limitations with minicomputers. Problems notwithstanding, minis have performed well on database problems in many situations and continue to attract small businesses. Moreover, the changing economics of hardware manufacturing, the surprising flexibility of minicomputers, the outlook for low cost data transmission, and the continuous developments in the field of database management systems combine to suggest that minicomputer-based database systems, at least for some users, may be the way of the future.[26,27]

CHAPTER 4

DATABASE MANAGEMENT SYSTEM FEATURE DESCRIPTIONS

In this chapter, we examine the more significant features of DBMS's: the reasons why each should be considered, the problems associated with their implementation, and the extent to which they are provided by current systems.

From a reasonably comprehensive menu of DBMS capabilities, a user can select those which best satisfy his data processing requirements. With an understanding of the importance of various features to his requirements, he can then evaluate candidate systems.

There is no implication that any one DBMS should, or could, or does offer all the capabilities covered. All the off-the-shelf DBMS packages do not provide the same set of functions, and the implementation of functions differs widely in depth and strength of effectiveness. A careful evaluation process is crucial. But without all the facts about each package and the considerable variety of features offered by modern DBMS's, an organization has difficulty in making a choice.

The features of database management systems are discussed under the following headings:[3, 5, 8, 12]

- General Description
- Computer Environment
- Data Structures

- Data Definition
- Data Manipulation
- Query Capabilities
- Output Presentation and Report Generation
- Security and Error Recovery
- Database Restructuring
- Application Programming
- System Installation

Each section is further subdivided, as necessary, to describe functions appropriate to each heading. It is important to recognize that, in discussing features of a database management system, the problem is multi-dimensional: a major database system capability, such as update, may appear many times in a feature list. There has been an effort to discuss those issues which affect more than one function as they arise.

We have chosen a particular database package, TRW's GIM-II operating on a PDP 11/70 under RSX-11D, to be evaluated in view of the capabilities presented. In the following chapter, GIM-II's capabilities are examined in the context of a hospital information system, and the advantages and disadvantages of the package are considered in light of the installation's ultimate needs.

4.1 General Description

4.1.1 History of Software

Historical information on the implementation of the database package indicates the maturity of the product in terms of extent of development and actual usage.

GIM-II

History	The software is being developed solely under contract to the U.S. Army.
Significant Installations	Federal Aviation Administration Department of Defense The FBI The CIA

4.1.2 Availability and Cost of Software

The database package to be installed at customer's site may be available by purchase or lease. Some of the more "popular" systems are also available through time-sharing service bureaus. In certain cases, systems may be available simply because they are already owned by the user's organization (particularly the government).

GIM-II

Availability	GIM-II software is government owned and should be available to the federal agencies from TRW at any time. Presumably, a support contract with TRW would be required. The GIM-II system is non-proprietary to the federal government; that is, any enhancement or modification developed by any one of the agencies would be available to all at no cost.
Cost	The Non-recurring cost for each copy of the system including application development (see 1.3) is about $250,000.
Maintenance Costs	

4.1.3 Type of System

A common characterization of database management systems is self-contained versus host-language. Self-contained DBMS's were primarily developed to minimize the overall need for programmers by providing facilities for the non-expert user. The non-expert user is so called because he does not write a program in some conventional programming language to access the database management capabilities. The system provides a self-contained query-update language which in addition to being non-procedural can provide powerful data manipulation facilities. Of course, there is a certain pre-defined processing structure within which the user must operate; certain functions, e.g., File Structuring, Updating and Retrieving, have had their logical flow pre-programmed. In this sense, specialized manipulation of data is not attainable.

On the other hand, host-language DBMS's capabilities are accessible only to programs written in a conventional programming language (e.g., COBOL, FORTRAN, PL/1). The capabilities, in effect, extend the programming language, making it easier for the programmer to access and manipulate the database in ways which were difficult or impossible using the host programming language by itself. In a sense, the system is essentially a sophisticated collection of input/output routines.

The current trend is towards systems that offer both self-contained and host-language facilities.

GIM-II

System Type

The GIM-II system is a self-contained, on-line database management system that provides a complete set of data management facilities for both the non-expert user and the more sophisticated user.

GIM-II contains a basic English type language, the User Language, that makes it possible for the user to query a database in a precise and timely manner. The User Language (UL) is used to initiate all transactions - updating and interrogating the database, defining and structuring the files and managing the GIM-II system.

GIM-II also contains a self-contained programming language, the Procedural Oriented Language, that permits the user to design, code, debug and execute routines called Procedure Lists. The Procedural Oriented Language (POL) makes use of the same software elements of the GIM-II User Language; however, it allows the user to exercise control over the program flow. The Procedure Lists may be used to produce reports, perform repetitive functions, update groups of files automatically or perform menu processing.

4.1.4 Modes of Operation

Batch processing implies that the user is not expecting an immediate response, whereas on-line processing implies that the user is waiting for results. DBMS's are designed to operate in batch mode as well as on-line with interactive capabilities. Use of interrogation and update functions can be provided in either batch or on-line mode. It is also possible in the batch mode to run several inquiries concurrently against one or more databases to increase throughput. The Data Base Administrator (DBA) might be able to restrict the available mode in order to use system resources efficiently.

GIM-II

Mode for Data Definition	Files may be created in batch or on-line.
Mode for Update	Database update can be performed by the on-line user through query language. The User Language verbs ADD, CHANGE and DELETE are used to update the database. GIM-II's on-line update language syntax is oriented toward infrequent, non-repetitive file updates. There is no single command for batch updates and it must be accomplished through user written programs.

4.2 Computer Environment

4.2.1 Main Frame, Operating System, Core Requirements and Source Language

The main frame refers to the particular computer on which the system (or a version of it) can operate. The overall operation of the database management system itself (its scheduling and its relation to other jobs running on the computer) is usually under the supervision of the operating system. The core memory requirements of a database package may vary depending on the number of users supported, and the amount of input-output buffer space in the system. Space may also be required to store pages of tables used internally, or for transient functions of the database package. These imply a minimum core requirement for operation of the system at a reasonable level. The source language is the programming language in which the system itself is written.

GIM-II

Main Frame and Operating System	IBM 360/370 under DOS and OS. PDP 11/45, 11/70 under RSX.11D.
Minimum Core Requirement	On the IBM 360/370, GIM-II operates as a single service channel program in an 80K bytes partition, or as a multi-service program in up to 500K bytes. Under RSX-11D, GIM-II operates within a fixed 32K bytes partition.

Source Language	GIM-II is written in Program Word Structure - a TRW macro programming language. It is an open-end program and can be modified or extended to meet specific requirements.

4.2.2 Teleprocessing and Concurrent Usage

A teleprocessing capability provides the mechanisms for establishing links between the remote terminal users and the system. While certain software packages are available which combine both DBMS and teleprocessing, it is unnecessarily complicated to consider both functions simultaneously. Generally, database software designed for on-line use does not include facilities for controlling terminals and lines. This is the function of the teleprocessing monitor. Teleprocessing monitors are concerned with the control and scheduling of transactions in an on-line environment. The data communication systems operate in single-thread or multi-thread mode. A single-thread communication system processes a single transaction to completion. That is, a single transaction introduced via the data communication system is transmitted to application programs which then carry out their functions until all requirements of the transaction have been satisfied and the end result delivered to the user via the data communication system. A multi-thread capability allows concurrent execution of application programs.

If the database is to be accessed by a number of terminal users, the software is likely to be re-entrant (code does not modify itself). In this manner, multiple users can concurrently be executing the one copy of the system.

GIM-II

Communication Software	GIM-II has its own teleprocessing monitor, ASSIST Monitor Software (AMS), operating under RSX-11D

in a fixed 32K bytes partition. AMS operates in a multi-thread mode and can support up to 6 active terminals.

Terminal Type

Any terminal compatible with teleprocessing software is supported. Modems are required for terminals located more than 1000 feet from the computer facility.

4.2.3 Mass Storage

The database system requires large amounts of peripheral storage space. Storage space is required not only for the database of interest to the community of users, but also for the database system as well.

GIM-II

Mass Storage Device	All PDP-11 storage devices are supported. Databases, stored on removable disk packs, must be mounted before the system is brought up.

4.3 Database Structure

4.3.1 Logical File Organization

The GIM-II databases are organized more or less on a network basis. The system accomplishes its networks by allowing the coupling of individual data fields in one file with data fields in a second file. This provides the user with a means for describing logical groupings of records.

The GIM-II data files are, in a logical sense, structurally quite simple. Traditional file and record layouts are used. A file (data list in GIM-II terminology) is a collection of related logical records. The relationship exists in the usual sense, i.e., that the records need to be maintained under the same file. Each logical record (item in GIM-II terminology) consists of a number of data fields which are defined in the dictionary for the file. A data field (attribute in GIM-II terminology) is the lowest level of definition in the logical file organization.

The construction of logical file organization is accomplished by dictionary definition (see 4.4). The dictionary contains a description of each attribute field in a data list and describes the relationship of the data list to other data lists in the database. Attribute fields may be defined as having a single value, multiple values or no values. In addition, an attribute field may have a parent-child relationship to another attribute field(s) in the same logical item.

The approach to logical database structuring in GIM-II is based on the premise that a user has a number of files that he wishes to maintain as a shared resource. The amount of redundant information content that can be contained within the database is a reflection of the user's skill in defining his files. There are two techniques which may be used in GIM-II to consolidate data. The first of these is to make a "super" file containing all the types of attribute fields which may possibly occur. The second and more general approach is to couple files.

Consider, for example, a database which contains a file of automobiles registered in a state; and another file of car owners.[8] Figure 1 shows some possible attribute fields in such a pair of files. In this example, the social security number may be used to couple the two files. Then, for example, a query may retrieve the names and addresses of all residents of Suffolk County who own a brown Chevrolet Nova built since 1967.

Database restructuring (see 4.9) is easily accommodated in the GIM-II system. New attribute fields may be added without reloading the database, or without changing any existing application programs. In our example, new attributes regarding the licensed driver can be added to the car-owner file. Then a file of traffic violations may be structured and coupled to the existing files (see Fig. 2). Now a GIM-II query can retrieve the names of all offenders who were convicted of speeding in 1977, were residents of Cambridge, and drove a red Chevrolet.

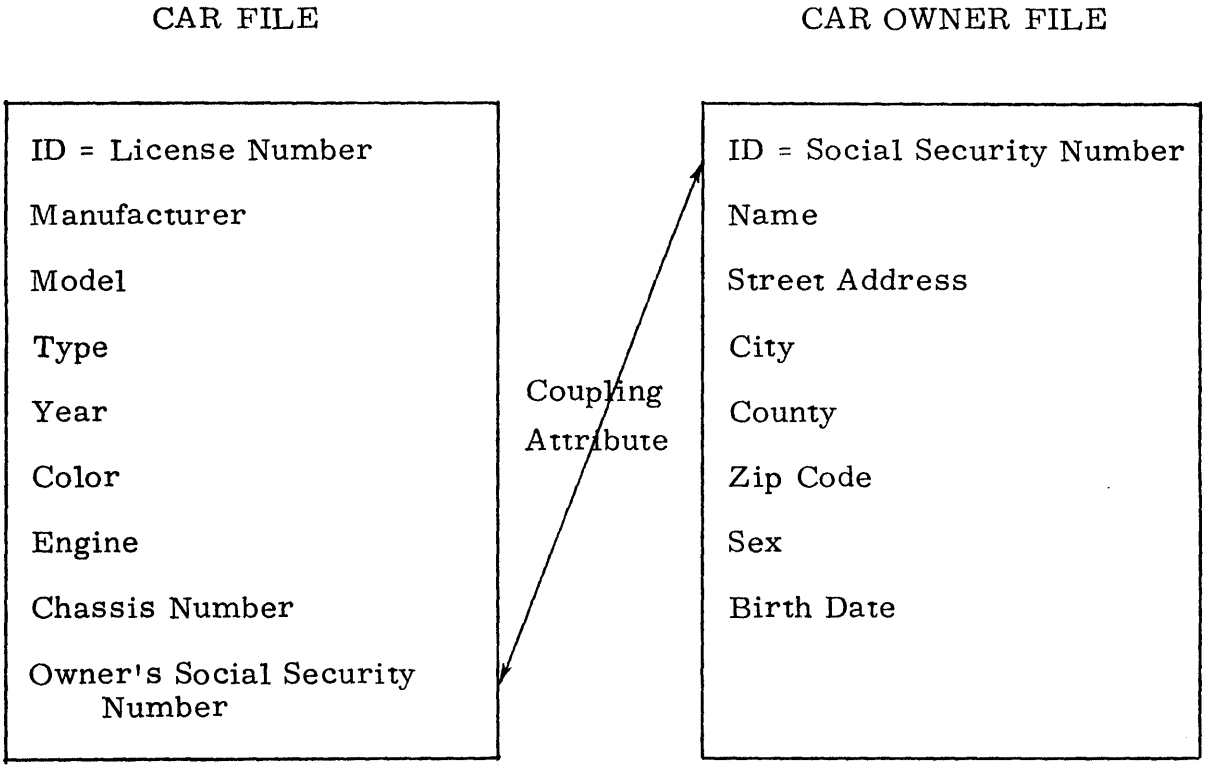
CAR FILE
CAR OWNER FILE
ID = License Number
Manufacturer
Model
Type
Year
Color
Engine
Chassis Number
Owner's Social Security Number
Coupling Attribute
ID = Social Security Number
Name
Street Address
City
County
Zip Code
Sex
Birth Date

