COMPARATIVE STUDY OF A HIERARCHICAL DATABASE MANAGEMENT SYSTEM WITH A RELATIONAL-LIKE DATABASE MANAGEMENT SYSTEM FOR A SCHEDULING PROBLEM

By

VINIT VERMA

Bachelor of Science

St. Stephen's College

University of Delhi

Delhi, India

1984

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
July, 1987

COMPARATIVE STUDY OF A HIERARCHICAL DATABASE MANAGEMENT SYSTEM

WITH A RELATIONAL-LIKE DATABASE MANAGEMENT SYSTEM

FOR A SCHEDULING PROBLEM

Thesis Approved:

Donald D Fisher

Thesis Adviser

[illegible]

[illegible]

Norman N. Durham

Dean of Graduate College

PREFACE

A comparative study of a hierarchical database management system, IMS (Information Management System), and a relational-like database management system, Model 204 was performed. The comparison of the two database management systems was limited to data definition, data manipulation, data independence, data protection, and storage organizations. A classroom reservation system was the scheduling problem which provided the comparative tool. An on-line classroom reservation system was developed using Model 204. A brief overview of of the capabilities of the two systems is provided under each topic, before a comparison is performed. Initially the underlying data models have been discussed to set the ground work for the following discussion.

I wish to thank my adviser, Dr. D. D. Fisher for his invaluable guidance, and help. I am also grateful to the other committee members, Dr. M. G. Kletke, and Dr. G. E. Hedrick for their advisement. The help of the Assistant Registrar, Mr. Glen Jones is appreciated for providing the requirements for the reservation system, and giving access to the class schedule data. Input from Ms. Beth Thorton of Sectioning was invaluable. Special thanks go to Mr. A.F. Curtis of Administrative Systems Development for providing the needed data, and information on IMS. I am indebted to Mr. Michael Barton, and Ms. Betty McDaniel for the needed encouragement. I appreciate the patience and support of my fiancee Vidhi.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Statement of the Problem

A comparative study of a hierarchical database management system, IMS (Information Management System), and a relational-like database management system, Model 204 is presented. The comparison of the two database management systems is limited to the following subtopics -

- data definition;
- data manipulation;
- data independence;
- data protection;
- storage organizations.

The objective of the thesis is to expose the capabilities of the two database management systems under the mentioned subtopics. It provides a comprehensive overview of the two systems at the external/conceptual level of the ANSI/SPARC (American National Standards Institute / Systems Planning And Requirements Committee) model. The ANSI/SPARC model is a three tier model with external, conceptual, and internal levels. The study is intended to help investigators of these two products. The comparison of the two approaches to database systems was made by applying both methods to the same problem, namely, the implementation of a classroom reservation system which is a subproblem of a general scheduling problem. A general scheduling problem is concerned with the assignment of resources and activities subject to constraints, Almond(1). The classroom reservation system has been implemented in IMS by staff members of administrative systems development at OSU. Inorder to be able to compare the two database management approaches, it was necessary to develop a similar system in Model 204. The data for the new system was abstracted from the existing IMS database. Once a common application problem was available, a study was performed to explore the above mentioned capabilities of the two systems. The study initially provides an overview of the data models involved. A description of scheduling systems, and the on-line system developed in Model 204 follows. The subsequent chapters discuss the data definition, data manipulation, data independence, data protection, and storage organization capabilities of the two systems.

The data definition comparison involves the data definition constructs available under Model 204 and IMS. The underlying data models cause the different data definition styles in the two systems. Under data manipulation a comparison is made of the data manipulation capabilities of DL/I(Data Language/One) under IMS, and user language under Model 204. Again the constructs in both these languages reflect on the data models they imitate. Data independence can be further segmented into physical data independence and logical data independence. The physical data independence implies program immunity to changes in the storage structure, while logical data independence implies program immunity to changes in the data model definition. Program immunity to change implies that if the logical/physical organization of a database is changed, then the application programs accessing this database need not be changed. This degree of data independence in the two classes of database management systems is discussed. The data protection study involves data integrity and data security in the two systems. The underlying storage structures and access methods in IMS and Model 204 are discussed under the storage organization chapter.

Literature Review

The hierarchic, inverted list, and network data models resulted after abstractions from implemented systems. The relational model was developed on set theory principles. Fry, Sibley (18), provide a good reference for the evolution of database management systems. A discussion on relational and inverted list data models is provided in Date (17). Codd (8), and Codd (9) provide a practical foundation for the relational database theory. Codd proposed the relational data model and provided the basic principles for the model. Chamberlin (6), and Kim (24) give a detailed summary of relational databases, and several implemented systems have been discussed by the two authors. Mcfadden, Hoffer, Jeffrey (27), and Wiederhold (35) are a good sources of definitional terms, and keywords. The inverted list data model has envolved after abstraction from systems which utilize the inverted file access method. Date (17) provides a description of the basic inverted list data model operations. Cardenas (5) provides a formal analysis of inverted structures incorporating implementation oriented considerations. In large highly inverted databases, the inverted directory or index becomes a large database itself. Thus such a directory should not be stored as a sequential file. The advantage of the multi-level inversion approach over the single inversion level and sequential structure was shown to increase exponentially with the degree of inversion and/or the number of distinct key values of inverted access keys. The search time through the directory was significant for large databases, and a function of complexity of queries, data distributions, and timing and blocking specifications of the storage devices. The hierarchic data model was also abstracted from an implemented system, namely IMS. McGee (29) gives a detailed system level discussion on the capabilities of IMS. Tsichritzis, Lochovsky (33) have given a general overview of hierarchic systems, mainly concentrating on IMS.

Literature was reviewed to come up with criteria to evaluate database management systems. Most of the reviewed literature focussed on the database architectural aspects, at the internal level of the ANSI/SPARC data model. There was no evidence of comparison of implemented systems. Most references provide an in depth discussion of specific systems. Cardenas (4) discusses evaluation of file organization models. Performance evaluation techniques like modelling, measuring system response time, and overhead costs have been discussed by Christodoulakis (7). McGee (28) describes criteria for data model evaluation at the three levels of the ANSI/SPARC model (i.e external, conceptual, and internal levels). Siler (31), and Stonebraker, Woodfill, Ranstrom, Murphy, Meyer, Allman (32) provide a good description of performance evaluation criteria. Michaels, Mittman, Carlson (30) have compared a relational system, with a network system, at a conceptual level. Data definition, data manipulation, and data independence were the primarv criteria used to compare the relational and network systems.

As a class room reservation system was used as a comparative tool, a fair amount of review work was done in this area. Algorithms to implement scheduling systems were reviewed. Lions (25) discusses the matrix reduction method for generation of school timetables. Implemented systems, like the Ontario school scheduling program, by Lions (26), and the class scheduling system at Oklahoma State University were studied. Gosselin, Truchon (19) discuss a linear programming problem of allocating classrooms to reservation requests. Csima, Gotlieb (16), Almond (1), Barraclough (2), Brittan, Farley (3) are the other primary references in the class scheduling area.

Model 204 systems reference manuals were used to obtain detailed information on the system. CCA File Manager's Guide (11) provides a detailed description of the storage structures, and the data definition capabilities of Model 204. CCA User language Manual (15) describes the data manipulation capabilities of the system. The general features of Model 204 are described in CCA Command Reference Manual (10), and CCA System Manager's Guide (12). Data security is detailed in CCA File Manager's Guide (11), and CCA Terminal User's Guide (14). IBM General information manual (21) provides an overview of IMS. The data manipulation capabilities of IMS are explained in the IBM Application Programming Reference Manual (20). The data definition, and data security features for IMS are discussed in IBM System Design Guide (22), and IBM System Programming Reference Manual (23). McGee (29) provides a good description of the access methods, and storage structures available in IMS.

Basic Definitions

Access Path: to a file is defined as the strategy used to address individual records in a file (e.g indexed, sequential);

Attribute: a column in a relational table;

Data Definition: description of data objects;

Data independence: immunity of application programs to a change in storage structure, or access strategy;

Data integrity: accuracy or correctness of data in a database;

Data Security: restriction of access to sensitive data;

DBMS: software that provides the interface between the database, and the end user;

Data Sub-Language: is a combination of two subordinate languages: a data definition language (DDL), which provides for the definition of database objects, and a data manipulation language (DML), which supports the processing of those objects;

Entity integrity: no attribute participating in the primary key of a base relation is allowed to accept null values;

Foreign key: a key which matches the primary key of another relation, or contains a null value;

Fourth Generation Language: an interface to an advanced application generator (rapid application development tool);

Inverted list: a set of indexes in which a secondary key leads to a set of primary key references;

Logical data independence: program immunity to changes in the data model definition;

Normalization theory: mathematical theory, which governs the conversion of relations into normal forms, to maintain data integrity;

Physical data independence: program immunity to changes in the storage structure;

Primary key: a key which uniquely identifies a tuple in a relation;

Pseudo-foreign key: a foreign key which does not necessarily match the primary key of another relation;

Referential integrity: a foreign key can either match the primary key of another relation, or have a null value;

Relation: a mathematical term for a table;

Secondary key: identifies tuples in a relation. It can contain non-unique values;

Segment: a record in IMS;

Tuple: a row in a relational table.

Overview of Relational and Inverted List Databases

Relational Databases

The term "relational" applies to the conceptual and external levels of the ANSI/SPARC architecture model, it has nothing to do with internal data structures of the system. A relational database is one that is perceived by it's users as tables, and nothing but tables. It is built on the principles of the relational data model. Such a system should have the operations select, project, and join without requiring any pre-definition of physical access paths to support those operations. A relation is another term for a table. The rows in a relation are called tuples, and the columns are called attributes. A relational table is shown in Figure 1. The table has two tuples and three attributes.

COURSE.ID	ROOM	DAY
12296	MS 222	M
12273	CLB 110	F

Figure 1. A Relational Table

No duplicate tuples are allowed in a relational table. The attributes values for a specific tuple are atomic i.e. repeating groups are disallowed. The relational table is unordered. Thus a relational database is perceived by it's users as a collection of time varying relations. The data manipulation language for a relational system can be classified as a relational calculus language. A relational calculus language uses set theory operations join, union, intersection etc, to perform data manipulations. The select operation takes a horizontal subset of a relation, and the project takes a vertical subset. The join combines two tables according to some qualifying conditions. Relational languages are primarily set-at-a-time languages. Set processing provides loop avoidance in application programs, which greatly enhances programmer productivity. Most relational systems provide a fourth generation language for query purposes.

A tuple must have a primary key, and such a key cannot have a null value. Foreign key values in relational systems must match primary key values (or be null). The mathematical foundations of a relational DBMS give rise to normalization theorems which provide algorithms to detect and remove dependencies among entries in the system. A synopsis of the normal forms used to remove data dependencies follows, Date(17). A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. A relation is in first normal form if it contains atomic values only, no repeating groups are allowed. A relation is in second normal form if and only if it is in first normal form, and every non-key attribute is fully dependent on the primary key. If all non-key attributes of a relation are mutually independent and fully dependent on the primary key of the relation , then the relation is said to be in third normal form. A relation is in Boyce/Codd normal form if and only if every determinant is a candidate key. A relation is in fourth normal form if it is in Boyce/Codd normal form and all multi valued dependencies in the relation are in fact functional dependencies. Finally, a relation is in fifth normal form if and only if every join dependency in the relation is a consequence of the candidate keys of the relation. Logical relationships in such systems are represented via foreign keys, and physical pointer chains are not utilized. In a relational system positional addressing is replaced by associative addressing. Every datum can be identified uniquely via a relation name, primary key value , and attribute name. The relational model was developed to tackle various data dependencies existing in systems. Codd (8), Codd (9) provide a practical foundation for the relational database theory. Three primary dependencies are ordering dependency, indexing dependency, and access path dependency. The ordering dependency speaks of the order in which the data records are stored and accessed. Some non-relational systems are restricted by the ordering of data records closely associated with the hardware determined ordering of addresses. Indexing dependence is associated with the indexes being performance oriented components of the data representation. Indexes are performance oriented components of data because, they exist to increase the performance of data access, and are not originally a part of the data. The dependency comes into the application programs which need to deal with dynamic indexes. Finally access dependence is the tree-like pre-defined access paths imposed on application programs in a hierarchical environment. All three of these dependencies have been removed explicitly in the relational data model. The requirements for a system to be called truly relational are extremely stringent, and such a system does not exist today. Kim (24), Date (17) provide a good overview of relational database management systems.

Inverted List Databases

The data model for inverted list systems can be regarded as an abstraction of the underlying inverted list storage structure and associated operators. The inverted list model, like the hierarchical model, explains the database concepts at the internal level of the ANSI/SPARC architecture model. The inverted list model is an abstraction of the indexed file organization. At the external levels, inverted list systems can have relational front-ends. Thus an inverted list database is a system comparable to the low level component of a relational database in which the users operate directly at the record-at-a-time level, instead of the set level. Model 204 has an inverted list file organization but provides set level operations. An inverted list system is similar to a relational database in that stored tables and indexes on those tables are visible to the user. It has a collection of tables each with it's own set of tuples and attributes. The tuples in such a system are ordered. The system may have a large number of search keys which may be composite. The operators in such a system are primarily record-at-a-time. The data manipulation operators fall under the two categories:

1) operator to obtain the address of a record;
2) operator to manipulate a record at an established address.

The operators in the first category can be further classified as:

a) direct search operators;
b) relative search operators.

The operators under category one are also termed as search/locate/find operators. Relative search operators locate a record relative to some established address. Established addresses are termed as "database address areas". The retrieval operations obtain data via access paths. These access paths for search operators need to be maintained for terms like "first" and "next" to be meaningful in search operators. This is similar to the notion of current database position in hierarchical systems. These access paths are maintained by the total database physical ordering and inverted list indexes. Let A be a database address area, T be a table for a database, and K the search key. Some inverted list data retrieval operators are:

- locate first: locate the first record in table T and return it's address in A;

- locate next: locate the first record following the one identified by A, and return it's address in A;

- locate first with search key equal: find the first record in table T with a search key of K, and return it's address in A;

- locate next with search key equal: locate the next record in table T which follows the record at address A, and whose search key is identical to K, and return it's address in A;
- locate first with search key greater: is the same operator as the previous operator, except the search key of the located record should be greater than the key of the record identified by A;
- retrieve: retrieve the record identified by A;
- insert, delete and update operators.

Overview of Hierarchical Databases

Hierarchical databases were not originally constructed from an abstract data model. The hierarchic model resulted by the process of abstraction from implemented systems (e.g. IMS), Tsichritzis, Lochovsky (33). Hierarchic systems are an abstraction of the parent/child data organization. A hierarchical database is formally defined as a forest of trees called database trees, whose record occurrences appear as nodes. The database trees form an ordered set consisting of multiple occurrences of a single type of a tree. The type of a tree is analogous to a distinct record type. A particular tree consists of a root, and an ordered set of zero or more dependent subtrees or record types. The subtrees are defined recursively by the same definition. Thus an entire database tree is basically a hierarchic arrangement of various types of records. The records can be subdivided into fields. Consider the hierarchic database shown in Figure 2.

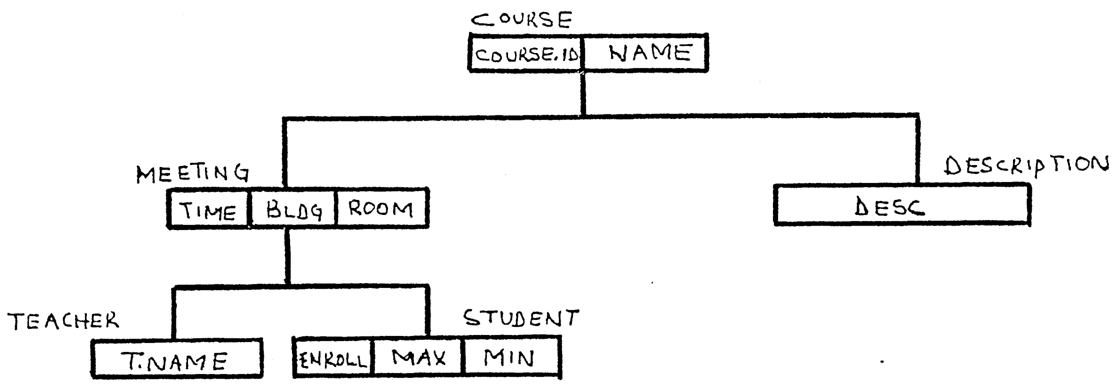


Figure 2. Hierarchic course Database

In Figure 2 course database has 'course' as the root record type, with two dependent record types 'meeting' and 'description'. The 'meeting' record type has two dependent record types 'teacher' and 'student'. The entire database has a definite ordering (top-to-bottom, left-to-right). The 'meeting' subtree precedes the 'description' subtree, and the 'teacher' subtree precedes the 'student' subtree. The entire 'meeting' subtree is traversed before the 'description' subtree. Under one 'course' root there can be multiple 'meeting' and 'description' subtrees, and under 'meeting' there can exist multiple 'teacher' and 'student' subtrees. Multiple 'teacher' segments under a 'meeting' segment signifies a hypothetical case where any one of the teachers could teach the course. The entire course database primarily consists of a forest of these 'course' database trees. Course, meeting, description, teacher, and student are the five distinct record types present in the database. The course forms the root record type, and meeting and description form the dependent record types. Course is also termed as a parent record type for meeting and description child record types. The entire database tree is threaded by explicit pointers between parents and children. Figure 3 explains multiple occurrences of a record type. The figure shows a single database tree in the course database.

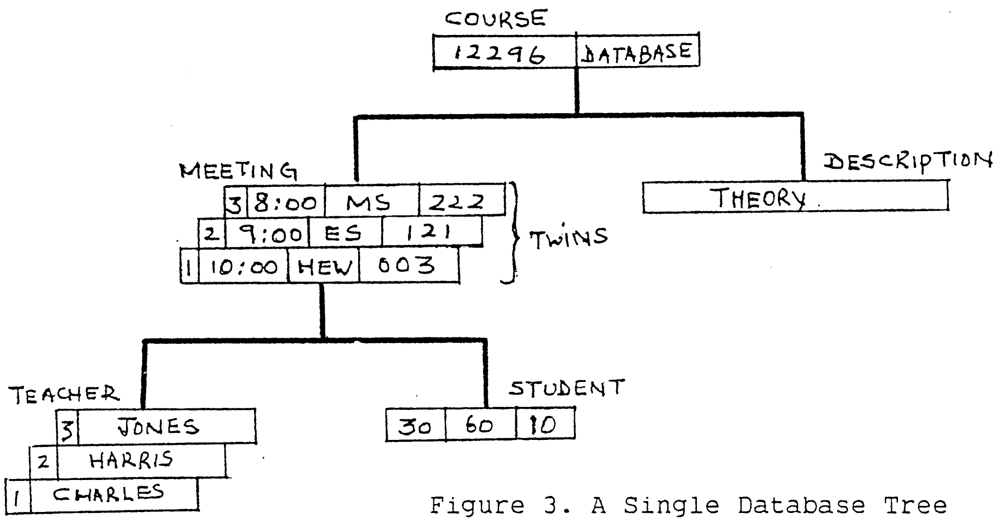


Figure 3. A Single Database Tree

The database has a single occurrence of the root record type 'course'. The course occurrence has an ordered subtree consisting of three occurrences of the meeting record type, and one occurrence of the description record type. The first meeting occurrence in turn has an ordered set of three teacher occurrences, and one student occurrence. All occurrences of a specific record type which share a common parent are termed as twins. In the previous diagram the meeting records are twins. As was mentioned earlier the dependent record types are always ordered. The hierarchic sequence for the given database tree is shown in Figure 4.

```
course:         12296
meeting:        1
teacher:        charles
teacher:        harris
teacher:        jones
student:        30,60,10
meeting:        2
.
.
meeting:        3
description:    theory
```

Figure 4. Hierarchic Sequence

The top-to-bottom and left-to-right ordering is evident from the previous sequence. The database is a forest of these course trees. If these course trees are depicted to be rooted at the system, then the entire database can be visualized as a tree. The hierarchic sequence applies to this entire database i.e. the database trees have a definite ordering left-to-right. The physical database records are thought of as being stored in this logical sequence. Referential integrity is maintained in the hierarchic model, because a child cannot exist without a parent. If a parent is deleted, all children are deleted automatically. A child type can be inserted only if the corresponding parent exists. A child can have only one parent. The data manipulation language provides operations to manipulate trees. Some operators on trees are:

- operator to locate a specific database tree in a forest;
- operator to process a forest of database trees in a hierarchic sequence;
- operator to step through the subtrees in a specific database

tree;
- insert operator;
- delete operator.

The operations on trees process a node at a time. Thus a hierarchic system is record-at-a-time. McFadden, Hoffer, Jeffrey (27) provide a good review of hierarchic databases.

CHAPTER II

CLASSROOM RESERVATION SYSTEM

Theoretical Background

A scheduling system is one that distributes a resource, into a finite workspace, in the most efficient manner. A good example of a scheduling system would be one that schedules teachers into a weekly timetable. A classroom reservation system is a scheduling system in which classrooms are allotted to room reservation requests. The class scheduling systems reviewed primarily involved scheduling teachers and courses into a weekly timetable. A set of course requirements was laid out and the day was broken into a fixed number of periods. Each teacher had a fixed teaching load. An algorithm was utilized to come up with a timetable which schedules the teachers and courses during the week. The algorithms to do the scheduling varied according to the timetable restrictions laid down at various schools e.g. (double periods, specific lunch hours). Gosselin, Truchon(19), developed an algorithm which performs a heuristic room allocation to courses. The algorithm presumes a categorized set of requests and a set of rooms which can satisfy each of the requested sets. The algorithm involves a first come first serve allocation and a two dimensional array is the primary data structure used. It results in a schedule where rooms get alloted to specific requests. An exhaustive search is performed and a conflict free schedule is found if one exists.

An explanation of the algorithm follows.

Suppose there are 5 hours in the day, 3 rooms in the category to be considered and 6 requests to be satisfied by rooms of this category. The columns in the matrix below represent the hours and the rows represent the rooms. The 6 courses are to be given at hours (1,2), (1,2), (4,5), (2,3), (4,5), (3,4). The 3 x 5 matrix is initialized to zeros. The algorithm tries to satisfy each request starting from room 1. Once the needed hours are found free in a certain room (row), the request number is placed in those slots, otherwise the next room down is tried. This goes on until we run into a request which remains unsatisfied after checking the last row. In such a case the request prior to this one is uprooted from the matrix and tried in the next room (row) down. The freed slot is replaced with zeros. Now the unsatisfied request is picked up again and the whole process is repeated until a conflict free schedule results. If finally the first request is uprooted and no schedule results after trying all combinations then the algorithm prints out the error message. In the above problem the schedule matrix looks as follows after the fifth request has been satisfied.

$$\begin{array}{llccccc} \text{hour} & & 1 & 2 & 3 & 4 & 5 \\ \hline \text{room} & 1 & 1 & 1 & 0 & 3 & 3 \\ & 2 & 2 & 2 & 0 & 5 & 5 \\ & 3 & 0 & 4 & 4 & 0 & 0 \end{array}$$

The sixth request cannot be satisfied in any of the rooms, so request five is uprooted and satisfied in a room further down. This results in the sixth request being satisfied by room 2. The final schedule matrix is shown below.

hour		1	2	3	4	5
room	1	1	1	0	3	3
	2	2	2	6	6	0
	3	0	4	4	5	5

The algorithm can be modified to catch the delinquent request. This algorithm is being used in the timetabling system set up at University Laval in Canada, Gosselin, Truchon(19). In some systems faculty was matched to courses. The instructors laid down a set of preferences in timings and courses, and a program came up with an appropriate schedule. Other systems incorporated the student enrollment data into the timetabling system. An interesting problem was scheduling students to classes in a compiled timetable. Each student selects a set of courses as his proposed schedule. The schedule was fed into the scheduling program which verified if it was compatible with the timetable. If a conflict results a workable schedule was computed by the program. Each student schedule was considered on it's own merit without reference to any other student's schedule. There was an excessive amount of processing time spent in scheduling each student. If the student body was large the above system proved extremely expensive. With a large student body the time to schedule each student was of major importance. The method used aimed at scheduling each student as quickly as possible by reducing the number of schedules tried, before a satisfactory one was found. This was achieved by categorizing the classes so that the most difficult one to fit into occurred first in the student schedule. For example if only one class was available then the student can either be fitted into it or not, so the rejection occured very early. The program addressed another important factor, the students were evenly distributed among different class sections, so fewer classes reach their maximum limit, and thus the students enrolling late could be easily enrolled.

Barraclough(2) has discussed an interesting technique in high school timetabling. The primary resources being addressed was classes to be taught and masters. The problem was to assign masters to certain classes. An intelligent use of bit strings and logical operations performed the needed operation. A master could meet a class for a single period or two successive single periods i.e. a double period. The school time table had pre-scheduled break timings, and a double period could not span across a break. The list of requirements was structured as follows: (Mi,Cj,s,d). Master i must teach class j for s single periods and d double periods. In order to assign these single periods it was necessary to know those periods when master i and class j were both free, and then s of them had to be chosen. The binary pattern formed by the logical "and" of the available digits for master i and available digits for class j gave the common periods available. If the available digits for master i and class j were:

```
PMi (master i) = 1 1 0 1 0 0 1

PCj (class j ) = 0 1 0 0 1 0 1
                 -------------
PMi & PCj      = 0 1 0 0 0 0 1
```

The logical "and" showed that master i and class j were both available in the second and last periods of the day. (Mi,Cj,0,d) was the general format of a double period assignment. The request list demands master i teach class j for d double periods. Pairs of digits that do not cross a break must be found by referring to the pattern showing when breaks can occur. The pattern of digits denoted by de showed where double periods could end. The digits showing where double periods can start can be found by the following logical operations. (Pmi & PCj & de) gives a 1 where the second period is free. Let (PMi & PCj & de)' denote the same pattern of binary digits but left shifted by one position. The 1s now occur at positions which could be the start of double periods. Then (PMi & PCj & de)' & (Pmi & PCj) gives a 1 where a double period may start. If the binary patterns for one day were:

```
        PMi (master i)      = 1 1 1 0 1 1 1
        PCj (class j )      = 0 1 1 0 1 1 0
        de ( double end )   = 0 1 0 1 0 1 1
        PMi & PCj           = 0 1 1 0 1 1 0
        PMi & PCj & de      = 0 1 0 0 0 1 0
        (PMi & PCj & de)'   = 1 0 0 0 1 0 0
(PMi & PCj & de)' & (PMi & PCj)= 0 0 0 0 1 0 0
```

Showing that a double period may start in the fifth period, but rejecting the two consecutive periods in the second and third periods since they cross breaks.

Csima,Gotlieb(16) proposed a method of constructing school timetables based on an iteration of boolean matrices. They utilized a three dimensional array each node of which represented a specific teacher, class and time. In the boolean array a value of zero implied the impossibility of the class and teacher meeting at that time. Conversely a value of one implied the possibility that the teacher and the class may meet at that time. The array was initialized to ones indicating that a teacher could meet any class at any time. But as the computation proceeded the ones were changed to zeros according to set rules and entries in the requirements matrix. At the end of the computation for a certain time slot it was possible for each teacher to meet only one class and each class to meet only one teacher. Each teacher could meet each class a number of times preset for that teacher and class. The resulting three dimensional array was the required timetable. This method required examination of each plane section of the three dimensional array at regular intervals. A plane section is a two dimensional array of zeros and ones belonging to a teacher, class or time. An examination of such an array has two phases:

1) The existence of at least one possible schedule is determined (feasibility test).

2) Any non-zero element which does not belong to some possible schedule is changed to zero (matrix reduction).

