PERFORMANCE CONSIDERATIONS IN RELATIONAL AND HIERARCHICAL DATA BASE MANAGEMENT SYSTEMS

by

Mary Kathleen Tod
BSc., Queens University, 1971

A thesis submitted in partial fulfillment of
the requirements for the degree of
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming to
the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
January, 1980

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date January 25, 1980

ABSTRACT

This paper will examine two data base management systems; IMS, an example of a hierarchical data base management system and System R, a relational system. Each system will be described in general terms followed by a discussion of their relative performance. A claim is made that IMS, because of its structure and the procedural nature of its language can be more efficient than System R. Some examples are given to illustrate this claim.

There is an increasing need today for readily available information; thus more and more information is being stored in disk files to facilitate rapid retrieval. A relational approach to data base management permits a high-level non-procedural interface for the user, which is important if more people require access to information in a flexible fashion and can not afford or wait for traditional application development. If however a relational data base management system is not as efficient as hierarchical or network systems then new retrieval methods must be found.

Associative processors are introduced as a possible solution to this problem and three examples are discussed. RAP, a relational associative processor developed at the University of Toronto, has been used in a performance study to demonstrate the dramatic performance improvements offered by associative processors. This is also discussed.

C O N T E N T S

Cont'd...

1.0 INTRODUCTION

Today information processing is vital to any large or small organization. Information processing includes the ability to extract information, to maintain and update information, to analyse information and to rely on the accuracy and currency of this stored information. Information is required to perform the daily business transactions, to plan for the future and to control the development towards the future.

Information processing is essential to support management and decision making roles. As a result there are trends in business towards large corporate data bases which are accessible to non-dp personnel using high-level query languages. Thus there is a movement away from designing specialized applications to access data in the most efficient way possible.

Online banking (including automated funds transfer), automated manufacturing, inventory control, personnel and payroll applications all involve access to data bases. With this trend to integrated data base solutions to business systems, the size of data bases and hence the amount of data stored on machine readable media is increasing. At the same time computer hardware costs are decreasing and this will lead eventually to an age where home terminals are a reality. Such terminals will provide numerous functions relying on large quantities of shared data.

With the growing number of areas accessing large data bases there is a demand for efficient data retrieval and flexible high-level tools for data access. The needs for data have changed dramatically since the early days of computing. The next section will discuss how data base management systems (DBMS) have changed over the years to support such new requirements.

2.0 EVOLUTION OF DATA BASE MANAGEMENT SYSTEMS

The first systems that processed information on a large scale used punched cards, the ultimate in sequential files. This medium was slow and unsuitable for large volumes of data. In addition, the fixed size of the card was often inappropriate although techniques were developed to overcome this restriction.

In the early 1950's magnetic tape was introduced as a faster, more flexible storage medium. Problems of fixed length were solved since variable length records could be maintained on tape; however, files remained sequential.

As more applications were written it was recognized that certain operations recurred, for example: sort, delete a record, add a record, generate a report. It seemed logical and very useful to generalize these file handling functions [1]. Also data sharing and integrated files were considered in an attempt to reduce the overall programming burden. The new field of data management had begun.

Having provided generalized functions for file maintenance, sorting and report generation, the computing community began exploring the concept of data sharing. It was recognized that there was little advantage to a business if one department hoarded its own data forcing a second department to acquire and maintain its own set of overlapping data. This led to a need to define data in a file in

some standard way so that two (or more) parties could access their relevant pieces of information. Hence data definitionslanguages were born, examples of which are, COMPOOL, JOVIAL and COBOL. With these new generalized methods of defining data, report generation toolssbecame more sophisticated in accessing the data. Some examples of these report generation tools are RPG, MARK I and MARK II, GIRLS and FFS.

If we look at the parallel developments in storage technology we see that storage media had undergone a change from punched card technology to magnetic tape. Thus the access speed was significantly increased but the access method remained sequential. Files were typically read from beginning to end, record by record, in order to extract the necessary information.