Figure 1

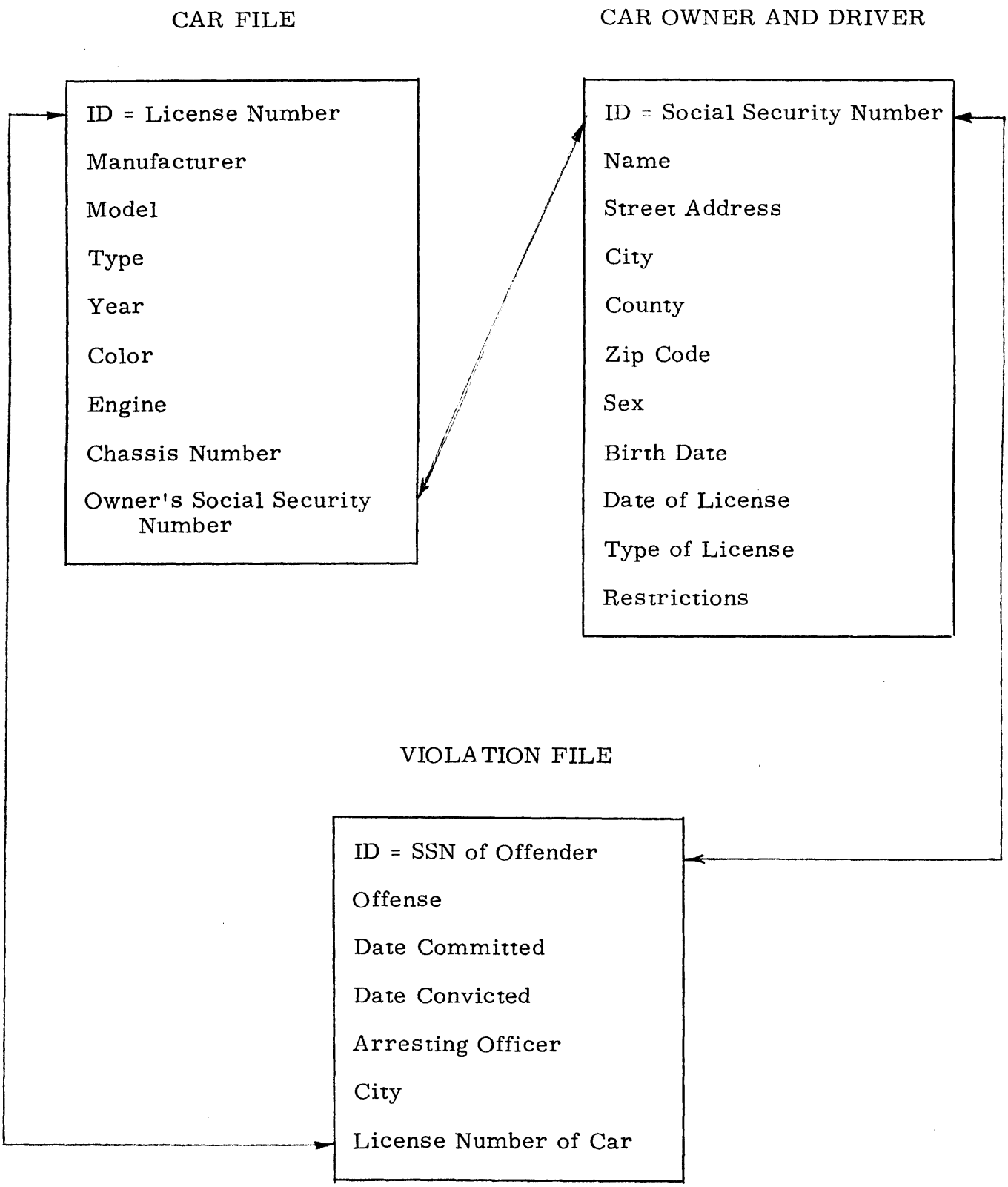
CAR FILE
ID = License Number
Manufacturer
Model
Type
Year
Color
Engine
Chassis Number
Owner's Social Security Number
CAR OWNER AND DRIVER
ID = Social Security Number
Name
Street Address
City
County
Zip Code
Sex
Birth Date
Date of License
Type of License
Restrictions
VIOLATION FILE
ID = SSN of Offender
Offense
Date Committed
Date Convicted
Arresting Officer
City
License Number of Car

Figure 2

4.3.2 Physical File Organization

In the GIM-II system, the user has few obligations and responsibilities with respect to the physical organization of records for files of the database. In effect, the user's responsibilities for record design ends, once the set of fields for the records of a particular file is defined. From this point on, the required record organization is developed by the system itself.

All physical records in GIM-II are fixed length records, with their size determined by the particular device type. These records usually contain a number of variable length logical records. A single logical record, however, may span several physical records. Each field within a logical record is also of variable length, preceded by a length code. All the fields of a single logical record occupy physically adjacent storage positions within the same physical record or within adjacent physical records. The fields are identified by their position in the logical record. New logical records may be created by a single command, but each field in the record must be assigned a value individually, using the field name to identify it. If the value for a field is not specified then the field is not present in the record and takes up no storage.

There are no physical limits to GIM-II databases. The maximum number of files in a database is constrained only by disk storage. There is no practical limit to the size of a logical record or a field within a record. And any number of fields can be defined for a logical record.

The physical records allocated to a file (data list) are described to the system, at dictionary definition time, via three parameters: base, modulo and separation. The base is the address or starting location of the first physical record assigned for storage of the data list (if not specified, the base will be assigned by the dictionary compiler). The modulo is the number of directly addressable groups of physical records allocated to a data list. The separation is the number of sequentially linked physical records that form a directly addressable group or modulo. For example, an assignment of 2 and 3 for modulo and separation, respectively, allocates two groups of three physical records each to a data list, i.e., a total data space of six physical records.

The logical records of a file are randomly distributed in the data space, and therefore a randomizing function must be used. The system attempts to place a synonym record as close as possible to its home address. However, if a logical record is addressed to a modulo with insufficient space to accommodate it, the group is dynamically linked to the physical records in the overflow area. All of this is automatically accomplished by the system and is transparent to the user. Nonetheless, modulo and separation assignments affect system processing time and disk space utilization and should be carefully selected (e.g., the higher the separation the more update will be favored).

4.4 Data Definition

The construction of the data list (file) dictionary is the GIM-II method for defining data lists. The data list dictionaries provide the detailed definitions of the user's application files and all procedures processing a specific data list will be controlled by its dictionary. Each data list dictionary comprises the description of every attribute field in the data list and describes the relationship of the data list to other data lists in the database (all the files in a database can be bidirectionally interrelated).

Dictionaries are treated as data lists that describe data, i.e., the user's application files. This permits the user to structure and maintain dictionaries as any other GIM-II data list. Since they are data lists themselves, they must have a dictionary that describes their contents. The dictionary that describes all other dictionaries is a system-file (SYSFILE) called the Master Dictionary, or M/DICT, that is supplied with the system and is established in the database at the time of database initialization (see 4.11). The Master Dictionary contains dictionary names, attribute field definitions of dictionary records, and pointers to all system and user dictionaries contained in the database. The pointer for the Master data list, M, is also contained in M/DICT. M contains all system verbs, user-defined functions and file names. As data lists are structured by the user, the names of the data lists are automatically entered into M and M/DICT.

User dictionaries are structured by the user. The ID of each dictionary

record is the name of the attribute field being defined. In the case of the dictionary record which describes the key of data records to be stored in the user application file, the ID is always the data list name. The data list Item ID in the dictionary contains the location of the user data list, security locks and data list relationships.

Each attribute field of a dictionary record describes a characteristic of a data field value. These include retrieval and update security codes (IR/SC, UPD/SC), type of value (numeric, alphanumeric), the maximum character size of an output value, the number of data values, value storage method (e.g., each unique value will be stored only once under the attribute, or the value will be algebraically added to the existing attribute value), record-to-record and/or attribute-to-attribute relationship (see 4.5.4), and various editing specifications (see 4.5.5).

All entries, changes, modifications or manipulations to any GIM-II data list, whether it is a user application file, a dictionary or a specialized data list (e.g., Procedure List) are made via the GIM-II User Language. Data list dictionaries exist in both source language and compiled versions. Before entering data into the data list, the source dictionary must be validated for correct structural definition, and compiled into GIM-II executable form. The compiled versions are interpreted and used at run time. The user makes changes to the source dictionaries. These changes become effective only after the revised source dictionaries are again validated and compiled.

Example:

To structure a new data list, the user assigns a name to it and assigns a portion of the physical storage to the data list and to its dictionary.

STRUCTURE-FILE "EMPLOYEE" "EMP" DICT/BMS ",1,2" DATA/BMS ",3,2" #

The Structure-File Processor verifies that the data list name (EMPLOYEE) and synonym (EMP) do not previously exist in M; assigns and initializes the data list dictionary (2 physical records) and the data list (6 physical records); creates a record in M for each data list name; creates a record in M/DICT for each data list name with /DICT appended to the name; and creates a record in the data list dictionary for each data list name.

The dictionary is activated with the command: COMPILE-DICT M #

Attribute fields may now be defined:

ADD EMP/DICT "EMP" "EMPLOYEE" V/TYPE "RN9" V/COR "YNRS" S/EDIT "V/MAX '9' V/MIN '9' TYPE 'N'" #

The record identifier (Item 1D) is defined to be right-justified (for output or sorting purposes), numeric and of 9 character length. Each unique value is to be stored only once under the attribute (YNRS), and the size and type of values on input are to be verified (S/EDIT specification).

ADD EMP/DICT "NAME" V/TYPE "LA30" V/COR "YSVX" S/EDIT "V/MAX '30'" #

The attribute field, NAME, is defined to be alphanumeric with a maximum character size of 30. The first value entered will be stored under the attribute; additional values will cause the input statement to reject (YSVX).

ADD EMP/DICT "SALARY" V/TYPE "RN6" V/COR "YSVX" S/EDIT "V/MAX '6' TYPE 'N'" #

This statement defines the SALARY attribute field in a similar manner.

The data area is activated by the command: COMPILE-DICT EMP #

Data may now be added to the file:

ADD EMP "165480001" NAME "JONES, JACK" SALARY "1200" #

There is now one logical record in the file.

New fields may be added to the existing file:

ADD EMP/DICT "AGE" #

and activated by:

COMPILE-DICT EMP #

The new field may be used for existing records:

FOR EMP "165480001" ADD AGE "30" #

4.5 Updating and Data Integrity

Updating is the process of retrieving a record, modifying the data within it, and returning the record to the database. Normally, the update function utilizes the basic interrogation capabilities of the package for selection and qualification (see 4.6). Provision may be made for updating to be performed as a transaction-oriented or batch-oriented feature.

4.5.1 Update Language

Since the approach to update processing in a host-language system is for the user to supply a transaction program for each type of transaction entering the system, the update language is primarily discussed in relation to self-contained systems. Typically, there is a common language for both interrogation and update. During update processing, initially a part of the database is selected and then it is updated rather than displayed.

GIM-II

The database can be updated in both query and procedural languages. The User Language verbs ADD, CHANGE, DELETE and RECREATE are used to update the database; these commands can be invoked within the POL. Syntactically, the update commands may be used at the beginning of a statement or after a FOR clause, with full selection and linkage capability (see 4.6). For each logical record selected, the Update

Processor passes control to the function within the processor that performs the appropriate updating task.

ADD

The ADD command is used to place all values and names into the GIM-II data lists and data list dictionaries. The following sample sentences exhibit the usage of the ADD verb.

ADD EMPLOYEE/DICT "NAME" "AGE" "SALARY" "ADDRESS" #

ADD NEW EMPLOYEE "123" #
The modifier NEW appears before the data-list-name to insure that the update is to occur only if the item value - record ID in this case - does not already exist.

FOR EMPLOYEE "123" ADD NAME "JONES, JACK" SALARY "12000" AGE "42" #
Only the fields being updated are referenced and their order of referencing is independent of their order in the record.

CHANGE

The CHANGE command is used to change any value and/or name in a GIM-II data list and/or data list dictionary. The specified values or names will be deleted from the file and the

replacement values specified in the CHANGE statement will be added to the file.

CHANGE EMPLOYEE "123" TO "345" #
When an item ID value is changed all secondary references to that item ID are also changed.

To protect against updating an incorrect record, the user should identify the "delete" value(s).
FOR EMPLOYEE "345" CHANGE SALARY "12000" to "(12000 * 1.05)" #
Note also that the update value may be specified as the result of a computation.

If no specific value is mentioned in the statement (CHANGE attribute-name TO ...) all values for the attribute will be deleted and the TO value will be added.
FOR EMPLOYEE "345" CHANGE EXTENSION TO "7350" ROOM-NUMBER TO "652" #

DELETE

The DELETE command is used to remove any value or name in the GIM-II data list and data list dictionary. A child value may be deleted without affecting the parent attribute. A deletion of a parent attribute, however, automatically deletes the child.

DELETE EMPLOYEE "123" #

When an item ID value is deleted, the record and all the associated attributes and all secondary references (correlatives) are also deleted.

FOR EMPLOYEE "345" DELETE SALARY "12250" #

FOR EMPLOYEE "123" DELETE SKILLS "CLERK" AND CODE "03" SKILLS "ADMIN" AND CODE "07" "08" #

RECREATE

This command is used to create an identical record in the same file with another item ID. For example,

RECREATE EMPLOYEE "123" AS "345" #

will cause a copy of the primary record "123" and all its associated attributes to be created in the EMPLOYEE file with an item ID "123345".

4.5.2 Batch Update

Batch update refers to the capacity for updating the database by reading sequentially stored update transactions from a batch medium such as tape, disk or cards. Most database systems require the user to program such an update.

GIM-II

Compiled procedure lists (see 4.10) are used to perform batch updates. The Procedural Oriented Language contains verbs which can read sequentially stored input transactions, parse them into strings and convert the strings to the symbolic parameters which can be used to direct the execution of the User Language update commands. The process can be repeated until an end-of-file is detected.

4.5.3 Bulk Update

Updating a few records (normal on-line update) differs in practice from bulk data loading in terms of volume, mode and input medium. In order to bulk load a database, most DBMS's require the user to format a bulk data tape as input to a restore utility (normally used to restore a database or a file that has been previously dumped as a database protection measure). As such, an entire file/database must be written. Some systems, on the other hand, allow for selective, incremental creation of the database.

GIM-II

A User Language command, SYSLOAD, evokes a program that has the ability to directly update a database (e.g., adding several records) with bulk inputs from a magnetic tape. The selection capability to specify qualifying criteria that must be met before the update takes place and the validation features (see 4.5.5.1) are available with either the basic update commands (ADD, CHANGE, etc.) or the SYSLOAD command.

The BULK-UPDATE verb provides a facility for updating the database with information from a file prepared outside of the GIM-II system. It is faster than handling individual updating statements or a SYSLOAD, but does not have the selection capability option. Therefore, very little validation can occur with the BULK-UPDATE command. It is designed for the transfer of large, already validated data. Speed improvements are obtained by:

1. A single user statement to be parsed.

2. A single set of accesses to the dictionary for a data list even though many changes are made to that data list.

3. No dictionary edits will be activated by the BULK-UPDATE processor.

4. Ignoring bridges to secondary data lists.

4.5.4 System Triggered Update and Retrieval

Some database management systems can upon updating (retrieving) one field in one record indirectly update (retrieve) another field in the same record, or in other records in the same or other files.