The algorithm suggested by Csima,Gotlieb(16) to process step 2 was of the order 2^n. Lions(25) came up with a more efficient algorithm to effect the matrix reduction in order n^2 time. The Ontario school scheduling program was successfully implemented using the above algorithm, Lions(26). The program handled a variety of special requirements in the timetabling process, as opposed to the structured requirements tested by Csima, Gotlieb(16).

Classroom Reservation System at OSU

The current classroom reservation system at Oklahoma State University is set up on IMS. The system was studied to obtain an understanding of classroom reservations at OSU. The system provides an on-line browse facility keying off the course identification number. Setting up the schedule for a semester and making reservations are still a batch oriented process.

A manual is created with thirty minute time slots for six days in the week and for each available classroom on campus. This in effect is a manual database maintained by the registrar's office. The registrar has created a turnaround document which is sent to each department on campus. A copy of the previous semester's schedule is sent with the turnaround document. The departments come up with lists of courses they will offer and expected room and reservation timings. Once the department's request reaches the registrar's office, entries are made in the manual database for each room requested. Any conflicts are resolved at this stage by re-adjustment requests to the departments concerned. The turnaround documents are forwarded to the data entry personnel who key in the data into the IMS database. A verification run is made and any conflicts that may have been overlooked are resolved at this stage. Once the data is entered into the IMS database the on-line system available is able to browse it. Any reservation request made after the initial schedule is set up is keyed into the on-line system. The conflict reports are generated in a batch process at the end of the day. The personnel then reconsult their manual database and resolve the conflicts with the departments concerned. On-line deletions do not cause any problems as they are not involved in any conflicts. Student enrollment data is also incorporated into this database. During pre-enrollment and enrollment each student's enrollment request is entered into the system, so the database remains current with the number of students enrolled for a specific course. Each course has a pre-specified lower and upper limit for the number of students and once these are crossed a indicator is set. The departments specify these limits for each course. Once a semester's schedule has been finalized by the above method it is ready to be printed.

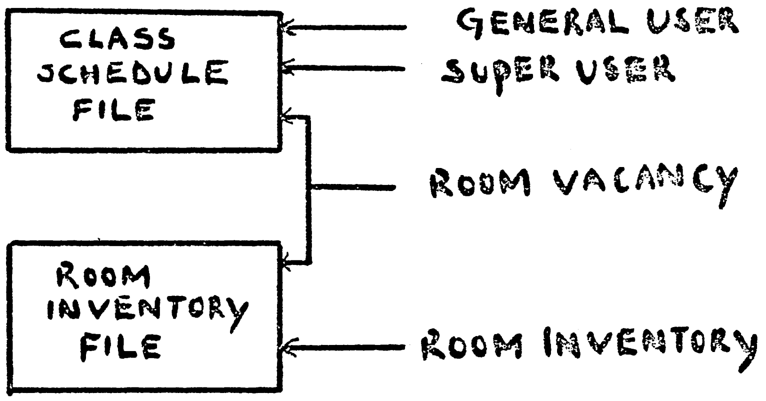
The amount of manual effort going into the current system is evident from the process described. The existing system is not truly on-line because the reservation requests are not instantly verified by the computer but instead are performed in a batch process at the end of the day. Both the departments and the registrar's office personnel spend a lot of time setting up the initial schedule for a semester and resolving conflicts. The whole process is primarily manual with the computer being utilized for data collection. The departments propose a schedule by inertia from the previous semester's schedule. The departments on campus have an understanding from prior years as to which rooms are available for assignments. The laboratories have been pre-assigned to each department. A room request in mid semester requires querying of the manual database. The browse facility is minimal just keying off the course identification number and the department name and course number. The departments on campus do not have any on-line system to check up on room reservations. The conflict detection system is redundant with both the personnel and the computer performing the exercise. Two databases are being maintained namely one in IMS and the room reservation manual. The turnaround time for a room reservation request is very slow.

The requirements for the on-line classroom reservation system developed on Model 204, were obtained from the registrar's office at OSU. The primary requirement was to get a truly on-line reservation system. The system was expected to provide multiple views of the data. It was to provide a list of courses meeting in a room, or courses being taught by an instructor or department. The manual database needed to be eliminated, therefore the conflicts had to be resolved by the reservation system. A system was needed which would allow the departments on campus to query the reservation system. The system was to handle two types of requests namely, room reservations for for a full semester, and temporary room reservations. The ad hoc room reservations needed to be dynamic, in that a classroom was freed once the expiration date for the reservation was reached. A room inventory database was needed which would house information on all reserveable rooms on campus. This room inventory database would allow for complex reservation requests. An example of such a request is: " provide a list of rooms which are case studys, have at least 40 seats, are free on

Mondays from 9:30 to 10:30 on specific dates". The system was expected to be user friendly, and a user's manual was needed for the reservation system. For security reasons a complete tape backup system was also needed.

Classroom Reservation System using Model 204

The first phase of the development of the classroom reservation system on Model 204 involved studying the existing system, and obtaining requirements for the new system. The second phase involved accessing the classroom data. A copy of the IMS database was made on tape. Segments were sequentially dumped to tape, to make a course meeting dataset and a course description dataset. IBMs IEHMOVE utility was utilized to copy these datasets onto disk. The schematic below outlines the subsystems of the classroom reservation system.



The data for the classroom reservation system was stored in a class schedule file, and a room inventory file. The class schedule file consists of of all course meeting data, and the room inventory file contains specifications for all reserveable rooms on campus. The class reservation system is sub-divided into four subsystems namely :

- general user;
- super user;
- room inventory;
- room vacancy.

The general user's option is intended for departments on campus. It provides extensive data retrieval functions on the class schedule database. No updates are provided through this window. A user can retrieve information by keying off the following fields :

a. course id number;

b. department name and course number;

c. department name;

d. building and room number;

e. instructor's name;

This option also provides a list of all courses offered by a department, sorted by day of the week, and time of the day. The program function key support, and the advanced cursor sensing capabilities of Model 204 provide a user friendly interface.

The super user's option is intended for the registrar's office. This option provides room reservation facilities. All the data retrieval operations mentioned above are available under this option also. Two types of reservation request can be satisfied, namely :

- room reservation for the whole semester;
- ad hoc room reservation request for limited amount of

time.

A user is expected to query the room inventory database via the room vacancy subsystem before making a room reservation for the whole semester. The day and time conflicts for a room are resolved in the room vacancy subsystem. The reservation system maintains unique course id numbers, and any attempt at duplication is rejected.

An ad hoc room reservation request requires the user to input a commencement date and an expiration date. The commencement date allows for a proactive reservation. The expiration date allows the system to free the room, once the expiration date is reached. This keeps the reservation system dynamic in that rooms are freed up again on expiration. The system checks for time, day and date conflicts for a room before allowing a successful reservation. The deletion process keys off the course id number.

The room inventory option provides information on all reserveable rooms on campus. Some rooms have been allocated to specific departments, and these do not appear in the available pool mentioned above. The room inventory database keeps information on the room capacity and room type. Case study, auditorium, fixed, table and chairs, arm, bolted arm, and lab are the valid room types. Information for any specific room, or for all rooms in a building can be retrieved. New rooms can be added to the database, and old ones may be deleted.

The room vacancy option is the main component of the reservation system. This option queries both the class schedule and the room inventory databases. A user inputs the start date, end date, days the course will meet, begin time, course end time, number of seats needed, and the type of room needed. The room inventory database is queried to come up with a list of rooms which match the needed type and capacity. Then the time, day and date conflicts for each of the rooms is resolved by querying the class schedule database. Finally a list of rooms is displayed which satisfies the request.

The BATCH204 utility of the Model 204 file manager was executed to set up the database files. The batch routine is included in Appendix B. The attributes of the class schedule file are:

file name	: M204.ACT10820.DATA
volume	: OSU201
device type	: 3380
organization	: PS
record format	: U
record length	: 6184
block size	: 6184
first extents	: 90
secondary extents	: 20
creation date	: 86/08/03

The data definition phase involved defining the fields in the Model 204 files. Two tables were set up, namely the class schedule table, and the room inventory table. Both were kept in first normal form (i.e all underlying domains contain atomic values). A description of the table attributes follows.

CLASS SCHEDULE TABLE

ATTRIBUTES

COURSE.ID (key) - course identification number

DEPARTMENT (key)	- department name
COURSE.NO (key)	- four digit course number
COURSE.TYPE	- course type (Th,LAB,IS,DS)
COURSE.DESCRP	- course title
INSTRUCTOR (key)	- instructor's surname
BUILDING (key)	- building code
ROOM (key)	- room number
SECTION	- section number
COURSE.DAYS	- meeting days (MTWHFS)
COURSE.BEGIN	- start time
COURSE.END	- end time
FREEFORM	- flag to indicate TBA reservation
MIN	- minimum # of hours for course number ending in 0
MAX	- maximum # of hours for course number ending in 0
START (key)	- commencement date
EXPIRE (key)	- expiry date
COMMENT	- comments on the course
NUMBER	- index into comments

ROOM INVENTORY TABLE

ATTRIBUTES

BUILDING (key)	- building code
CAPACITY (key)	- total number of seats
ROOM (key)	- room number
TYPE	- 1 case study
	2 auditorium
	3 fixed
	4 table & chairs
	5 arm
	6 bolted arm
	7 lab

The key specification against the field name signifies that a hash index is created for that attribute.

The data extraction, and the file initialization was done via the imaging facility of user language. This facility allows an application program to access a non-Model 204 file (i.e the IMS datasets). The application programs, and the hierarchy chart for this abstraction process are included in Appendix B.

In the next phase the application programs were developed to constitute the four subsystems. The algorithms used in these subsystems make use of inverted lists, and set theory operations. The order of processing for these algorithms is polynomial. The class schedule table is created with eight hash indexes. A query for all courses for the computer science department is executed in the following manner. The department code COMSC is hashed, and the secondary index entry is located. This entry has a list of internal record numbers for all tuples which belong to the queried department. This located set is then displayed to the user. The hash function has a processing time of O(1), and if 'n' tuples exist in the located set, the query takes O(n). More complex queries are executed using intersections, and joins on located sets. A typical query for the room vacancy subsystem would be: " generate a list of all rooms which are free on Monday and Wednesday, from 7:30 to 8:30, with a capacity of at least 30 seats, and are case studys". The room inventory table is queried, and two sets are located. The first set contains all rooms with 30 or more seats, and the second set contains all case study rooms. An intersection on these two sets provides a set of rooms which are case studys, and have 30 or more seats. For each of the rooms in this reduced set, the class schedule table is queried for the vacancy at the needed time, and day. A join is performed between the room inventory table, and the class schedule table with day and time restrictions. Suppose MS 222 is a case study, and has 30 seats, thus it is a member of the reduced set due to the intersection. A set is located which consists of all courses meeting in MS 222. The timings, and meeting days for each of these courses is checked. If a conflict results MS 222 is deleted from the set, and the next room is verified in a similar manner. The operations just described are performed in the join. As a result of the join a set of rooms is obtained which satisfy all the request criteria. The previous example showed how the set level algorithm executes a query. Suppose there are 'n' rooms with at least 30 seats, and 'm' rooms which are case studys. It takes O(n) to locate the set with capacity restrictions, and O(m) to locate the set with room type restrictions. The intersection is O(n X m). Suppose 'p' of these rooms satisfy the day and time restrictions. The total processing time for the request after the final join is O(n X m X p). In the general case, if each located set has 'n' tuples, and there exist a total of 'k' intersections and joins for a request. The processing time is $O(n^{k+1})$. Thus querying the reservation system requires a polynomial time. The processing time is directly dependent on the number of intersections, and joins in the request. Insertions, and deletions require a index access and a union or difference on a set. The index access utilizes a hash function thus it has a constant order, and the union or difference require an addition or removal of a tuple, which again takes a constant amount of time. Thus the insertions, and deletions in the classroom reservation system are O(1) operations.

The application programs which form the four subsystems, and the data abstraction programs are included in Appendix B. Appendix C has a users manual which provides directions on how to perform reservation queries, and updates. It provides directions on the usage of program function keys, and a description of each of the four subsystems is provided with user interface screens. The final section of Appendix C contains a set of test cases which verified the system. The test cases checked both the ad hoc reservations, and the full semester reservations. The dynamic nature of the system was verified as expired requests were successfully eliminated at system startup. All the data retrieval functions were verified with entries in the schedule book. Additions, and deletions on the room inventory database were successfully executed. The system correctly caught course identification number duplications, and conflicting reservation requests. The ten minute restriction between reservations was checked. Both proactive, and retroactive reservations were successfully made. All program function keys were debugged to perform correctly. A tape dump procedure provides the needed back up for the classroom reservation system.

The extraction of data from the IMS database highlighted the external file access and imaging facilities in Model 204. The data definition phase brought out the tabular data format of the relational-like system. The indexing options were invisible to the user. Redefinition was easily performed without requiring changes in the application programs. Coding in user language brought forth the set-level processing capability, and provided a good understanding of the retrieval, file maintenance, and flow of control constructs. Studying the execution of a user language request exposed an unexpected inefficiency. Each request is compiled and executed at each execution, and no load modules are stored. The requirements for the developed system were obtained from the registrar's office at Oklahoma State University. Changing requirements over time, tested the data independence capabilities of Model 204. No data integrity checks are performed by Model 204. Integrity checks were implemented via procedural code. No primary keys, or foreign keys are supported by Model 204. The data security features proved to be extremely powerful. The file level, record level, and field level security features have been used in the developed system. Model 204 proved to be lacking in the storage organization area as IFAM (Inverted File Access Method) was the only access method available.

CHAPTER III

DATA DEFINITION

Data Definition in Model 204

The data definition in Model 204 can be done via a batch job, or interactively using user language. A Model 204 file needs to be allocated and initialized before any file definitions can be executed. The tables associated with a Model 204 file have been described in the storage organizations chapter. The data in Model 204 can be visualized to be organized in a tabular format. A record is the primary data object, and it is divided further into data fields. A Model 204 file can have multiple record types. The data definition discusses no physical placement or access paths to the data. The DEFINE FIELD command is used to define the fields in a Model 204 record. A field can be assigned the KEY attribute. The execution of the DEFINE FIELD command with the KEY attribute, creates an index on the mentioned field. The whole process is invisible to the user. At any stage in the life of a database application the REDEFINE command can be used to create indexes dynamically on existing data. The data can be normalized before the records are defined. Model 204 provides a variety of attributes which can be assigned to fields in a record. Fields which are used frequently for retrievals should be assigned the KEY attribute. If a field is updated heavily and is occasionally used for retrievals, it should be defined as NON-KEY. Range retrievals can be performed on a field if it is defined to have the NUMERIC-RANGE attribute. Logical relationships can be set up between records using the INVISIBLE attribute. The FOR-EACH-VALUE attribute allows the system to keep track of the number of unique entries for the specified field. The updates to an index in a Model 204 file can be deferred to a batch run if the key field has the DEFERABLE attribute . This provides efficiency and space reductions in an on-line data entry environment, as updates to the index are deferred to a batch execution. There are two field attributes that control the way the value of a field occurrence is changed: UPDATE IN PLACE and UPDATE AT END. If the UPDATE IN PLACE is specified, changing the value of a field occurrence will not change it's position relative to other occurrences of the same field. If UPDATE AT END is specified, a change in the value of a field occurrence is accomplished by deleting the existing occurrence, and adding a new one following the others. A user can secure against unauthorized access by including the LEVEL clause in the field's description. Field level security has negligible impact on both performance and storage usage. A field can be stored in a record in one of the following formats: string/binary, coded/non-coded, or float. The data definition of the class schedule database set up in Model 204 is shown in Figure 5.

```
DEFINE FIELD COURSE.ID (KEY)
DEFINE FIELD DEPARTMENT (KEY)
DEFINE FIELD COURSE.NO (KEY)
DEFINE FIELD COURSE.TYPE
DEFINE FIELD COURSE.DESCRP
DEFINE FIELD INSTRUCTOR (KEY)
DEFINE FIELD BUILDING (KEY)
DEFINE FIELD ROOM (KEY)
DEFINE FIELD SECTION
DEFINE FIELD COURSE.DAYS
DEFINE FIELD COURSE.BEGIN
DEFINE FIELD COURSE.END
DEFINE FIELD FREEFORM
DEFINE FIELD MIN
DEFINE FIELD MAX
DEFINE FIELD START (KEY)
DEFINE FIELD EXPIRE (KEY)
DEFINE FIELD COMMENT
DEFINE FIELD NUMBER
```

Figure 5. A Database Definition in Model 204

All retrieval fields have been assigned the KEY attribute. Eight hash indexes are dynamically set up for this database. CCA File Manager's Guide (11), provides detailed information on the data manipulation capabilities of Model 204.

Hierarchical Approach

IMS has two principal data defintion constructs: DBD (Data Base Definition), and PCB (Program Communication Block), McGee (29). Each IMS database is defined by a DBD, which specifies the tree-like hierarchic structure of the database. The tree-like structure is outlined via a hierarchic arrangement of segments. A PCB defines a logical view of the database. The PCB is derived from the underlying DBD. The sensitive segments and fields are enumerated in the PCB definition. A segment is the primary data object in IMS. A segment can be divided further into constituent fields. Figures 6 and 7 list a simplified DBD and PCB definition for a physical database. The definition is for the hierarchical database introduced in Chapter 1.

```
DBD       NAME = SCHEDULE
SEGMENT   NAME = COURSE, BYTES=25
FIELD     NAME = (COURSE.ID,SEQ), BYTES=5, START=1
FIELD     NAME = NAME, BYTES=20, START=6
SEGMENT   NAME = MEETING, PARENT=COURSE, BYTES=10
FIELD     NAME = TIME, BYTES=4, START=1
FIELD     NAME = BLDG, BYTES=3, START=5
FIELD     NAME = ROOM, BYTES=3, START=8
SEGMENT   NAME = DESCRIPTION, PARENT=COURSE, BYTES=15
FIELD     NAME = DESC, BYTES=15, START=1
SEGMENT   NAME = TEACHER, PARENT=MEETING, BYTES=10
FIELD     NAME = T.NAME, BYTES=10, START=1
SEGMENT   NAME = STUDENT, PARENT=MEETING, BYTES=6
FIELD     NAME = ENROLL, BYTES=2, START=1
FIELD     NAME = MAX, BYTES=2, START=3
FIELD     NAME = MIN, BYTES=2, START=5
```

Figure 6. A Simplified Database Definition in IMS

```
PCB       DBDNAME = SCHEDULE
SENSEG    NAME = COURSE, PROCOPT = G
SENFLD    NAME = COURSE.ID, START = 1
SENSEG    NAME = MEETING,PARENT=COURSE,PROCOPT=G,I,R,D
SENSEG    NAME = TEACHER, PARENT = MEETING, PROCOPT=G
```

Figure 7. A Simplified PCB Definition in IMS

The DBD statement assigns a name to the DBD definition. The second statement specifies COURSE as the root segment type, with a length of 25 bytes. Statements 3-4 define the fields which make up the course segment. The length of the field is given in bytes and the start position within the segment is listed. COURSE.ID is defined to be the sequence field for the course segment. SEQ signifies that COURSE.ID values are unique, and course trees are in an ascending COURSE.ID order in the course database. Statement five defines the meeting segment as being a dependent of the course segment. The definitions of fields within the segments follow. The rest of the statements can be explained in a similar manner. The PCB definition is a subset of the DBD. It can be derived from the underlying DBD. Any field or segment can be omitted in the PCB definition. If a given segment is omitted, then all dependents must be omitted too. Segments and fields that exist in a user's view are termed as sensitive. The PCB definition shows the course, meeting and teacher segments as being sensitive. The user remains unaware of the existence of the description and student segments. The sensitive segments and sensitive fields are specified by the SENSEG and SENFLD statements respectively. If a segment is specified to be sensitive and no sensitive fields are defined, then all fields default to being sensitive. The user is unaware of the course name field in the course segment, and all fields are sensitive by default in the meeting and teacher segments. The PROCOPT statement specifies the operations that can be carried out on the segment. Get, insert, replace, and delete are the valid options. The previous definitions describe the format of the physical database to IMS, but the file definitions need to be performed separately.

Discussion

The data definition is simpler in Model 204 than in IMS. In IMS the data is formatted in a tree-like structure. A fair amount of effort goes into deciding the content of segments, and the hierarchical layout of segments. In Model 204 the data is laid out in a tabular format. Once fields are assigned to a record, the database can be defined in Model 204. Fields can be added to a record type at any stage in the life of an application in Model 204. Such an addition, requires a complete redefinition in IMS. The physical database structure supports one primary key at the root of an IMS database. Any number of key fields can be defined for a Model 204 record. The REDEFINE command allows for dynamic indexing after the data has been loaded. The primary data object is a segment in IMS, and a record in Model 204. Normalization theory can be applied before setting up a Model 204 database to maintain data integrity. Data definition in IMS addresses both physical placement and access paths to the data. This data dependency does not occur in data definitions under Model 204. Logical relationships are set up via pseudo foreign keys in Model 204, and via pointer chains in IMS. Setting up of indexes and deciding on access methods is a separate step in IMS. In Model 204 indexes are created dynamically at the data definition phase.

CHAPTER IV

DATA MANIPULATION

Data Manipulation in Model 204

Model 204 provides data manipulation via two windows; namely, a host language interface, and a fourth generation language: 'user language'. Cobol, Fortran and PL/I are the host languages which can make IFAM (Inverted File Access Method) calls to perform data manipulations. The term relational is applicable only at the external and conceptual levels of the ANSI/SPARC architecture model. Interface with a user at an external level comes via a data sublanguage. A data sublanguage is further divided into a data manipulation language and a data definition language. In Model 204 the user language provides a relational like interface. The data can be viewed in a tabular format. Operators process records a set-at-time as opposed to procedurally a record-at-a-time. At the internal level Model 204 imitates the inverted list data model as the inverted file access method is utilized. At the external level the operations provided are closer to the relational model. The inverted list model primarily processes information one record-at-a-time. User language has no notion of database address areas which is a primary concept in inverted list data manipulation. A relational data sublanguage has four primary data manipulation operators; namely, select, update, delete and insert. Find, change, delete and store operators of user language correspond directly to the ones mentioned for the relational systems.

User language consists of statements of eight different types:

- selection statements based on the values of one or more fields, combined with a full range of boolean facilities;
- reporting statements provide facilities for simple ad hoc requests and sophisticated multifile reports;
- logical and control statements provide record set looping, request termination, and data dependent operations;
- database maintenance statements provide a wide range of update facilities;
- online application development statements provide for the creation, storage, modification and execution of user language requests;
- a full range of numeric and string computation functions;
- full screen formatting statements;
- external file access statements.

Statements of the type one through four and six will be discussed. The others are added features provided by Model 204. User language provides arithmetic and logical operations using constants, field values, special temporary variables, and functional values. Variable names start with a percent sign. More than 60 built-in functions provide facilities for data editing and validation, character string manipulation, global variables, and current date and time. The Fortran set of 26 mathematical functions is also available. Function names always begin with a dollar sign. Expressions are formed by combining constants, variables, functions or field names. Parentheses can alter the normal sequence of operations. Model 204 processes a user language request in two phases: compilation and evaluation. During compilation, the text is checked for proper syntax, and statements are translated into an internal format that Model 204 can execute. During evaluation, certain types of user responses are entered and the request is executed. Compilation errors must be resolved before a request can be executed. A user has a variety of options for evaluation time errors. Automatic evaluation continuation, request control, request cancellation, and user restart are some of the valid options for an evaluation error.

The data manipulation statements in user language fall under six classes:

- retrieval statements;
- loop statements;
- output statements;
- file maintenance statements;
- flow of control statements;
- miscellaneous statements.

Retrieval statements in user language begin with the keyword FIND. A set which can be acted on by loop statements, is located. The three main retrieval statements are :

```
1. FIND ALL RECORDS
2. FIND ALL RECORDS FOR WHICH  fieldname = value
                               fieldname = value1 OR value2
                               fieldname = NOT value
                               fieldname = value1 AND NOT value2
                               fieldname = NOT value1 NOR value2
3. FIND ALL RECORDS FOR WHICH fieldname IS BEFORE value
                                           AFTER value
```

The retrieval under the first choice locates all records in the current database. In the second choice all records which satisfy the condition of the FIND are located. The third choice illustrates range retrievals in Model 204.

Loop statements act on a set of records located by retrieval statements. Each record in a located set is acted on by a loop statement.

The format for a loop statement is as follows :

```
FOR EACH RECORD IN label
```

Label refers to the statement label for a retrieval statement. This loop statement shows the set-at-a-time processing capability of user language. Looping seems to utilize the most amount of code in a procedural language, thus this set level looping reduces the coding requirements.

Output statements are used to display results from database retrieval operations. A selected set of output operations is :

```
a. PRINT ALL INFORMATION;
b. PRINT  fieldname;
c. PRINT COUNT IN label.
```

Statement a is placed in a loop statement to print all fields of the current record in a located set. The output is in the fieldname = value pair form. Statement b is utilized in a similar manner except that a specific field value is output. Statement c prints the number of records in a located set. The label refers to the retrieval statement.

File maintenance statements perform the database updates in Model 204. The primary statements under this class are :

```
1. ADD fieldname = value;
2. CHANGE fieldname TO value;
3. DELETE fieldname;
4. DELETE RECORD;
5. DELETE ALL RECORDS IN label;
6. STORE RECORD
```

```
fieldname = value
fieldname = value
  .
  .
  . .
```

The ADD statement allows new fields to be added to records. This statement can only be used within a FOR EACH RECORD loop. The CHANGE statement is used to alter the value of a field within a record. If CHANGE is applied to a record that does not contain the specified field, then the fieldname and value are added to the record. The CHANGE statement can be used only within a loop statement. The DELETE fieldname statement is also only allowed in a FOR EACH RECORD loop. It removes a field from a record. If the DELETE statement is applied to a record that does not contain the field to be deleted, no action is taken on that record. The DELETE RECORD statement deletes the current record in a FOR EACH RECORD loop. The DELETE ALL RECORDS statement, deletes sets of records in a Model 204 file. The STORE RECORD statement is used to add new records to a Model 204 file. The fieldname = value pairs that constitute the new record must follow the STORE RECORD statement.

Flow of control statements allow for conditional processing in a user language request. A selected list of statements under this class is :

1. IF;
2. ELSE, ELSEIF;
3. JUMP TO label;
4. STOP;
5. SUBROUTINE;
6. CALL;
7. RETURN;
8. ON unit.

IF, ELSE and ELSEIF allow for condition testing. JUMP TO label is analogous to a go to statement. STOP terminates the program execution. SUBROUTINE, CALL and RETURN allow for structured programming as subroutines can be invoked to perform specific tasks. ON unit statements provide powerful error checking in case of run time errors. The purpose of the ON unit statement is to provide the user with a means of overriding the normal system response in case of a run time error. The three main types of ON units are :

- ON ATTENTION;
- ON ERROR;
- ON FIND CONFLICT.

ON ATTENTION allows the programmer to specify the action to be taken when a user presses the break/attention key during program execution. ON ERROR allows the programmer to specify the action before Model 204 cancels a request due to a run time error. ON FIND CONFLICT is used to resolve record level enqueueing conflicts when more than one user is trying to obtain update access to a specific record.

Some miscellaneous statements which perform powerful retrieval and manipulative operations are :

1. COUNT RECORDS IN label;
2. IS PRESENT;
3. SORT RECORDS IN label BY key AND key ....;
4. LIST;
5. VALUE LOOPS.

The COUNT RECORDS statement counts the number of records in a located set. A typical use of this statement is to check if a FIND statement resulted in an empty set or not. The IS PRESENT clause is utilized in a FIND statement to check if a specific field is present in a record. The SORT RECORDS statement invokes a system sort routine and the field names for the sort need to be specified. This is a logical sort and the actual records are not sorted. Records located via a FIND statement are placed in a buffer and sort is applied to records in this buffer. LIST is an extremely useful feature provided in user language. Retrieved records can be placed on a logical list. Then retrievals and updates can be performed on this list as opposed to the whole database. This adds efficiency, and provides localized processing. A FOR EACH VALUE statement initiates a loop that is executed once for each unique value of the specified field. It is useful for counting and printing records which have a given set of values and for detecting control breaks.