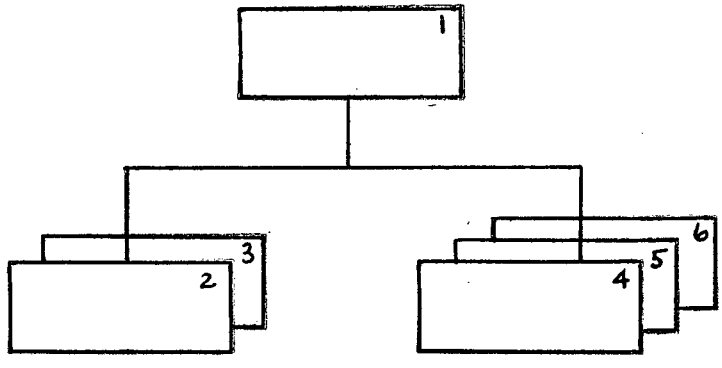
Because pieces of information are inter-related it became necessary to add facilities for data structuring to the list of data management facilities. The first structuring technique that was used in DBMS was hierarchic. "One of the main advantages of these structures is their inherent computer efficiency with regard to storage space and peripheral storage accesses. Another advantage is a correspondence of hierarchic records to the report structures often required by commercial enterprises." [2]

A hierarchy is a very natural structuring and, to a generation brought up thinking and using sequential access methods, the ability to flatten a hierarchy into a sequential storage medium proved its

worth. For example, the first IMS system demanded requests for data in a top to bottom, left to right fashion corresponding with the way the hierarchy was flattened for storage on tape.

Greater flexibility in data access was brought about with the widespread availability and increased reliability of direct access storage devices. Random access became truly feasible. This led to developments of network structures in data base management systems; IDS developed by General Electric, TOTAL and ADABAS are examples. The Codasyl DBTG played a major role in the thrust toward network systems.

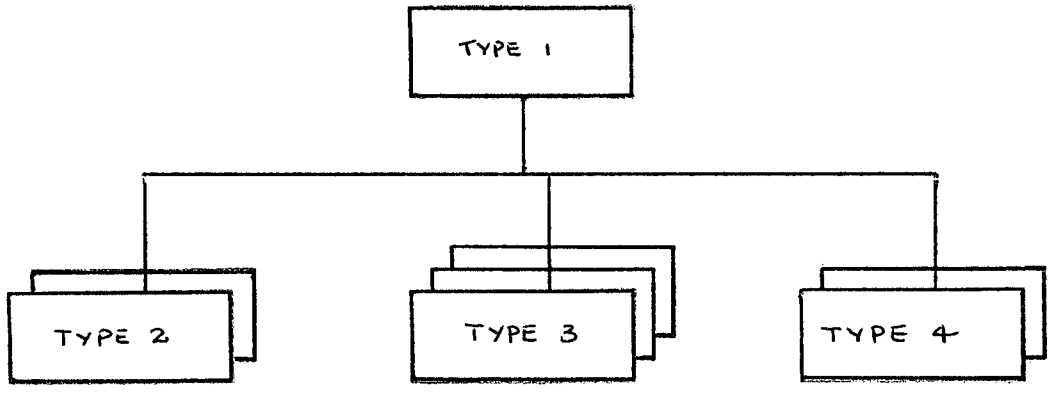
Hierarchic systems also benefitted from direct access methods. A perfectly flat structure was no longer necessary when implementing a hierarchy.



Originally stored and accessed in the order 1, 2, 3, 4, 5, 6, this hierarchy could be accessed randomly if direct access pointers are used.

It is interesting to note that IMS developed from a hierarchy implemented in a "flat" fashion to one with pointers and eventually became a network data base management system, thus taking advantage of the changing storage and access methods.

Another development in data base management systems stemmed from the need to define multiple views of one set of data. Schemas or logical data bases allow one program to access one subset of data from the entire data base. In this example:



a logical data base could be created consisting only of segments of type 1, 3, 4. In this case the program need not know about type 2 records and in fact is not permitted to access type 2 records. Hence security measures are introduced.

There are still problems to be solved. Improved data independence, high-level access languages, completeness of the data base model and improved data security and integrity are some areas of concern.

With data independence it is possible to change the physical storage structure of the data or the data access strategy without affecting any application programs. Current data base management systems provide some independence but as demonstrated in later sections it is quite limited.