GIM-II

The GIM-II correlatives are the mechanism through which data elements are associated within the GIM-II database. They are stored in dictionary attribute field definitions and carry out their defined function automatically whenever the referenced attribute field is accessed.

Indirect Update

The I-Correlative is used to cause the value applied to a directly updated field to indirectly and simultaneously update the value of another field in the same data list record. For example, whenever a "budget salary" value is ADDed to a department record, the "total budget" for that department could be updated indirectly via an I-Correlative.

The B-Correlative (BRIDGE) permits values being updated in one record of a data list to be simultaneously stored in fields of another record of another data list (joint store). The following example exhibits the typical specification of the bridge correlative:

ADD PURCHASE-ORDER/DICT DL/SECONDARY
"BINV, V, PART-NO, QTY-ORD, INVQTY-SUB,
QTY-RET, INVQTY-SUM" #

Here, the inventory data list is to be updated as a result of an updating of the purchase order data list. The "V" specifies that part number must be verified in the secondary data list. The quantity ordered is used to update an inventory quantity by subtracting the number ordered. Quantity returned is to update the inventory quantity by adding the number returned.

Indirect Retrieval

The D-Correlative (DEPENDENT) is used to establish a parent-child relationship. This specification is accomplished by defining one of the attribute fields (e.g., Schools-Attended) as primary, i.e., as a D1, and those attributes which are related to it (e.g., Degree and Year) as secondary, or D2. Upon retrieval of the parent, the children will also be listed. An attribute field defined as a D2 can independently be retrieved as well.

The M-Correlative is used to retrieve the values of a selected number of attribute fields upon the specification of one special field ID. For example, a special attribute called "employee information"

may be established in an employee data list which will retrieve name, organization and title whenever the employee ID is specified.

The S-Correlative (SPAN) is used to allow attribute fields of a secondary data list to be simultaneously retrieved along with the directly accessed fields of the first data list. The secondary attributes must be named in the statement. When automatic retrieval is desired, the R-Correlative must also be specified on the attribute. The following sequence of specifications establishes automatic retrieval of the attribute "description" from skills-file upon request of the attribute "skill-code" for an employee.

```
ADD EMP/DICT DL/SECONDARY "SSKILLS,
SKILL-CODE" #

ADD EMP/DICT "SKILL-CODE" V/COR
"RSKILLS, DESCRIPTION" #
```

4.5.5 Data Integrity

Data integrity is concerned with maintaining the quality of the database through:

1. Data validation, to insure that the data always conforms to its definition (a definition which includes validation criteria on the stored data), and
2. Controlling concurrent processes which update the database.

4.5.5.1 Data Validation

Data validation applies to all data entering, stored in, or leaving the system. In other words, the validation and conformity of data elements to their definitions covers creation input, update data, and data when retrieved by the system. Basically, the validation process may cover item type, size and pattern verification, value validation (e.g., magnitude ranges), and relational validation (e.g., different data item value relations).

GIM-II

An extensive facility for predefined field edits is supported. The editing feature of GIM-II permits the user to define one or more directives to validate, convert or calculate a value on entry, during selection, for update or on output. Editing rules are defined in the dictionary and, once established, are evoked automatically whenever the corresponding attribute field is mentioned in a processing statement.

The S/EDIT field of the dictionary for a data list is used to define specifications to control values on input. It is the purpose of the S/EDIT expression to verify the type size, pattern and magnitude range of values on input. For example,

ADD EMP/DICT "SALARY" S/EDIT "TYPE 'N' PAT '/5000; 34000'" #

specifies that the value which may be validly input to the salary field of an employee-file must be numeric and in the range 5000 to 34000.

Several other edition specifications may be placed on an attribute. The Q/EDIT specification is evoked on retrieval and can calculate a value for output and/or to print-limit a value on output. The I/EDIT is evoked whenever the name of the associated attribute appears in an ADD statement. It is used - prior to the actual update - to test the validity of input values, to compare an input value to other input values, stored values or literals, or to derive a replacement value for an input value.

4.5.5.2 Concurrent Update

The problem of concurrent update can be illustrated by the following scenario. Suppose that two processes (or users), P and Q, desire to update the same record, A, at the same time. Since all requests to access the database are originated from a single process (the database manager), such requests are ultimately handled in a sequential fashion. Suppose further that the following sequence of events takes place:

1. Process P requests and receives a copy of record A in its buffer.

2. Process Q requests and receives a copy of record A in its buffer.

3. Process P modifies record A (e.g., subtract quantity ordered) and requests that it be written back into the database.

4. Process Q then modifies its copy of record A (e.g., add quantity returned) and requests that it be written back into the database.

As a result of Q's action, the update of A performed by P is lost and the integrity of the database is breached.

Lockout provides a solution to the above loss of integrity by giving an updating process exclusive control over a part of the database - no other process is allowed to look at it or change it. Such a mechanism, however, leads to the more complex problem of deadlock: process P holds resource A and asks for B, while process Q holds B and asks for A. Now neither can continue. Adequate mechanisms are therefore needed to prevent, or at least detect the problem of deadlock.

GIM-II

In the Procedural Oriented Language, the system automatically locks each record but requires the user to specifically release previously updated records.

4.6 Query Capabilities

The query capabilities of a database system may be described as the capacity for examining the database and extracting from it data records that meet certain search criteria.

4.6.1 Query Language

When formulating queries in a host-language system, the user writes a program that implements the query. As such, host-language systems do not offer a query language.

Self-contained systems, on the other hand, provide a user with a non-procedural language for on-line interrogation. Typically, there is a common language for both query and update. This, of course, does not preclude the separation and control of these functions in the actual structure of the system. It is also possible to have multiple interrogation and update languages with different syntaxes and semantics in order to meet the requirements of specific classes of users.

Implementing a query language requires a query translator, the function of which may be thought of as the selection and loading of one or more query programs written in a host or source language. These programs then determine the query parameters needed to produce the required formal references to the database through the database package.

GIM-II

The User Language (UL) is the on-line facility for non-procedural query and update.

Statement Syntax

Each transaction may be composed of one or more phrases. A phrase is a simple unit consisting of one or more words that may be qualified or unqualified by a modifier. FOR EACH EMPLOYEE or FOR EMPLOYEE are examples of a simple FOR phrase unit, with and without a qualifier. The different types of phrases are:

- LINK Phrase - dynamically associates one data list (file) to another.
- FOR Phrase - addresses or identifies the target data list to be accessed for the transaction.
- SELECTION Phrase - describes the criteria or conditions for selecting items (records) within the data list that has been addressed.
- VERB Phrase - identifies the action or command to be performed. This phase is mandatory in each statement.
- LIMITER Phrase - specifies what fields or values are to be stored, changed, deleted (input), or printed (output).

Since the verb is the primary directive of each transaction, all other statement units may be referenced syntactically by their position relative to the verb. Some phrases, therefore, must appear preceding the verb phrase in a statement (e.g., LINK phrases, FOR phrases, and Selection phrases); and some are only permissible after the verb (e.g., Limiter phrases).

Statement Rules

1. Each statement must contain one verb and only one verb.

2. All values (record keys, data values) must be enclosed in double quotation marks.

3. Each statement must end with a pound sign (#).

4. Each conditional statement must contain a FOR clause. A clause is a group of one or more phrases joined or nested within one statement. For example, the FOR phrase that has a WITH-phrase nested within it becomes a FOR clause (FOR EMPLOYEE WITH SKILL-CODE "3892").

5. Maximum number of characters for each statement:

   IBM = 691

   PDP = 509

4.6.2 Selection Criteria

The data selection criteria and qualification (which is discussed next) provide the means by which the user identifies specific instances of data which he wishes to access or update. The selection criteria are usually in the form of a Boolean combination (AND, OR, NOT) of simple relational expressions. These expressions are composed of subject (an attribute name, e.g., SALARY), relational operator (e.g., EQ), and object (usually a literal - either numeric or alphabetic - such as "8000").

GIM-II

The typical format of a conditional statement is:

FOR [data-list-name / data-list-name item-ID] WITH attribute-relation [AND / OR] [NOT]

WHERE attribute-relation . . . . .

Each WITH phrase applies to the FOR phrase. Conjuncted WITH-phrases form a WITH clause and independently share the preceding FOR criteria. A WHERE phrase applies to the criterion portion of the previous WITH phrase.

The normal relational operators provided are equals (EQ), not equals (NE), greater than (GT), less than LT), and combinations of these (GE, LE).

The object of a relational operator in a selection clause can be a literal,

an attribute name or an arithmetical expression. The arithmetical expression can consist of attribute names, operators (+, -, *, /, **, =), literals and functions that can be assembled in such a way that they represent a meaningful and solvable mathematical entity. Elements of the expression may be grouped by standard algebraic parenthetical notation.

GIM-II provides limited text string scanning expressions for record selection criteria. The functions $SCAN and $SCANX can be used to scan for an argument within an attribute.

The following query statements help to illustrate the diversity of selection criteria:

FOR EMPLOYEE WITH MAJOR EQ "MATH" LIST SCHOOL#

FOR EMPLOYEE WITH MAJOR EQ "MATH" AND DEGREE EQ "BS" LIST SCHOOL DEGREE MAJOR#

This query may be paraphrased: "Find those employees who majored in and received a B.S. degree in math. List the school(s) attended, the degree(s) received, and the major(s).

FOR EMPLOYEE WITH MAJOR EQ "MATH" AND WITH DEGREE EQ "BS" LIST SCHOOL DEGREE MAJOR#

In contrast to the previous query for which the resulting output is confined to those employees who majored in and received a B.S. degree in math; this query, in addition, will result in the listing of those who

majored in math but did not receive a degree, and those who obtained a B.S. degree in a subject area other than mathematics.

FOR EMPLOYEE (WITH MAJOR EQ "MATH" (WITH DEGREE EQ "BS" (WITH START-DATE LT "01/01/70"))) LIST . . .

This is an example of nested criteria which allows the user to specify the order of evaluation.

FOR EMPLOYEE WITH MAJOR EQ "MATH" AND WHERE DEGREE EQ "BS" AND WHERE START-DATE LT "01/01/70" LIST . . .

This query produces the same output as the previous one. In the processing, however, the two separate WHERE clauses are satisfied independently.

FOR EMPLOYEE EQ "165480001" WITH . . .

In this query the data-list-name is followed by a limiter phrase (the item (record) ID).

FOR EMPLOYEE WITH (SALARY GT "12000" AND AGE GT "50") OR WITH (SALARY LE "12000" AND AGE LT "40") LIST#

FOR EMPLOYEE WITH $SCAN (NAME, "ADASH") LIST NAME #

This query is used to list an employee's name that contains the string ADASH. $SCANX differs from $SCAN since the argument supplied with $SCANX assumes a preceding blank.

4.6.3 Qualifications on Searches

Besides the logically connected conditions within the search criteria, various qualifications may be imposed to further narrow the search. For example, the user may retrieve records based on the presence or absence of a specified field, or he may declare upper and lower values and retrieve data whose value falls within the range.

GIM-II

Attribute-relationals can be qualified with NEAREST, FIRST, LAST, PRESENT, NULL, GREATEST, SMALLEST, ABSOLUTE, EVERY, ONLY, JUST, and NEW.

Examples:

FOR EMPLOYEE WITH NULL ADDRESS DELETE #

FOR EMPLOYEE WITH LAST DEGREE EQ "MS" LIST SCHOOL #

FOR EMPLOYEE WITH GREATEST SALARY AND SMALLEST AGE CHANGE SALARY TO "SALARY * .07" #

FOR ACCOUNT EQ "1075" LIST TRANS-DATE TRANS-CODE WHEN EVERY TRANS-DATE LT "03/18/72" #

In this query, a WHEN phrase is used with the LIST verb to limit the printing of a multi-valued attribute.

The qualifiers ONLY and JUST may be used with LIST (LIST - VERTICAL) commands whenever a listing of the primary attributes or the primary item IDs is required.

4.6.4 Multi-File Searching

Multi-file searching refers to the capacity for dynamically (at run time) establishing links between files so that for a single transaction data elements within a set of files may be examined and retrieved.

GIM-II

A LINK phrase may be used at the beginning of a transaction statement to specify a secondary data list (file) name, and link two data lists together regardless of any predefined (dictionary) links. The target data list (primary) will contain the link to a secondary data list through an attribute whose values are the item (record) ID of the secondary data list. The link is temporary and operational only during the execution of the statement. Multiple links are permitted on a statement.

Example:

LINK EMPLOYEE TO SKILLS FOR EMPLOYEE WITH SALARY LT "12000" AND SKILLS-DESCRIPTION EQ "ANALYST" LIST NAME #

In this query, two data lists are involved in the selection phrase. As each primary item (employee record) is considered for selection, the link established through SKILL-CODE (a primary attribute and a secondary ID) will also be exercised.

4.6.5 Predefined Queries

Database systems may allow the user to store frequently used queries. These queries may then be invoked by calling them by name. Such a facility reduces the amount of input typing necessary, and makes data available to authorized personnel without a need for them to know the language rules. Explicit modification capability through the use of parameter list substitution may also be available in some systems.

GIM-II

This capability is supported through user-written application programs. Procedure Lists may be used to cause processing of a series of GIM-II User Language statements as an entity. By storing these statements in a Procedure List, they may be executed by entering a few words.

4.6.6 Aids on Search Formulation

During on-line interrogation, most database management systems require the user to conform to the precise format of query statements. In addition the user must be aware of the exact name of attributes defined within a file, as well as which fields are keyed elements. However, some "forgiving" features may be built in to provide flexibility for the non-expert user. For example, the system may support a list of same sounding but differently spelled attribute names, or it may display the relevant information from the file dictionary upon request. The system may also permit reserved words or noise words for making the statements easier to read. A particularly useful feature in the interactive interrogation environment is to provide the user with on-line documentation, so that information about the commands including function, format, or required parameters may be displayed.

GIM-II

None of the above features is supported. On-line documentation, however, can be implemented as a predefined query.