User language has various other features which do not directly fall under data manipulation. They provide facilities for effective input/output interfaces, external file access and application subsystem development support. The full screen formatting and cursor sensing capabilities of Model 204 are fairly advanced. More detailed information on the capabilities of user language are provided in CCA User language Manual (15).

Hierarchical Data Manipulation

IMS is a hierarchic database management system in which the data is stored in the form of an ordered set of trees. A tree consists of a root segment, together with a set of dependent segments. Each record is stored as a tree. Hierarchical data manipulation consists of operators for processing data represented in the form of trees. Hierarchic sequence in a tree traversal is obtained by a top-to-bottom, left-to-right ordered traversal. Each tree in the database can be regarded as a subtree of the system root. Thus the entire hierarchic database is in effect a single tree. The notion of hierarchic sequence applies to the entire database as well as to the individual trees. Many IMS data manipulation operators are defined in terms of this hierarchic sequence. An IMS database is queried via a CALL interface called DL/I (Data Language/One). The DL/I calls are made from application programs written in Cobol, Fortran, PL/I or 360 assembler, IBM Application Programming Reference Manual (20). IMS does not provide a fourth generation language. All data manipulation in IMS is done via record level operators, and no set level operations may be performed. Examples of some of the operators are :

- operator to locate a specific tree in the database;
- operator to move from one such tree to the next;
- operator to access segments within a record;
- operator to access segments in a hierarchic sequence for the entire database;
- operator to insert a new segment;
- operator to delete a specified segment.

The major DL/I operators available are :

- GU get unique;
- GN get next;
- GNP get next within parent;

- GHU,GHN,GHNP get hold (unique/next/next within parent);
- ISRT insert;
- DLET delete;
- REPL replace.

Get unique and insert operations require segment search arguments (SSA) to specify a hierarchic path. Delete and replace do not involve SSAs at all.

Get unique allows for direct retrieval from a hierarchic database. It retrieves the first segment in hierarchic sequence which satisfies the SSA. Status indicators are set which need to be checked after each GU operation.

Get next is defined in terms of the current database position i.e. the position last accessed by a "get" or an "insert" operation. This operator allows for sequential retrieval, once a database position is established.

Get next within parent is defined in terms of the current database position, as well as the current parent position. This allows for retrieval of dependent segments under a specific parent segment.

A get hold operation establishes the current database segment being addressed. The variations on the get hold operation allow for direct retrieval, sequential retrieval and sequential retrieval under current parent. Once the needed segment is obtained by a get hold operation, it can be replaced or deleted.

The insert operation allows for a new segment to be inserted, at a specific position in the database established by a get hold operation. The field values for the segment to be inserted need to be maintained in a temporary I/O buffer.

A segment to be deleted must be identified first by a get hold operation. The delete operation deletes the specified segment and all dependent segments if they exist.

The replace operator allows for a segment update operation. Again the segment to be updated needs to be located via a get hold operation. The update operation occurs in a temporary I/O buffer.

Enhanced retrieval operations can be performed by utilizing command codes. The data manipulation operators in IMS are record level. The operators are very closely related to the data model they operate on. The bulk of the coding in an IMS query procedure is performed in the host language. DL/I does not provide variables or conditional statements. The very basic database operations can be performed via DL/I calls, and any other processing requirements are left to the host language.

Discussion

IMS and Model 204 have completely different data manipulation operators, primarily because of the underlying data models involved. IMS does not have a fourth generation language interface like Model 204. One of the main differences is that IMS only supports record-at-a-time processing, while as Model 204 allows for set-at-a-time processing. The set level processing capability of Model 204 is embedded in the user language query statements. Set processing eliminates the need for explicit looping statements. The DL/I calls provide for record-at-a-time access, therefore explicit looping constructs present in a host language need to be utilized. The retrieval operators in IMS query an ordered set of trees in a hierarchic sequence. There is no ordered sequence that a user language request needs to adhere to. User language itself is not a truly relational language. The only similarity is the set level processing. The set theoretic operations select, project and join are not directly supported by user language. User language does not fall under an inverted list data model language as it has no notion of database address areas, and the processing is set-at-a-time. DL/I does not provide flow of control constructs which are available in user language. Retrieval is conceptually straight forward in user language as opposed to the tree structured access needed in IMS. An application programmer must know the underlying tree structure in an IMS database before a request can be coded. On the other hand if the field names of a Model 204 record are known, a user language request could be easily coded. The task for an application programmer is definitely easier using a fourth generation language in Model 204. The illustration below highlights this point.

QUERY: "get a list of all courses taught by the computer science department".

The hierarchic database is presumed to have only one type of segment with the course id and department.

DL/I calls

```
          GU COURSE WHERE DEPARTMENT = 'comsc'
          while more COURSES do
```

```
        print course.id
        GN COURSE WHERE DEPARTMENT = 'comsc'

    end while
```

The Model 204 database is presumed to have records with just two fields namely, course.id and department.

user language

```
        LABEL: FIND ALL RECORDS FOR WHICH DEPARTMENT = 'comsc'
                   END FIND

        FOR EACH RECORD IN LABEL
            PRINT COURSE.ID
        END FOR
```

The IMS retrieval has been pseudo-coded and simplified to highlight the main points. The while loop needs to be coded explicitly in the host language. After each get unique or get next operation the status of the operation needs to be checked, to verify if the search was successful or not. The coding in user language is less taxing. The FIND operation locates all the needed records, and the FOR EACH RECORD loop prints off the needed course.id numbers. The set level operators are responsible for the simplicity of user language. This example was extremely simplified as the root segment had the needed information. If dependent segments are involved, then coding requirements increase. The user language request on the other hand stays the same, independent of the data format.

A user language request is executed in a two step process namely, the compilation phase and the execution phase. In the compilation phase the user language request is parsed for syntax errors. If errors occur, they are flagged and the compilation is aborted. If no errors occur the compiler generates command tables which contain hexadecimal code. In the execution phase this hexadecimal code is executed to perform a database operation. An example helps clarify this compilation and execution process. Suppose a user language FIND is to be executed to locate all tuples with NAME=JONES. Initially the syntax of the request is verified, then a loop is created for each segment in the database. A segment consists of 50,000 records. The request is executed in segments to gain efficiency. The name JONES is hashed to obtain the correct entries in the index for the NAME field. All this information is stored in hexadecimal code in temporary command tables. During the execution phase this command table code is executed. Variations to the indexing process are possible. Instead of a hash index, a B-tree index may be used. The application subsystem facility allows a user to save the compiled command tables, so that each time a request is executed, only the execution phase is triggered.

Data manipulation is performed via a host language interface in IMS. The host language program has DL/I operators to perform database queries. These DL/I operators are executable subroutines in the resident load module library in IMS. Thus IMS has subroutines to perform the get unique, get next, get next within parent and other IMS operations. While the user program is running it issues calls to these resident action modules to retrieve data from and update data in IMS databases. An application program communicates with the system through a set of Program Communication Blocks (PCBs). The PCBs for a program are produced and stored in a library at the time the program is defined to IMS. PCBs contain user declared program attributes as well as parameters that are passed between the program and the system during execution. Application programs invoke system services through calls to a standard interface routine, specifying the function to be performed, the PCB to be used to communicate the parameters and results of the call, and additional parameters as appropriate to the function being invoked. As a result of the call, control goes to the interface routine and thence to various system modules to carry out the requested function. The system places feedback information in the designated PCB and returns control to the program. A call statement to the interface routine has the following form (in PL/I programs):

```
  CALL PLITDLI ( parmcount, function, pcbptr, workarea, ssal, ssa2,..)
```

where :

- parmcount designates the number of parameters in the call;

- function designates a character string variable that holds the name of the function to be performed;

- pcbptr designates a pointer variable that points to a database pcb;

- ssal, ssa2,.. designate character string variables that hold Segment Search Arguments (SSAs), that collectively designate the segment or the segment path to be accessed;

- workarea designates an area in the program where segments and segment paths are deposited and picked up by the system. A segment is designated by specifying an SSA for each level in the record hierarchy, down to and including the level of the segment in question. Each SSA has the following form:

```
segment-type-name *    command-codes (condition)
```

and designates the first segment under the designated parent that is of the specified type, and meets the specified condition. Conditions consist of one or more logical predicates that are separated by AND and OR operators. The system responds to a data manipulation call by performing the function called for by placing feedback information in the PCB specified in the call, including the following:

- status code, to indicate that the function has been performed successfully, or that it has not been performed, for a reason indicated in the code;

- level, type, and concatenated key of the segment accessed, or of the segments that define the path accessed.

The selection power of a data manipulation language is defined as it's ability to express a database query, whose answer is contained in the database. Any query expressable in relational algebra or relational calculus is expressable in user language. IMS queries are restricted by the access path dependency. Conciseness of a data manipulation language is defined as the lack of verbosity in the language. It provides a measure of the quantity of code needed to express a query. A user language query proves to be concise due to the presence of control and exception testing statements, and the increased scope of language expression.

CHAPTER V

DATA INDEPENDENCE

Discussion

Data independence is defined as the immunity of application programs to change in storage structure and/or access strategy. In a data dependent system the knowledge of the data organization and access technique is built into the application logic and code. Data independence can be divided further into physical and logical data independence. Physical data independence implies program immunity to changes in the storage structure, while logical data independence implies program immunity to changes in the data model definition. Physical data independence allows application programs to execute correctly after the storage has been tuned to optimize overall performance, to take advantage of new hardware technology, and/or to implement new standards in the storage structure. Logical data independence allows application programs to execute correctly after the data model has been changed in response to changing requirements. Logical data independence can be studied under two aspects: growth, and restructuring. Growth implies addition of another field to a record, or addition of a new record type, or deletion of a field or record type. Restructuring refers to a change in the database such that although the information content of the database stays the same, the placement of information within the database changes; i.e., allocation of fields to records is altered. There may exist a need to split a record vertically, so that commonly required fields may be stored on a faster device, and less frequently desired ones stay on a slower one. Physical data independence is guaranteed by the nature of the relational data model. The relational model is at an external/conceptual level of the ANSI/SPARC database architecture model, thus any changes at the internal level do not affect the applications. The hierarchic and inverted list systems are data dependent as they are at the internal level. In Model 204 inverted file access is the only form of access method utilized. Thus, a change in access strategy is not possible. The applications in Model 204 are independent of the type of a file. Hashed, sorted and indexed are the file types available. Any file type can be changed to any other without affecting a user language program. HSAM, HISAM, HDAM, and HIDAM are some of the access methods available under IMS. A change in the access mechanism does not require a change in the host language programs, provided the logical structure of the database stays the same. Thus the degree of physical data independence under the two systems is identical. Degree of data independence is an indication of the extent to which a system is data independent.

Model 204 has a high degree of logical data independence. A field can be added or deleted from an existing record in a Model 204 file, without affecting existing applications. New record types may be introduced into a file without requiring any changes in existing applications. Thus new applications can be readily added to an existing database. Model 204 does not perform well under the restructuring aspect of logical data independence. A split of a record type requires the introduction of foreign keys, and thus requires modifications in application programs. IMS is logically data dependent due to the hierarchic model it imitates. Addition or deletion of a field from a segment requires a modification of the application programs, and re-definition of the database. Addition of a new segment type requires the same changes too. This dependency in IMS is called access path dependence, which occurs due to the pre-defined access paths in application programs. Restructuring in IMS requires a complete database redefinition. Thus the degree of physical data independence is similar under the two systems, but Model 204 is more logically data independent.

CHAPTER VI

DATA PROTECTION

Data Integrity

Data integrity means the accuracy or correctness of data in a database. Most systems today are weak in integrity checks, they only provide concurrency control i.e. two users cannot concurrently update the same database record, Michaels, Mittman, Carlson (30). Both Model 204 and IMS provide this type of control. IMS provides for both referential and entity integrity, but Model 204 proves to be lacking in this area. Most integrity checks are done by user-written procedural code in Model 204.

There are two types of integrity rules, one pertains to primary keys, and the other to foreign keys. The entity integrity rule states that no attributes participating in the primary key of a base relation are allowed to accept null values. Referential integrity states that if a base relation R2, includes a foreign key FK, matching the primary key PK, of some base relation R1, then every value of FK in R2 must either be equal to the value of PK in some tuple of R1, or be wholly null. Let us consider the integrity rules under the data models involved in the comparison. The relational data model supports both the entity and referential integrity rules stated above. The inverted list data model provides no integrity rules. The hierarchic data model has automatic support for certain forms of referential integrity. No child is allowed to exist without its parent.

If a parent is deleted, the system automatically deletes the (sub)tree rooted at the parent. Similarly a child cannot be inserted unless its parent already exists. Therefore the hierarchic data model enforces the following rules:

- nulls not allowed;
- delete cascades;
- update cascades.

Model 204 does not support primary or foreign keys. Two records can have the same value for the primary key. A foreign key in Model 204 may not be wholly null, or match the primary key of another record type. There exists no domain constraints in the data definition part of Model 204. The notion of domains is similar to that of data types. There is no support for range constraints either. No data type checking exists in user language. Model 204 accepts null values for any field in a database record. Thus Model 204 closely reflects the underlying inverted list data model's data integrity capabilities.

IMS on the other hand supports all the integrity checks imposed by the hierarchical data model. It supports additional constraints by means of it's logical database and secondary indexing capabilities. The locking protocols to control concurrent updates in both systems are similar. A segment can be exclusively locked in IMS, and a record can be locked in Model 204. DL/I does not provide any locking protocol statements, but user language under Model 204 provides the following:

- FIND AND RESERVE RECORDS;
- ON FIND CONFLICT;

- COMMIT;
- TRANSACTION BACKOUT.

The FIND AND RESERVE statement obtains exclusive access for a set of database records. The ON FIND CONFLICT statement is an error check clause to react to a condition when an application is trying to access record(s) held by another user. The COMMIT statement completes a database update and removes exclusive access from a set of records. The TRANSACTION BACKOUT feature undoes any database update operation which left the database in an erroneous state. More detailed information on data integrity checks is provided in the CCA User language Manual (15).

Data Security in Model 204

The file manager in Model 204 is responsible for the data security of a database. Eight basic types of Model 204 security features are : login security, file security, group security, record security, field level security, procedure security, subsystem security and terminal security, CCA File Manager's Guide (11).

The login security feature limits access to the Model 204 system by requiring a user to enter a valid password while logging on to the system. The system manager maintains a system access table with privileges for each user identification number. These privileges are granted once a user logs on successfully.

File security is related to protected access to Model 204 files. Files in Model 204 can be password protected. If a user successfully opens a file with the correct password, then appropriate privileges pertaining to the data and application programs, as well as a user class number that is used with procedure security, and field level security levels, are assigned to the user. The security level is segregated into three levels namely public, semi-public, and private. A public file is not password protected and default privileges are assigned to the user. A semi-public file requires a password to grant a user the needed privileges. An incorrect password for a semi-public file results in default privileges being assigned to the user. A private file requires a correct password to give a user access to the data, and application programs residing in that file. The privileges granted to a user determines the type of operation that the user can perform. The system manager maintains a password table, which contains password and privilege information for files, file groups and login accounts. A file can have several passwords defined for it, and each password may have a different set of privileges associated with it. A user can be assigned the privilege to override record security. Data update by ad hoc requests or host language programs comes under file privileges. The ability to run, view, update or delete application programs comes under file privileges.

Group security is identical to file security except a set of Model 204 files comprise a group. If files of a group were individually defined to the system then the user's privileges are the intersection of the individual privileges of each file. Only privileges that every file has specified for it are granted. If a file of a group is accessed individually then the group privileges are imposed on the file. But if a file is opened both as a member of a group as well as an individual file then individual file privileges are imposed on the file. If a file is referenced and it is concurrently a member of two groups, the user's privileges are the union of the privileges associated with the groups.

Record security limits user access to individual records in a Model 204 file. Retrieval or update of protected records is limited to privileged users. A file needs to be defined to have record security during the file creation phase. If a file has record security active then every record stored in the file has the user identification number appended to it. Thus during retrievals a match with the user id and the security key in a record allows access to the record. To allow for multiple access to records, the user needs to explicitly append user ids which can be granted access to the record.

Field level security protects sensitive fields in a Model 204 record. This restricts the kinds of access to fields within a record. Field level security is imposed on the file when it is opened for access. Field level security comes into effect only if access to a data record has been granted by previous file level and record level security measures. Field level security is implemented via the following access privileges:

- select, ability to locate records with a user language statement;
- read, ability to display field contents;
- update, ability to change the contents of a field;
- add, ability to add new occurrences of a field.

The above access privileges are termed as user levels. The add level can be used in the following environment: data entry clerks can add new field occurrences or records without being able to change or even examine them. Field levels are defined as the access privileges associated with a field when it is defined. Levels are numbered from 0 to 255. Zero implies no security and 255 implies the highest security. Fields can be assigned levels in a hierarchical manner.

FIELD	FIELD LEVEL SECURITY(READ)
course.id	0
course name	0
department	20
enrolled students	30
instructor name	40

Figure 8. A Field Security Scheme in Model 204

The fields are listed in the order of increasing sensitivity. Instructor name is the most sensitive field; course.id and course name have no security level imposed on them. Each user has field level security access levels associated with each file opened. These correspond to the four field access levels: select, read, update and add. These user levels also range from 0 to 255. When a user attempts to access a field in a particular way, the system compares the user's access level with the field's access level. If the user's access level for the desired access type (e.g update) is greater than or equal to the field's access level, then the particular operation is allowed. Taking the previous field level security example, if a user has read user level set at 30, then read access is permitted to the fields with field access levels less than or equal to 30 (i.e. the user can access the course.id, course name, department, and enrolled student fields, but access to the instructor field is disallowed). The user levels reside in the password table for a file.

Procedure security is related to access to application programs. A

Model 204 file can contain both application programs as well as data. Separate sections in the file are dedicated for these specific purposes. Procedure security restricts an unauthorized user from invoking an application program and accessing sensitive data.

An application subsystem in Model 204 refers to a group of data files and application programs which perform related functions for a specific application. All the previously mentioned security measures can be assigned in a concise manner to an entire application subsystem.

Terminal security restricts access to certain user ids, files, or groups from specific terminals. Each login id, file or group can be made accessible from a restricted list of terminals only. Terminal security is used with hardwired terminals only, because the node name for a terminal remains a constant. Dial up terminals can have the node number change for a terminal on each dialling, so terminal security is not possible for these types of terminals.

Data Security in IMS

To protect against unauthorized use of the system, IMS provides two types of security; basic security and security tables, McGee (29). The basic security restricts IMS commands to the master terminal only. The master terminal is used for monitoring of on-line execution, startup, shutdown, and enabling and disabling of lines and terminals. Attempts to enter such a command from a terminal other than the master terminal are rejected. Basic security is implemented via security tables that are built with the security maintenance utilities program. The security definitions from these tables come into effect at startup. The master terminal can be used to override these constraints. A typical entry in the security tables has the following format:

```
transaction-type-code ,    logical-terminal-name
command name
```

This limits particular transactions or commands to particular logical terminals. The security tables may also contain entries of the following type:

```
transaction-type-code, remote-execution-id
```

This requires that particular transaction types originate from particular remote executions. A different set of definitions may be supplied for each logical terminal that is associated with a given physical terminal, thus giving the physical terminal different security attributes, depending on which one of it's logical terminals is enabled. The security tables may further control terminal user access via entries of the following form:

```
transaction-type-code     , password
command name
```

These entries require that transactions or commands of the specified type contain a specific password, before they can be accepted by the system, regardless of the logical terminal from which they are entered. Data security features are enforced in IMS at the data definition phase also. A sensitive segment is one which can be viewed by a user. These segments, and fields which form the segments are included in the user's view. A user of the view is not aware of any other segments or fields, thus providing security to these hidden segments and fields. Sensitive segments and sensitive fields are specified by the SENSEG and SENFLD statements respectively. If a certain sensitive segment has no sensitive fields defined, then by default all fields in that segment are sensitive. The PROCOPT (processing option) entries in a SENSEG definition specify the valid operation types. I(insert), R(replace), D(delete), and G(get) are the possible PROCOPT options. A definition of a sensitive segment is given below.

```
SENSEG      NAME = MEETING, PROCOPT = I,R,D,G
```

The statement specifies that the meeting segment is in the user's view, and insert, delete, replace and get operations may be performed against this segment type. IMS relies on file level security on another package RACF (Resource Allocation & Control Facility). Data entry data bases allow data to be input via certain logical terminals only.

Discussion

The file level security features are more advanced in Model 204 than in IMS. IMS depends on file level security on RACF (Resource Allocation & Control Facility). The basic data object that can be protected in IMS is segment, and for Model 204 it is a record. Field level security is assigned to these objects when they are defined. Both systems allow for fields to be defined as sensitive. The overhead due to security options in both systems is heavy. In IMS each transaction on a segment is verified with the processing options allowed for that segment type. Similarly every record update in Model 204 is checked for field security levels. Terminal security features are comparable under the two systems. Both systems provide audit trails to detect unauthorized access. Model 204 provides data encryption in way of the CODED field attribute. Application programs are protected in Model 204 via the procedure security option. Application programs in IMS are developed in a host language, thus the development environment is responsible for their security and not IMS.

CHAPTER VII

STORAGE ORGANIZATIONS

Storage Organizations in Model 204

The Model 204 file system supports the following kinds of data structures: flat structures, relations, hierarchies, and networks, CCA File Manager's Guide (11). No physical linkage is used between data items, and the relationships are maintained at a logical level by the use of value indexes. The primary access technique utilized in Model 204 is inverted list. A field is the smallest data item possible in Model 204. A collection of fields forms a record. A file is defined as an arbitrary collection of records. Each field has a name and a value, and the various data manipulations are performed on these field name = value pairs. These fields can be assigned certain pre-defined attributes that define indexing options, and internal storage structure formats. Model 204 allows a maximum of 4000 different field names in a single file. The records are variable length, with no limit to the number of fields in a record. The record does not have a pre-defined format, and any number of fields can appear any number of times. Each Model 204 record is assigned an internal record number, which is used by the system to build index entries for the record. A file can contain records with varied formats. 16.7 million records is the maximum limit for a Model 204 file. These files can be logically linked via field values. Any number of files can be logically linked in such a manner. One Model 204 job can access a maximum of 32,767 files. Due to the logical nature of the relational data model the files and records have a flexible format. A new field can be added to an existing record even though it was not previously defined. The file supports additions of completely new types of records. New logical relationships can be developed among fields, records or files, without modification to the underlying structure. The flexible nature of this data model can be contrasted with the data dependent hierarchical data model. The records in Figure 9 help in explaining the inverted list format of a Model 204 file.

INTERNAL RECORD NUMBER	COURSE.ID	DEPT.	ROOM	INSTRUCTOR
0	12296	COMSC	MS 212	RAY,HOLMES
1	12325	COMSC	MS 121	BATES
2	29132	MATH		JONES
3	31298	ECEN	ES 212	JAMES
4	32915	MATH	MS 212	JOHN

INVERTED LIST

DEPT = COMSC	0,1
DEPT = MATH	2,4
DEPT = ECEN	3
ROOM = MS 212	0,4
ROOM = MS 121	1
ROOM = ES 212	3

Figure 9. An IFAM example

The example shows five records and their index entries. There are multiple occurrences of a field in a record(e.g. record 0 instructor field), and a record may have a missing field value (e.g. record 2 room field). When a Model 204 file is created certain fields can be assigned the KEY attribute, so that they can be indexed. Each index entry contains one field name = value pair, and a list of records in which the pair occurs. When a KEY field is defined, it's internal record number is noted in the index. To retrieve all MATH department courses which meet in MS 212 a search of the file index is performed. MATH appears in records 2 and 4, and MS 212 appears in records 0 and 4. Model 204 then compares the two lists and pulls out record 4 which satisfies the request criteria. Fields are given the NON-KEY attribute in case keyed access is not needed. This saves index space. But if a retrieval performs a search by one of these NON-KEY fields, the performance greatly reduces as a sequential search of the database is performed. This feature of allowing a NON-KEY field to query the database may result in highly inefficient application programs. In a hierarchical database like IMS each key field has to be explicitly stated, and access paths pre-defined, so this performance deterioration can never occur.

A Model 204 database consists of one or more physical datasets. These datasets consist of fixed length records called pages. A Model 204 file is divided into 5 tables or sections.

1) FCT - File Control Table keeps track of the file parameters, file definition names of all datasets on the field, and other control information. The FCT is of a fixed size; usually it is fairly small in comparison to the rest of the file.

2) TABLE A - is a dictionary of the field names and coded field values in the file. It is further divided into sections for field names, values of

FEW-VALUED fields, and MANY-VALUED fields. The field name section should be as small as possible to aid efficient access. Table A is fairly small as compared to the other tables.

3) TABLE B - contains the retrievable data in a Model 204 file. This is the largest section of the file. Records in Table B are stored in internal file segments to minimize storage and optimize retrieval.

4) TABLE C and TABLE D - make up the indexing structure necessary for key retrieval of records. There is an entry in Table C for every field name = value pair that occurs in the file for fields defined as key. If the field name = value pair is not unique in the file, Table C contains a pointer to an entry in Table D. Table C is a hashed file divided into entries of 7 bytes each. As mentioned earlier it stores index information for a KEY field. A chain of entries is stored in Table C for each value stored in a KEY field. The head of each chain is called a "property entry". The property chain identifies the field name = value pair that is indexed by other entries in the chain. An entry is placed in the chain for each segment of the containing records that have the field name = value pair in the property. Table D contains lists of Table B record numbers for all of the KEY field name = value pairs that occur more than once in the file. It also contains user language procedures, a procedure dictionary (used to store procedure name and procedure classes).

There is some free space available to the file on unassigned pages in the free space pool. A KEY field has an index in Tables C and D, while the data records reside in Table B. The KEY fields allow for quick index based retrieval, but insert, delete or update operation is slowed down as indexes have to be updated. These operations on an ordinary NON-KEY field require very little processing time as records in Table B only need to be manipulated. A search on a NON-KEY field involves a sequential search of Table B records. The sequential search costs can be reduced in cases where both KEY and NON-KEY fields are specified in retrieval conditions. In this case, Model 204 diminishes the number of records to be searched directly by performing the indexed selection first. Records that are eliminated, based on KEY conditions, are not searched sequentially. The Model 204 file system provides an extremely powerful operation to redefine a field as KEY, after the file has been pre-defined and loaded with this field as NON-KEY. The setting up of indexes and pointers is done dynamically and is invisible to the user. Sufficient space should be left in Table C for this inversion. The Table C size is computed using the following formula:

$$\text{CSIZE} = 1.2 * \frac{(14*V_u) + 7(N+1)(V_n + V_r)}{\text{Usable Page Size}}$$

V_u = total number of fieldname = value pairs that usually appear in only one record in the file (e.g. course.id).
V_n = total number of fieldname = value pairs that usually appear in more than one record in the file (e.g. course.name).
V_r = total number of extra entries required for all numeric range retrieval fields.

V_u and V_n apply to fields with key or numeric range attributes.

This is a productivity booster for an application programmer in Model 204. In a hierarchical system such a field attribute change would require a complete redefinition for access by the new key.