How easy to use and flexible are the current data base accessing languages? The CODASYL report [3] describes host-language and self-contained systems. A host-language system permits data base access from a host-language such as PL/1 and subroutine calls to perform a well defined set of operations against the data base (eg. retrieve, delete, insert). A self-contained system provides an interface suitable for a non-programmer. This interface usually consists of a highly specialized language that is as english-like as possible. Because of the increased demand for flexible data access to support business functions, better data base accessing languages must be developed.

Does the data base model have the ability to represent real-world relationships or is the model too confining? The hierarchic and network data models impose constraints on the ability to represent relations and thus on the ability to model the real-world.

As data sharing increases so does the importance of security and data integrity, and the size of any one data base. Hence efficient access becomes even more important.

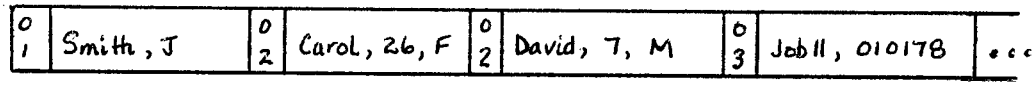
Data independence, high-level languages, the ability to represent relations, security and integrity and fast access are important concerns. Relational data base management systems may solve some of them.

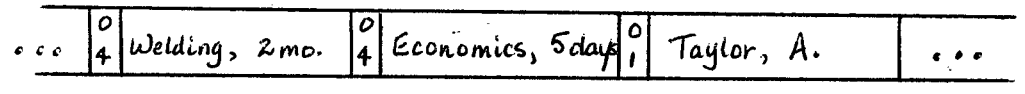
In the following sections a hierarchical DBMS and a relational DBMS will be compared. A discussion of associative processors in Section 7 will show that they permit faster access than conventional random access disks.

3.0 CURRENT DATA BASE MANAGEMENT SYSTEMS -- AN EXAMPLE

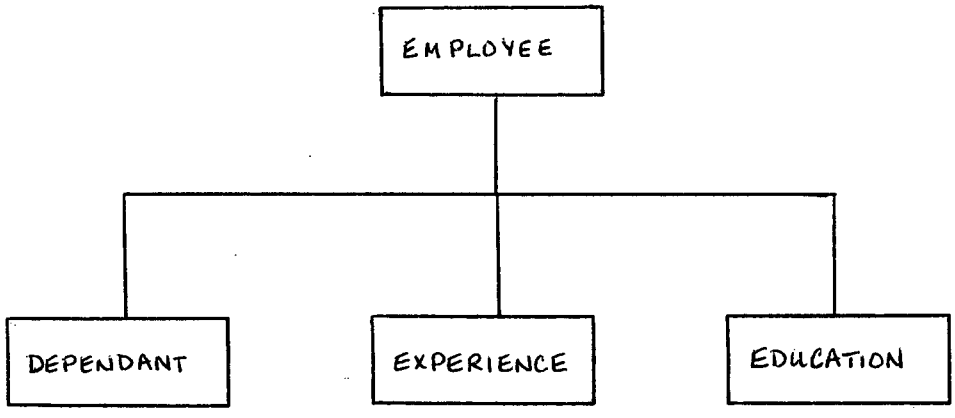
In order to illustrate the current state of commercially available DBMS IMS, an IBM product, will be discussed. The concepts introduced can, I feel, be generalized to other systems such as TOTAL, ADABAS and SYSTEM 2000.

IMS is a hierarchical data base management system, that is, it uses a hierarchical structure to represent whatever world it models (eg. business organization, personnel information, inventory systems). If we look back to the days of tape files or card files we can see the beginnings of hierarchical data management. Suppose we have an employee with certain dependants, job experience and educational skills. In early days this information could have been represented as a series of records, each with a record type stored in a predefined order:





In IMS this information would be represented with the following diagram:



In this example, the "world" consists of any number of employees and for each employee, three relationships:

1. dependants of the employee.
2. job experience of the employee.
3. educational skills of the employee.