4.7 Output Presentation and Report Generation

The output is usually the display of some parts of the database as a result of query. System output can vary from a standard format to a full report-writing capability. Most systems have both standard output for use during interactive sessions, and bulk quantity report generation facilities to be run as needed.

4.7.1 Default Formatting of Reports

Automatic standard output is usually in either columnar or tubular form. It may include a heading line displaying the data, time and page number.

GIM-II

Automatic formatting of LIST'ed retrievals is in columnar or non-columnar format. Output will be printed horizontally across the page if all the attributes can be accommodated in one line. Otherwise, the output format will automatically be vertical. Column width is the larger of the attribute name or the maximum size length of the values to be output. Examples of both columnar and non-columnar automatically formatted output follow:

```
FOR EMPLOYEE WITH SEX "M" LIST NAME SALARY#

  E M P              N A M E              S A L A R Y

0 . . . .      0 . . . . . . . . . . .      0 . . . . .
1 9 0 1 4      P A Y N E , W . P .            1 0 0 0 0
8 4 0 1 1      B E L L , L . C .                4 5 0 0
```

```
LIST EMP#

EMP:    19014
NAME:   PAYNE, W. P.
ST-ADDRESS:   1227 SOUTHWOOD DRIVE
CITY:   MORGANTOWN
STATE:   VIRGINIA
ZIP:   22900
SEX:   M
SALARY:   10000

EMP:   84011
NAME:   BELL, L. C.
ST-ADDRESS:   1512 WOODBERRY LANE
CITY:  COLUMBUS
STATE:  GEORGIA
ZIP:   65432
SEX:  M
SALARY:   4500
```

4.7.2 Report Format Specification

The actual formatting of reports includes the following capabilities with varying degrees of user control.

- Report titles
- Headings and page footings
- Automatic page numbering
- Report body formatting - vertical or horizontal data placement specification
- Matrix or two-dimensional presentation
- Line spacing
- Complete editing and formatting of data item values - the specification of template forms to allow insertion of decimal points, commas, etc.
- Control breaks based on data item value changes or other predefined criteria
- Report summary line

GIM-II

A Generalized Report Writer is included in the GIM-II package. GRW provides the capability to prepare short, formatted reports. Its basic capabilities are pagination, positioning on the print line, sorting and summarization. These capabilities are provided through the User Language (REPORT/FORMAT) with the same syntax as other retrieval oriented verbs of the GIM-II system.

GRW does not compete with any of the commercially available report generators. Its primary benefits are the simplicity of use in either a batch or interactive mode and the elimination of support requirements from operating personnel, e.g., tape handling.

The GIM-II system can be interfaced with a generalized report generator through the EXTRACT and EXTRACT-TO-FORMAT capabilities (see 4.7.6). Complete interactive report formatting can be provided with the aid of the Procedural Oriented Language.

4.7.3 Output Media Flexibility

Database systems may furnish the user with control over the types of output devices. If the user is using a teletypewriter, the system usually responds on the same teletypewriter. However, for a lengthy report, the user may direct output to a system printer, to another terminal, or suppress listing, save report (on disk or tape), and later reinitiate listing.

GIM-II

GIM-II supports all of the above capabilities.

4.7.4 Arithmetic Capability

The arithmetical capability of a database package is the ability to evaluate expressions containing arithmetical operators and/or functions. These computations are usually built-in support routines working in conjunction with retrieval and update services of the database system. The range of arithmetic and statistical functions supported, and the ability to include user-defined functions are important elements of consideration.

GIM-II

The basic drivers for the arithmetic capability of GIM-II are editing and the User Language. The editing capability is a dictionary driven function, i.e., editing instructions (including arithmetical processes) are defined in the dictionary and, once established, are evoked automatically whenever the appropriate attribute field is mentioned in a processing statement.

The user Language provides for two distinct uses of the arithmetical capability: selection and calculated output. An arithmetic expression can appear as the object of a relational operator in a selection clause (see 4.6.2), or it may be used to perform certain computations on retrieved numeric data values prior to output.

In general, all expressions are evaluated for a target item (a logical record). They may be evaluated for every logical record of a target data list, but will not combine information from more than one logical record of the target data list.

As an example, consider a given set of data at the same structural level:

A. NAME	B. NAME
8	4
6	10
16	2
10	1

FOR . . . WITH A. NAME LT (A. NAME + B. NAME)/"2" . . .

In this query an arithmetic expression is used in the selection clause and will result in the second item to be chosen.

FOR . . . LIST "LABEL" = A. NAME * B. NAME #

This would yield the following value results:

LABEL = 32; 60; 32; 10

Several functions are provided as part of the basic GIM-II arithmetic capability. To illustrate the range we will only mention two:

NEAREST - The purpose of the NEAREST function is to acquire the value of an attribute "nearest to" a specified value.

Example:

FOR EMPLOYEE WITH NEAREST SALARY GT "10000" . . .

$DISTM - This is an example of a system function ($Function). It can

be used as a verbal command or within the selection and LIST clauses. It is intended to allow a user to get answers to such questions as "How many points are within X miles of this particular point?".

There is also a capability to define and incorporate new, general or tailored, functions into the GIM-II system. Once properly integrated into the system, these new functions can be used as a direct extension of the User Language.

4.7.5 Sorting

Sorting on the selected, retrieved data is usually provided by database systems. Some packages accommodate the sort function as part of their DML commands (on-line), while in others the sort function is performed by a utility program external to the system, but operating under the operating system as a library routine.

GIM-II

GIM-II can only sort items that are to be REPORTed. However, there is a way to circumvent this problem. The COPY function can be used to take selected logical records in one data list and place them in another data list. The ORDER command will then cause the newly formed data list to be sorted in ascending or descending order, right or left justified, using the item ID as the sort key.

4.7.6 Data Extraction

The data, satisfying the selection criteria and qualification process, may be extracted for intermediate processing or may be output in another machine readable form for use outside of the system.

GIM-II

The retrieval commands, EXTRACT and EXTRACT-TO-FORMAT, may be used for data extraction. EXTRACT may be used to produce a tape file that can be read by a user application program, for example, for off-line report generation. EXTRACT-TO-FORMAT is used to produce a tape with selected data values from GIM-II files in another machine readable form. For example, the data contained in a file and defined to be numeric, may be written to tape as floating point, packed decimal or fixed point binary to ease processing by a higher-level language program (such as FORTRAN).

4.8 Security and Error Recovery

As we have said in Chapter 3, a fundamental concept behind the integrated database approach to information management is to view all of the organization's data as a common, nonredundant resource available to all applications. However, not every user in an organization or department should have free access to all available data. Therefore, it becomes mandatory to control and restrict data access in a database system.

Security in a database management system refers to the protection of data against the unauthorized user, and it involves access control with respect to data (including system data such as data definitions), data manipulation commands, and even hardware. It is important to recognize that in a multi-user environment, applications software such as DBMS can not be any more secure than the hardware, operating system or communications software.

Error recovery refers to the protection of data from system failure due to physical equipment or software error, and depends on two basic steps. The first is reconstitution of the data and the second reestablishing the relationship between the various records of the database so as to include all mocifications up to the failure point.

4.8.1 Database Level Security

Typically, users are identifiable as individuals or groups via a mechanism such as account number/password combination. The system retains a profile of users involving identification information and their level of authorization. If an invalid account number/password combination is presented at SIGNON time, the user is denied access to the database.

GIM-II

GIM-II provides an extensive system of security checking that is intended to protect the user of databases, functions or verbs, programs, files, attributes and devices. This responsibility is ascribed to the Security Processor which interfaces with the other processors of the GIM-II system and provides:

- Protection when accessing the database, and
- Protection while processing transactions.

SIGNON Security

Access to the database is restricted by a valid SIGNON statement. It is the function of the SIGNON statement to identify the user to the system, to allow the selection of the user's database, to activate a series of security locks and keys which are applied during transaction processing, and to assign any default priorities for processing.

The SIGNON verb is followed by a series of specified

parameters (e.g., department name, operator name, and the name of the database) which have been established by the DBA. The processing of the SIGNON statement will use a security file (a system supplied data list) to verify that the organization name is correct and, for that organization, the operator, database and device names specified are acceptable. The input device name is not supplied by the terminal operator in his SIGNON statement but it is available to AMS (GIM-II executive) when he first logs on to ASSIST. If all items are verified, the operator will be requested to enter the password. When the password is evaluated (via the security file) the SIGNON will be processed.

If any of the required SIGNON checks fail, the SIGNON is considered invalid and the user is so notified. If the user exceeds the system threshold for the number of SIGNON statements that can be attempted without a valid SIGNON, the Master Terminal will be notified of a Possible Security Violation and the device "disconnected" from the system.

4.8.2 File and Field Level Security

Most systems provide protection on the file level and on the field level (e.g., the salary attribute in an employee file) based on independent authorities for retrieval and update.

GIM-II

The entry of the user to the system through the valid SIGNON statement activates a series of locks and keys which are applied during his processing of transactions.

File Level Security	The data dictionary of a data list (file) defines separate retrieval and update locks (IR/SC, UPD/SC) for the file. If the user has not been assigned the keys, he can not access the data list for retrieval or update
Field Level Security	GIM-II provides field level security on the basis of LOCK and KEY. Locks are character codes, defined in the dictionary, that may be associated with any attribute in the data list. The user must have the associated keys in order to be allowed access. In addition, LOCKS may be placed on any verb command and thus a user may not be able to perform certain functions. During the transaction cycle, all the verbs and attributes, referenced in either the primary or secondary data lists, will be

security checked. It should be noted that any attribute used in a selection clause must also pass its retrieval locks.

4.8.3 Record Level Security

A reasonably secure access control capability should provide for protection on the basis of the value of an attribute. For example, a manager may be authorized to access all records in a file except for his own, or he might be permitted to access "salary" unless the "employee-number" equals his own.

GIM-II

This capability is not supported. However, by defining a file to be segmented, different access authorities can be established for records in different segments.

4.8.4 Hardware Security

Access control may also exist relative to hardware. The system retains a profile for terminals or other output devices and controls messages and the output generated from the user's transaction that may be allowed to be sent to a given device.

GIM-II

Device LOCKS may be specified on any verb or attribute. These locks are evaluated using keys associated with the user's input or output device. Salary information, for example, may be restricted to certain terminals within the organization. Security checking will also be performed on message routing (output queued messages) before actual transmission and on directed output (output generated from the user's transaction). If any of the security checks fail, processing of the statement is rejected and a Possible Security Violation is logged.

4.8.5 Protection Against Direct Access to the Database

A database management system should provide a built-in mechanism which precludes any attempt by a programmer to directly access the database. This facility may simply trap all direct input/output calls which do not originate from the database management software.

GIM-II

This capability is supported.

4.8.6 Backup and Recovery

In an environment of sharing of data and on-line update, database backup and recovery takes on added importance. Backup is the generation of a copy of the database at some point in time established by the user or the system (a checkpoint). It may be retained on-line or off-line, and furthermore, it may be segmented to allow for selective reloading upon loss of a portion of the database.

Recovery means restoring the data to a previous state. The time of this state may coincide with the time of a backup dump or some later time during which the database has undergone changes. Transaction logging is a recovery feature that involves recording of actual changes to the database as updating occurs. After a system failure, the most recent backup copy of the database in conjunction with the input transactions logged subsequent to the dump allow for database recovery. A log file may contain either before or after images of a portion of the database each time it changes. Most systems, however, provide for both the before and after image of the original data on the log file. Loading a backup copy and reprocessing after image changes is referred to as roll forward. Taking the present database and applying before image changes allows the user to roll back his database to an earlier point in time (e.g., before the unsuccessful completion or erroneous action of an update process).

Some systems provide audit trails that keep records of transactions tagged with information sufficient to identify the source, cause, and

location of the change (user, program, date, transaction, database record, etc.). In some cases (e.g., two or more processes were updating the database concurrently when one of them failed), audit trails provide additional information for recovery or selective reprocessing of events and the data related to those events.

Finally, in order to provide continuous on-line availability of the databases it may be necessary, though not sufficient, to maintain two separate copies of each database on-line and synchronously updated by the database management system.

GIM-II

The operator is provided with commands in order to periodically dump and restore the databases of interest. As such, the database can not be restored to the point of system crash. However, a History Tape facility is optional in GIM-II. If implemented, the operator can issue a REPROCESS NEW command to update the RESTOREd database from the History Tape up to the last completely processed transaction. In addition, a HISTORY-ANALYSIS verb can be used to note which databases were active at the time of system failure or other error conditions.

Each time during the course of processing that a record is written to the History Tape the following information will be recorded:

- Transaction number, date and time.
- User and database number.
- A replica of the original statement.

- Various output messages and directed output information.
- Record images of any database records which are updated or changed.

Consequently, through the History Tape, the database administration has access to both old and new records, either of which may be REPROCESSed to return all or a portion of the database to a specified state at a given point in time.

4.9 Database Restructuring

The capability of a database system to grow, in terms of its applications and the data to which they refer, is fundamental to the database concept. Given an established database it is usually necessary over time to modify either the physical or logical structure organization of the data for such needs as efficiency or new applications.

Logical reorganization is the process of modifying data definition; such as changing validation information, modifying a data type, adding a new attribute field to a file, creating or removing a level in a tree structure, or adding a new relationship to a network. One way of doing this is to unload the database, redefine, and load again. However, some systems permit modification or incremental redefinition of an existing database, on-line.

The purpose of physical structure reorganization in a database system is usually to achieve efficiency in either space or time utilized in the storage and access of the logical records. In some database packages, the physical file organization used is of a specific type. No options or variations exist and the user's reference to all the logical records in any context is via this one physical file organization, established and operated by the system. As such, physical reorganization is not available as a direct function of the system. Other packages may make several physical organizations potentially available and leave it to the user to specify which of these should be used.

While physical structure reorganization normally will not require the rewriting of a user's application program (unless the program was written to take advantage or be dependent upon aspects of the storage structure), logical restructuring always carries the potential need to rewrite the programs accessing the database. Normally, however, modification of a program is in order if the program is to utilize the changed view of data.

GIM-II

There is relatively little difficulty in the logical reorganization of GIM-II files. This is true because of GIM-II's dictionary driven structure and the fact that GIM-II data files are, in a logical sense, structurally quite simple. Dictionaries may be changed at any time by ADDing, CHANGing or DELETing entries, and almost anything done when files are initially defined may be changed at a later date without reloading. In terms of the effect of structural modifications on the application programs, GIM-II users are relatively unaffected. It goes without saying that any application program requires adjustment if the data fields it is accessing have been modified.