A field can have a NUMERIC RANGE attribute so retrievals for field values numerically equal to, less than, or greater than, or in between certain values can be performed. The INVISIBLE attribute allows the user to store logical relationships between physical records. A set of physical records can be retrieved by an INVISIBLE attribute field. Under normal circumstances, the storing and updating of logical records in a Model 204 file is done at one time i.e Tables A,B,C,D are changed simultaneously. When there is a high volume of updates, efficiency and space reductions can be gained by deferring the updates to the index (Tables C and D). The deferred update feature is provided via the DEFERRED attribute for a field. The CODED attribute can be used to save space. When a value that has the CODED attribute is defined, the character string is stored in Table A (the internal file directory), and a four byte value code pointing to that character string is stored in a logical record in Table B. Space is saved when there are several records that contain the same value. The string is stored only once in Table A, and the four byte code is stored in each of the several records in Table B. The coding and decoding of these values may slow down updates and retrievals at the cost of saved space. Field-value encoding is entirely transparent to the user. Data is returned exactly as it was entered, and codes are system generated. As Table B contains the data records, a file structure may be imposed on this Table. Entry ordered, reuse-direct-file-space, sorted and hashed files can be used to store Table B records. Model 204 primarily utilizes the inverted file access method. Figure 10 explains this inverted list format. A primary index on course.id is referenced by two secondary indexes on department and room. A -1 in the pointer field signifies the end of a pointer chain. Files such as our inverted indices, in which a secondary key leads to a set of one or more primary keys, are called inverted lists. The inverted aspect comes in when a secondary key works its way back to a primary key. After the primary key is located, the primary key index helps locate the physical record.

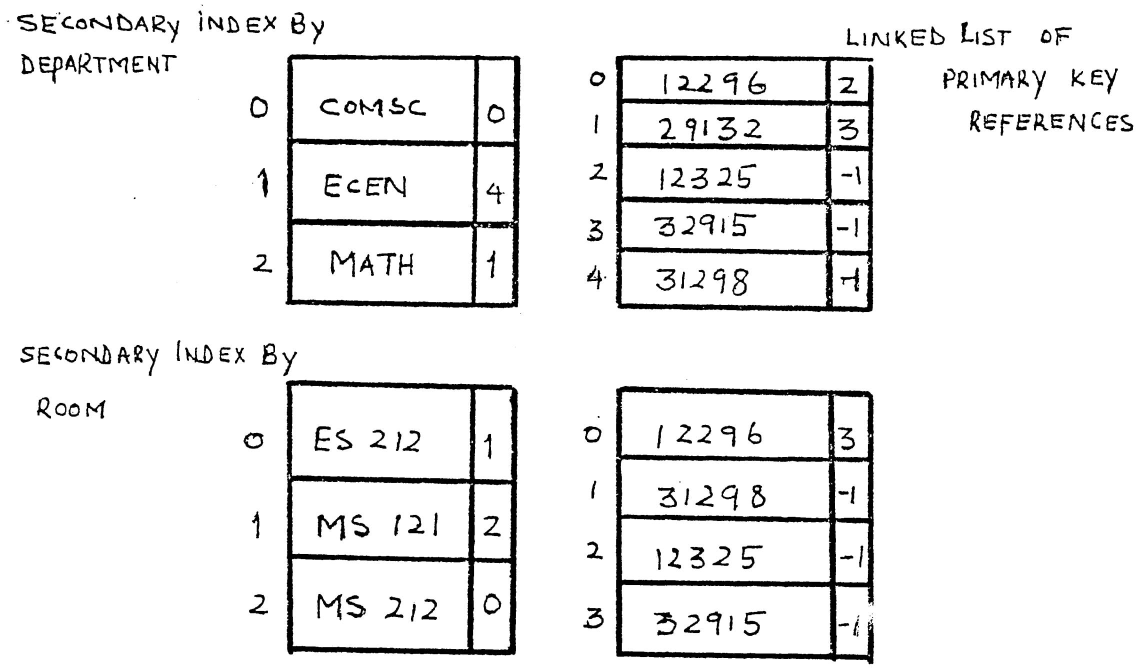


Figure 10. Inverted Lists

Entry ordered files store records in a chronological order in Table B. These files provide inverted list capabilities. When a record in an entry ordered file is deleted, the internal record number is not used. New records are always appended to the end of Table B. This mode, called append only is the least expensive for updates. However Table B gets filled up even though space has been released by record deletions. When the reuse option is specified, then the entry order of records is disrupted. Such files are called reuse-direct-file-space files. There are two append modes in such files namely reuse-first mode and append-first mode. In the reuse-first mode Table B record numbers freed by deletions are reused for additions whenever possible. This mode optimizes Table B space utilization at the expense of update time. In the append-first mode records are appended to the end of Table B, as long as there is space available. Record numbers are reused only when there is no space left at the end of the file. In a sorted Model 204 file a particular field is designated as a sort key, and logical records are stored in Table B in a sorted order by key. A sort key can be alphabetic, thus it provides a convenient method of doing alphabetic ranging on a single key without doing an actual search. This sorted organization of Table B is similar to ISAM (Indexed Sequential Access Method) with master and overflow areas. The logical records of a sorted file are stored on the pages of Table B. The pages in a sorted file are grouped by the range of the key values. A sort group consists of a fixed number of pages of master area in which the records are stored in strict order, and a fixed number of pages of overflow area to accommodate overflow records from the master area. A number of extra overflow areas are also reserved at the beginning of Table B. A sorted file needs to be loaded with pre-sorted input so that the records will be stored sequentially on the master area pages. For example, a file with each master area occupying two pages, each overflow area occupying one page, and three extra overflow areas at the beginning of Table B, after the initial load is shown in Figure 11.

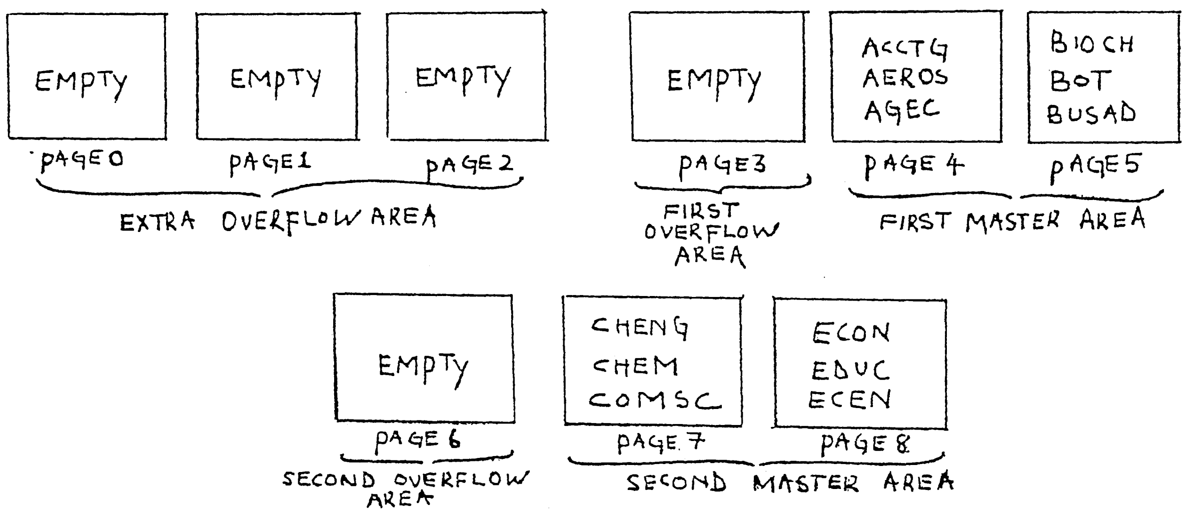


Figure 11. File Load in Model 204

In the figure above the first sort group (pages 3 through 5) contains all records with the sort key from ACCTG to BUSAD. A new record insertion is first attempted in the master area. If it cannot be inserted in the proper order, it will be stored instead on a page of that sort group's overflow area. If all of the pages of this preferred overflow area are full, the record is said to "spill" to a previous or extra overflow area. Indexed sequential access is a term used to describe situations in which a user wants both sequential access to records in order by key and indexed access to the same records. ISAM can be provided by the B+ tree data structure.

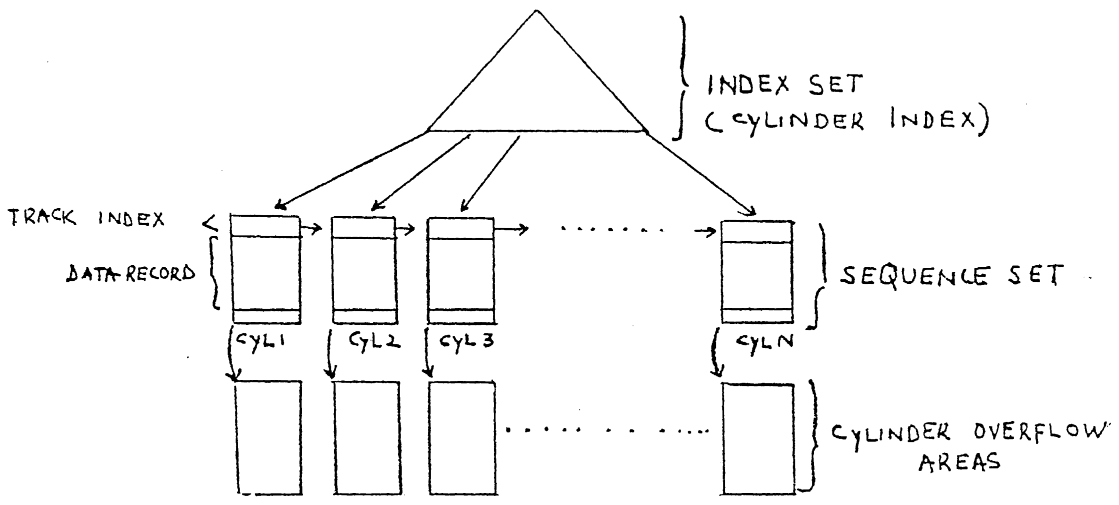


Figure 12. Indexed Sequential Access Method

A B+ tree consists of an index set providing the indexed access, and a sequence set for the sequential access. All the data records are stored in the sequence set. Insertions and deletions of records are handled by splitting, concatenating, and re-distributing blocks in the sequence set. The index set which is a B tree is used as a locater for the blocks in the sequence set. When read in a logical order, block after block, the sequence set lists all of the records in order by key. The index set can be viewed as a cylinder index, with pointers to all cylinders used to store an indexed sequential file. The sequence set can be viewed as a linked list of cylinders. A certain portion of each cylinder is dedicated as a track index. This index locates the track in a cylinder on which a record is stored. Each cylinder has its overflow area which is primarily a linked list. There are various optional independent overflow areas present too. The tracks are filled to a maximum of 80% when the file is initially loaded. This allows for later insertions. If a track gets filled up then the overflow area is utilized. Each track has a pointer to the overflow area. If a cylinder gets filled up then either the file needs to be reorganized or the optional independent overflow area needs to be utilized.

In a Model 204 hash key file, a particular key is chosen as a hash key, and logical records are randomly placed in Table B according to their hash key values. As Model 204 hashes directly to the Table B records , this reduces disk I/O. Hash key files are recommended for query operations where a single primary key performs the retrievals. Extra space in Table B is needed to accommodate the large number of bucket slots. Hash key files are thus suited for applications which retrieve and process records one at a time. Model 204 typically requires two disk transfers to accomplish this: one to index in Table C, and one to the logical record in Table B. Using hash files eliminates the the disk access for the index entry. When a record is stored in a hash key file, it is stored on an apparently random page in Table B; the page number is governed by the record's hash key. Records which collide to the same page number will be stored in a progressive overflow manner in Table B.

Model 204 gives the file manager an option of storing data in a single physical file, in multiple files or in a file group. Several logical files can be incorporated into a single physical file, because multiple logical record types are supported in each physical file. This saves disk accesses as explicit file cross referencing is avoided. Data redundancy is reduced by combining logical files. The smaller indexes reduce disk storage overhead. Separate logical files have their advantages too. If the on-line storage is limited, some unnecessary files can be left off-line.

Heavily manipulated data can be targeted for checkpointing and backup. Separate logical files make controlled access to data easier to implement. A group is a collection of physically distinct files, which appears to the user as a single logical entity. Thus advantages of both the previous file types are incorporated into a group. File grouping is ideal for data aging applications. Each member file forms a replacable aging unit. Data sharing is facilitated by a file belonging to multiple file groups.

Storage Organizations in IMS

IMS provides two distinct classes of data structures: DA class, which are seen by the data administrator; and the AP class, which are seen by the application programmer, McGee (29). The DA class is primarily hierarchic, but with provisions for interconnection of hierarchies into networks to reduce redundancy. This class is designed for efficient storage and retrieval of data. The AP class is strictly hierarchic, and is a subset of the DA class. This class provides a simplified view of the data, appropriate for application development. The AP structures can be defined in terms of DA structures, and all operations performed on the AP class are automatically mapped to the underlying DA class. The DA data structure class, and the the implementation of DA structures will be described.

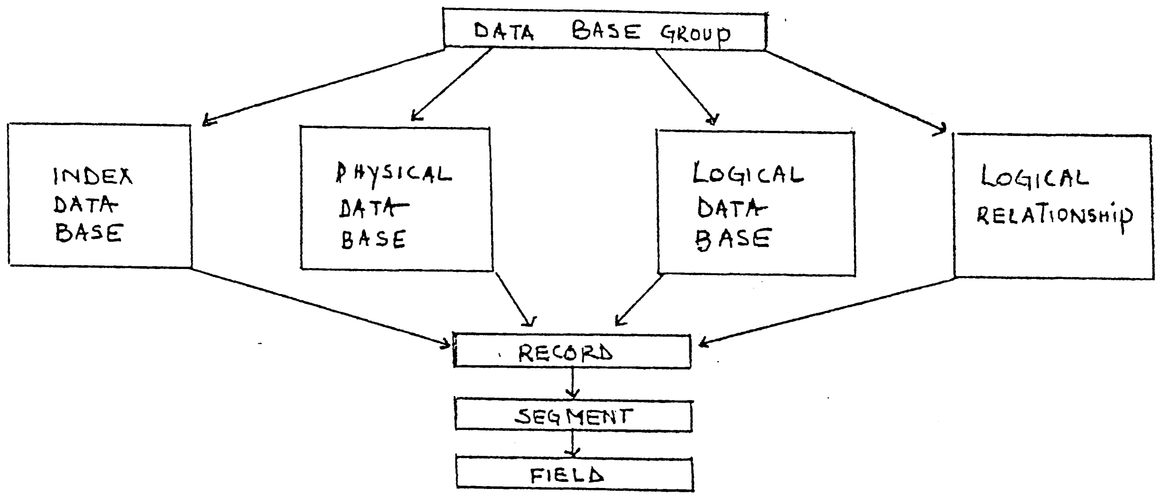


Figure 13. Data Administrator Data Structure Types

Figure 13 illustrated the various DA data structure types, and the way in which the structures are composed of other structures. A segment is the basic structure in the DA class. It is primarily a string of bytes. Segments can either be fixed or variable length, and can comprise of one or more fields. A field is a string of bytes within a segment. An example of a segment with associated fields would be a course segment with fields describing the course number, course name and course department. One of these fields may designate the segment sequence field, or key. A record is a single rooted tree of segments that is produced from it's record type in accordance with the following rules:

- The root segment type produces a single segment of that type;
- Each dependent (child) segment type produces zero or more segments of that type (twins) under each instance of it's parent segment type. The twins are sequenced by a key, if the child segment type has a key.

The root segment represents a major application entity type, and the dependent segment types represent hierarchically subordinate entity types, or collection of attributes that occur optionally or with variable frequency. Concatenated keys are used to identify a segment uniquely within a database record. A concatenated key is constructed by concatenation of key values along the tree access path to that segment. A physical database is a set of records of a single type. The sequence of records in a database is determined by the method used to implement it. The various methods to implement physical databases will be described in detail later. To reduce redundancy in physical databases the logical relationship structure type is utilized. It permits shared access to stored data. Many-to-many relationships are supported between entity types. These relationships can be both unidirectional and bidirectional. The application programmer's view is limited to hierarchic records, and virtual hierarchic structures are defined in terms of physical databases and logical relationships. These virtual structures are named as logical databases. They are a logical view of the stored database, and segments of these logical databases are materialized from underlying physical databases. A logical database definition is the same as view definitions in a relational system. The index database construct is provided for fast direct access to physical databases. The index database structure is composed of primary index and secondary index databases. Primary index databases provide direct access to record root segments. The index database record contains a key and a pointer to the indexed root segment. Secondary index databases provide fast direct access to a segment within a physical or logical database, by means of data within the segment, or some dependent segment. The segment to be accessed is called the target segment, and the search field is termed as the source segment. At most, one primary index database, and any number of secondary index databases may be associated with a given physical database. All physical databases connected by logical relationships, together with their associated index databases, form the database group.

IMS has four access methods to implement the previous data structures:

- SAM (Sequential Access Method);
- ISAM (Indexed Sequential Access Method);
- VSAM (Virtual Storage Access Method);
- OSAM (Overflow Sequential Access Method).

OSAM is utilized to supplement ISAM. IN OSAM records may be accessed sequentially or directly by relative byte number. Both fixed length unblocked and blocked records can be stored and accessed from a disk. The storage organizations in IMS use physical pointers. A physical pointer contains a four byte number, which is the relative number of a byte in an access method dataset. A physical pointer points to a dataset record or a byte sequence therein by specifying the relative byte number of the first byte of the record or sequence. The physical databases can be implemented by one of the following methods:

- HSAM (Hierarchical Sequential Access Method);
- HISAM (Hierarchical Indexed Sequential Access Method);
- HDAM (Hierarchical Direct Access Method);
- HIDAM (Hierarchical Indexed Direct Access Method);
- GSAM (Generalized Sequential Access Method);
- DEDB (Data Entry Data Base);
- MSDB (Main Storage Data Base).

Each access method has different performance and storage characteristics, and the user is given a choice to select an appropriate method for implementation of a specific database. In all implementations a segment is broken up into two parts: (1) a prefix part that contains a segment code and other implementation related information; and (2) the data part that contains the DA segment byte sequence.

PREFIX
DATA
CONTIGUOUS STORAGE
PREFIX
DATA
SEPARATE STORAGE

Figure 14. DA Segment Byte Sequence

The prefix and data parts are stored contiguously, unless the length of a variable length segment increases beyond the space originally allocated. In that event, the data part is stored separately, and linked to the prefix part by a physical pointer.

In HSAM, a physical database is implemented as a single SAM dataset with fixed length unblocked records. The segments of each logical database record are stored in hierarchic sequence in one or more consecutive physical database records. Thus the hierarchic sequence is represented by physical contiguity. The only operations that can be performed on an HSAM database are ISRT (Insert) (allowed only when a database is being built), and GU,GN,GNP (only for an existing database). Updating in a HSAM database is done by reading an existing version of a database and writing a new one. It is evident HSAM does not support direct access operations.

HISAM provides indexed access to root segments, and sequential access from roots to dependent segments. HISAM can be implemented via two methods: VSAM or ISAM/OSAM. The two implementation techniques are similar. The index for the roots is implemented via an ISAM dataset, and successive dependent segments are in an OSAM dataset for the ISAM/OSAM case. In the VSAM implementation the root index is in the way of a KSDS (Key Sequenced Data Set), and the dependent segments reside in an ESDS (Entry Sequenced Data Set). HISAM permits both sequential and direct access by root segment key. The implementation is efficient if the frequency of insertions and deletions is low. The use of OSAM/ESDS for overflow dependent segments permits variable length records, and physical partitioning of high usage and low usage data within a record.

Both HDAM and HIDAM use physical pointers to link segments. The pointers are of two types:

- Hierarchic pointers, each segment points to the next in sequence, to link all database segments in a hierarchic sequence. Both forward and backward pointers are allowed;
- Child-twin pointers, each parent points to it's first child of each type, and each child points to it's next twin. Again backward and forward pointers are allowed.

One or both type of pointers may be used to implement a record in a HD database. Hierarchic pointers provide access to a record in hierarchic sequence, and child-twin pointers provide access to any segment within a record.

HDAM provides hash access to the roots, and pointers from the roots to the dependent segments. Thus only direct access of roots is supported , and not sequential access. The primary and secondary datasets in HDAM can be implemented as ESDS or OSAM datasets. The primary dataset is partitioned into a root-addressable area, and an overflow area. Root segments are stored by a hash function in the root-addressable area, and the dependent segments are stored in one or more secondary datasets. Collisions in the root-addressable area are handled by chaining of roots which hash to the same address. HDAM is useful in a heavy insert/delete environment, and it provides fast direct access via root keys.

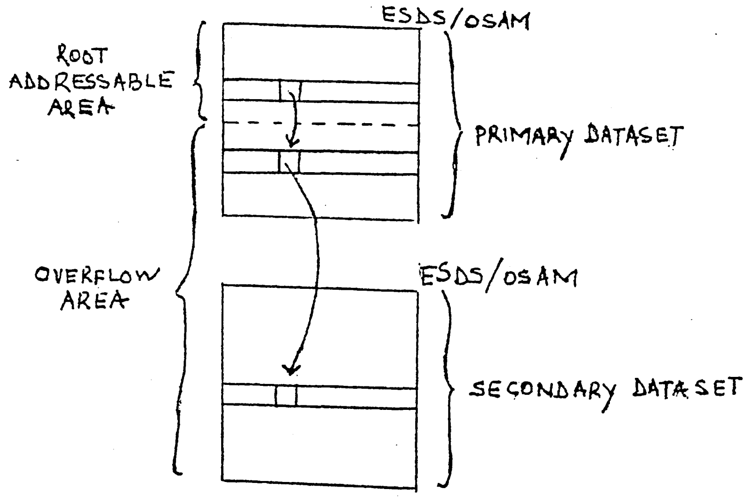


Figure 15. HDAM Implementation

In HIDAM there exists an indexed access to the roots (primary dataset), and pointer access to dependent segments (secondary dataset). The index to the roots in the primary dataset is provided via a VSAM dataset. The VSAM dataset records have pointers to the roots in the primary ESDS, which in turn have pointers to dependent segments in secondary ESDS. HIDAM is useful in heavy segment insertion/deletion environment, and when both sequential and direct access to the roots is necessary.

GSAM (Generalized Sequential Access Method) can be implemented with SAM datasets or with VSAM entry sequenced datasets. These are primarily used to implement root only databases with fixed length root segments. GSAM is structured to support data exchange between application programs, and other user programs which access SAM or entry sequenced VSAM files.

DEDBs (Data Entry Data Bases) are built for specialized applications in which a large number of key driven terminals enter data for later processing by batch programs. A record in this implementation is limited to a single root type, and a single dependent segment type. A root segment identifies a terminal, and dependents represent entries from that segment. DEDB utilizes VSAM and it is a slight modification of HDAM. Roots are hashed and dependents are chained via pointers. Dependent segments are sequenced by time of entry. Thus the last segment inserted is the first one on the chain from the root.

MSDB (Main Storage Data Base) is intended for those databases with very high access to roots and which can be held in main storage. A record is restricted to a single root segment type of fixed length. The root segments can be keyed via two methods. The first method is the normal one where a key is stored with the root. In the second method a 1:1 correspondence is set up between the logical terminals and the associated root segments. The first method is appropriate when many terminals need to access the same set of roots. The second method is suited in an environment where each terminal requires dedicated storage e.g. teller records. An in depth discussion of IMS storage structures is provided in McGee (29).

Discussion

The physical storage structures in IMS are far more advanced than those in Model 204. At the internal level Model 204 imitates the inverted list data model. The data is presented in an indexed format. In IMS the under-lying data structure is tree-like. Entry ordered, reuse-direct-file-space, sorted and hashed files are the secondary file types supported by Model 204. The primary access method in Model 204 is IFAM, while as IMS utilizes either HSAM, HISAM, HDAM, HIDAM, GSAM, DEDB or MSDB. Due to the variety of access methods available under IMS customized applications can be developed. If an application desires heavy insertions/deletions then HDAM may be used, or if the frequency of insertions/deletions is low then HISAM may be used. If an application has a high rate of updates and and requires both sequential and direct access to root segments then HIDAM may be used. DEDB react to specific data entry terminals. Hashed and sorted access is the only variation provided in Model 204. The access method is generalized for all application types. Model 204 is said to be relational because the user has no notion of the underlying storage structure. At the external and conceptual level data may be viewed in a tabular format. The data definition phase requires the keys to be defined so that the field indexes may be set up. After the data definition phase no mention is made of indexes. This leads to possible inefficient retrievals. An application programmer may be retrieving data via a non-indexed field. Such a situation will not arise in IMS, as access is provided in a limited manner via root segments, which are indexed. At the physical level Model 204 imitates the inverted list data model. This model is relational, because the term relational applies only at the external and conceptual levels and not at the internal level. The applications developed in IMS are extremely data dependent. Once a physical database has been set up with the hierarchic tree structure, it needs redefinition to make any changes. The data definition language in Model 204 provides the REDEFINE command to set up separate indexes on existing data. This is a useful feature as user requirements change with time, and data may need to be keyed via a new field. The REDEFINE command sets up a new index based on the mentioned key field. The whole process remains transparent to the user. Such a change would require a redefinition of the physical database in IMS. Both Model 204 and IMS support logical relationships. The INVISIBLE attribute in Model 204 allows logical relationships between physical records. The Data Base Group in IMS can directly support logical relationships.

CHAPTER VIII

SUMMARY AND CONCLUSIONS

An on-line classroom reservation system was developed on Model 204. The system utilizes an algorithm which uses set theory operations to make room reservations. The algorithm has a polynomial processing order. It takes $O(n^{k+1})$ to process a request, where 'n' is the total number of tuples in each of the located sets in the query, and 'k' is the total number of intersections and joins. Model 204 does not directly relate to any specific data model. At the physical level it has an inverted file access method, thus it is assumed to be an inverted list database. But at the conceptual and external levels it imitates the relational data model. In Model 204 data manipulation is set-at-a-time similar to relational systems, and no inverted list data manipulation operators are provided. The requirements for a system to be called 'truly relational' are extremely stringent, and such a system does not exist today. Model 204 is termed relational-like because it deviates from the data model. The attributes and tuples are ordered in a Model 204 file. A field can have multiple values in a record. Primary keys, foreign keys, and the associated entity and referential integrity rules are not supported. User language does not directly provide the select, project and join operators. In spite of these drawbacks Model 204 provides a relational interface to the user. The data definition is simpler in Model 204 than in IMS. In IMS the data is set up in a tree structured format, and a fair amount of work goes into deciding the content of segments, and the hierarchical layout of segments. In Model 204 the data is set up in a tabular format, which is conceptually simpler. A physical database in IMS supports only one primary key at the root. Any number of key fields may be defined for a Model 204 record. The data definition in IMS addresses both physical placement and access paths to the data. This data dependency does not occur in data definitions under Model 204.

IMS and Model 204 have completely different data manipulation operators, primarily because of the underlying data models involved. IMS does not have a fourth generation language interface like Model 204. One of the main differences is that IMS supports record-at-a-time processing, and Model 204 allows for set-at-a-time processing. The retrieval operations in IMS query an ordered set of trees in a hierarchic sequence. There is no ordered sequence that a user language request needs to adhere to. DL/I does not provide flow of control constructs which are available in user language. The set-level looping in user language provides loop avoidance, thus making the application programmer's task easier.

Physical data independence is guaranteed by the nature of the relational data model. The relational model is at the external/conceptual level of the ANSI/SPARC database architecture model, thus any changes at the internal level do not affect the applications. The hierarchic systems are data dependent as they are at the internal level of the ANSI/SPARC model. A change in the access mechanisms do not require major changes in application programs under IMS or Model 204. Thus the degree of physical data independence under the two systems is identical. Model 204 has a high degree of logical data independence. A field can be added or deleted from an existing record in a Model 204 file, without affecting existing application programs. New record types may also be introduced without requiring any modifications. IMS is data dependent as additions or deletions of fields from segments require modification of application programs, and a re-definition of the database. This dependency is called access path dependence in IMS, and it occurs due to the pre-defined access paths in application programs. IMS provides for both referential and entity integrity, but Model 204 proves to be lacking in this area. Most integrity checks are done by user written procedural code in Model 204. Entity and referential integrity checks are not provided by Model 204 as it does not support primary and foreign keys. File level security features are more advanced in Model 204 as compared to IMS. IMS depends on file level security on another package RACF.