Notice that these relationships are not explicitly stored in the data base. Rather, they are implied by the structure of the data base. Also, if the employee has no dependants this information is represented by having no dependant records in the data base. The information explicitly stored is the various data fields within each record -- or segment to use IMS terminology. In our example the boxes labelled DEPENDANT, EXPERIENCE, EDUCATION and EMPLOYEE are

segments. The EMPLOYEE segment may contain such information as SEX, SALARY, BIRTHDATE, ADDRESS, etc. Similarily the segment EDUCATION may contain such information as GRADE, EDUCATIONAL INSTITUTION, COURSE DATES, COURSE COST and so on. In IMS the basic unit of retrieval, update, insertion and deletion is the segment.

IMS is a sophisticated data management tool and it is definitely outside the scope of this discussion to fully describe all of IMS's many features. However to prepare the way for further discussion it is necessary to describe the basic data manipulation commands and logical data bases, and then to discuss IMS storage structures in some detail. Finally, several examples will be given to demonstrate how the typical programmer uses IMS efficiently.

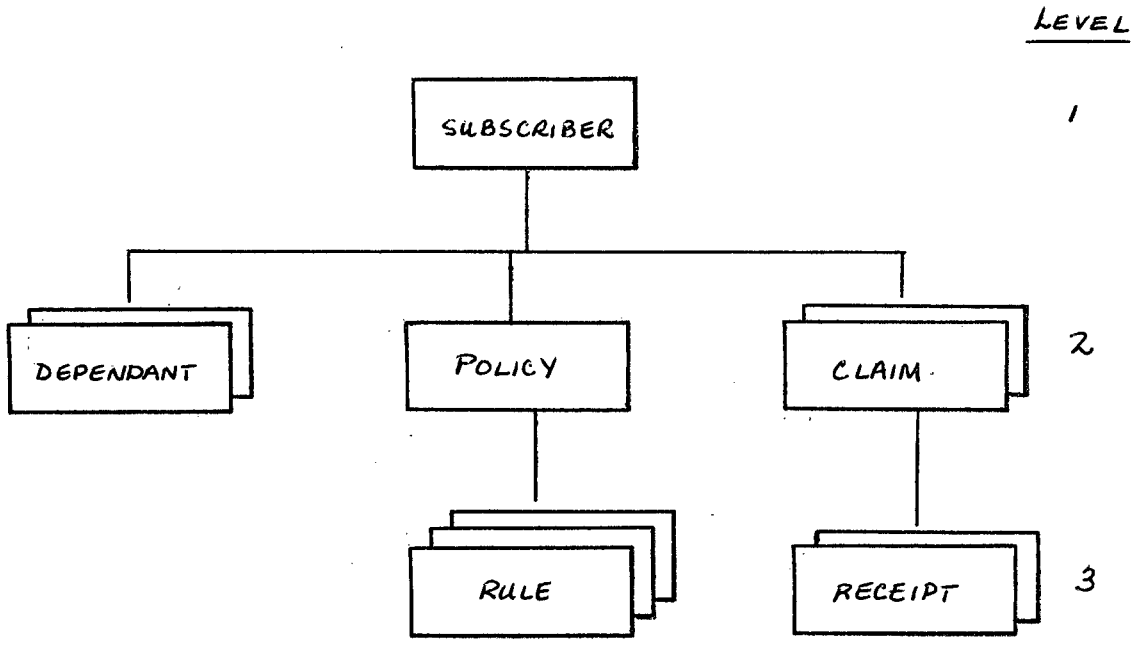
3.1 DATA BASE ACCESS COMMANDS

IMS provides facilities to retrieve, update, insert and delete segments from a data base. Included are:

IMS COMMAND		FUNCTION
GU	(GET UNIQUE)	direct segment retrieval
GN	(GET NEXT)	get next segment in sequence
ISRT	(INSERT)	add a new segment
REPL	(REPLACE)	update existing segment
DLET	(DELETE)	delete a specific segment

These commands are best explained with some examples.

In order to maintain some consistency within this paper I will introduce an application with which I have been involved. This particular system is an insurance application containing information on policies, subscribers (those who have policies with the company) and claims. The exact details of the data bases will not be given here but where relevant some of the structures will be shown.