4.10 Application Programming

The GIM-II User Language provides extensive facilities for the natural expression of data retrieval and update operations, and is particularly well suited to arbitrarily complex queries. However, a UL statement at a time in the only way the user can communicate with the system. In this sense, the user is unable to do much in the way of programmed manipulation of data.

The GIM-II Procedural Oriented Language (POL), which includes the UL as a subset, allows the user to control the action taken on data and to combine it for various purposes in precise detail. It permits the user to construct programs called Procedure Lists in which the program flow can be controlled by using conditionals, loops, etc.

Procedure lists can be used to:

1. Execute a series of GIM-II User Language statements as an entity. By storing these statements in a procedure list, they may be invoked from a terminal by entering a couple of words. This reduces the amount of typing necessary, and makes data available to authorized personnel without requiring them to know the GIM-II User Language rules.
2. Interact with the on-line user. Procedures can be written that allow prompting, printing and reading terminal replies. This allows "menus" to be a standard way of getting data into and out of the database.
3. Generate reports. Procedure lists can be written to select

data from anywhere in the database (as opposed to a single data list and its associated records) and combine it into a single report in any format desired.

4. Update several files from a single input. Procedure lists allow for passing a number of UL update statements to GIM-II. This provides for updating beyond the scope of Spans, Bridges, etc.
5. Perform batch updates. The POL contains verbs which can read sequentially stored input transactions, parse them into strings and convert the strings to the symbolic parameters which can be used to direct the execution of the UL update commands.
6. Get data produced outside the system and incorporate it into the database. Procedure lists can be written to get data from a non-GIM file into the database.
7. Create a non-GIM file. Procedure lists can be written that select data from the database and write a file that can be passed to other software systems.
8. Perform specialized processing that is not part of the generalized capabilities built into the GIM-II system.

Procedure lists are in the form of data lists and can be created on-line by using the standard UL update verb ADD. The Procedure List Dictionary defines the fields in a procedure list logical record, i.e., one program statement. These fields are:

Item ID - the label of the statement. The first statement that is compiled for every POL routine has the Item ID "START".

Operation Code (OP) - indicates what activity is to be accom-

plished at exectuion time.

Parameters (PARM) - specify data that is to enter into the execution of the Operation Code.

PL-Next-True (N/T) - the label of the next POL statement to be executed under normal program flow.

PL-Next-False (N/F) - the label of an alternate statement that can be given control.

PL-Next-Error (N/E) - specifies the statement to be executed after the detection of an error. This field is optional.

Consider, as a simple example, an interactive program called WHY that determines the area of a circle. STRUCTURE-PROCEDURE-LIST "WHY" #

```
ADD WHY "START" OP "VOPREAD" PARM " 'WHAT RADIUS', RADIUS"
N/T "2" #
ADD WHY "2" OP "EXPR" PARM "R = $NUM (RADIUS)" N/T "3" #
ADD WHY "3" OP "EXPR" PARM "A = 3.1416 *R * R" N/T "4" #
ADD WHY "4" OP "VOPRINT" PARM " 'THAT GIVES AREA OF' //
$ALF(A)" N/T "5" #
ADD WHY "5" OP "EXIT" #
```

The VOPREAD command is used to pass a message (WHAT RADIUS) to the user and to read the user's reply (radius). The first EXPR command is used to assign a value to a variable (R). The second EXPR command evaluates the area of the circle. The VOPRINT command is used to display the calculated area. The procedure list must be compiled (COMPILE WHY #) before it can be executed (EXECUTE WHY #).

The Procedure list will be executed using paging techniques that require only one page of the compiled procedure to be in core at any time. The same Procedure list user - written software will operate efficiently in any GIM-II environment. Procedure lists can contain subroutine calls to other procedure lists. In addition, new verbs or functions can be defined for an installation which upon being invoked will transfer control to a user written routine outside the GIM-II system.

4.11 System Generation

System generation refers to the installation of the database package software in the host computing system environment. As with operating systems, the software represented by a database package may allow several major to minor options that can be incorporated into the system at the time of installation. The degrees of control made available by the system are important elements of consideration.

A major area of concern is the ability to include or exclude a module of the database management system. For example, a database package may permit the user to remove a function such as on-line update upon system generation. This usually implies that the system has separate code modules for its interrogation and update facilities. Such capabilities therefore depend on the underlying design philosophy of the implementor.

During system generation the user specifies the start-up parameters for the system such as disk requirements, core allocation, and maximum number of concurrent transactions. Such items as number of files, limits on file sizes, and types of accessing methods might also require specification. Some DBMS's support an interactive dialog with the user at the time of installation.

GIM-II

Architectural Choices	A basic design criterion of the GIM-II system was extendibility. As such the system is totally

modular. A simplified block diagram of the various processors within a GIM-II system is presented in Figure 3. Different functions are ascribed to different processors. In principle, any of the modules or a given function within a module could be excluded from the system. However, such a capability is extraneous to system generation.

System Initialization

A GIM-II system consists of a minimum of four "databases". All databases are stored on direct access storage devices. The databases which comprise a GIM-II system are:

- Page Database - where the executable GIM-II software is stored.
- Two Extended-Storage Databases (ESDB) - which are used for buffering and message queuing. There must be one ESDB of page-size records and one of record-size records.
- System Database - where the SYSTEM FILES (SYSFILES) are stored.
- Any number of User Databases.

Only the User Database is utilized for permanent storage of user data. The other four databases are required for the execution of the GIM-II system.

The initialization of a GIM-II system is accomplished

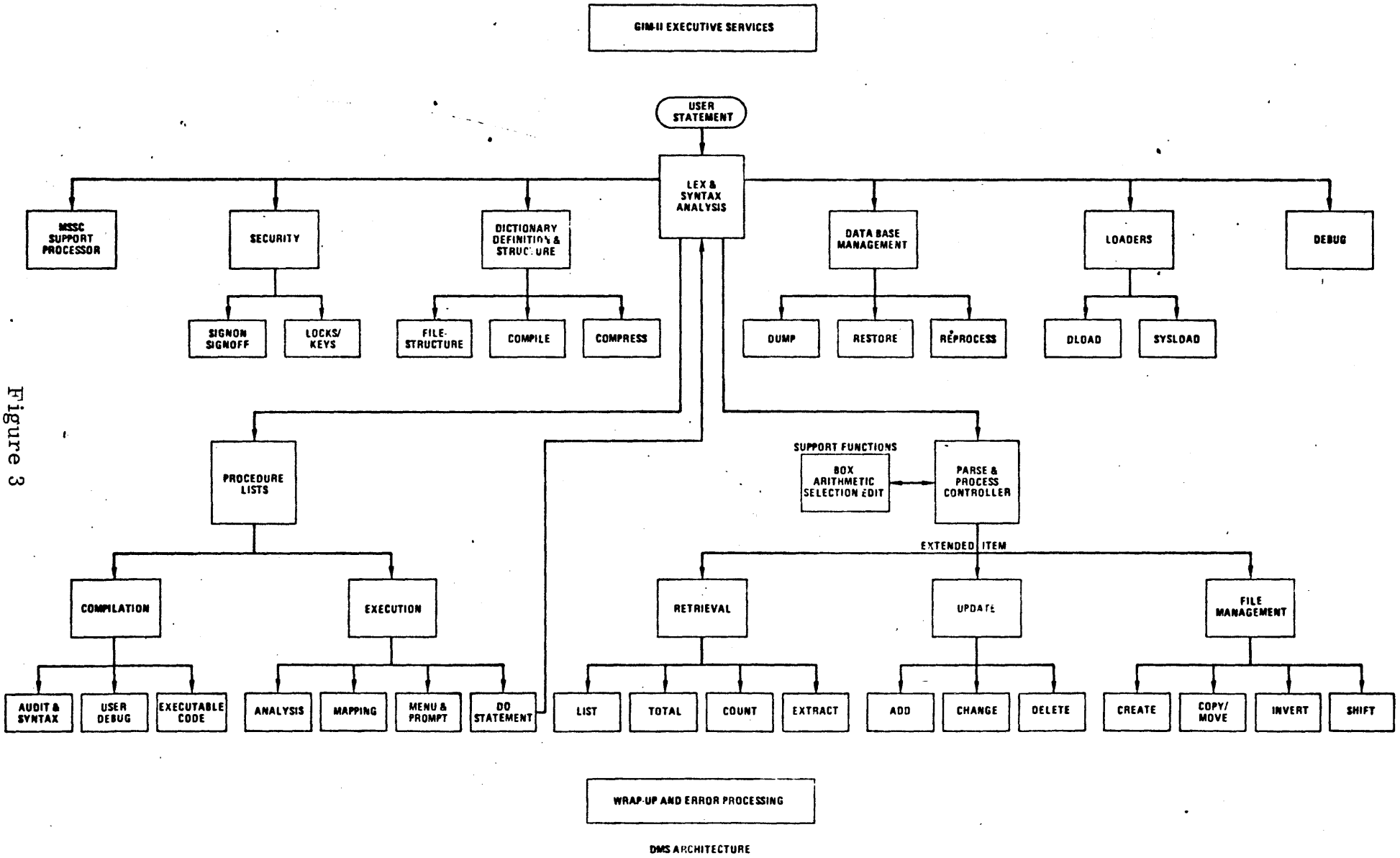

Figure 3

DMS ARCHITECTURE

by the following steps:

1. Using the appropriate system utility of the host operating system, allocate contiguous space for each of the four databases. It is not necessary that all the databases reside on the same volume.

2. Using a load utility write the GIM-II's executable software on the page database.

3. Using a GIM-II supplied library program, format the ESDB's and the system database into GIM-II record format. This process includes the establishment of the GIM-II physical record links, and the establishment of the BOOTSTRAP record containing the database parameters.

4. Using a normal GIM-II batch set-up, perform an initial data load (ZLOAD) to cause GIM-II system dictionaries (SYSFILES) to be written on the system database.

CHAPTER 5

HOSPITAL INFORMATION SYSTEM: AN APPLICATION FOR THE DATABASE APPROACH

Hospitals are very complex organizations of specialized resources, problems, and motivation. A modern hospital, perhaps more than any other social institution, is dependent upon rapid and accurate information flow. American hospitals spend between $7 and $10 billion annually to acquire and communicate patient information, a sum that represents between 24% and 33% of the total hospital budget.[28] It has been generally estimated that over half of this spending is for functions that can be automated.[29] Hospital information systems are seen by many leaders of the medical data processing field and health professionals as a solution to at least some of the major information handling problems in today's health care delivery systems. In this chapter, we consider the design and implementation of a hospital information system in a database environment.

5.1 Hospital Activities and Hospital Information System

There are four principal functional areas within a hospital: hospital management, patient management, medical services and medical research, and public health. Hospital management is concerned with the diverse tasks necessary for the operation of the hospital. Patient management deals with the individual patient before, during, and after his stay in the hospital. Medical services and medical research are the specific medical activities in dealing with the patient as an individual and

in general. Of increasing concern is the area of public health, implying both the introduction of tasks related to regional health care delivery systems and problems of preventive medicine.

All these areas are linked together by the tasks of communication and integration. It is in this area that the hospital information system renders specific services.

5.2 Background: Hospital Computer Systems

Historically, computers first entered hospitals in the administrative area. They were applied to the problems of patient billing/accounts receivable, payroll, accounts payable, general ledger, and inventory control. Computers are still being used in this way in the overwhelming majority of hospitals employing data processing facilities.

There are two types of hospital business systems, the stand-alone system, requiring the hospital to have its own computer, and the shared system in which several hospitals utilize the same equipment on a batch or partially on-line basis. The stand-alone systems are supplied by virtually all computer hardware vendors. Shared systems are available from various service bureaus that have risen to serve this market. Some of the systems in common use are IBM's Shared Hospital Accounting System,[30] Honeywell's Hospital Computer Sharing System,[31] and MEDINET.[30]

The system supplied by Honeywell, for example, consists of nine related subsystems designed to perform most of the routine tasks usually

assigned to the hospital business office. These are: patient accounting (including insurance apportionment, billing, and complete accounts receivable aging and control), general ledger/responsibility reporting, cost allocation, inventory reporting, property ledger, accounts payable, personnel records, payroll, and preventive maintenance.

The hospital business system solved only a small portion of the information processing needs of a hospital. The fact remained that while the practice of medicine had required more sophisticated information in greater quantities, the professional medical personnel had less time to process it. Thus, the need for computerized medical information processing was both present and urgent.

The initial efforts directly related to patient care activity were the bed census, admission scheduling, and outpatient appointment scheduling subsystems. These, consequently, led to the development of computer applications in the areas of pharmacy, dietary, and clinical laboratory.

Meanwhile, several institutions undertook projects for developing medical information systems, in which computerized patient medical records are maintained and the system is programmed to handle the information flow in the hospital. The various systems of Massachusetts General Hospital[30] and the project at Kaiser Foundation Hospital[30] are prime examples. While conceptually in the right direction, the software development effort for building and maintaining a medical database proved to be prodigious.

Commercial vendors entering the market focused upon computer-based automation to support, supplement, or replace the all-manual methods in communicating patient care information. Systems of this variety have been quite successful. Examples include: Total Hospital Information System by Medelco Inc.,[30] the REACH system by National Data Communications Inc.,[30-32] Technicon Medical Information System,[30] MediData,[30] and McDonnell Douglas Automation Company's shared hospital data processing service.[32]

The REACH (Real-time Electronic Access Communications for Hospitals) system, for instance, is essentially a hospital communications and data collection system. The only instrument used by hospital personnel in the operation of the system is a special terminal designed and built to NDC specifications by Raytheon. The terminal consists of a standard CRT display and keyboard with two unique features: a badge reader for operator identification, and a series of twenty-line selector buttons along the left margin. Each hospital data input and output function has a set of special displays and an associated logical tree structure of options designed to capture all orders, convey all messages, and store all relevant information within the scope of the system.

In a typical medication ordering sequence,

1. The physician inserts his badge in the badge reader. The card coupled with the terminal identifier code establishes authorized access to the REACH system and those system programs available to that user.
2. The system responds with a list of patients being cared for by

this doctor (in the case of a nurse, the list would consist of patients on her ward). The doctor selects the desired patient by depressing the button next to his name.