The physical storage structures in IMS are far more advanced than those in Model 204. The primary access method in Model 204 is IFAM, while as IMS utilizes HSAM, HISAM, HDAM, HIDAM, GSAM, DEDB and MSDB. Due to the variety of access methods available under IMS, customized applications can be developed. Applications in Model 204 may perform inefficient retrievals as data may be accessed via a non-indexed field. Such a situation will not arise in IMS as access is provided in a limited manner via root segments, which are indexed. The applications developed in IMS are highly data dependent as compared to those in Model 204.

Both Model 204 and IMS provide host language interfaces. Model 204 provides an added capability in way of a fourth generation language, user language. The development time for an application is reduced under Model

204. IMS may finally provide better run time performance because applications can be customized, and efficient access paths to the tree structured-data reduce the retrieval costs. The customized system may not readily accept other applications. The logical data structures in IMS are biased towards some applications and against others, because they closely reflect the physical data structure. The logical data structures in IMS application programs (e.g PL/I structures) closely reflects the tree-like IMS database organization. A tree-like data structure is biased towards applications which require only one primary key field. A table supports applications with any number of key attributes. The tables I through IV and Figures 16 and 17 summarize the comparative study of the two database systems.

This study exposed the data definition, data manipulation, data independence, storage organization, and data protection capabilities of IMS and Model 204. The classroom reservation system developed in Model 204 not only provided a common application problem for the study, it also made available a reservation system which can be used by the registrar's office, and all departments at Oklahoma State University. Future work in this area can involve studying the involved systems at an internal level of the ANSI/SPARC data model. The database architectures involved can be compared, and a set of benchmarks can be generated to compare the internal levels of IMS and Model 204.

TABLE I

DATA DEFINITION

	IMS	MODEL 204
Data formats	tree-like	tabular
Primary data object	segment	record
Dynamic indexing	not available	available
Logical relationship	pointer chains	pseudo foreign keys

TABLE II

DATA MANIPULATION

	IMS	MODEL 204
Host language interface	available	available
Fourth generation language	not available	available
Query processing	record-at-a-time	set-at-a-time
Database ordering	hierarchic sequence	unordered
Flow of control constructs	not available	available
Selection power of DML	low	high
non-keyed, non-indexed access	not possible	possible
Database query	execution	compilation & execution

TABLE III

DATA INDEPENDENCE

	IMS	MODEL 204
Degree of physical data independence	high	high
Degree of logical data independence (growth)	low	high
Degree of logical data independence (restructuring)	low	low

TABLE IV

DATA INTEGRITY AND SECURITY

	IMS	MODEL 204
Concurrency control	available	available
Exclusive access	segment	record
Primary keys	available	not available
Foreign keys	available	not available
Entity integrity checks	available	not available
Referential integrity checks	available	not available
Concurrency control statements in DML	not available	available
Audit trail	available	available
File level security	low	high

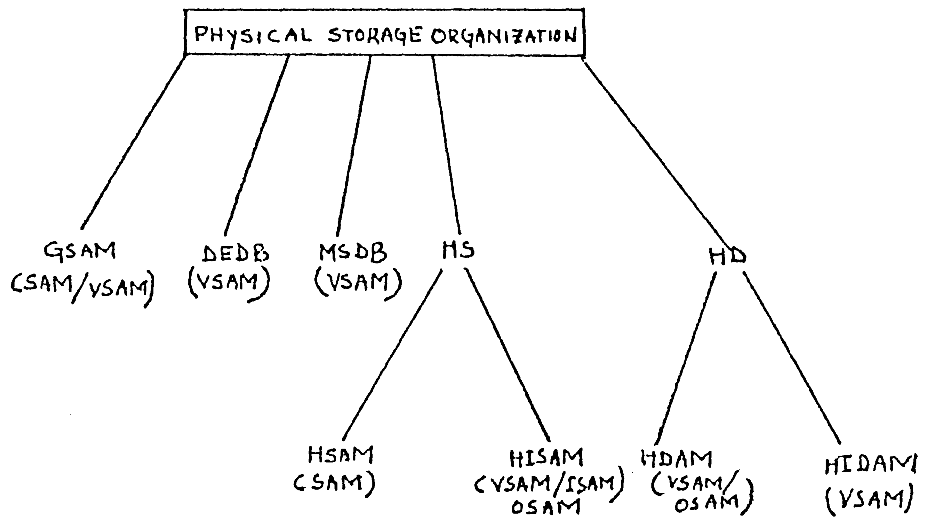


Figure 16. Storage Organizations and Access Methods in IMS

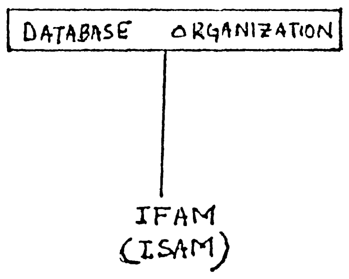


Figure 17. Storage Organizations and Access Methods in Model 204

SELECTED BIBLIOGRAPHY

1. Almond, Mary. An algorithm for constructing university timetables. Comput. J. 8, 1(January 1966), 331-340.

2. Barraclough, Elizabeth D. The application of a digital computer to the construction of timetables. Comput. J. 8, 1(April 1965), 136-164.

3. Brittan, J.N.G., Farley, F.J.M. College timetable construction by computer. Comput. J. 14, 4(November 1971), 361-365.

4. Cardenas, A.F. Evaluation and selection of file organization model and system. Commun. ACM 16, 9(Sept 1973), 540-548.

5. Cardenas, A.F. Performance analysis of inverted database structures Commun. ACM 18, 5(May 1975), 253-263.

6. Chamberlin, Donald D. Relational data-base management systems. Computing Surveys 8, 1(March 1976), 43-66.

7. Christodoulakis, S. Implications on certain assumptions in database performance evaluation. ACM TODS 9, 2(June 1984), 163-186.

8. Codd, E.F. A relational model for large shared data banks. Commun. ACM 13, 6(June 1970), 377-387.

9. Codd, E.F. Relational database: a practical foundation for productivity. Commun. ACM 25, 2(February 1982), 109-117.

10. Computer Corporation of America. Model 204 DBMS reference series, Command reference manual, release 8.1, September 1985.

11. Computer Corporation of America. Model 204 DBMS reference series, File Manager's Guide, release 8.1, September 1985.

12. Computer Corporation of America. Model 204 DBMS reference series, System Manager's Guide, release 7.1, April 1985.

13. Computer Corporation of America. Model 204 DBMS reference series, System messages manual, release 8.1, September 1985.

14. Computer Corporation of America. Model 204 DBMS reference series, Terminal user's guide, release 7.1, April 1985.

15. Computer Corporation of America. Model 204 DBMS reference series, User language manual, release 8.1, September 1985.

16. Csima, J., Gotlieb, C.C. Tests on a computer method for constructing school timetables. Commun. ACM 7, 3(March 1964), 160-163.

17. Date, C.J. An introduction to database systems. Reading, Massachusetts: Addison Wesley Publishing Company, Vol 1, 4th edition, April 1986.

18. Fry, James P., Sibley, E.H. Evolution of data-base management systems . Computing Surveys 8, 1(March 1976), 7-42.

19. Gosselin, Karl., Truchon, Michel. Allocation of classrooms by linear programming. J. Opl Res. Soc. 37, 6(June 1986), 561-569.

20. IBM, information management system / virtual storage (IMS / VS), Application programming reference manual, SH20-9026-2,1975.

21. IBM, information management system / virtual storage (IMS / VS), General information manual, GH20-1260-3,1975.

22. IBM, information management system / virtual storage (IMS / VS), System/application design guide, SH20-9025-2,1975.

23. IBM, information management system / virtual storage (IMS / VS), System programming reference manual, SH20-9027-2,1975.

24. Kim, Won. Relational database systems. Computing surveys 11, 3(September 1979), 185-211.

25. Lions, John. Matrix reduction using the Hungarian method for the generation of school timetables. Commun. ACM 9, 5(May 1966), 349-354.

26. Lions. John. The Ontario school scheduling program. Comput. J. 10 ,2(August 1967), 14-21.

27. McFadden, Fred R., Hoffer, Jeffrey A. Database Management. California: The Benjamin/Cummings Publishing Co., 1985.

28. McGee, William C. On user criteria for data model evaluation. ACM TODS 1, 4(Dec 1976), 370-387.

29. McGee, W.C. The information management system IMS/VS. IBM systems journal 16, 2 (1977), 84-168.

30. Michaels, Ann S., Mittman, Benjamin., Carlson, Robert C. A comparison of relational and CODASYL approaches to database management. Computing Surveys 8, 1(March 1976), 124 - 151.

31. Siler, K.F. A stochastic evaluation model for database organization in data retrieval systems. Commun. ACM 19, 2(Feb. 1976), 84-95.

32. Stonebraker, M., Woodfill, J., Ranstrom, J., Murphy, M., Meyer, M., Allman, E. Performance enhancements to a relational database system. ACM TODS 8, 2(June 1983), 167-185.

33. Tsichritzis, D.C., Lochovsky, F.H. Hierarchical database management - a survey. Computing Surveys 8, 1(March 1976), 104 - 123.

34. University Computer Center (Oklahoma State University), User manual Third edition, November 1985.

35. Wiederhold, Gio. Database design. New York: McGraw Hill Book Co. Inc., 1977.

APPENDIX A

GLOSSARY OF TERMS

ANSI/SPARC	American National Standards Institute / Systems Planning and Requirements Committee
AP	Application Programmer: a data structure class in IMS
DA	Data Administrator: a data structure class in IMS
DBD	Data Base Definition: a definitional construct in IMS
DEDB	Data Entry Data Base
DL/I	Data Language / One: host language interface calls in IMS
DLET	Delete: DL/I operator
ESDS	Entry Sequenced Data Set for VSAM
FCT	File Control Table: a file section in a Model 204 file
GHN	Get Hold Next: DL/I operator
GHNP	Get Hold Next within Parent: DL/I operator
GHU	Get Hold Unique: DL/I operator
GN	Get Next: DL/I operator
GNP	Get Next within Parent: DL/I operator
GSAM	Generalized Sequential Access Method
GU	Get Unique: DL/I operator
HD	Hierarchical Direct
HDAM	Hierarchical Direct Access Method
HIDAM	Hierarchical Indexed Direct Access Method
HISAM	Hierarchical Indexed Sequential Access Method
HS	Hierarchical Sequential
HSAM	Hierarchical Sequential Access Method
IFAM	Inverted File Access Method
IMS	Information Management System: vendor IBM
ISAM	Indexed Sequential Access Method
ISRT	Insert: DL/I operator
KSDS	Key Sequenced Data Set for VSAM
MODEL 204	Relational Database: vendor Computer Corp. of America
MSDB	Main Storage Data Base
OSAM	Overflow Sequential Access Method
PCB	Program Communication Block: a definitional construct in IMS
PF	Program Function
RACF	Resource Allocation and Control Facility
RDFS	Reuse Direct File Space: a file type in Model 204
REPL	Replace: DL/I operator
SAM	Sequential Access Method
SSA	Segment Search Argument
TABLE A	Dictionary of field names in a Model 204 file
TABLE B	Data section of a Model 204 file
TABLE C	Indexing section of a Model 204 file
TABLE D	Indexing section of a Model 204 file
USER LANGUAGE	Fourth Generation Language in Model 204
VSAM	Virtual Storage Access Method

APPENDIX B

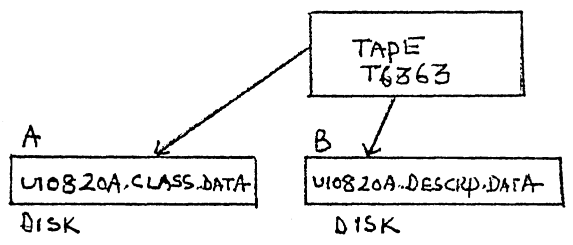
CLASS RESERVATION SYSTEM PROGRAMS

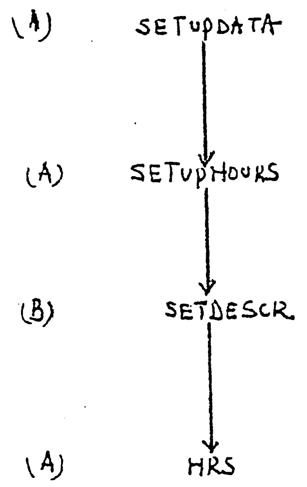
DATA ABSTRACTION

The class room reservation system database was set up using a tape dump of an IMS course database. The tape consists of a hierarchical sequence of segments. The tape dataset attributes are:

```
DSN = (RECFM=VB, LRECL=1004, BLKSIZE=8000, DEN=3),
VOL=SER=T6363
```

The tape was copied onto a two disk datasets to perform the abstraction. The two disk datasets are: u10820a.class.data; u10820a.descrp.data. The hierarchy chart lists the procedure calls for the abstraction process.





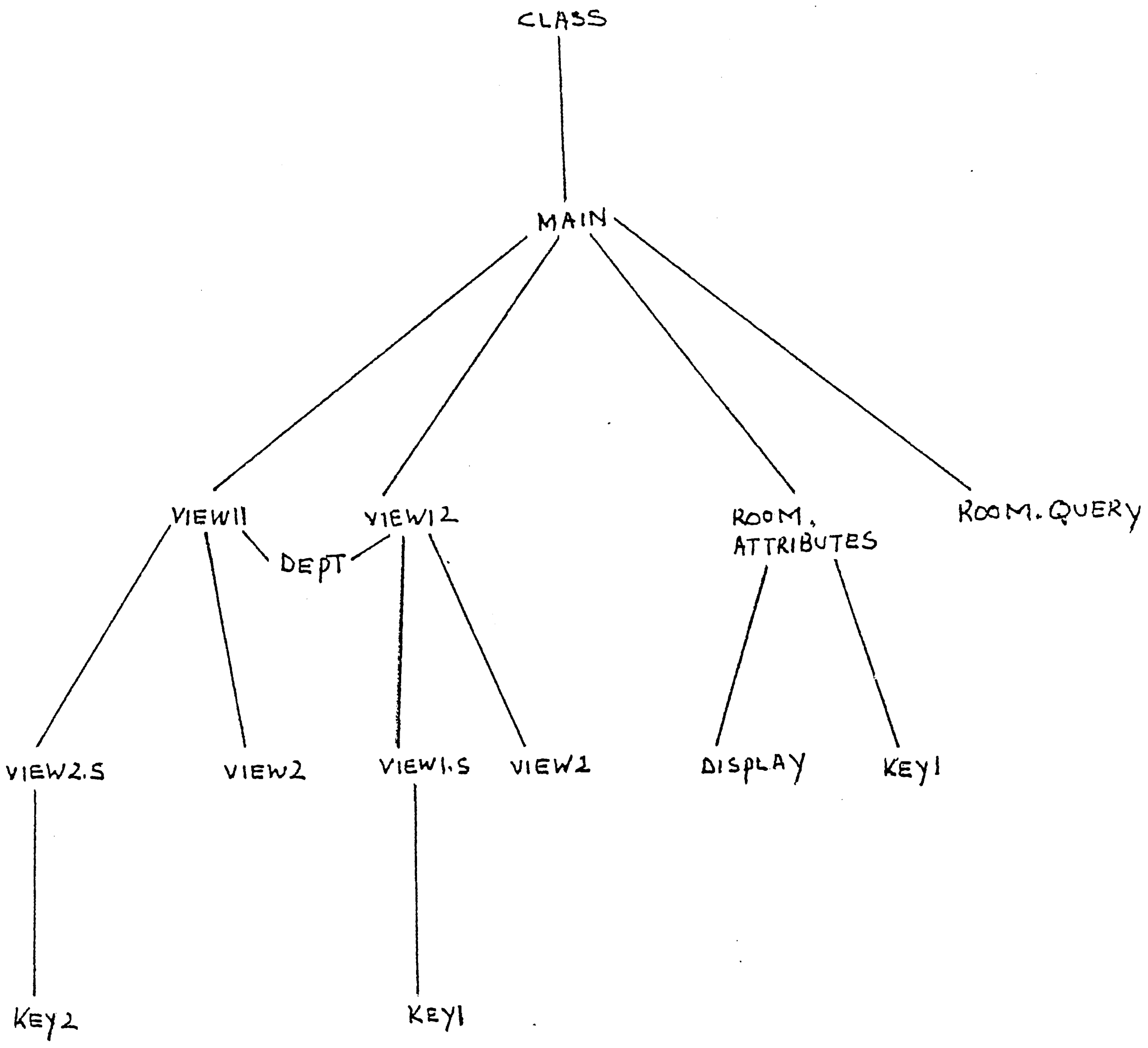
HIERARCHY CHART

The pcb definitions for the course and meeting segments of the IMS database are:

```
                    COURSE
                          12 crsid                    picture x(5).
                          12 crskey.
                             16 crssem.
                                20 year               picture xx.
                                20 semester           picture x.
                             16 crsname.
                                20 dept               picture x(5).
                                20 course-number      picture x(4).
                                20 type-crs           picture x.
                                   88 theory               value '1'
                                   88 discussion           value '2'
                                   88  independent-study     value   '3'
                                   88 lab                  value '4'
                                20 sec                 picture x(3).
                          12 filler                    picture x(50).



                    MEETING
                          12 seq                      picture x(3).
                          12 meeting-time-place
                             16 meeting-time
                                20 days
                                   24 mon             picture x.
                                   24 tue             picture x.
                                   24 wed             picture x.
                                   24 thr             picture x.
                                   24 fri             picture x.
                                   24 sat             picture x.
                                20 begin-time
                                   24 b-hr            picture x(2).
                                   24 b-min           picture x(2).
                                20 end-time
                                   24 e-hr            picture x(2).
                                   24 e-min           picture x(2).
                       16 meeting-time-free-form redefines meeting-time.
                                   20 indicator           picture x.
                                   20  meeting-time-ff    picture  x(13).
                       16 meeting-place
                                20  bldg                      picture  x(4).
                                20  room                      picture  x(4).
                       12  instructor                        picture  x(12).
                       12  filler                            picture  x(50).
```



HIERARCHY CHART

```
//U10820A    JOB (10820,204-BD-FILE),VINIT,TIME=(0,40),
//      MSGCLASS=X,MSGLEVEL=(2,0),CLASS=A,NOTIFY=*
/*PASSWORD ????
//       EXEC BATCH204
//******************************************************************
//*  PURPOSE - THIS BATCH ROUTINE WAS USED TO BUILD THE CLASS SCHEDULE
//*            DATABASE FILE.  THE FILE WAS INITIALLY ALLOCATED WITH
//*            THE NAME M204.ACT10820.DATA.  THE BATCH204 PROGRAM IS
//*            EXECUTED TO SET UP THE FILE AND DEFINE THE FIELDS.
//******************************************************************
//DATA DD DSN=M204.ACT10820.DATA,DISP=SHR
//CCAIN DD *
PAGESZ = 6184
CREATE FILE DATA
PARAMETER ASTRPPG = 384,ATRPG = 1,
PARAMETER FVFPG = 1,MVFPG = 1,BRECPPG = 47
PARAMETER BSIZE = 51,BRESERVE = 100
PARAMETER FOPT=X'08',FILEORG=X'04'
PARAMETER PDSTRPPG=256,PDSIZE=3
PARAMETER CSIZE = 5,DSIZE = 30
END
OPEN DATA
INITIALIZE
DEFINE   COURSE.ID (KEY)
DEFINE   DEPARTMENT (KEY)
DEFINE   COURSE.NO (KEY)
DEFINE   COURSE.TYPE
DEFINE   COURSE.DESCRP
DEFINE   INSTRUCTOR (KEY)
DEFINE   BUILDING (KEY)
DEFINE   ROOM (KEY)
DEFINE   SECTION
DEFINE   COURSE.DAYS
DEFINE   COURSE.BEGIN
DEFINE   COURSE.END
DEFINE   FREEFORM
DEFINE   MIN
DEFINE   MAX
DEFINE   START (KEY)
DEFINE   EXPIRE (KEY)
DEFINE   COMMENT
DEFINE   NUMBER
CLOSE DATA
EOJ
/*
//
--------------------------------------------------------------------
//U10820A JOB (10820,M24-TO-TAPE),VINI,TIME=(0,40),USER=*,
//      MSGCLASS=X,MSGLEVEL=(1,1),CLASS=A,NOTIFY=*
/*PASSWORD ???
/*JOBPARM ROOM=E
//**********************************************************
//* PURPOSE - THIS PROCEDURE IS USED TO BACKUP THE CLASS
//*           SCHEDULE DATABASE ONTO TAPE.  THE TAPE NUMBER
//*           AND THE LABEL NUMBER ON THE TAPE NEED TO BE
//*           FILLED UP.
//**********************************************************
/*MESSAGE PLEASE MOUNT Txxxxx
//STEP1 EXEC M204FLOD,REGION=3000K
//DATA DD DSN=M204.ACT10820.DATA,DISP=SHR
//DUMPDATA DD DSN=M204.DUMP.DATA,UNIT=TAPE,
//    VOL=(PRIVATE,RETAIN,,,SER=(T1xxxx)),
//    DISP=(NEW,KEEP,DELETE),
//    LABEL=xx
//CCAIN DD *
PAGESZ = 6184
OPEN DATA
DUMP TO DUMPDATA
CLOSE DATA
EOJ
//STEP2 EXEC PRTLBS,VOL=Txxxxx
//
-----------------------------------------------------------------
//U10820A    JOB (10820,DUM-PM-2040),VINI,TIME=(0,40),                 00000100
//      MSGCLASS=X,MSGLEVEL=(1,1),CLASS=A,NOTIFY=*,USER=*              00000200
/*PASSWORD ????                                                        00000300
/*JOBPARM ROOM=E                                                       00000400
/*MESSAGE PLEASE MOUNT TXXXXX--NFP
//**********************************************************
//* PURPOSE - THIS PROCEDURE IS USED TO RESTORE THE CLASS
//*           SCHEDULE DATABASE FROM TAPE.  THE TAPE NUMBER
//*           AND THE LABEL NUMBER ON THE TAPE NEED TO BE
//*           FILLED UP.
//*           m204.act10820.data needs to be allocated with
//*           the same attributes as those in chapter 2 of
//*           the thesis prior to running this job.
//**********************************************************
//STEP1 EXEC M204FLOD,REGION=3000K                                     00000500
//DATA  DD DSN=M204.ACT10820.DATA,DISP=SHR                             00000600
//DUMPDATA DD DSN=M204.DUMP.DATA,UNIT=TAPE,                            00000700
//   VOL=(PRIVATE,RETAIN,,,SER=(Txxxxx)),DISP=(NEW,KEEP),LABEL=xx
//CCAIN DD *                                                           00010900
PAGESZ = 6184
CREATE FILE DATA
END
OPEN DATA
INITIALIZE
RESTORE 128 FROM DUMPDATA
CLOSE DATA
EOJ                                                                    00011270
/*                                                                     00011270
//                                                                     00011300
-----------------------------------------------------------------
*****************************************************************
* FILENAME CLASS- THIS SETS UP SUCCESSIVE CALLS FOR THE MAIN DRIVER
* THE USER CAN QUIT THE PRIMARY MENU BY PRESSING THE ATTENTION KEY
*****************************************************************
INCLUDE MAIN
******************************************************************
* FILENAME MAIN
*                                                   AUTHOR - VINIT  VERMA
*                                                   CREATED 11-29-86
* THIS PROCEDURE IS A DRIVER FOR THE CLASS SCHEDULE SYSTEM
* THE PROGRAM SENSES THE MENU SELECTION AND APPROPRIATELY
* CALLS THE VIEW11, VIEW12, ROOM.ATTRIBUTES OR ROOM.QUERY
* PROGRAMS.  THE PA1 KEY IS NOT DEACTIVATED FOR PROGRAMMING
* PURPOSES.  THE LOG_OUT PROCEDURE ALLOWS A USER TO EXIT
* MODEL 204.
******************************************************************

RESET MSGCTL = X'02'
OPENC DATA
RESET MSGCTL = X'01'
UTABLE LSTBL = 5000
UTABLE LQTBL = 1200
UTABLE LVTBL = 200
UTABLE LNTBL = 100

BEGIN
*----------------------------------------------------------------*
MENU MAIN
TITLE ' OSU CLASS SCHEDULE SYSTEM ' AT 21
SKIP 4 LINE
PROMPT '        GENERAL USER (DEPARTMENTS)' AT 15
SKIP 1 LINE
PROMPT '        SUPER USER (REGISTRAR)' AT 15
SKIP 1 LINE
PROMPT '        ROOM INVENTORY DATABASE' AT 15
SKIP 1 LINE
PROMPT '        ROOM VACANCY' AT 15
SKIP 1 LINE
PROMPT '        EXIT' AT 15
END MENU
120. READ MENU MAIN
130. IF $SETG('SELECTION',%MAIN:SELECTION) THEN STOP
END

******************************************************************
* MAKE A CHOICE DEPENDING ON THE SELECTION
******************************************************************

IF SELECTION=1,VIEW11
IF SELECTION=2,VIEW12
IF SELECTION=3,ROOM.ATTRIBUTES
IF SELECTION=4,ROOM.QUERY
IF SELECTION=5,LOG_OUT
---------------------------------------------------------
*****************************************************
* FILENAME VIEW11
*                                        AUTHOR- VINIT VERMA
*                                        CREATED 10-23-86
* PURPOSE - THIS PROCEDURE PROVIDES THE GENERAL USER'S
* OPTION FOR THE DEPARTMENTS.  NO UPDATES CAN BE
* PERFORMED VIA THIS PROCEDURE.  ONLY DATA
* RETRIEVAL OPERATIONS MAY BE PERFORMED.
* THE QUERY FIELDS ARE -
*
*          .COURSE.ID
*           DEPARTMENT AND COURSE NUMBER
*           DEPARTMENT
*           BUILDING AND ROOM
*           INSTRUCTOR'S NAME
*
* THE FILES INCLUDED BY THIS PROCEDURE ARE
*
*             VIEW2.S - THE USER INTERFACE SCREEN
*             VIEW2 - THE DATA DISPLAY PROCEDURE
*
*****************************************************

RESET MSGCTL = X'02'
O DATA

BEGIN

***************************************************
*       THE INDEX ARRAY IS SET UP TO DO THE SCROLLING
***************************************************

%INDEX IS STRING LEN 7 ARRAY(200)
%SCAN IS FIXED
%COUNT IS FIXED

************************************************
*  THE SCREEN DEFINITION
************************************************

IN SCHED INCLUDE VIEW2.S
IN DATA INCLUDE DEPT

*********************************************************
*      SET UP THE ON CONFLICT CONDITION
*********************************************************

 ON FIND CONFLICT

   %CLASS:MESSAGE = '*** PLEASE TRY LATER ***'
   JUMP TO START

END ON
*********************************************************
```

```
*     DEACTIVATE THE ON ATTENTION KEY
****************************************************************

 ON ATTENTION

   %CLASS:MESSAGE = '*** ATTENTION KEY DEACTIVATED ***'
   JUMP TO START

END ON

*-------------------------------------------*
*  THIS IS THE STARTING POINT FOR THE PROGRAM
*-------------------------------------------*

START: READ SCREEN CLASS

%CLASS:MESSAGE = ' '

**********************************************
*          VERIFY THE PFKEY VALUES
**********************************************

IF %CLASS:PFKEY /= 3 AND %CLASS:PFKEY /= 5 AND %CLASS:PFKEY /= 6 AND
   %CLASS:PFKEY /= 7 AND %CLASS:PFKEY /= 8 AND %CLASS:PFKEY /= 9 THEN

    %CLASS:MESSAGE = '*** PF' WITH %CLASS:PFKEY WITH '  IS NOT ACTIVE'
    JUMP TO START
END IF

**********************************************************************
*     CHECK FOR THE PF3 EXIT KEY
**********************************************************************

IF %CLASS:PFKEY = 3 THEN

   JUMP TO FINISH

END IF

**********************************************************************
*     CHECK FOR THE PF9 DEPARTMENT QUERY KEY
**********************************************************************

IF %CLASS:PFKEY = 9 THEN

   CALL DEPT
   JUMP TO START

END IF

**********************************************************************
*     CHECK FOR THE PF6 REFRESH KEY
**********************************************************************

IF %CLASS:PFKEY = 6 THEN

   PREPARE SCREEN CLASS
   JUMP TO START

END IF
**********************************************************************
*     CHECK FOR THE PF5 REFRESH KEY
**********************************************************************

IF %CLASS:PFKEY = 5 THEN

CLEAR LIST SCROLL

**********************************************************************
*     CHECK IF THE QUERY IS KEYING OFF THE COURSE NAME AND NUMBER
*********************************************************************

IF %CLASS:ITEMID = 3 THEN


NUMBER: IN DATA FD DEPARTMENT  = %CLASS:DEPARTMENT AND -
        COURSE.NO = %CLASS:COURSE.NO AND COMMENT IS NOT PRESENT
END FIND

CT.NUMBER: COUNT RECORDS IN NUMBER

IF COUNT IN CT.NUMBER EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF
 PLACE RECORDS IN NUMBER ON LIST SCROLL

**********************************************************************
*     CHECK IF THE QUERY IS KEYING OFF THE COURSE ID NUMBER
*********************************************************************

ELSEIF %CLASS:ITEMID = 1 THEN


NUMBER1: IN DATA FD COURSE.ID  = %CLASS:COURSE.ID
END FIND

CT.NUMBER1: COUNT RECORDS IN NUMBER1

IF COUNT IN CT.NUMBER1 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER1 ON LIST SCROLL

******************************************************************
*     CHECK IF THE QUERY IS KEYING OFF FOR A CERTAIN DEPARTMENT
*****************************************************************

ELSEIF %CLASS:ITEMID = 2 THEN


NUMBER2: IN DATA FD DEPARTMENT = %CLASS:DEPARTMENT AND -
         COMMENT IS NOT PRESENT
END FIND

CT.NUMBER2: COUNT RECORDS IN NUMBER2

IF COUNT IN CT.NUMBER2 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER2 ON LIST SCROLL
******************************************************************
*     CHECK IF THE QUERY IS KEYING OFF FOR A CERTAIN ROOM
*****************************************************************

ELSEIF %CLASS:ITEMID = 5 THEN


NUMBER3: IN DATA FD ROOM = %CLASS:ROOM AND BUILDING = %CLASS:BUILDING
END FIND

CT.NUMBER3: COUNT RECORDS IN NUMBER3

IF COUNT IN CT.NUMBER3 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER3 ON LIST SCROLL
******************************************************************
*     CHECK IF THE QUERY IS KEYING OFF FOR AN INSTRUCTOR
*****************************************************************

ELSEIF %CLASS:ITEMID = 6 THEN


NUMBER4: IN DATA FD INSTRUCTOR = %CLASS:INSTRUCTOR
END FIND

CT.NUMBER4: COUNT RECORDS IN NUMBER4

IF COUNT IN CT.NUMBER4 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER4 ON LIST SCROLL

END IF

%COUNT = 0

ERR: COUNT RECORDS ON LIST SCROLL

IF COUNT IN ERR GT 200 THEN
   %CLASS:MESSAGE = '*** OVER 200 RECORDS TO SCROLL ***'
   JUMP TO START
END IF

 FOR EACH RECORD ON LIST SCROLL

    %COUNT = %COUNT + 1
    %INDEX(%COUNT) = $CURREC

END FOR

****************************************************************
*      SET UP THE CATENATED SCAN POINTER
****************************************************************

%SCAN = 1

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

              %CLASS:DEPARTMENT = %DEPARTMENT
             %CLASS:COURSE.NO = %COURSE.NO

CALL DISPLAY

JUMP TO START
```

```
END IF

************************************************************
*    CHECK FOR THE PF8 SCROLL FORWARD KEY
************************************************************

IF %CLASS:PFKEY = 8 THEN

 %SCAN = %SCAN + 1
IF %SCAN LE %COUNT THEN

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

              %CLASS:DEPARTMENT = %DEPARTMENT
             %CLASS:COURSE.NO = %COURSE.NO

 CALL DISPLAY
ELSE
    %CLASS:MESSAGE= '*** LAST RECORD ***'
    %SCAN = %SCAN - 1
END IF
 JUMP TO START

 END IF
************************************************************
*    CHECK FOR THE PF7 SCROLL FORWARD KEY
************************************************************

IF %CLASS:PFKEY = 7 THEN

 %SCAN = %SCAN - 1
IF %SCAN GE 1 THEN

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

              %CLASS:DEPARTMENT = %DEPARTMENT
             %CLASS:COURSE.NO = %COURSE.NO

 CALL DISPLAY
ELSE
     %CLASS:MESSAGE = '*** FIRST RECORD ***'
     %SCAN = %SCAN + 1
END IF
 JUMP TO START

 END IF
FINISH: PREPARE SCREEN CLASS
        COMMIT
****************************************************************
* INCLUDE THE SCROLL.LIST FILE
****************************************************************
IN SCHED INCLUDE VIEW2

END
O SCHED
RESET MSGCTL = X'01'
----------------------------------------------------------------
***************************************************************
* FILENAME VIEW2.S
*                                                AUTHOR - VINIT VERMA
*                                                CREATED - 10-23-86
* PURPOSE - THIS PROCEDURE PROVIDES THE USER INTERFACE SCREEN
* FOR THE GENERAL USER'S OPTION.  THIS PROCEDURE IS INVOKED
* BY THE VIEW11 PROGRAM.
*
***************************************************************

SCREEN CLASS
TITLE '------------------ Oklahoma State University Class Schedule ------
MAX PFKEY 12
PROMPT MESSAGE  AT 40 DEFAULT ' '
PROMPT 'id number      :' AT 2 INPUT COURSE.ID LEN 6 ITEMID 1 -
PROMPT 'instructor   :' AT 25 INPUT INSTRUCTOR LEN 13 ITEMID 6 -
PROMPT 'days  :' AT 53 INPUT COURSE.DAYS LEN 6 -
PROMPT 'min:' AT 68 INPUT MIN LEN 2
PROMPT 'department     :' AT 2 INPUT DEPARTMENT LEN 6 ITEMID 2 -
PROMPT 'building     :' AT 25 INPUT BUILDING LEN 5 ITEMID 4 -
PROMPT 'start :' AT 53 INPUT COURSE.BEGIN LEN 4 -
PROMPT 'max:' AT 68 INPUT MAX LEN 2
PROMPT 'course number :' AT 2 INPUT COURSE.NO LEN 5 ITEMID 3 -
PROMPT 'room         :' AT 25 INPUT ROOM LEN 5 ITEMID 5 -
PROMPT 'end   :' AT 53 INPUT COURSE.END LEN 4
PROMPT 'type           :' AT 2 INPUT COURSE.TYPE LEN 2 -
PROMPT 'section      :' AT 25 INPUT SECTION LEN 3 -
PROMPT 'hours :' AT 53 INPUT COURSE.HOURS LEN 2
PROMPT 'title          :' AT 2 INPUT COURSE.DESCRP LEN 22 -
PROMPT 'undecided hour   :' AT 45 INPUT FREEFORM LEN 14
PROMPT 'begin(yy-mm-dd):' AT 19 INPUT START LEN 8 -
PROMPT 'expiry(yy-mm-dd):' AT 45 INPUT EXPIRE LEN 8
PROMPT 'comment:' AT 2 INPUT COMMENT1 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT2 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT3 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT4 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT5 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT6 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT7 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT8 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT9 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT10 LEN 60
PROMPT '          ' AT 2 INPUT COMMENT11 LEN 60
PROMPT '          ' AT 2 INPUT COMMENT12 LEN 60
PROMPT '          ' AT 2 INPUT COMMENT13 LEN 60
PROMPT '          ' AT 2 INPUT COMMENT14 LEN 60
PROMPT 'pf3 exit  pf5 refresh  pf-6 clear  pf-7 <-  pf-8 ->
        pf9 dept. query
DEFAULT CURSOR COURSE.ID
END SCREEN
----------------------------------------------------------------------
**********************************************************************
* FILENAME DEPT
*                                              AUTHOR - VINIT VERMA
*                                              CREATED  01-10-87
* PURPOSE
* THIS SUBROUTINE LISTS ALL COURSES WITH MEETING TIMES FOR A SPECIFIC
* DEPARTMENT.  THE LIST IS SORTED BY DAY OF THE WEEK AND TIME OF THE
* DAY.  THE VIEW11 PROGRAM INVOKES THEIS SUBROUTINE.
*
*
**********************************************************************

DEPT: SUBROUTINE


A: IN DATA FD DEPARTMENT EQ %CLASS:DEPARTMENT AND COURSE.DAYS IS
   PRESENT
END FIND

CT.A: COUNT RECORDS IN A

IF COUNT IN CT.A EQ '0' THEN

   %CLASS:MESSAGE = '** NO MATCH **'
   RETURN

END IF
%DAYS IS STRING LEN 2 ARRAY(6)

* INITIALIZE THE DAYS ARRAY

            %DAYS(1) = 'M'
            %DAYS(2) = 'T'
            %DAYS(3) = 'W'
            %DAYS(4) = 'H'
            %DAYS(5) = 'F'
            %DAYS(6) = 'S'
%I = 0
SET HEADER 1 'COURSE.ID' AT 3 WITH 'DAY' AT 15 WITH 'BEGIN' AT 22 WITH -
 'END' AT 30 WITH 'TITLE' AT 40
NP
AGAIN: %I = %I + 1

    IF %I EQ '7' THEN

         JUMP TO FINITO

  END IF

CLEAR LIST P

PLACE RECORDS IN A ON LIST P

S: SORT RECORDS ON P BY COURSE.BEGIN

FR IN S
%X = $INDEX(COURSE.DAYS,%DAYS(%I))

IF %X /= 0 THEN

PRINT COURSE.ID AT 3 WITH %DAYS(%I) AT 15 WITH COURSE.BEGIN AT 22 -
 COURSE.END AT 30 WITH COURSE.DESCRP AT 40

END IF

END FOR

JUMP TO AGAIN

FINITO:

RETURN
END SUBROUTINE DEPT
----------------------------------------------------------------------
**********************************************************************
* FILENAME VIEW2
*                                              AUTHOR - VINIT VERMA
*                                              CREATED  10-23-86
* PURPOSE
* THIS SUBROUTINE SETS UP THE SCREEN VARIABLES TO BE DISPLAYED.
* THIS ROUTINE IS CALLED EACH TIME FOR A SCROLL FUNCTION. ONCE A
* RECORD HAS BEEN IDENTIFIED IN THE DATABASE THIS SUBROUTINE
* INITIALIZES THE SCREEN VARIABLES.  BOTH THE VIEW11 AND VIEW12
* PROCEDURES INVOKE THIS SUBROUTINE.
*
*
**********************************************************************

DISPLAY: SUBROUTINE

NEXT.RECORD: %SEARCH = %INDEX(%SCAN) + 1


LOCATE: IN DATA FD POINTS %INDEX(%SCAN) AND NOT %SEARCH
END FIND
```

```
     FR IN LOCATE

          %CLASS:START = START
          %CLASS:EXPIRE = EXPIRE
          %CLASS:COURSE.ID = COURSE.ID
          %CLASS:INSTRUCTOR = INSTRUCTOR
          %CLASS:COURSE.DAYS = COURSE.DAYS
          %CLASS:COURSE.BEGIN = COURSE.BEGIN
          %CLASS:COURSE.END = COURSE.END
          %CLASS:DEPARTMENT = DEPARTMENT
          %CLASS:SECTION = SECTION
          %CLASS:COURSE.TYPE = COURSE.TYPE
          %CLASS:COURSE.NO = COURSE.NO
          %CLASS:ROOM = ROOM
          %CLASS:BUILDING = BUILDING
          %CLASS:COURSE.DESCRP = COURSE.DESCRP
          %CLASS:COURSE.HOURS = $SUBSTR(COURSE.NO,4,1)
          %CLASS:FREEFORM = FREEFORM
          %CLASS:MIN = MIN
          %CLASS:MAX = MAX

COM1: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '010'
END FIND
FR IN COM1
%CLASS:COMMENT1 = COMMENT
END FOR
COM2: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '020'
END FIND
FR IN COM2
%CLASS:COMMENT2 = COMMENT
END FOR
COM3: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '030'
END FIND
FR IN COM3
%CLASS:COMMENT3 = COMMENT
END FOR
COM4: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '040'
END FIND
FR IN COM4
%CLASS:COMMENT4 = COMMENT
END FOR
COM5: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '050'
END FIND
FR IN COM5
%CLASS:COMMENT5 = COMMENT
END FOR
COM6: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '060'
END FIND
FR IN COM6
%CLASS:COMMENT6 = COMMENT
END FOR
COM7: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '070'
END FIND
FR IN COM7
%CLASS:COMMENT7 = COMMENT
END FOR
COM8: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '080'
END FIND
FR IN COM8
%CLASS:COMMENT8 = COMMENT
END FOR
COM9: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT = %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '090'
END FIND
FR IN COM9
%CLASS:COMMENT9 = COMMENT
END FOR
COM10: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT= %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '100'
END FIND
FR IN COM10
%CLASS:COMMENT10 = COMMENT
END FOR
COM11: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT= %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '110'
END FIND
FR IN COM11
%CLASS:COMMENT11 = COMMENT
END FOR
COM12: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT= %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '120'
END FIND
FR IN COM12
%CLASS:COMMENT12 = COMMENT
END FOR
COM13: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT= %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '130'
END FIND
FR IN COM13
%CLASS:COMMENT13 = COMMENT
END FOR
COM14: IN DATA FD COMMENT IS PRESENT AND DEPARTMENT= %CLASS:DEPARTMENT -
     AND COURSE.NO = %CLASS:COURSE.NO AND NUMBER = '140'
END FIND
FR IN COM14
%CLASS:COMMENT14 = COMMENT
```

```
END FOR

     END FOR
RETURN
END SUBROUTINE DISPLAY
------------------------------------------------------------
*******************************************************
* FILENAME VIEW12
*                                        AUTHOR- VINIT VERMA
*                                        CREATED 11-07-86
* PURPOSE - THIS PROCEDURE PROVIDES THE SUPER USER'S
* OPTION FOR THE DEPARTMENTS.  UPDATES ARE ALLOWED TO BE
* PERFORMED VIA THIS PROCEDURE.  THE QUERY FIELDS FOR THE
* RETRIEVAL OPERATIONS ARE -
*
*            COURSE.ID
*            DEPARTMENT AND COURSE NUMBER
*            DEPARTMENT
*            BUILDING AND ROOM
*            INSTRUCTOR'S NAME
*
* THE PF1 KEY PERFORMS THE RESERVE OPERATION, AND THE
* PF2 KEY THE DELETE.  UNIQUE COURSE.ID NUMBERS ARE
* MAINTAINED FOR THE RESERVE OPTION.  THE DELETE
* KEYS OFF THE COURSE.ID NUMBER.
*
* THE FILES INCLUDED BY THIS PROCEDURE ARE
*
*              VIEW1.S - THE USER INTERFACE SCREEN
*              VIEW2 - THE DATA DISPLAY PROCEDURE
*
*******************************************************
RESET MSGCTL = X'02'
O DATA

BEGIN

**********************************************************
*      INVOKE THE EXPIRE ROUTINE TO DO THE GARBAGE COLLECTION
**********************************************************

CALL EXPIRE
%INDEX IS STRING LEN 7 ARRAY(200)
%SCAN IS FIXED
%COUNT IS FIXED

**************************************************
*  THE SCREEN DEFINITION
**************************************************

IN SCHED INCLUDE VIEW1.S

*****************************************************
* CLEAR UP THE MESSAGE INCASE EXPIRE DELETED RECORDS
*****************************************************

%CLASS:MESSAGE = ' '

************************************************************
*     SET UP THE ON CONFLICT CONDITION
************************************************************

 ON FIND CONFLICT

   %CLASS:MESSAGE = '*** PLEASE TRY LATER ***'
   JUMP TO START

END ON
************************************************************
*     DEACTIVATE THE ON ATTENTION KEY
************************************************************

 ON ATTENTION

   %CLASS:MESSAGE = '*** ATTENTION KEY DEACTIVATED ***'
   JUMP TO START

END ON

*----------------------------------------------*
*  THIS IS THE STARTING POINT FOR THE PROGRAM
*----------------------------------------------*

START: READ SCREEN CLASS

%CLASS:MESSAGE = ' '

************************************************
*          VERIFY THE PFKEY VALUES
************************************************

IF %CLASS:PFKEY /= 1 AND %CLASS:PFKEY /= 2 AND %CLASS:PFKEY /= 3 AND -
   AND %CLASS:PFKEY /= 5 AND %CLASS:PFKEY /= 7 -
   AND %CLASS:PFKEY /= 8 AND %CLASS:PFKEY /= 6 THEN

    %CLASS:MESSAGE = '*** PF' WITH %CLASS:PFKEY WITH '  IS NOT ACTIVE'
    JUMP TO START
END IF

******************************************************************
*     CHECK FOR THE PF1 RESERVE KEY
******************************************************************

IF %CLASS:PFKEY = 1 THEN

     CALL RESERVE
     JUMP TO START

END IF

*******************************************************************
*      CHECK FOR THE PF2 DELETE KEY
*******************************************************************

IF %CLASS:PFKEY = 2 THEN

     CALL DELETE
     JUMP TO START

END IF

*******************************************************************
*      CHECK FOR THE PF3 EXIT KEY
*******************************************************************

IF %CLASS:PFKEY = 3 THEN

   JUMP TO FINISH

END IF
*******************************************************************
*      CHECK FOR THE PF6 REFRESH KEY
*******************************************************************

IF %CLASS:PFKEY = 6 THEN

   PREPARE SCREEN CLASS
   JUMP TO START

END IF
*******************************************************************
*      CHECK FOR THE PF5 REFRESH KEY
*******************************************************************

IF %CLASS:PFKEY = 5 THEN

CLEAR LIST SCROLL

*******************************************************************
*      CHECK IF THE QUERY IS KEYING OFF THE COURSE NAME AND NUMBER
******************************************************************

IF %CLASS:ITEMID = 3 THEN


NUMBER: IN DATA FD DEPARTMENT  = %CLASS:DEPARTMENT AND -
        COURSE.NO = %CLASS:COURSE.NO AND COMMENT IS NOT PRESENT
END FIND

CT.NUMBER: COUNT RECORDS IN NUMBER

IF COUNT IN CT.NUMBER EQ O THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF
 PLACE RECORDS IN NUMBER ON LIST SCROLL

*******************************************************************
*      CHECK IF THE QUERY IS KEYING OFF THE COURSE ID NUMBER
******************************************************************

ELSEIF %CLASS:ITEMID = 1 THEN


NUMBER1: IN DATA FD COURSE.ID  = %CLASS:COURSE.ID
END FIND

CT.NUMBER1: COUNT RECORDS IN NUMBER1

IF COUNT IN CT.NUMBER1 EQ O THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER1 ON LIST SCROLL

*******************************************************************
*      CHECK IF THE QUERY IS KEYING OFF FOR A CERTAIN DEPARTMENT
******************************************************************

ELSEIF %CLASS:ITEMID = 2 THEN


NUMBER2: IN DATA FD DEPARTMENT = %CLASS:DEPARTMENT AND -
         COMMENT IS NOT PRESENT
END FIND

CT.NUMBER2: COUNT RECORDS IN NUMBER2

IF COUNT IN CT.NUMBER2 EQ O THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START
```

```
END IF

 PLACE RECORDS IN NUMBER2 ON LIST SCROLL
*******************************************************************
*     CHECK IF THE QUERY IS KEYING OFF FOR A CERTAIN ROOM
******************************************************************

ELSEIF %CLASS:ITEMID = 5 THEN


NUMBER3: IN DATA FD ROOM = %CLASS:ROOM AND BUILDING = %CLASS:BUILDING
END FIND

CT.NUMBER3: COUNT RECORDS IN NUMBER3

IF COUNT IN CT.NUMBER3 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER3 ON LIST SCROLL
*******************************************************************
*     CHECK IF THE QUERY IS KEYING OFF FOR AN INSTRUCTOR
******************************************************************

ELSEIF %CLASS:ITEMID = 6 THEN


NUMBER4: IN DATA FD INSTRUCTOR = %CLASS:INSTRUCTOR
END FIND

CT.NUMBER4: COUNT RECORDS IN NUMBER4

IF COUNT IN CT.NUMBER4 EQ 0 THEN

   PREPARE SCREEN CLASS
   %CLASS:MESSAGE = '*** NO MATCH ***'
   JUMP TO START

END IF

 PLACE RECORDS IN NUMBER4 ON LIST SCROLL

END IF

ERR: COUNT RECORDS ON LIST SCROLL

IF COUNT IN ERR GT 200 THEN

    %CLASS:MESSAGE = '*** OVER 200 RECORDS TO SCROLL ***'
    JUMP TO START

END IF
%COUNT = 0

 FOR EACH RECORD ON LIST SCROLL

    %COUNT = %COUNT + 1
    %INDEX(%COUNT) = $CURREC

END FOR

*****************************************************************
*      SET UP THE CATENATED SCAN POINTER
*****************************************************************

%SCAN = 1

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

               %CLASS:DEPARTMENT = %DEPARTMENT
              %CLASS:COURSE.NO = %COURSE.NO

CALL DISPLAY

JUMP TO START

END IF

************************************************************
*'    CHECK FOR THE PF8 SCROLL FORWARD KEY
************************************************************

IF %CLASS:PFKEY = 8 THEN

 %SCAN = %SCAN + 1
IF %SCAN LE %COUNT THEN

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

               %CLASS:DEPARTMENT = %DEPARTMENT
              %CLASS:COURSE.NO = %COURSE.NO

 CALL DISPLAY
ELSE
    %CLASS:MESSAGE= '*** LAST RECORD ***'
    %SCAN = %SCAN - 1
```

```
END IF
 JUMP TO START

 END IF
*************************************************************
*    CHECK FOR THE PF7 SCROLL FORWARD KEY
*************************************************************

IF %CLASS:PFKEY = 7 THEN

 %SCAN = %SCAN - 1
IF %SCAN GE 1 THEN

%DEPARTMENT = %CLASS:DEPARTMENT
%COURSE.NO = %CLASS:COURSE.NO

PREPARE SCREEN CLASS

              %CLASS:DEPARTMENT = %DEPARTMENT
              %CLASS:COURSE.NO = %COURSE.NO

 CALL DISPLAY
ELSE
     %CLASS:MESSAGE = '*** FIRST RECORD ***'
     %SCAN = %SCAN + 1
END IF
 JUMP TO START

 END IF
FINISH: PREPARE SCREEN CLASS
        COMMIT
*****************************************************************
* INCLUDE THE SCROLL.LIST FILE
*****************************************************************
IN SCHED INCLUDE VIEW2

*****************************************************************
* SUBROUTINE RESERVE
*****************************************************************

RESERVE: SUBROUTINE

*****************************************************************
* CHECK FOR A UNIQUE COURSE IDENTIFICATION NUMBER
*****************************************************************

UNIQUE: IN DATA FD COURSE.ID = %CLASS:COURSE.ID
END FIND

CT.UNIQUE: COUNT RECORDS IN UNIQUE

IF COUNT IN CT.UNIQUE /= '0' THEN

   %CLASS:MESSAGE = '*** DUPLICATE COURSE ID ***'
   RETURN

END IF
*****************************************************************
* CHECK CONFLICTS ONLY IF ROOM BUILDING DAYS BEGIN AND END ARE
* FILLED UP BY THE USER ON THE SCREEN
*****************************************************************

IF %CLASS:BUILDING /= '' AND %CLASS:ROOM /= '' AND -
   %CLASS:COURSE.DAYS /= '' AND %CLASS:COURSE.BEGIN /= '' AND -
   %CLASS:COURSE.END /= '' THEN

*****************************************************************
* LOCATE ALL THE RECORDS FOR THIS ROOM
*****************************************************************

ROOM: IN DATA FD ROOM = %CLASS:ROOM AND BUILDING = %CLASS:BUILDING AND -
      COURSE.DAYS IS PRESENT
END FIND

*****************************************************************
* SET TEMPORARY VARIABLES TO DAY BEGIN AND END TIMES
*****************************************************************

%DAYS = %CLASS:COURSE.DAYS
%BEGIN = %CLASS:COURSE.BEGIN
%ENDS = %CLASS:COURSE.END

******************************************************************
* SUBTRACT TEN MINUTES FROM THE START TIME TO GIVE A 10 MIN OVERLAP
******************************************************************

%HRS = $SUBSTR(%BEGIN,1,2)
%MINS = $SUBSTR(%ENDS,3,2)

%TOTAL IS STRING DP 5
%FILL= '0000'

******************************************************************
* SUBTRACT TEN MINUTES FROM THE START TIME TO GIVE A 10 MIN OVERLAP
******************************************************************

%HRS = $SUBSTR(%BEGIN,1,2)
%MINS = $SUBSTR(%BEGIN,3,2)

%TOTAL = %HRS * 60
%TOTAL = %TOTAL + %MINS
%TOTAL = %TOTAL - 10
%TOTAL = %TOTAL / 60

%I = $INDEX(%TOTAL,'.')
%HRS = $SUBSTR(%TOTAL,1,%I - 1)
```

```
%MINS = $SUBSTR(%TOTAL,%I)
%MINS = $ROUND(%MINS * 60.0,0)
%BEGIN = %HRS WITH %MINS

%LEN = $LEN(%BEGIN)
%LEN = 4 - %LEN
%FILL = $SUBSTR(%FILL,1,%LEN)
%BEGIN = %FILL WITH %BEGIN


******************************************************************
* SET THE CONFLICT FLAG TO LOW
******************************************************************


%CONFLICT = 0


******************************************************************
* SET REPEAT EQUAL TO THE LENGTH OF THE DAY STRING
******************************************************************

%REPEAT = $LEN(%DAYS)

%DAY1 = %DAYS

******************************************************************
* MAIN FOR LOOP WHICH CHECK EACH COURSE MEETING IN THE NEEDED ROOM
******************************************************************

FOR EACH RECORD IN ROOM

%DAYS = %DAY1

******************************************************************
* LOOP TO CHECK EACH DAY IN THE DAYS FIELD
******************************************************************

FOR %LOOP FROM 1 TO %REPEAT

%WEEK = $SUBSTR(%DAYS,1,1)
%DAYS = $SUBSTR(%DAYS,2)
%PRESENT = $INDEX(COURSE.DAYS,%WEEK)


******************************************************************
* IF THE DAY IS PRESENT IN THE SEARCHED RECORD THEN CHECK THE TIME
******************************************************************

IF %PRESENT /= 0 THEN


******************************************************************
* CHECK THE VARIOUS COMBINATION OF POSSIBLE CONFLICTS
******************************************************************


IF %BEGIN GT COURSE.BEGIN AND %BEGIN LT COURSE.END THEN
*PRINT 'FIRST IN BETWEEN CONFLICT'
%CONFLICT = 1
END IF

IF %ENDS GT COURSE.BEGIN AND %ENDS LT COURSE.END THEN
*PRINT 'LAST IN BETWEEN CONFLICT'
%CONFLICT = 1