3. The system displays to the physician a list of options that include X-ray, laboratory, dietary, and pharmacy. The physician selects pharmacy.
4. Subsequent displays allow the physician to select generic or trade names for drugs or to order them alphabetically; to specify the dosage and frequency of administration; and to view the finished prescription before transmission to the pharmacy. Printer units supply a hard copy of the order where required; for example, in pharmacy, a printed gummed label is produced for use in dispensing the medication. The above medication order would normally be entered in 15 to 20 seconds.
5. The physician may now return to the list of options on his patient roster, or sign off, removing his badge and thus disabling the terminal.

A similar operating procedure is followed by the nursing staff, admission clerks, technicians and all other hospital personnel interacting with the system.

The REACH system maintains a complete computer patient chart reflecting all orders and services applicable to the patient during his hospitalization. The system captures the order of a chargeable item and thereby records the charge connected to the delivery of the item automatically. Upon discharge, the patient chart is unloaded to magnetic tape for subsequent reference.

One of the most interesting features of the REACH system is its dedication to extreme reliability. This is accomplished by duplicating virtually every piece of hardware. The system is completely proprietary and never sold in source form. While the system supports a broad range of application programs, the user can not modify anything, and hence from this viewpoint flexibility is extremely low.

5.3 The Objectives and Requirements of a Hospital Information System[30, 32]

It is not possible to state accurately all long-term goals and objectives of a hospital information system. As medical science modifies medical practice the goals and objectives change. Nevertheless, the principal function of a hospital information system remains in the area of communication and integration of patient medical data.

The usual objectives are to:

1. Streamline hospital administrative procedure, thereby freeing professional personnel for direct patient contact.
2. Improve the quality, quantity and utility of patient data, as well as providing a more rapid means of communicating this information.
3. Communicate patient data from the professionals providing care (doctors, nurses, technicians, etc.) to the patient's computer medical record, and then to other professionals (e.g., dietitian) or other departments in the hospital (e.g., radiology or pharmacy).
4. Communicate patient data from various subsystems

(e.g., automated multiphasic health testing) into patient's computer medical record.

5. Establish communication between clinical services (i.e., nursing stations) and ancillary services (e.g., laboratory, pharmacy).
6. Increase the efficiency, economy and safety of the logistics of patient care; for example, in the ordering and administration of medication, menu planning, patient scheduling and utilization of medical care facilities.
7. Establish a database for administrative and business functions.
8. Establish a medical database that maintains clinically and administratively relevant, readily accessible medical data for each patient served in the hospital (including the outpatient department), and that provides a source for the systematic retrieval of medical data across large numbers of patients.
9. Provide data necessary for projection of health care needs for the hospital and also the community.
10. Permit growth and development without major reorganization effort.

In order to achieve the above objectives, the hospital information system must meet the following broad requirements.

A. All data should be entered directly into the computer from source. For example, the physicians should enter their medical orders and diagnoses directly into the system, ideally using a free text approach.

B. Extensive automatic validation of data at the input level should be supported.

C. To maintain an integrated computer record for all data for each individual patient for every hospitalization and outpatient visit.

D. The system should readily make available any part or all of the data in the patient's computer medical record.

E. The system should support message switching functions. That is, the process by which the computer receives a message from an input device at point A (e.g., a nursing station) and routes the message to an output device at point B (e.g., laboratory).

F. The system should have the capacity to support administrative functions, such as:

    1. Patient scheduling, including outpatient appointments; admissions; bed census; and scheduling of medical care facilities.
    2. Scheduling and control functions for personnel, supplies and equipment.
    3. Business functions, including posting of charges, billing, payroll, etc.

G. To provide database interrogation capabilities for the purpose of (1) health service research, (2) hospital service evaluation and planning, and (3) medical education.

H. To satisfy stringent security requirements.

5.4 The Architecture of a Hospital Information System

5.4.1 Approach

There are two different approaches to the development of a hospital information system: the "total" systems approach, and the "modular" approach. In the "total" approach extensive effort is made to identify all of the information handling tasks of the hospital, and to prepare a comprehensive picture of the total data flow. A universal system is then developed that attempts to (1) do everything for everyone in the hospital, and (2) prohibit tasks that do not go through the system. This approach presupposes the availability of a large computer facility, a great number of input/output devices, an extraordinarily powerful data-base management system and communications software that can handle simultaneously many specialized and diverse applications, and total acceptance by medical personnel. The sheer enormity of this approach has not yet found anyone with the vast resources needed for successfully implementing a total hospital information system.

In the "modular" approach the hospital is considered to be a set of nodes in a data flow diagram. The nodes are chosen in a way that will clarify the procedures involved and still preserve some relationship to the usual global view of hospital operations. The functional information processing needs of each node are identified and installed in a modular fashion. To insure the success of modular implementation, however, it is absolutely essential to plan from the beginning to integrate the various operational modules into the eventual total hospital information system.

Many hospital information system projects failed because when they tried to integrate already successfully implemented subsystems (e.g., patient scheduling, admission, bed census, medical records, pharmacy, clinical laboratory, etc.), they discovered serious incompatibilities between the various modules that required major reprogramming at prohibitive costs. The solution to this problem is to develop a common central database containing an integrated, continuing computer record for all data from each individual patient for every hospitalization and outpatient visit, and to require the different modules (with or without small dedicated computer) interacting with the database only through the database management system.

5.4.2 Architecture

The architecture to be considered here for the V.A. health care delivery system is shown in functional block form in Figure 5.1. At the heart of the system are the medical database, the DBMS, and the teleprocessing monitor system. The medical database serves as the central register of continuous, integrated patient records. The DBMS permits building and maintaining the integrated database, and functions as the single source for retrieval and storage of data in the database. The teleprocessing monitor system handles transactions with the DBMS, and provides communication between the various periphery modules.

Each of the outlying application subsystems would be responsible for its special area of interest, and for assuring proper input of patient data to the central record. For example, the admission subsystem is an

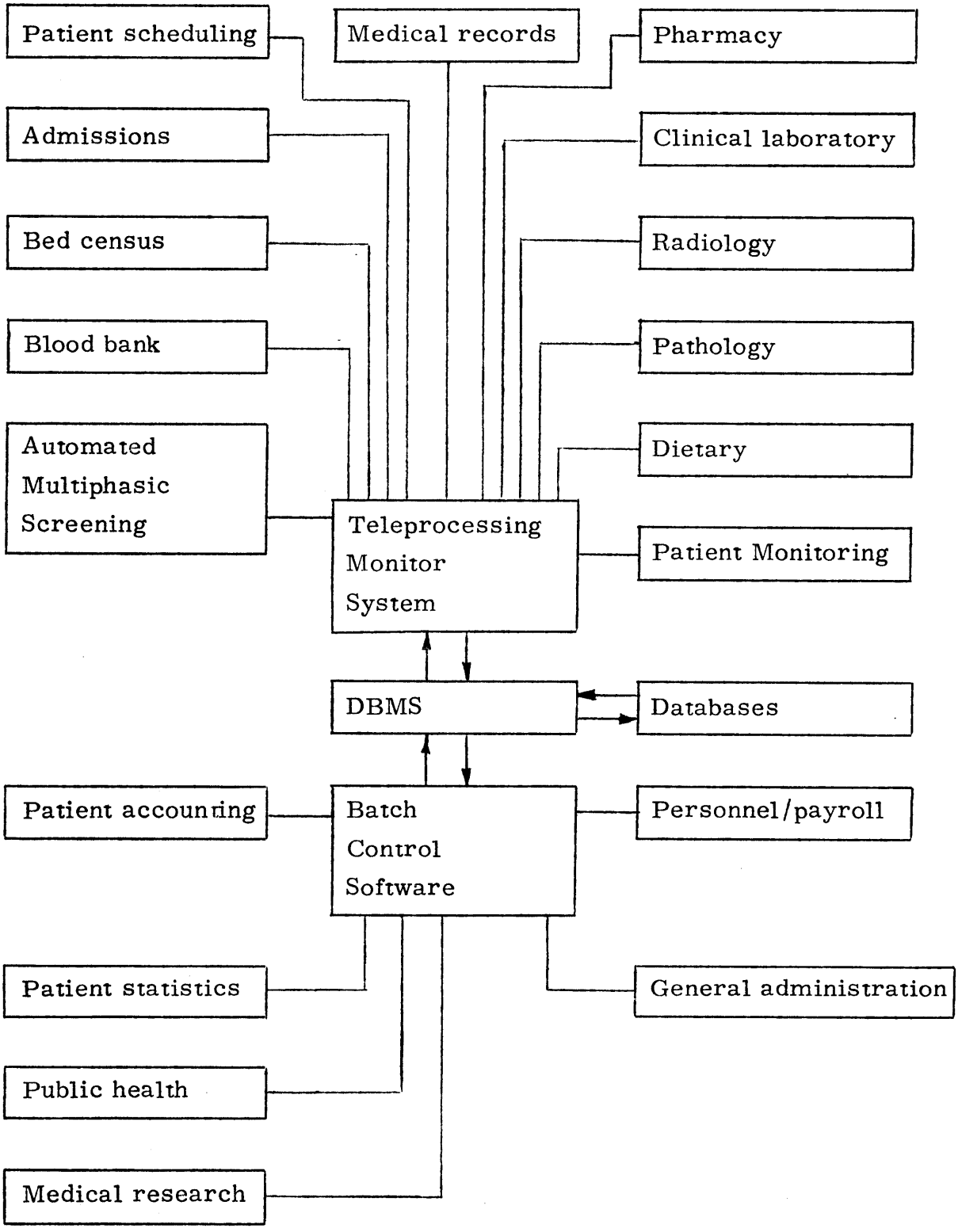
Patient scheduling
Medical records
Pharmacy
Admissions
Clinical laboratory
Bed census
Radiology
Blood bank
Pathology
Automated Multiphasic Screening
Dietary
Teleprocessing Monitor System
Patient Monitoring
DBMS
Databases
Patient accounting
Batch Control Software
Personnel/payroll
Patient statistics
General administration
Public health
Medical research

Figure 5.1

application program that maintains a dialogue with the admitting clerk. The admission dialogue would consist of questions about the patient's identification, next of kin, insurance type, referring physicians and so on. If a previous record does not already exist, a record is created; otherwise the existing record is updated and checked for formal errors. A second type of application area (clinical laboratory, patient monitoring) would require the application program to interact with an automated system for acquisition of patient data. Finally, those application areas that do not require real-time response can be served by programs that interact with the database in a batch mode.

It should be noted that many of the application subsystems will maintain their own special files for solving information problems internal to the area; however, extreme prudence must be observed in assuring that these files will also serve the overall clinical service, research, and administrative needs of the total health care delivery system.

5.5 Designing the Hospital Information System

Having decided to utilize the database approach to solve the communication and integration problems inherent in the medical environment, we must now examine how the various application modules can be implemented using the capabilities of the candidate database management system. In contrast to Chapter 4, where the capabilities of the DBMS were evaluated in isolation from any specific application, our discussion here will bring to focus those capabilities in light of the specific requirements of the hospital information system. This will eventuate in the determination of the extent of suitability of the package.

5.5.1 Medical Records

Statement of Problem

The medical record is the fundamental repository and basic source of patient information in the medical environment. It would be necessary for each patient's computer record to have the capability of being a continuous lifetime record containing all essential past, present, and future inpatient and outpatient data. The record would also contain patient's data from any facility within the health care delivery system.

This continuous integrating record concept is clearly a major undertaking; however, once the continuous integrating medical record is available, adding application subsystems becomes a much easier task. Even with the need for this flexibility, it is evident that the resultant amount of essential medical data will surpass acceptable storage costs

on currently available direct access devices in a system with a large number of patients.[33] This problem notwithstanding, the computer processing of large records may be undesirable. Consequently, it is questionable whether it will be feasible to maintain indefinitely an integrated comprehensive file of medical records in a dynamic direct access system.

A revised strategy is to maintain for all patients condensed relevant information on-line. A record in this master file would contain identification information (hospital number, name, address, sex, date of birth, occupation, etc.); critical medical information (blood type, allergies, etc.); information from previous inpatient visits (date, admitting diagnosis, treatment, status at time of discharge); and information from previous outpatient visits (date, diagnosis, treatment).

The information flow from daily hospital work can be handled by an inpatient file. The prime purpose of an inpatient computer record is to serve the functions of both the medical record and the medical chart during the patient's hospitalization. As a result, the record would contain selected history, physical examination, and progress report data; all diagnoses; all physician-reported examinations; doctors' orders; all clinical laboratory test results; all drugs administered; and essential summarized patient monitoring data from intensive care areas.

When the patient is discharged, the master file is updated with summary information, and remaining data are then unloaded on tape so that they may be subject to reloading or statistical searches.

Implementation

The medical field lends itself nicely to strictly defined file structures in the sense that complex data structures are usually not required. For example, while the capacity for defining the various files to be inter-related is required, data structures within a record beyond the simple parent-child relationships are typically not required.[34] This is the case for the medical record files discussed above.

A typical GIM-II master patient record would have the structure shown in Figure 5.2. Note that most of the fields are multi-valued. In other words, each time the patient visits the hospital for some type of clinical examination or surgical procedure over the year(s), pertinent data is recorded in the appropriate field.

The inpatient file would be structured as a segmented GIM-II datalist in which the logical records are directed to different segments of the same file, based on some specified criteria, e.g., the appropriate ward. This will allow the concentration of information for each nursing station, and will thereby reduce the security requirements. The structure of an inpatient record is shown in Figure 5.3.

Restrictions and Limitations

GIM-II does not place any restrictions upon the size of a field, logical record, or file. The only limitation is that the file structure does not extend beyond the subfield level, i.e., root plus two levels.