END IF

IF %BEGIN EQ COURSE.BEGIN OR %BEGIN EQ COURSE.END THEN
*PRINT 'FIRST EQUAL CONFLICT'
%CONFLICT = 1
END IF

IF %ENDS EQ COURSE.BEGIN OR %ENDS EQ COURSE.END THEN
*PRINT 'LAST EQUAL CONFLICT'
%CONFLICT = 1
END IF

IF %BEGIN LT COURSE.BEGIN AND %ENDS GT COURSE.END THEN
*PRINT 'THE BIG SPAN CONFLICT'
%CONFLICT = 1
END IF


******************************************************************
* IF THERE EXISTS A CONFLICT THEN SET A MESSAGE AND RETURN
******************************************************************



******************************************************************
* IF THERE IS A CONFLICT THEN CHECK THE START AND EXPIRE DATES
* ON THE CURRENT RECORD.  IF THE START ON THE SCREEN IS AFTER THE
* EXPIRE OF THE CURRENT RECORD, OR IF THE EXPIRE ON THE SCREEN IS
* BEFORE THE START ON THE CURRENT RECORD THEN THERE IS NO CONFLICT.
******************************************************************

IF %CONFLICT = '1' THEN

  IF EXPIRE NE '' AND START NE '' THEN

     IF %CLASS:START GT EXPIRE OR %CLASS:EXPIRE LT START THEN

*     NO CONFLICT THEN

     ELSE
```

```
       %CLASS:MESSAGE = '*** CONFLICT1 ***'
       RETURN

     END IF

 ELSE
       %CLASS:MESSAGE = '*** CONFLICT2 ***'
       RETURN

END IF

END IF
END IF


END FOR

END FOR

END IF

*****************************************************************
* STORE THE ORDINARY RECORD IN FILE DATA
*****************************************************************

IN DATA STORE RECORD

      START = %CLASS:START
      EXPIRE = %CLASS:EXPIRE
      COURSE.ID = %CLASS:COURSE.ID
      COURSE.NO = %CLASS:COURSE.NO
      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.TYPE = %CLASS:COURSE.TYPE
      SECTION = %CLASS:SECTION
      COURSE.DAYS = %CLASS:COURSE.DAYS
      COURSE.BEGIN = %CLASS:COURSE.BEGIN
      COURSE.END = %CLASS:COURSE.END
      BUILDING = %CLASS:BUILDING
      INSTRUCTOR = %CLASS:INSTRUCTOR
      COURSE.DESCRP =  %CLASS:COURSE.DESCRP
      MIN =  %CLASS:MIN
      MAX =  %CLASS:MAX
      ROOM =  %CLASS:ROOM
      FREEFORM =  %CLASS:FREEFORM

END STORE

*****************************************************************
* STORE THE COMMENT RECORD IN FILE DATA ONLY IF THERE IS ONE
*****************************************************************


NONE: IN DATA FD DEPARTMENT = %CLASS:DEPARTMENT AND -
       COURSE.NO = %CLASS:COURSE.NO AND COMMENT IS PRESENT
END FIND

CT.NONE: COUNT RECORDS IN NONE

IF COUNT IN CT.NONE EQ 0 THEN

IF %CLASS:COMMENT1 /= '' THEN

   IN DATA STORE RECORD

       DEPARTMENT = %CLASS:DEPARTMENT
       COURSE.NO = %CLASS:COURSE.NO
       NUMBER = '010'
       COMMENT = %CLASS:COMMENT1

  END STORE

END IF
IF %CLASS:COMMENT2 /= '' THEN

   IN DATA STORE RECORD

       DEPARTMENT = %CLASS:DEPARTMENT
       COURSE.NO = %CLASS:COURSE.NO
       NUMBER = '020'
       COMMENT = %CLASS:COMMENT2

  END STORE

END IF
IF %CLASS:COMMENT3 /= '' THEN

   IN DATA STORE RECORD

       DEPARTMENT = %CLASS:DEPARTMENT
       COURSE.NO = %CLASS:COURSE.NO
       NUMBER = '030'
       COMMENT = %CLASS:COMMENT3

  END STORE

END IF
IF %CLASS:COMMENT4 /= '' THEN

   IN DATA STORE RECORD

       DEPARTMENT = %CLASS:DEPARTMENT
       COURSE.NO = %CLASS:COURSE.NO
       NUMBER = '040'
       COMMENT = %CLASS:COMMENT4

  END STORE
END IF
IF %CLASS:COMMENTS5 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '050'
      COMMENT = %CLASS:COMMENTS5

  END STORE

END IF
IF %CLASS:COMMENTS6 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '060'
      COMMENT = %CLASS:COMMENTS6

  END STORE

END IF
IF %CLASS:COMMENT7 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '070'
      COMMENT = %CLASS:COMMENT7

  END STORE

END IF
IF %CLASS:COMMENTS8 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '080'
      COMMENT = %CLASS:COMMENTS8

  END STORE

END IF
IF %CLASS:COMMENTS9 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '090'
      COMMENT = %CLASS:COMMENTS9

  END STORE

END IF
IF %CLASS:COMMENT10 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '100'
      COMMENT = %CLASS:COMMENT10

  END STORE

END IF
IF %CLASS:COMMENT11 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '110'
      COMMENT = %CLASS:COMMENT11

  END STORE

END IF
IF %CLASS:COMMENT12 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '120'
      COMMENT = %CLASS:COMMENT12

  END STORE

END IF
IF %CLASS:COMMENT13 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '130'
      COMMENT = %CLASS:COMMENT13
  END STORE

END IF
IF %CLASS:COMMENT14 /= '' THEN

   IN DATA STORE RECORD

      DEPARTMENT = %CLASS:DEPARTMENT
      COURSE.NO = %CLASS:COURSE.NO
      NUMBER = '140'
      COMMENT = %CLASS:COMMENT14

  END STORE

END IF
    %CLASS:MESSAGE = '** ROOM RESERVED **'
ELSE
    %CLASS:MESSAGE = '** COMMENT FOR COURSE NO. EXISTS ***'
END IF
COMMIT
END SUBROUTINE RESERVE

*******************************************************************
* SUBROUTINE DELETE
*******************************************************************

DELETE: SUBROUTINE

COURSE: IN DATA FD COURSE.ID = %CLASS:COURSE.ID
END FIND

CT.COURSE: COUNT RECORDS IN COURSE

IF COUNT IN CT.COURSE EQ O THEN

   %CLASS:MESSAGE = '*** NO MATCH ***'
   RETURN

END IF

*******************************************************************
* DELETE THE MAIN RECORD
*******************************************************************

FR IN COURSE

     %DEPT = DEPARTMENT
     %NO = COURSE.NO
     DELETE RECORD
COMMIT
END FOR

*******************************************************************
* CHECK IF ANOTHER COURSE.ID HAS THE SAME COMMENT RECORD
*******************************************************************

IND: IN DATA FD DEPARTMENT = %DEPT AND COURSE.NO = %NO -
     AND COURSE.ID IS PRESENT
END FIND

CT.IND: COUNT RECORDS IN IND

IF COUNT IN CT.IND EQ O THEN

DET: IN DATA FD DEPARTMENT = %DEPT AND COURSE.NO = %NO AND -
     COMMENT IS PRESENT
END FIND

FR IN DET

 DELETE RECORD

END FOR
END IF

   %CLASS:MESSAGE = '*** COURSE DELETED ***'

   COMMIT
END SUBROUTINE DELETE

*******************************************************************
* SUBROUTINE EXPIRE
*******************************************************************

EXPIRE: SUBROUTINE

REMOVE: IN DATA FD EXPIRE IS PRESENT
END FIND

FR IN REMOVE

*******************************************************************
* DELETE ALL RECORDS FOR WHICH THE EXPIRY DATE IS L.E. TODAYS DATE
*******************************************************************

   IF EXPIRE LT $DATE THEN
        %CLASS:COURSE.ID = COURSE.ID
        CALL DELETE
   END IF

END FOR

END SUBROUTINE EXPIRE
END
O SCHED
```

```
RESET MSGCTL = X'01'
-----------------------------------------------------------------
*****************************************************************
* FILENAME VIEW1.S
*                                             AUTHOR - VINIT VERMA
*                                             CREATED - 10-23-86
* PURPOSE - THIS PROCEDURE PROVIDES THE USER INTERFACE SCREEN
* FOR THE SUPER USER'S OPTION.  THIS PROCEDURE IS INVOKED
* BY THE VIEW12 PROGRAM.
*
*****************************************************************

SCREEN CLASS
TITLE '----------------- Oklahoma State University Class Schedule ------
MAX PFKEY 12
PROMPT MESSAGE  AT 40 DEFAULT ' '
PROMPT 'id number      :' AT 2 INPUT COURSE.ID LEN 6 ITEMID 1 -
PROMPT 'instructor  :' AT 25 INPUT INSTRUCTOR LEN 13 ITEMID 6 -
PROMPT 'days  :' AT 53 INPUT COURSE.DAYS LEN 6 -
PROMPT 'min:' AT 68 INPUT MIN LEN 2
PROMPT 'department     :' AT 2 INPUT DEPARTMENT LEN 6 ITEMID 2 -
PROMPT 'building    :' AT 25 INPUT BUILDING LEN 5 ITEMID 4 -
PROMPT 'start :' AT 53 INPUT COURSE.BEGIN LEN 4 -
PROMPT 'max:' AT 68 INPUT MAX LEN 2
PROMPT 'course number :' AT 2 INPUT COURSE.NO LEN 5 ITEMID 3 -
PROMPT 'room        :' AT 25 INPUT ROOM LEN 5 ITEMID 5 -
PROMPT 'end   :' AT 53 INPUT COURSE.END LEN 4
PROMPT 'type           :' AT 2 INPUT COURSE.TYPE LEN 2 -
PROMPT 'section     :' AT 25 INPUT SECTION LEN 3 -
PROMPT 'hours :' AT 53 INPUT COURSE.HOURS LEN 2
PROMPT 'title          :' AT 2 INPUT COURSE.DESCRP LEN 22 -
PROMPT 'undecided hour  :' AT 45 INPUT FREEFORM LEN 14
PROMPT 'begin(yy-mm-dd):' AT 19 INPUT START LEN 8 -
PROMPT 'expiry(yy-mm-dd):' AT 45 INPUT EXPIRE LEN 8
PROMPT 'comment:' AT 2 INPUT COMMENT1 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT2 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT3 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT4 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT5 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT6 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT7 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT8 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT9 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT10 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT11 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT12 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT13 LEN 60
PROMPT '        ' AT 2 INPUT COMMENT14 LEN 60
IN SCHED I KEY1
DEFAULT CURSOR COURSE.ID
END SCREEN
-----------------------------------------------------------------------
* FILENAME KEY1
PROMPT 'pf-1 reserve pf-2 delete pf3 exit pf5 refresh pf-6 clear pf-7<--
      pf8 -->'
***********************************************************************
* FILENAME - ROOM.ATTRIBUTES
*                                               AUTHOR - VINIT VERMA
*                                               CREATED 11-17-86
* PURPOSE -
* THIS PROCEDURE MANIPULATES THE ROOM INVENTORY DATABASE IN FILE SCHED.
* THE PROCEDURE PERFORMS RETRIEVALS VIA THE BUILDING AND ROOM FIELDS,
* AND ALSO JUST BY THE BUILDING FIELD.  THE DELETE KEYS OFF THE
* BUILDING AND ROOM FIELDS.  THIS PROCEDURE INVOKES THE DISPLAY
* SUBROUTINE TO INITIALIZE THE SCREEN VARIABLES.
*
***********************************************************************

BEGIN

*******************************************************
*      THE INDEX ARRAY IS SET UP TO DO THE SCROLLING
*******************************************************

%INDEX IS STRING LEN 7 ARRAY(20)
%SCAN IS FIXED
%COUNT IS FIXED

****************************
*  THE SCREEN DEFINITION    *
****************************

SCREEN ROUTE
TITLE 'Oklahoma State University Room Inventory Database' AT 15
MAX PFKEY 12
PROMPT MESSAGE  AT 40 DEFAULT ' '
SKIP 1 LINE
PROMPT 'building:' AT 2 INPUT BUILDING LEN 4
PROMPT 'room    :' AT 2 INPUT ROOM LEN 4
PROMPT 'type    :' AT 2 INPUT TYPE LEN 1
PROMPT 'capacity:' AT 2 INPUT CAPACITY LEN 3
SKIP 1 LINE
PROMPT '1 - case study' AT 40
PROMPT '2 - auditorium' AT 40
PROMPT '3 - fixed' AT 40
PROMPT '4 - table & chairs' AT 40
PROMPT '5 - arm' AT 40
PROMPT '6 - bolted arm' AT 40
PROMPT '7 - lab' AT 40
SKIP 7 LINE
I KEY1
DEFAULT CURSOR BUILDING
END SCREEN

**************************************************************
*     SET UP THE ON CONFLICT CONDITION
```

```
************************************************************

 ON FIND CONFLICT

   %ROUTE:MESSAGE = '*** PLEASE TRY LATER ***'
   JUMP TO START

END ON
************************************************************
*     DEACTIVATE THE ON ATTENTION KEY
************************************************************

 ON ATTENTION

   %ROUTE:MESSAGE = '*** ATTENTION KEY DEACTIVATED ***'
   JUMP TO START

END ON

*----------------------------------------------------*
*          THIS IS THE STARTING POINT                 *
*----------------------------------------------------*

START: READ SCREEN ROUTE

%ROUTE:MESSAGE = ' '

*********************************************
*         VERIFY THE PFKEY VALUES
*********************************************

IF %ROUTE:PFKEY /= 1 AND %ROUTE:PFKEY /= 2 AND %ROUTE:PFKEY /= 3 -
   AND %ROUTE:PFKEY /= 5 AND %ROUTE:PFKEY /= 6 AND %ROUTE:PFKEY /= 7 -
   AND %ROUTE:PFKEY /= 8 THEN

    %ROUTE:MESSAGE = '*** PF' WITH %ROUTE:PFKEY WITH '  IS NOT ACTIVE'
    JUMP TO START
END IF

******************************************************************
*     CHECK FOR THE PF1 SAVE KEY
******************************************************************

IF %ROUTE:PFKEY = 1 THEN

   CALL SAVE
   JUMP TO START

END IF
******************************************************************
*     CHECK FOR THE PF2 DELETE KEY
******************************************************************

IF %ROUTE:PFKEY = 2 THEN

   CALL DELETE
   JUMP TO START

END IF
******************************************************************
*     CHECK FOR THE PF3 EXIT KEY
******************************************************************

IF %ROUTE:PFKEY = 3 THEN

   JUMP TO FINISH

END IF

******************************************************************
*     CHECK FOR THE PF6 CLEAR KEY
******************************************************************

IF %ROUTE:PFKEY = 6 THEN

   PREPARE SCREEN ROUTE
   JUMP TO START

END IF
******************************************************************
*     CHECK FOR THE PF5 REFRESH KEY
******************************************************************

IF %ROUTE:PFKEY = 5 THEN

CLEAR LIST SCROLL

IF %ROUTE:ROW = 5 THEN

NUMBER: FD BUILDING  = %ROUTE:BUILDING AND ROOM = %ROUTE:ROOM_
END FIND

CT.NUMBER: COUNT RECORDS IN NUMBER

IF COUNT IN CT.NUMBER EQ 0 THEN

   PREPARE SCREEN ROUTE
   %ROUTE:MESSAGE = '*** NO MATCH *** '
   JUMP TO START

END IF
IF COUNT IN CT.NUMBER GE 20 THEN

   PREPARE SCREEN ROUTE
   %ROUTE:MESSAGE = '*** TOO MANY ENTRIES TO SCROLL **'
   JUMP TO START
```

```
END IF

PLACE RECORDS IN NUMBER ON LIST SCROLL

ELSE

 IF %ROUTE:ROW = 4 THEN

NUMBER1: FD BUILDING  = %ROUTE:BUILDING
END FIND

CT.NUMBER1: COUNT RECORDS IN NUMBER1

IF COUNT IN CT.NUMBER1 EQ 0 THEN

   PREPARE SCREEN ROUTE
   %ROUTE:MESSAGE = '*** NO MATCH *** '
   JUMP TO START

END IF
IF COUNT IN CT.NUMBER1 GE 20 THEN

   PREPARE SCREEN ROUTE
   %ROUTE:MESSAGE = '*** TOO MANY ENTRIES TO SCROLL **'
   JUMP TO START

END IF

PLACE RECORDS IN NUMBER1 ON LIST SCROLL

END IF
END IF

%COUNT = 0

 FOR EACH RECORD ON LIST SCROLL

    %COUNT = %COUNT + 1
    %INDEX(%COUNT) = $CURREC

END FOR
*********************************************
*      SET UP THE CATENATED SCAN POINTER
*********************************************

%SCAN = 1

CALL DISPLAY
JUMP TO START

END IF

************************************************************
*    CHECK FOR THE PF8 SCROLL FORWARD KEY
************************************************************

IF %ROUTE:PFKEY = 8 THEN

 %SCAN = %SCAN + 1
IF %SCAN LE %COUNT THEN
 CALL DISPLAY
ELSE
    %ROUTE:MESSAGE= '*** LAST RECORD ***'
    %SCAN = %SCAN - 1
END IF
 JUMP TO START

 END IF
************************************************************
*    CHECK FOR THE PF7 SCROLL FORWARD KEY
************************************************************

IF %ROUTE:PFKEY = 7 THEN

 %SCAN = %SCAN - 1
IF %SCAN GE 1 THEN
 CALL DISPLAY
ELSE
     %ROUTE:MESSAGE = '*** FIRST RECORD ***'
     %SCAN = %SCAN + 1
END IF
 JUMP TO START

 END IF
FINISH: PREPARE SCREEN ROUTE
        COMMIT
******************************************
*     INCLUDE THE DISPLAY FILE
******************************************

INCLUDE DISPLAY

******************************************
*     SUBROUTINE SAVE
******************************************

SAVE: SUBROUTINE

LOG: IN SCHED FD BUILDING = %ROUTE:BUILDING AND ROOM = %ROUTE:ROOM
END FIND

CT.LOG: COUNT RECORDS IN LOG

IF COUNT IN CT.LOG NE 0 THEN
```

```
   FR IN LOG
      CHANGE TYPE TO %ROUTE:TYPE
      CHANGE CAPACITY TO %ROUTE:CAPACITY

END FOR
%ROUTE:MESSAGE = '** ENTRY CHANGED **'
ELSE
       STORE RECORD

         BUILDING = %ROUTE:BUILDING
         ROOM = %ROUTE:ROOM
         TYPE = %ROUTE:TYPE
         CAPACITY = %ROUTE:CAPACITY

       END STORE
%ROUTE:MESSAGE = '** ENTRY STORED **'
END IF
END SUBROUTINE SAVE

*********************************************
*      SUBROUTINE DELETE
*********************************************

DELETE: SUBROUTINE

DEL: IN SCHED FD BUILDING = %ROUTE:BUILDING AND ROOM = %ROUTE:ROOM
END FIND

CT.DEL: COUNT RECORDS IN DEL

 IF COUNT IN CT.DEL EQ 0 THEN
    %ROUTE:MESSAGE = '** NO MATCH **'
     RETURN
END IF

FR IN DEL

   DELETE RECORD

END FOR

%ROUTE:MESSAGE = '** ENTRY DELETED **'

RETURN
END SUBROUTINE DELETE
END
----------------------------------------------------------------------
* FILENAME KEY2
PROMPT 'pf3 exit  pf5 refresh  pf-6 clear  pf-7 scroll back
       'pf8 - scroll back'
*************************************************************************
* FILENAME- DISPLAY
*                                               AUTHOR - VINIT VERMA
*                                               CREATED 11-9-86
* PRUPOSE -
* THIS SUBROUTINE SETS UP THE SCREEN VARIABLES TO BE DISPLAYED.
* THIS ROUTINE IS CALLED EACH TIME FOR A SCROLL FUNCTION.
* THE ROOM.ATTRIBUTES PROGRAM INVOKES THIS SUBROUTINE.
*************************************************************************

DISPLAY: SUBROUTINE

NEXT.RECORD: %SEARCH = %INDEX(%SCAN) + 1


LOCATE: FD POINTS %INDEX(%SCAN) AND NOT %SEARCH
END FIND



     FR IN LOCATE
         %ROUTE:BUILDING = BUILDING
         %ROUTE:ROOM = ROOM
         %ROUTE:TYPE = TYPE
         %ROUTE:CAPACITY = CAPACITY
     END FOR


RETURN
END SUBROUTINE DISPLAY
------------------------------------------------------------------------
*************************************************************************
* FILENAME - ROOM.QUERY
*                                               AUTHOR - VINIT VERMA
*                                               CREATED 11-09-86
* PURPOSE -
* THIS PROCEDURE QUERIES DATA IN THE ROOM INVENTORY AND CLASS SCHEDULE
* DATABASES.  THE ROUTINE PERFORMS DATA RETRIEVAL FUNCTIONS ONLY.
* THE USER IS EXPECTED TO SELECT ONE OF THE ROOMS DISPLAYED
* VIA THE QUERY OPTION.  IF NO ROOMS MATCH THE REQUEST
* CRITERIA A NO MATCH MESSAGE IS DISPLAYED.  NOTE SOME
* ROOMS ON CAMPUS ARE ASSIGNED TO DEPARTMENTS, AND THEY
* DO NOT OCCUR IN THE AVAILABLE POOL OF RESERVEABLE
* ROOMS.
*************************************************************************

BEGIN

%CAPACITY IS FIXED
%LIMIT IS FIXED

*****************************
*  THE SCREEN DEFINITION    *
*****************************

SCREEN QUERY
```

```
TITLE 'Oklahoma State University Room Inventory Database' AT 15
MAX PFKEY 12
PROMPT MESSAGE  AT 40 DEFAULT ' '
PROMPT 'start(yy-mm-dd) :' AT 2 INPUT START.DATE LEN 8
PROMPT 'expire(yy-mm-dd):' AT 2 INPUT EXPIRE LEN 8
PROMPT 'days            :' AT 2 INPUT DAYS LEN 6
PROMPT 'begin time      :' AT 2 INPUT START LEN 4
PROMPT 'end time        :' AT 2 INPUT TERMINATE LEN 4
PROMPT 'capacity        :' AT 2 INPUT CAPACITY LEN 3
PROMPT 'room type       :' AT 2 INPUT TYPE LEN 1
SKIP 4 LINES
PROMPT '1 - case study' AT 25
PROMPT '2 - auditorium' AT 25
PROMPT '3 - fixed' AT 25
PROMPT '4 - table & chairs' AT 25
PROMPT '5 - arm' AT 25
PROMPT '6 - bolted arm' AT 25
PROMPT '7 - lab' AT 25
SKIP 2 LINES
PROMPT '     PF5 - QUERY              PF3 - EXIT'
DEFAULT CURSOR START.DATE
END SCREEN

****************************************************************
*     SET UP THE ON CONFLICT CONDITION
****************************************************************

 ON FIND CONFLICT

   %QUERY:MESSAGE = '*** PLEASE TRY LATER ***'
   JUMP TO START

END ON
****************************************************************
*     DEACTIVATE THE ON ATTENTION KEY
****************************************************************

 ON ATTENTION

   %QUERY:MESSAGE = '*** ATTENTION KEY DEACTIVATED ***'
   JUMP TO START

END ON

*----------------------------------------------------------*
*          THIS IS THE STARTING POINT                      *
*----------------------------------------------------------*

START: READ SCREEN QUERY

%QUERY:MESSAGE = ' '

**********************************************
*          VERIFY THE PFKEY VALUES
**********************************************

IF %QUERY:PFKEY /= 5 AND %QUERY:PFKEY /= 3 AND %QUERY:PFKEY /= 6 THEN

    %QUERY:MESSAGE = '*** PF' WITH %QUERY:PFKEY WITH '  IS NOT ACTIVE'
    JUMP TO START
END IF

**********************************************************************
*     CHECK FOR THE PF3 EXIT KEY
**********************************************************************

IF %QUERY:PFKEY = 3 THEN

   JUMP TO FINISH

END IF

**********************************************************************
*     CHECK FOR THE PF6 CLEAR KEY
**********************************************************************

IF %QUERY:PFKEY = 6 THEN

   PREPARE SCREEN QUERY
   JUMP TO START

END IF
**********************************************************************
*     CHECK FOR THE PF5 REFRESH KEY
**********************************************************************

IF %QUERY:PFKEY = 5 THEN
  CALL GET.ROOM
  JUMP TO START
END IF
FINISH: PREPARE SCREEN QUERY
**********************************************************************
*     SUBROUTINE GET.ROOM
**********************************************************************

GET.ROOM: SUBROUTINE

*******************************************************************
* GET ALL ROOMS WHICH MATCH THE ROOMS SPECIFICS
*******************************************************************


UNIQ: IN SCHED FIND ALL RECORDS FOR WHICH TYPE = %QUERY:TYPE
END FIND

CLEAR LIST P
```

```
FOR EACH RECORD IN UNIQ

  %CAPACITY = CAPACITY
  %LIMIT = %QUERY:CAPACITY

  IF %CAPACITY GE %LIMIT THEN
     PLACE RECORD ON LIST P
   END IF

END FOR

CT.UNIQ: COUNT RECORDS ON LIST P

IF COUNT IN CT.UNIQ = 'O' THEN

   %QUERY:MESSAGE = '*** NO ROOM WITH SUCH ATTRIBUTES **'
   RETURN

END IF

******************************************************************
* SET TEMPORARY VARIABLES TO DAY BEGIN AND END TIMES
******************************************************************

%DAYS = %QUERY:DAYS
%BEGIN = %QUERY:START
%ENDS = %QUERY:TERMINATE

*******************************************************************
* SUBTRACT TEN MINUTES FROM THE START TIME TO GIVE A 10 MIN OVERLAP
*******************************************************************

%HRS = $SUBSTR(%BEGIN,1,2)
%MINS = $SUBSTR(%ENDS,3,2)

%TOTAL IS STRING DP 5
%FILL= '0000'

*******************************************************************
* SUBTRACT TEN MINUTES FROM THE START TIME TO GIVE A 10 MIN OVERLAP
*******************************************************************

%HRS = $SUBSTR(%BEGIN,1,2)
%MINS = $SUBSTR(%BEGIN,3,2)

%TOTAL = %HRS * 60
%TOTAL = %TOTAL + %MINS
%TOTAL = %TOTAL - 10
%TOTAL = %TOTAL / 60

%I = $INDEX(%TOTAL,'.')
%HRS = $SUBSTR(%TOTAL,1,%I - 1)
%MINS = $SUBSTR(%TOTAL,%I)
%MINS = $ROUND(%MINS * 60.0,0)
%BEGIN = %HRS WITH %MINS

%LEN = $LEN(%BEGIN)
%LEN = 4 - %LEN
%FILL = $SUBSTR(%FILL,1,%LEN)
%BEGIN = %FILL WITH %BEGIN


*******************************************************************
* SET REPEAT EQUAL TO THE LENGTH OF THE DAY STRING
*******************************************************************

%REPEAT = $LEN(%DAYS)

%DAY1 = %DAYS

FOR EACH RECORD ON LIST P

*******************************************************************
* SET THE CONFLICT FLAG TO LOW
*******************************************************************


%CONFLICT = O

%PRESENT.ROOM = ROOM
%PRESENT.BLDG = BUILDING

*******************************************************************
* PRESENT ROOM BEING REFERRED
*******************************************************************

ROOM: IN DATA FD ROOM EQ %PRESENT.ROOM AND BUILDING EQ %PRESENT.BLDG
      AND COURSE.DAYS IS PRESENT
END FIND

FOR EACH RECORD IN ROOM

%DAYS = %DAY1

*******************************************************************
* LOOP TO CHECK EACH DAY IN THE DAYS FIELD
*******************************************************************

FOR %LOOP FROM 1 TO %REPEAT

%WEEK = $SUBSTR(%DAYS,1,1)
%DAYS = $SUBSTR(%DAYS,2)
%PRESENT = $INDEX(COURSE.DAYS,%WEEK)
```

```
*****************************************************************
* IF THE DAY IS PRESENT IN THE SEARCHED RECORD THEN CHECK THE TIME
*****************************************************************

IF %PRESENT /= 0 THEN

*****************************************************************
* CHECK THE VARIOUS COMBINATION OF POSSIBLE CONFLICTS
*****************************************************************

IF %BEGIN GT COURSE.BEGIN AND %BEGIN LT COURSE.END THEN
*PRINT 'FIRST IN BETWEEN CONFLICT'
%CONFLICT = 1
END IF

IF %ENDS GT COURSE.BEGIN AND %ENDS LT COURSE.END THEN
*PRINT 'LAST IN BETWEEN CONFLICT'
%CONFLICT = 1
END IF

IF %BEGIN EQ COURSE.BEGIN OR %BEGIN EQ COURSE.END THEN
*PRINT 'FIRST EQUAL CONFLICT'
%CONFLICT = 1
END IF

IF %ENDS EQ COURSE.BEGIN OR %ENDS EQ COURSE.END THEN
*PRINT 'LAST EQUAL CONFLICT'
%CONFLICT = 1
END IF

IF %BEGIN LT COURSE.BEGIN AND %ENDS GT COURSE.END THEN
*PRINT 'THE BIG SPAN CONFLICT'
%CONFLICT = 1
END IF


*****************************************************************
* IF THERE EXISTS A CONFLICT THEN SET A MESSAGE AND RETURN
*****************************************************************


IF %CONFLICT = '1' THEN

*****************************************************************
* IF THERE IS A CONFLICT THEN CHECK THE START AND EXPIRE DATES
* ON THE CURRENT RECORD.  IF THE START ON THE SCREEN IS AFTER THE
* EXPIRE OF THE CURRENT RECORD, OR IF THE EXPIRE ON THE SCREEN IS
* BEFORE THE START ON THE CURRENT RECORD THEN THERE IS NO CONFLICT.
*****************************************************************


  IF EXPIRE NE '' AND START NE '' THEN

     IF %QUERY:START.DATE GT EXPIRE OR %QUERY:EXPIRE LT START THEN

*     NO CONFLICT THEN
      %CONFLICT = 0

     ELSE

      JUMP TO OUTER

    END IF

 ELSE
      JUMP TO OUTER

END IF

END IF
END IF

END FOR
END FOR
OUTER: IF %CONFLICT = '1' THEN
    REMOVE RECORD FROM LIST P
    END IF
END FOR

CT.P: COUNT RECORDS ON LIST P

IF COUNT IN CT.P EQ 0 THEN
  %QUERY:MESSAGE = '** NO FREE ROOM **'
ELSE
FOR EACH RECORD ON LIST P
  PRINT BUILDING AT 20 WITH ROOM AT 25 WITH CAPACITY AT 33
END FOR
END IF

END SUBROUTINE GET.ROOM
END
-----------------------------------------------------------
*********************************************************
* FILENAME SETUPDATA IN TEMP
*                                      AUTHOR- VINIT VERMA
*                                      CREATED 10-20-86
* PURPOSE - THIS PROCEDURE ABSTRACTS DATA FROM THE COURSE
* DATABASE.  THE IMAGING FACILITY IS USED.  THE FIELDS THAT
* ARE STORED IN A RECORD ARE:
*
* COURSE RECORD
*
*           DEPARTMENT
*           COURSE.NO
*           COURSE.ID
*           COURSE.TYPE
*            SECTION
*            COURSE.DAYS
*            COURSE.BEGIN
*            COURSE.END
*            BUILDING
*            ROOM
*            INSTRUCTOR
*
*  COMMENT RECORD

*            DEPARTMENT
*            COURSE.NO
*            NUMBER
*            COMMENT
*
*
* THE EXTERNAL FILE ACCESSED BY THIS PROCEDURE IS:
*
*              U10820A.CLASS.DATA
*
********************************************************

ALLOCATE INDATA WITH OLD SEQUENTIAL DSNAME=U10820A.CLASS.DATA
BEGIN
IMAGE COURSE
  NAME IS STRING LEN 8
  COURSE.ID IS STRING LEN 5
  SKIP 3 POSITIONS
  DEPARTMENT IS STRING LEN 5
  COURSE.NO IS STRING LEN 4
  COURSE.TYPE IS STRING LEN 1
  SECTION IS STRING LEN 3
  SKIP 51 POSITIONS
IMAGE MEETING
   NAME1 IS STRING LEN 8
   SKIP 3 POSITIONS
   COURSE.DAYS IS STRING LEN 6
   COURSE.BEGIN IS STRING LEN 4
   COURSE.END IS STRING LEN 4
   BUILDING IS STRING LEN 4
   ROOM IS STRING LEN 4
   INSTRUCTOR IS STRING LEN 12
   SKIP 35 POSITIONS
IMAGE COMMENT
SKIP 8 POSITIONS
NUMBER IS STRING LEN 3
COMMENTS IS STRING LEN 77
END IMAGE

    OPEN DATASET INDATA FOR INPUT
    IF $STATUS EQ 1 THEN
        JUMP TO DONE
    ELSEIF $STATUS EQ 2 THEN
         PRINT 'OPEN ERROR: ' WITH $ERRMSG
         STOP
    END IF
READ: READ COURSE FROM INDATA

     IF $STATUS EQ 1 THEN
           JUMP TO DONE
     ELSEIF $STATUS EQ 2 THEN
           PRINT 'READ ERROR: ' WITH $ERRMSG
           STOP
      END IF
IF %COURSE:NAME EQ 'COURSE' THEN
       %DEPARTMENT =  %COURSE:DEPARTMENT
       %COURSE.NO =  %COURSE:COURSE.NO
       %COURSE.ID =  %COURSE:COURSE.ID
       %SECTION = %COURSE:SECTION
       %COURSE.TYPE = %COURSE:COURSE.TYPE

IF %COURSE.TYPE = '1' THEN
   %COURSE.TYPE = 'TH'
ELSEIF %COURSE.TYPE = '2' THEN
     %COURSE.TYPE = 'DS'
ELSEIF %COURSE.TYPE = '3' THEN
       %COURSE.TYPE = 'IS'
ELSEIF %COURSE.TYPE = '4' THEN
       %COURSE.TYPE = 'LB'
END IF

ELSEIF %COURSE:NAME EQ 'MEETING' THEN

IDENTIFY IMAGE MEETING

IN DATA STORE RECORD
       DEPARTMENT = %DEPARTMENT
       COURSE.NO = %COURSE.NO
       COURSE.ID = %COURSE.ID
       COURSE.TYPE = %COURSE.TYPE
       SECTION = %SECTION
       COURSE.DAYS = %MEETING:COURSE.DAYS
       COURSE.BEGIN = %MEETING:COURSE.BEGIN
       COURSE.END = %MEETING:COURSE.END
       BUILDING = %MEETING:BUILDING
       ROOM =  %MEETING:ROOM
       INSTRUCTOR = %MEETING:INSTRUCTOR
END STORE

ELSEIF %COURSE:NAME EQ 'COMMENT' THEN
IDENTIFY IMAGE COMMENT
IN DATA STORE RECORD
 DEPARTMENT = %DEPARTMENT
 COURSE.NO = %COURSE.NO
 NUMBER = %COMMENT:NUMBER
 COMMENT = %COMMENT:COMMENTS
```

```
END STORE
END IF
  JUMP TO READ
DONE: PRINT 'NORMAL TERMINATION'
CLOSE DATASET INDATA
END
FREE INDATA
-------------------------------------------------------------
*******************************************************
* FILENAME SETUPHOURS IN TEMP
*                                       AUTHOR- VINIT VERMA
*                                       CREATED 10-20-86
* PURPOSE - THIS PROCEDURE ABSTRACTS DATA FROM THE COURSE
* DATABASE.  THE IMAGING FACILITY IS USED.  THE FIELDS THAT
* ARE STORED IN A RECORD ARE:
*
* COURSE RECORD
*
*           MIN
*           MAX
*
*
* THE EXTERNAL FILE ACCESSED BY THIS PROCEDURE IS:
*
*             U10820A.CLASS.DATA
*
*******************************************************

ALLOCATE INDATA WITH OLD SEQUENTIAL DSNAME=U10820A.CLASS.DATA
BEGIN
IMAGE COURSE
  NAME IS STRING LEN 8
  COURSE.ID IS STRING LEN 5
  SKIP 3 POSITIONS
  DEPARTMENT IS STRING LEN 5
  COURSE.NO IS STRING LEN 4
  COURSE.TYPE IS STRING LEN 1
  SECTION IS STRING LEN 3
  SKIP 51 POSITIONS
IMAGE HOURS
   NAME1 IS STRING LEN 8
   SKIP 1 POSITIONS
   MIN IS PACKED LEN 2
   MAX IS PACKED LEN 2
END IMAGE

    OPEN DATASET INDATA FOR INPUT
    IF $STATUS EQ 1 THEN
        JUMP TO DONE
    ELSEIF $STATUS EQ 2 THEN
         PRINT 'OPEN ERROR: ' WITH $ERRMSG
         STOP
    END IF
READ: READ COURSE FROM INDATA

     IF $STATUS EQ 1 THEN
           JUMP TO DONE
     ELSEIF $STATUS EQ 2 THEN
           PRINT 'READ ERROR: ' WITH $ERRMSG
           STOP
      END IF
IF %COURSE:NAME EQ 'COURSE' THEN
       %COURSE.NO =  $SUBSTR(%COURSE:COURSE.NO,4,1)
       %COURSE = %COURSE:COURSE.ID
ELSEIF %COURSE:NAME EQ 'ENRLINFO' AND %COURSE.NO EQ 'O' THEN
IDENTIFY IMAGE HOURS
A: IN DATA FD COURSE.ID = %COURSE
END FIND
FR IN A
 ADD MIN = %HOURS:MIN
 ADD MAX = %HOURS:MAX
END FOR
END IF
  JUMP TO READ
DONE: PRINT 'NORMAL TERMINATION'
CLOSE DATASET INDATA
END
FREE INDATA
-----------------------------------------------------------
*******************************************************
* FILENAME SETDESCR IN TEMP
*                                       AUTHOR- VINIT VERMA
*                                       CREATED 10-20-86
* PURPOSE - THIS PROCEDURE ABSTRACTS DATA FROM THE COURSE
* DATABASE.  THE IMAGING FACILITY IS USED.  THE FIELDS THAT
* ARE STORED IN A RECORD ARE:
*
* COURSE RECORD
*
*           COURSE.DESCRP
*
*
* THE EXTERNAL FILE ACCESSED BY THIS PROCEDURE IS:
*
*             U10820A.DESCRP.DATA
*
*******************************************************

ALLOCATE INDATA1 WITH OLD SEQUENTIAL DSNAME=U10820A.DESCRP.DATA
BEGIN
IMAGE COURSE
  SKIP 11 POSITIONS
  DEPARTMENT IS STRING LEN 5
  COURSE.NO IS STRING LEN 4
  COURSE.DESCRP IS STRING LEN 22
END IMAGE
```

```
    OPEN DATASET INDATA1 FOR INPUT
    IF $STATUS EQ 1 THEN
        JUMP TO DONE
    ELSEIF $STATUS EQ 2 THEN
         PRINT 'OPEN ERROR: ' WITH $ERRMSG
         STOP
    END IF
READ: READ COURSE FROM INDATA1

     IF $STATUS EQ 1 THEN
           JUMP TO DONE
     ELSEIF $STATUS EQ 2 THEN
           PRINT 'READ ERROR: ' WITH $ERRMSG
           STOP
      END IF
MATCH: IN DATA FD DEPARTMENT = %COURSE:DEPARTMENT AND -
   COURSE.NO = %COURSE:COURSE.NO
END FIND

FR IN MATCH
ADD COURSE.DESCRP = %COURSE:COURSE.DESCRP
END FOR

JUMP TO READ
DONE: PRINT 'NORMAL TERMINATION'
CLOSE DATASET INDATA1
END
FREE INDATA1
--------------------------------------------------------------
*******************************************************
* FILENAME HRS IN TEMP
*                                     AUTHOR- VINIT VERMA
*                                     CREATED 10-20-86
* PURPOSE - THIS PROCEDURE CHANGES THE CODE FOR THURSDAY
* IN THE COURSE.DAYS FIELD TO 'H'.
*
*
*******************************************************

BEGIN
A: IN DATA FD COURSE.DAYS IS PRESENT
END FIND
FR IN A
%X = $SUBSTR(COURSE.DAYS,4,1)
IF %X = 'T' THEN
%Y = $SUBSTR(COURSE.DAYS,1,3) WITH 'H' WITH $SUBSTR(COURSE.DAYS,5)
CHANGE COURSE.DAYS TO %Y
END IF
END FOR
END
```

APPENDIX C

CLASS RESERVATION SYSTEM USER'S MANUAL

TABLE OF CONTENTS

Logging onto the class room reservation system

A user can access the class room reservation system via an IBM 3270 terminal which is hooked onto the network in a bisynchronous mode. The system can also be accessed via the asynchronous network, but this is not advisable due to the lower baud rate.

The following set of instructions allow a user to access the class room reservation system. The prompts in caps are from the computer, and the lower case prompts are the user's response.

The following initial prompt appears on a terminal which is ready for access -

```
OKLAHOMA STATE UNIVERSITY COMPUTER CENTER
ENTER APPLICATION NAME
m204

86.365   DEC 31          12.45.05                    PAGE 1
logon (your user id)

*** M204.0353:PASSWORD
(your password)

*** M204.0353:   U10820A    U10820A      LOGIN 86   DEC 31    12.46
open sched

*** M204.0620: FILE SCHED OPENED
*** M204.1203: SCHED WAS LAST UPDATED ON 86.364    DEC 30    22.58.26

include class
```

At this stage the primary options menu for the class room reservation system will be displayed.

Program function keys (PF keys)

The program function keys perform various user interface functions. The functions assigned to the keys are listed below. To access a program function key, press the ALT key and simultaneously press the number of the needed function key.

pf1 -	reserve a room
pf2 -	delete a room reservation
pf3 -	exit to primary options menu
pf5 -	data retrieval
pf6 -	clear screen
pf7 -	scroll back one entry
pf8 -	scroll forward one entry
pf9 -	department query

Primary options menu

Once the user has successfully opened the SCHED file and included the class program, the following primary options menu is displayed.

```
--------- Oklahoma State University Class Schedule System -------------

                1     general users (departments)
                2     super user (registrar)
                3     room inventory database
                4     room vacancy
                5     exit
```

To make a selection, the user needs to move the cursor to the desired option number and then press the enter key. Each of the options on the primary menu are described below.

option 1

This option provides data retrieval functions only, no database updates can be performed via this window. Data retrievals are performed on the class schedule data base. The departments on campus can utilize this option to check up the status of room reservations. The user interface screen for this option is displayed below.

```
------ Oklahoma State University Class Schedule System -------------

  id number    :        instructor:      days:         min:
  department   :        building  :      start:        max:
  course number:        room      :      end   :
  type         :        section   :      hours:
  title        :                         undecided hours:
                        begin(yy-mm-dd):            expiry(yy-mm-dd):
  comment:


pf3 exit  pf5 refresh  pf6 clear  pf7  --  pf8 --    pf9 dept. query
```

The prompts on the screen correspond directly to the entries in the class schedule book.

id number	-	course identification number
department	-	department code
course number	-	four digit course number
type	-	TH/LAB/IS/DS
title	-	course description
instructor	-	instructor's name
building	-	building code
room	-	room number in the building
section	-	course section
days	-	days of the week the course is offered (MTWHFS)
start	-	commencement time for the course (army time)
end	-	course end time (army time)
hours	-	credit hours for the course
min	-	minimum # of hrs. for course ending in 0
max	-	maximum # of hrs. for course ending in 0
undecided hour	-	for TBA meeting
begin(yy-mm-dd)	-	date of commencement of course
expiry(yy-mm-dd)	-	date of termination of course
comment	-	comments on the course

option 2

This option is meant for the registrar's office. A user can perform data retrievals and room reservations via this window. The user interface screen for this option is identical to the option 1 screen except two extra pf keys are provided. Pf1 for reserving a room, and pf2 for deleting a room reservation.

option 3

The room inventory database keeps information on the rooms on campus. The database stores the room type and capacity for each room. A menu provides options to add or delete rooms. Information for a specific room can be pulled up. Information from this database is utilized to perform ad hoc reservations.

option 4

The room vacancy option provides advanced data retrieval operations from the class schedule, and room inventory databases. A user can input the start date, end date, days the course will meet, begin time, course end time, number of seats needed and the type of room needed. The program will query the room inventory database for the rooms which match the type and capacity of the request. Once a list of all such rooms is obtained, the class schedule database is queried for time and date conflicts. Finally the program comes up with a set of rooms on campus which will satisfy the request.

option 5

This option allows the user to exit form the class room reservation system.

<u>Data retrievals</u>

The pf5 key needs to be pressed to invoke any data retrieval function. The cursor should be kept on the field for which the retrieval is to be performed. A list of the retrieval fields for the different options on the primary menu follows -

- options 1 and 2 of the primary menu

    1. course id number
    2. department and course number
    3. department name
    4. building and room number
    5. instructor's name

- an option to query the database for all courses offered by a department, sorted by day of the week, and time of the day.

- option 3 of the primary menu

    1. building
    2. building and room

- option 4 of primary menu

All prompt fields under this option need to be filled out before a data retrieval may be executed.

<u>Reservations</u>

This option allows a user to make a room reservation. The reservation request are categorized as follows -

a. room reservation for the full semester

b. ad hoc room reservation with commencement and expiry date.

The course id number is the primary key for the class schedule database. Every room reservation request has to maintain the primary key uniqueness constraint. Duplicate course id numbers are rejected by the system.

a. room reservation for the full semester

This type of a reservation request requires the user to utilize the data retrieval functions. The user should have the following information before making such a reservation -

- course id number
- department
- course number
- type
- title
- instructor
- section
- days of the week for meeting
- start time
- end time
- undecided hour if TBA type
- minimum credits for course number ending in zero
- maximum credits for course number ending in zero
- comments

The objective is to come up with a suitable room for the above course. The room vacancy program needs to be executed to come up with a set of eligible rooms. The user can choose one of the rooms on the list. Once the building and room numbers are available the reservation can be made. Enter the information for the prompted fields and press the pf1 key. An 'entry stored' message signals a successful reservation.

b. room reservation for an ad hoc request

Again the user needs to have all the previously mentioned information ready, except the building and room numbers. This reservation requires the user to input the start date and expiry date. Again prior to making the reservation the room vacancy program needs to be executed to come up with a list of eligible rooms. Besides checking for the time conflicts the room vacancy program also checks the date conflicts. Once the building and room number is available the reservation can be made. Enter the prompted information and press the pf1 key. An 'entry stored' message signals a successful reservation. This request is dynamic in nature. Each time the reservation system is invoked, all reservations which have an expiry date less than the system date are purged. Thus the rooms are dynamically freed for ad hoc requests.

Deletions

To delete a certain room reservation, the user needs to enter the course id number and press the pf2 key. If the needed id number is located, a deletion is performed, otherwise a 'no match' message is displayed.

Room inventory database

The room inventory database houses information on the available rooms on campus. The user interface screen for the database is described below.

```
-------------------- OSU Room Inventory Database ----------------------

         building:
         room    :
         type    :
         capacity:


                                        1 - case study
                                        2 - auditorium
                                        3 - fixed
                                        4 - table & chairs
                                        5 - arm
                                        6 - bolted arm
                                        7 - lab


pf1 reserve  pf2 delete  pf3 exit  pf5 refresh  pf6 clear pf7  --  pf8
```

The database search key is building and room. If the building and room entries are filled up on the screen and the pf5 key is pressed, then the type and capacity for the room will be displayed. Information about all rooms in a specific building may be obtained by entering the building code and pressing the pf5 key. The needed information is put on a scrollable list, and the pf7 and pf8 keys provide the needed scroll functions. To save an entry in the database, enter the information for the four prompts on the screen and press the pf1 key. If an entry with the same building and room codes exists, it will be updated with the current type and capacity, otherwise a new database record is stored. An 'entry stored' message will be displayed to signal a successful insertion. To delete a information for a specific room, enter the building code and the room number and press the pf2 key. If a match is found, then the record is deleted, otherwise a 'no match' message is displayed. The pf6 key may be used to clear the screen input fields at any stage.

Room vacancy

This program queries the class schedule database and the room inventory database, to come up with a list of rooms which satisfy certain time and date constraints. The user interface screen for this option looks as follows -

```
-------------------- OSU Room Inventory Database ----------------------

         start (yy-mm-dd) :
         expire (yy-mm-dd):
         days             :
         begin time       :
         end time         :
         capacity         :
         room type        :


                                      1 - case study
                                      2 - auditorium
                                      3 - fixed
                                      4 - table & chairs
                                      5 - arm
                                      6 - bolted arm
                                      7 - lab


             pf5 query            pf3 exit
```

Each of the prompted fields are explained below.

start	- date of commencement of the request
expire	- expiry date for the request

days	- days of the week for reservation
begin time	- starting time for the request
end time	- ending time for the request
capacity	- seating capacity for the request
room type	- one of the 7 codes from above

The user needs to input all the prompted fields, and then press the pf5 key. The program comes up with a list of rooms which will satisfy the request. If no rooms exist which can satisfy the request, then a message to that effect is displayed. The pf3 key gets the user back to the primary options menu.

Logging off the system

To log off the system the user needs to place the cursor on option 5 of the primary options menu and press the enter key.

Test cases

1) Test cases for query fields

   a. retrieval via course id number - 12296
   b. retrieval via department and course number - comsc 4113
   c. retrieval via department - comsc
   d. retrieval via building and room - ms 222
   e. retrieval via instructor's name - smith

2) (N.B. The conflict verification is activated if the building, room, start,end and days fields are filled in)

   To test the reservation routine complete information about room MS 222 was obtained using the data retrieval functions. The busy hours for the room are listed below.

W	1230 - 1320
M	1530 - 1620
MWF	1030 - 1120

MWF	0930 - 1020
T	0900 - 1015
T	1030 - 1145
M	1900 - 2200

(N.B. The reservation routine subtracts 10 minutes from the incoming start time for the needed overlap between meetings)

a. full semester reservation

```
course id : 99999     instructor: john       days   : MH
department: comsc     building  : ms         start  : 1045
course #  : 1111      room      : 222        end    : 1200
type      : th        section   : 001        hours  : 1
title     : basic

comment: $5.00 charge for micro computer usage
```

press the pf1 key to make the reservation

response: conflict (1045 - 10 = 1035 falls between a course meeting)
(start time falls between a course meeting time)

b. all entries stay the same except start time: 0800
end time : 1000

response: conflict (1000 falls between a course meeting)
(end time falls between a course meeting time)

c. all entries stay the same except start time: 0940
end time : 0900

response: conflict (0940 - 10 = 0930 is the start time for existing course) (start time equal to the commencement time for existing course)

d. all entries stay the same except start time: 0800
end time : 1030

response: conflict (1030 is the start time for an existing course)
(end time equal to the commencement time for existing course)

e. all entries stay the same except start time: 0800
end time : 1300

response: conflict (a course meets between 0800 and 1300)
(start and end time span the meeting time of a course)

f. all entries stay the same except start time: 0700
end time : 0800

response: room reserved

g. clear the screen using the pf6 key, and recall the entry for course id number: 99999

h. press the pf2 key to delete the course

i. try to refresh the screen for course id number: 99999
response: no match

3) ad hoc reservation request for temporary room assignment

a. query the room inventory database for building: MS
room : 310

response: type : case study
capacity: 63

b. query the class schedule database via the general user's option for the meetings in MS 310. The scroll key should be used.

```
TH      1230 - 1345
TH      1400 - 1515
MW      0830 - 0920
MWF     1430 - 1520
M       1830 - 2120
W       1830 - 2120
MWF     1230 - 1320
TH      0730 - 0845
MWF     0730 - 0820
MWH     1530 - 1620
MWF     0930 - 1020
MWF     1330 - 1420
MWF     1130 - 1220
MWF     1430 - 1520
TH      1030 - 1145
TH      0930 - 1020
MWF     1030 - 1120
T       1830 - 2120
```

c. enter the room vacancy option to check up on an empty slot

```
start   : 87-07-01
expire  : 87-07-01
days    : MH
begin   : 0600
end     : 0700
capacity: 30
type    : 1
```

press the pf5 key.
response: a list of rooms which are available at the needed time,days and date is displayed. MS 310 is a valid choice. Now a reservation can be made via the super user's option.

4) ad hoc reservation request for temporary assignment, with expiry date active for the automatic purge process.

a.

```
 course id : 99999     instructor: john       days   : MH
 department: comsc     building  : ms         start  : 0700
 course #  : 1111      room      : 222        end    : 0800
 type      : th        section   : 001        hours  : 1
 title     : basic     begin : 87-01-aa    expire: 87-01-bb
                       (yy-mm-dd)                  (yy-mm-dd)
                        aa - today's day - 2
                        bb - today's day - 1
                        (e.g if the date today is 86-12-31 then
                             aa - 28
                             bb - 29)
 comment: $5.00 charge for micro computer usage
```

press the pf1 key to reserve the room.

response: the room is reserved

b. press the pf6 clear key, and recall the room reservation for course id number: 99999

c. return to the primary options menu

d. re-enter the super user's option. recall the entry for course id number: 99999
response: no match. The automatic delete process checked the expiry date for the request and purged it.

5) check the start date logic for proactive reservations

a. select the super user's option and make the following reservation

```
 course id : 99999     instructor: john       days   : MH
 department: comsc     building  : ms         start  : 0600
 course #  : 1111      room      : 310        end    : 0700
 type      : th        section   : 001        hours  : 1
 title     : basic     begin : 87-07-04    expire: 87-07-06
                       (yy-mm-dd)                  (yy-mm-dd)
 comment: $5.00 charge for micro computer usage
```

b. select the room vacancy option

```
start     : 87-07-05
expire    : 87-07-07
days      : MH
begin     : 0600
end       : 0700
```

```
   capacity : 30
   type     : 1

press the pf5 key to come up with a set of available rooms

response: MS 310 is not in the list.

c.  keep all entries the same except     start: 87-07-03
                                         end  : 87-07-04

response: MS 310 is not in the list.

d.  keep all entries the same except     start: 87-07-03
                                         end  : 87-07-08

response: MS 310 is not in the list.

e.  keep all entries the same except     start: 87-07-01
                                         end  : 87-07-03

response: MS 310 is a valid choice.

f.   select the super user's option.  Check out the reserve logic
     by trying to reserve a through e above for course id: 79999
     response: only choice e will end up in a valid reservation.
               (comment for 79999 and 99999 are identical message
                will be displayed, and the entry will be reserved)

g.   delete course id: 99999 and course id: 79999
```

VITA 2

Vinit Verma

Candidate for the Degree of

Master of Science

Thesis: A COMPARATIVE STUDY OF A HIERARCHICAL DATABASE MANAGEMENT SYSTEM WITH RELATIONAL-LIKE DATABASE MANAGEMENT SYSTEM FOR A SCHEDULING PROBLEM

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Lucknow, India, October 30, 1962, the son of Dr. Indrapal Singh Verma.

Education: Graduated from Mayo College, Ajmer (India), in May 1981; received Bachelor of Science Degree in Mathematics, Physics and Chemistry from University of Delhi in May 1984; completed requirements for the Master of Science Degree at Oklahoma State University in July 1987.

Professional Experience: Research Assistant, and Applications Programmer, Department of Grants Contracts and Financial Administration, Oklahoma State University, January, 1985, to April, 1987; member Association of Computing Machinery, New York.