Key = hospital record number

Name

Address

Sex

Birth date

Blood type

Allergies

Drug profile record number

Admission dates

- Physician
- Diagnosis (coded)
- Operation date
- Status at discharge (text)

Outpatient - Visits

- Clinic
- Physician
- Diagnosis
- Treatment (text)

Figure 5.2

Key = inpatient record number

Hospital record number

Next of kin

Phone number

Bed

Referring physician

Admission date

Appointment file record number

Admitting doctor

Medical history (text)

Diagnoses

Physical monitoring data - Date

Time

Pulse rate

Blood pressure

Temperature

Laboratory tests - Type

Result

X-ray (text)

Pathology (text)

Doctors' orders (text)

Drug profile record number

Progress report (text)

Figure 5.3

5.5.2 Patient Scheduling and Booking

Statement of Problem

Patient care involves qualified personnel and expensive equipment distributed over several clinical specialities and service units within the hospital. For purely economic reasons, it becomes necessary to use the hospital facilities to the maximum extent. An application program for scheduling patient admissions and outpatient visits is instrumental in achieving efficient utilization of existing hospital facilities and stabilizing staff workloads. Greater efficiency in handling a large number of patients not only would reduce the cost per patient to the hospital, but would shorten the waiting times of patients and the idle time of health personnel.

The primary functions of a patient scheduling and booking system are:

1. To schedule patient admissions and surgical procedures.
2. To schedule appointments in the outpatient clinics.

The two functions, though dissimilar in details, are analogous in a significant, basic way - the scheduling program must coordinate the resources required for the patient's course through the health care delivery system.

A typical admission scheduling cycle begins with a request by a physician for a patient booking. The request is made for admission on a particular date, usually with pre-operation examinations (such as EKG, X-ray, etc.) on the same date and the surgery to follow on the next day. If the facilities and bed space are available the booking is confirmed; otherwise

another date must be negotiated. Keeping track of the available times in various facilities is relatively straightforward; however, forecasting the availability of bed space proves to be a more challenging problem. A bed costs a considerable amount of money to maintain per day, whether there is a patient in it or not. Nevertheless, a hospital should have a certain number of empty beds to provide for the day-to-day variation in the number of patients. Once a decision is made as to how many extra beds the hospital needs, data may be stored on estimated lengths of stay according to diagnosis, surgical procedure, age of patient, and other relevant information. Then, any time a booking is made, the program would adjust the projection of the total number of available beds according to the estimated length of stay.[35]

Outpatients' clinic scheduling involves several possible levels of complexity.[36, 37] If a sufficiently higher priority is given to personnel time than to patient time, then several patients may be scheduled for the same time (block scheduling). If patient convenience with respect to waiting is given a sufficiently high priority, then each patient is scheduled for his own appointment. The basic scheduling function must simply keep track of the current time available for appointments at each clinic and for each physician (so that the patient would be seen by the same doctor in multiple visits).

Implementation

The patient admission scheduling and booking program would be written in POL (see 4.10), and would be parameterized. The program is invoked from a terminal, which results in the display of a scheduling form (the

list of parameters). The parameters are used to guide the scheduling process and include:

1. Patient Hospital Number - This is the patient's master file record number.
2. Medical Procedure Groups - A medical procedure group is a number of medical activities to be scheduled on the same day (e.g., pre-operation examinations). Usually two medical procedure groups are separated by a number of days (e.g., surgery to follow pre-operation examination).
3. Facility Selection - There are several units within a hospital capable of performing a medical procedure (e.g., operating rooms). As such, the program would select a facility from the group. If, however, a specific facility is desired, this parameter would indicate the special consideration.
4. Date Specification - The first available date is always sought. But, if a specific date is requested, only that date would be checked. A variation is to request admission after (or before) a specified date.
5. Priority Specification - Emergency time is reserved for each facility. A priority specification would allow the program to schedule the patient as soon as possible by using the reserved time.

Several files are required by the admission scheduling program.

A. Unbooked Time Slots File - This file contains information about the currently available time slots at each facility in the hospital for which a schedule is to be maintained.

The file encompasses a period of two or three months. Included in the file are the projection of available bed space. (See Figure 5.4.)

B. Medical Procedure File - This file contains information about each medical procedure (operations, examinations, etc.), the facility that can provide the service, the expected duration of the procedure, pre-examination and post-examination time, and the estimated length of hospitalization. (See Figure 5.5.)

C. Facilities File - This file contains information about the various facilities, such as name and location of each unit, reserved time for priority cases, etc.

The above files are defined and initially loaded using the GIM-II User Language. The scheduling program uses them to update the appointment file. (See Figure 5.6.)

Scheduling Logic

After the scheduling form is completed, the program retrieves information (facility and duration) for each appointment requested. The expected length of hospitalization for the surgical procedure is used to determine the first date nearest to the specified date for which a bed is available and this availability continues for the duration of stay. This date is the target date for scheduling the first medical procedure group.

The available time for each unit within a facility can be broken up into consecutive 15 minute slots. Such a representation is then encoded as a bit string. For example, an 8 hour day (9 to 5) is initially represented

Key = date

Facility 1 (e.g., Operating-Room)

Unit (coded)

Unbooked time slots (bit string)

Facility 2

Unit

Unbooked time slots

•

•

•

Facility n

Unit

Unbooked time slots

Number of private beds available

Number of semi-private beds available

Figure 5.4

Key = medical procedure code

Procedure name

Facility

Duration

Pre-examination time

Post-examination time

Estimated length of hospitalization

Figure 5.5

Key = date

Facility 1

Unit

Appointment time

Hospital record number

•

•

•

Facility n

Unit

Appointment time

Hospital record number

Figure 5.6

as 32 one's, each byte denoting 2 hours. Then, if an appointment for 10 - 11:15 is made, the unbooked time slots attribute of the unit would be changed to 11110000011···1. Now, if another appointment is to be made that requires 115 minutes (8 time slots), the attribute is examined for the pattern ---11111111---. This would result in the following alternatives: 11:15 - 1:15, 11:30 - 1:30, ···, 3 - 5.
The target date is thus examined for possible appointment times for each procedure in the group.

In order to determine a compatible schedule, the available appointment times for procedures must be checked against one another. The appointment times are compatible if and only if the time slots do not overlap. An AND operation can be used for testing.

Once a compatible schedule is found, the program retrieves the record corresponding to the date on which the next medical procedure group is to be scheduled. (Note that this date is implied by the interval specification in the scheduling form.) A similar process of determining a compatible schedule is performed for this date. Failure would result in revising the target date.

Projected appointment schedules are displayed on the terminal for final selection. Selected schedule is used to update both the appointment file and the unbooked time slots file.

Program

This section is intended to be only representative of the coding of the

scheduling program in POL.

Scheduling parameters can be received through prompting, i.e., passing a message to the user and reading the user's reply.

Stmt. Label	Operation-Code	Parameters
Start	VOPREAD	Medical Procedure Group 1, $A, 1, ',', E
	VOPREAD	Interval 1, F Days
	EXPR	I1 = F Days
	VOPREAD	Medical Procedure Group 2, $B, 1, ',', E
	VOPREAD	Interval 2, S Days
	EXPR	I2 = S Days
	VOPREAD	Surgical Procedure, Operation
	EXPR	SP = Operation
	VOPREAD	Date Specification, Day
	EXPR	Start-Date = $I DATE (day)

The first statement would cause the input character string (procedure codes separated by comma) to be scanned and each procedure code to be stored in the global array $A.

For each appointment requested, pertinent information must be retrieved from the medical procedure file.

Stmt. Label	Operation-Code	Parameters
	EXPR	I = 1
NO	DO	FOR MP '$A(I)' ACQUIRE Facility Duration
	GFAV	Facility
	EXPR	$C = $W
	GFAV	Duration
	EXPR	$D = $INT ($W/15)
	TEST	I EQ $AM NF = No

Values from the database are always made available, one at a time, in an area to be addressed as $W. The DO/ACQUIRE statement places the Item ID in $W. The GFAV statement places the value stored in the attribute field Facility in $W. Since this value is to be used later in the procedure it must be stored. $AM is the maximum current entry of the array.

The target date must now be found.

Stmt. Label	Operation-Code	Parameters
	DO	FOR UTS Nearest to 'start-date' WITH Bed GT 'O' ACQUIRE Date

```
                    EXPR        Target-Date = $W
                    EXPR        I = 1
Loop                EXPR        Next = Target-Date + I
                    DO          FOR UTS '& Next'
                                ACQUIRE Bed
                    GFAV        Bed
                    EXPR        B = $W
                    TEST        B GT 'O'        NF = Fail
                    TEST        I EQ ELOS       NF = Done
                    EXPR        I = I + 1       NT = Loop
Fail                EXPR        Start-Date = Next + 1
```

For the target date possible appointment times for each procedure must be determined. The program branches to this part according to the number of procedures in the group.

```
Stmt. Label         Operation-Code      Parameters
TWO                 DO                  FOR UTS '& Target-Date'
                                        Acquire & A &B
                    GFAV                & A
Loop 1              EXPR                Unbooked-Time = $W
```

```
                    EXPR        I = 1

Loop 2              EXPR        Test-String = $BS (32, I,
                                Duration)

                    TEST        $MATCH (Test-string,
                                Unbooked-Time)
                                EQ '1'          NF = No

                    EXPR        $D = Test-String

NO                  EXPR        I = I + 1

                    TEST        I EQ 32         NF = Loop 2

                    GNAV        & A             NF = Loop 1

                    TEST        $DM EQ 'O'      NF = Done

                    EXPR        Start-Date = Target-Date + 1
```

The GNAV statement places the next value stored in the attribute A (i.e., unbooked time slots of various units within the facility) in $W. The function $MATCH performs an AND operation on test-string and unbooked-time, and compares the result with test-string.

Once possible appointment times for each procedure in the group is found (e.g., the entries in the global array $D), a compatible schedule must be determined.

```
Stmt. Label         Operation-Code          Parameters

                    EXPR                    I = 1
```

Loop 1	EXPR	J = 1	
Loop 2	TEST	NOT ($ANY (D(I) AND E(J)))	NF = Next
	EXPR	$F = D(I) $G = E(J)	
Next	TEST	J EQ $EM	NF = NO2
	TEST	I EQ $DM	NF = NO1
	TEST	$FM GT 'O'	NF = Fail
NO2	EXPR	J = J + 1	NT = Loop 2
NO1	EXPR	I = I + 1	NT = Loop 1

The function $ANY gives a numeric 1 (true) if any one or more bits in the argument are one bits.

Once scheduling is completed and final selection is made by the terminal operator (e.g., the 5th alternative), the appointment file is updated:

```
DO          FOR APP '& target-date' ADD & A 'F(5)' & B 'G(5)'
```

As it can be seen from the above description, the coding of the scheduling program is quite involved. Query capabilities by themselves are not sufficient. However, the POL as a programming language is equipped with the facilities needed to implement the scheduling logic.

Scheduling Response Time

The scheduling response time is defined as the elapsed time between the end-user's final specification for a scheduling transaction and the appear-

ance of the first output character on the terminal. Almost by definition database applications are input/output intensive. Every query will involve at least one target datum, and many target references will involve access to secondary storage. As such, it can be expected that the significant portion of response time is due to the cost of interacting with secondary storage. In the scheduling program, the CPU is used considerably for non-database computations. However, these costs can be considered small since the computations are done in main memory and any processing time spent here is small compared with the cost of interacting with secondary storage. Thus, we will consider here the number of database accesses made to retrieve the required information.

In the GIM-II system, logical records are grouped into fixed length physical records (pages). The physical record is the unit of data transmission between secondary storage and memory. The physical record size for user data is 1024 bytes. A request for a target logical record proceeds as follows:

1. Determine the index entry for the record.
2. Search the buffer area (kept in main memory) for the page containing the target record. If the page is missing, then fetch the physical record (and if necessary remove a page to make room).
3. Find the target record within the page. Acquire target attributes and perform desired operations.

When a file is randomly stored in secondary storage and is not clustered according to any criteria, the number of database accesses required to

resolve a query on the same attribute of n records will be n. On the other hand, if the file is organized sequentially, one database access often will satisfy several target requests, especially when the logical record size is small compared with the page size.

There are two types of queries involved in the scheduling program:

1. Direct access to the medical procedure file.
2. Sequential search of the unbooked time slots file.

The medical procedure file is distributed randomly. However, because of its static nature it can be considerably clustered. This means that if there are k appointments to be scheduled, the number of database accesses is significantly less than k and closer to the number of procedure groups (the average record size is hardly greater than 30 bytes).

The unbooked time slots file is organized sequentially. Assuming that there are 25 facilities for which a schedule is to be maintained, the record size would be approximately 130 bytes. As a result the entire file (90 records) would occupy 10 contiguous physical records. And in the worst case 10 database accesses will be required.

Remarks

The outpatient clinic scheduling program differs from the admission scheduling system in that only the clinic and physician availability need to be examined and that a shorter period (e.g., one month) of unbooked time slots needs to be maintained.

The byproducts of the patient scheduling and booking system are:

1. On-line schedules of each facility, clinic, and physician.
2. On-line cancellation of appointments.
3. Preparation of admission lists and schedules for the following day.
4. Gathering data for evaluating and adjusting estimated length of stays, broken appointments, etc.

5.5.3 Bed Census

There is no specific application programming required for a bed census subsystem. A segmented file of all available beds (see Figure 5.7) is maintained, and when a patient is admitted the patient hospital number is joint-stored in the inpatient file (see 5.5.1) and the assigned bed record. Inquiries and transfers are directed to the bed file using the normal GIM-II User Language.

	Bed Number	Patient	Private Semi-Private	Phone
Ward 1				
•				
•				
•				
Ward k				

Data for all beds in the hospital

Figure 5.7

5.5.4 Pharmacy

Statement of Problem

Pharmacy is an integral component of the health care delivery system for both the inpatient and the outpatient. Input to the pharmacy consists of medication orders or prescriptions and patient medical record summaries. The latter allow the pharmacist to examine the kind and amount of drugs being taken by the patient. This is important for two reasons. First, drugs have effects in combination with other drugs or treatments not always known to the average physician. Second, a patient may be seeing several doctors who may prescribe the same drug independently, the total of which would be an overdose. It is the pharmacist's responsibility to notice such incompatibilities.

The screening of drug orders for allergies, possible drug interactions, contraindications, and overdose is a task of almost impossible magnitude in a hospital pharmacy without some kind of automation aids. Hundreds of orders are processed each day by the pharmacy of a medium-sized hospital. There is no time to examine manually the medical record of each patient receiving a drug.

Implementation

Two kinds of data are required for the operation of the pharmacy subsystem:

1. Data on each drug in the formulary with usual or maximum doses, potential drug interactions, interference with certain

foods, contraindications, and interference with certain laboratory or diagnostic tests. (See Figure 5.8.)

2. Patient drug profile information. (See Figure 5.9.)

It has been estimated that every hospital patient receives between two and three prescriptions on the average each day he is in the hospital. For this reason, it would be reasonable to maintain independently an inpatient drug profile file to accommodate the temporary increase in activity. (See Figure 5.10.)

Note that patient identification information need not be stored in the drug profile record, as it would automatically be retrieved from the patient's master record by the GIM-II system. Furthermore, once a drug inventory file is established, the dispensing of drugs would result in simultaneous updating of inventory status. Also, on the addition of an entry in the inpatient drug profile record the system can trigger the calculation of the charge connected to the delivery of that item in the patient's billing record (see 4.5.4). The interrelationships that exist between the various files are depicted in Figure 5.11.

Key = drug code

Generic name

Trade name

Usual dosage

Maximum dosage

Drug interactions

Food interactions

Contraindications

Remarks (text)

Figure 5.8

Key = drug profile record number

Hospital record number

Date of dispension

Drug code

Amount dispensed

Instructions for administration

Number of refills permissible

Doctor ordering

Prescription number

Pharmacist identification

Figure 5.9

Key = inpatient drug profile record number

Inpatient record number

Date of medication order

Drug code

Dosage

Route of administration

Frequency of administration

Instructions for administration

Amount dispensed

Doctor ordering

Pharmacist identification

Time of drug administration

Individual administering the drug

Patient reaction

Figure 5.10

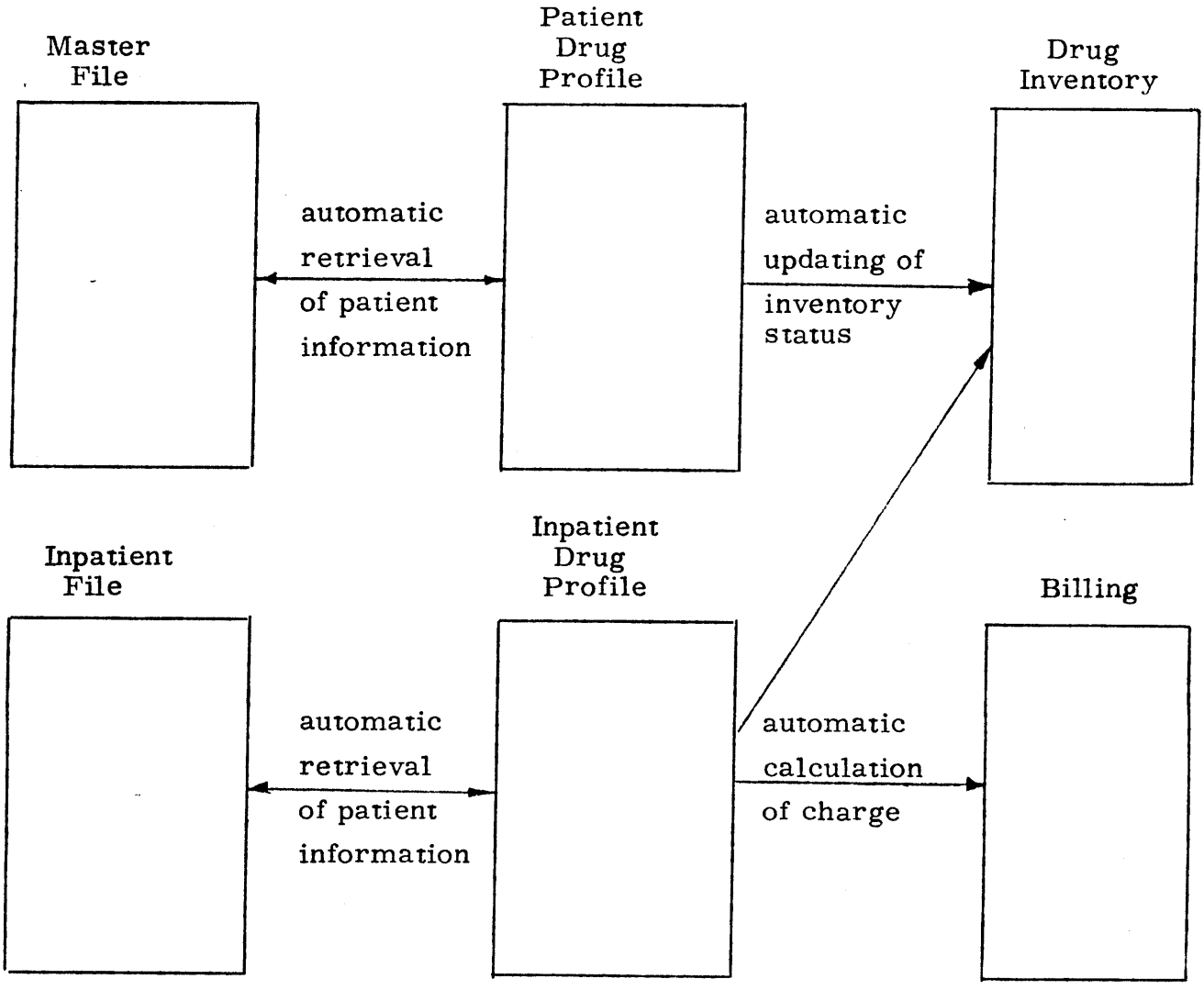
Master File
Patient Drug Profile
Drug Inventory
automatic retrieval of patient information
automatic updating of inventory status
Inpatient File
Inpatient Drug Profile
Billing
automatic retrieval of patient information
automatic calculation of charge

Figure 5.11

5.6 Assessing the Results

Does the GIM-II system have the capabilities to function as the heart of an automated health care delivery system? Does it satisfy the requirements of the Veteran's Administration Hospital? The answers, based on the evaluation of system capabilities presented in Chapter 4 and the examination of user requirements in this chapter, are yes. Some qualifications, however, are necessary.

Strong Points

1. Non-procedural User Language for on-line query and update.
2. Procedural Oriented Language capability (see 4.10).
3. The capacity to support transaction-oriented functions such as patient scheduling through pre-defined POL routines with parameter substitution capability.
4. The ASSIST teleprocessing system has the ability to communicate with other computers with a minimal amount of programming. This compounded with the fact that procedure lists can be written to read non-GIM files, analyze the data fields and cause them to be used for updating the database, permits the integration of stand-alone computer systems (such as Laboratory) with the GIM-II based hospital information system.
5. Applications are transportable to and from the V.A.'s GIM-II systems.
6. GIM-II will quite easily accommodate growth of the database. New fields can be added to records without any modification of the programs which use the database. In addition, the ability to

structure new files at any time, then couple the new file to an existing file through a common attribute adds a great deal of flexibility to the growth potential of the GIM-II system.

7. GIM-II file sizes are limited only by PDP-11 disk storage capacity. For the PDP-11/70, this is 88 million bytes/disk x 8 disks, or 704 million bytes. GIM-II handles all space management, i.e., automatically reallocates released and deleted storage areas.
8. All data going to or coming from the database is handled by the GIM-II system. All security locks and editing specifications are active. This maintains the security of the database, while making the database available to user written application programs both for retrieval and update.
9. GIM-II imposes little formal structure on the logical records. The user's view of the record is quite natural. The basic problem is one of organizing the database into a structure that reflects the needs and requirements of the user. For the user who will use the User Language only for retrieval and update (this covers almost all medical personnel), the skill requirements are indeed minimal.

Weak Points

1. Even though GIM-II supports textual data, its textual processing capabilities are limited.
2. The GIM-II system uses a pseudo-random method of addressing logical records. Consequently, one record is as easy or as hard to access as any other record, and the efficiency with which it is accessed is somewhat independent of what preceded the access.

This hashing capability to any record within a file is well suited for applications that are oriented toward individual record retrieval. However, it does not facilitate an efficient method for resolving queries. Even though the GIM-II system permits the user to create cross-reference files, the normal method for resolving queries is by examining each logical record.

Fortunately the problem is not as bad as it seems. There are two reasons for this. First, most patient care retrieval applications are individual record oriented. And secondly, the user will usually tolerate a long response time to his query.

3. There are some questions regarding the number of active terminals and concurrent queries the GIM-II/ASSIST system can support. The capabilities, however, are expected to be significantly increased with the later versions of the system.

CHAPTER 6

SUMMARY

The process of selecting a database management system consists of the following steps:

1. Define the organization's requirements.
2. Develop a compatible set of user needs as they relate to DBMS concepts.
3. Map the user requirements into the features of database management systems which can satisfy them. Assign weights to each capability.
4. Identify candidate systems that satisfy the prerequisite or mandatory requirements.
5. Evaluate each candidate system against the desired features, and make final selection.

It is almost impossible to find an off-the-shelf DBMS which does everything and is exactly tailored for an intended application. This is especially true with a complex application such as a hospital information system.

Generally speaking, the following capabilities which are closely associated with DBMS design decisions will be required for the successful implementation of a DBMS-based hospital information system.

A. Non-procedural user language for on-line query and update. The ability for the user to directly interrogate or access a specified file without programmer intervention, and even without program-

ming knowledge is crucial to the acceptance of the system by the medical personnel.

B. Procedural language interface. Allowing the user to interface with a higher level programming language gives him a powerful tool for tailoring his applications while taking advantage of the data management functions provided by the system.

C. Transaction-oriented processing. The queries of the database can be either unanticipated or pre-determined. Transactions are normally pre-defined functions such as daily reports with known set of parameters (e.g., bed census). The ability to store procedures which can later be invoked from a terminal accommodates transaction-oriented processing.

D. Data structure. Most files encountered in the medical field are structurally quite simple. They are patient-oriented files, i.e., they are set up by patient name. Most fields are multi-valued. Data structures beyond the simple parent-child relationships (e.g., associating each admission date with the corresponding treatment) are usually not required.

An important characteristic of the medical field is that patient information is required for diverse applications. In order to eliminate redundancy and to consolidate data, the DBMS should provide the capacity for coupling files.

E. Access method. Most hospital applications demand fast retrieval and constant changes to the database. The random method of accessing records is well suited for individual patient record-oriented applications. However, unanticipated queries would have to pay the penalty of a sequential scan if the needed attributes

are not inverted.

F. The capacity to support large databases. The database approach to the hospital information system is based on the concept of a continuous, lifetime, integrated patient medical record. As such, the volume of data that needs to be maintained and the number of users interacting with the system are quite significant. In addition, the DBMS should easily accommodate growth. Not all DBMS's are built with large databases in mind. Projected performance criteria can truly be evaluated only when the database management system has been subject to benchmark tests.

REFERENCES AND BIBLIOGRAPHY

1. CODASYL Systems Committee, "A Survey of Generalized Data Management Systems", ACM, New York, 1969.

2. CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems", ACM, New York, 1971.

3. CODASYL Systems Committee, "Selection and Acquisition of Data Base Management Systems", ACM, New York, 1976.

4. Anderson, R. K., "Index of Data Management Software Packages", National Bureau of Standards, Report 10-932, 1972.

5. Fong, E., et al., "Six Data Base Management Systems: Feature Analysis and User Experiences", National Bureau of Standards, Technical Note 887, 1975.

6. Datapro Research Corporation, "A Buyer's Guide to Data Base Management Systems", 1974.

7. Palmer, I. R., "Data Base Systems: A Practical Reference", Q.E.D. Information Sciences, Wellesley, Massachusetts, 1975.

8. Cohen, L. J., "Data Base Management Systems: A Critical and Comparative Analysis", Q.E.D. Information Sciences, Wellesley, Massachusetts, Fourth Edition, 1975.

9. TRW Systems Group, "GIM-II User Reference Manual", July 1975.

10. TRW Systems Group, "ASSIST DMS Elementary User's Guide With Sample Application", June 1976.

11. TRW Systems Group, "GIM-II Procedural Oriented Language", August 1976.

12. AUERBACH Associates, "Minicomputer DMS Study", AUER-2324-FR, October 1975.

13. Weitzman, C., "Minicomputer Systems", Prentice-Hall, 1974.

14. Zack, B. A., "Selecting A Minicomputer: A Framework and Application to the Sloan School of Management", M.S. Thesis, Sloan School of Management, M.I.T., June 1977.

15. Digital Equipment Corporation, "PDP 11 Software Handbook", 1975.

16. Digital Equipment Corporation, "RSX-11D User's Guide", 1975.

17. Katzan, H., Information Management System, "Computer Data Management and Data Base Technology", Van Nostrand Reinhold, 1975.

18. Date, C. J., "An Introduction to Database Systems", Addison-Wesley, 1975.

19. Stonebraker, M. R., et al., "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976.

20. Yasaki, E. K., "The Mini: A Growing Alternative", Datamation, May 1976.

21. Stein, P., "The Adolescent Minicomputer", Computer Decisions, November 1976.

22. Theis, D. J., "The Midicomputer", Datamation, February 1977.

23. Boylan, D. T., "DBMS for Minis", Computer Decisions, January 1976.

24. Boylan, D. T., "Minicomputer DBMS: less than meets the eye", Computer Decisions, January 1977.

25. Floam, G., "Putting a Database on a Mini", Datamation, June 1976.

26. Foster, J. D., "Distributive Processing for Banking", Datamation, July 1976.

27. Lynch, A., "Distributed Processing Solves Mainframe Problems", Data Communications, November/December 1976.

28. Ball, M. J., "Computers: Prescription for Hospital Ills", Datamation, September 1975.

29. Jackson, G. G., "Information Handling Costs in Hospitals", Datamation, May 1969.

30. Collen, M. F. (editor), "Hospital Computer Systems", John Wiley & Sons, 1974.

31. Garrett, R. D., "Hospitals - a systems approach", Auerbach, 1973.

32. Bekey, G. A., Schwartz, M. D. (editors), "Hospital Information Systems", Dekker, 1972.

33. Mills, C. A., "Architecture of a Community Medical History Data Base", M.S. Thesis, University of Illinois, 1973.

34. Ludwig, H. R., "Computer Applications and Techniques in Clinical Medicine", John Wiley & Sons, 1974.

35. Wood, C. T., Lamontagne, A., "Computer Assists Advanced Bed Booking", Hospitals, March 1, 1969.

36. Jessiman, A., Erat, K., "Automated Appointment System to Facilitate Medical Care Management", Medical Care 8, 1970.

37. Shonick, W., Klein, B. W., "An Approach to Reducing the Adverse Effects of Broken Appointments in Primary Care Systems", Medical Care 5, 1977.