This structure represents an individual (SUBSCRIBER segment) who is enrolled with the insurance company for a certain type of coverage (POLICY and RULE segments). This individual has a wife and children (DEPENDANT segment) and over the years has submitted certain claims to the company (CLAIM and RECEIPT segments). The level of this hierarchy is three. Each data base must have a root segment and the root segment must have a key field. In this data base the SUBSCRIBER segment is the root segment and its key is a social insurance number (SIN). Non-root segments do not necessarily have keys.

GET UNIQUE

In IMS a GU provides random access to a segment with a given key field value. When issued a GU ignores any current positioning information maintained by IMS. Instead, using either an index or a hashing algorithm IMS finds the specified segment directly. A GU is normally used to access a new root with no relationship to the root previously accessed.

Given a specific SIN in the SUBSCRIBER data base IMS will position itself for processing that root and all of its child segments. Child segments are DEPENDENT, POLICY, RULE, CLAIM and RECEIPT segments pertinent to that root.

GET NEXT

This command directs IMS to move forward in the data base from it's current position. To understand its use one must understand the implied ordering of an IMS data base: top to bottom and left to right in the hierarchy. For a given SUBSCRIBER, as established by a GET UNIQUE, successive GN commands would retrieve DEPENDANT segments, then the POLICY segment (assume only one policy per individual), all RULE segments, then the first CLAIM segment, all RECEIPT segments for that CLAIM, the second CLAIM segment and its RECEIPT segments, etc. GET NEXT commands may be qualified. For example if an inquiry is issued for all claims for a given subscriber, the following could be done:

1. GU for the given SUBSCRIBER, its key field (SIN) must be supplied.
2. GN qualified to indicate that only CLAIM segments are to be retrieved.

Note that step 2 would be repeated until IMS indicated no more claims for that subscriber exist.

INSERT

Whenever a new segment must be added to the data base the ISRT command is used. If a dependent segment is added its parent must first exist. IMS provides certain rules for insertion. For example, a last-in-first-out rule can be applied or segments may be inserted so that their key field is in a specific order.

REPLACE

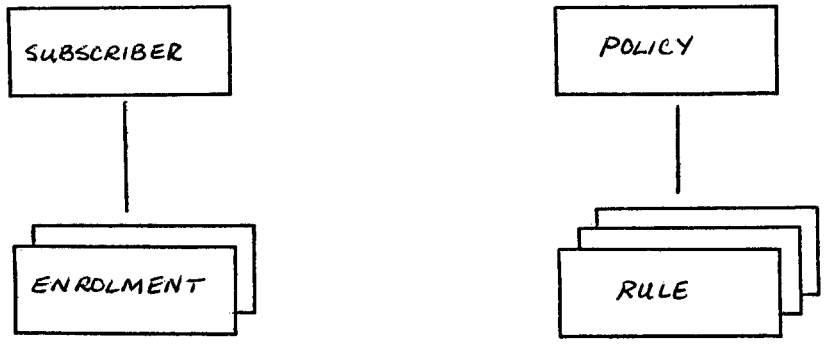
REPL is used when updating fields in a segment. The segment must have been previously accessed by a special type of retrieval command, GHU or GHN. These commands operate exactly as GU and GN but perform the extra function of marking the segment for subsequent replacement (or deletion). When a segment is marked in this fashion it is unavailable for other access.

DELETE

DLET is used to delete segments from the data base. When a segment is deleted all of its children are also deleted. Note that the GHU and GHN are also used with the delete function.

3.2 LOGICAL DATA BASES

During the early design phases of the insurance application the data base design underwent a significant change. Initially there were thirteen unrelated physical data bases. All relational information was stored as data fields in the physical data base or as logic within some particular program. For example:



Here we have two separate data bases, the SUBSCRIBER and the POLICY data bases. In the ENROLMENT segment there is a field called POLICY-NUMBER. If we attempt to find details on all policies in which a subscriber is enrolled we would construct the following program:

1. Issue a GU to the SUBSCRIBER segment giving the social insurance number of the particular subscriber.
2. Do until no more ENROLMENT segments.
    3. Get the next ENROLMENT segment.
    4. Using the POLICY-NUMBER stored in that segment issue a GU to the POLICY segment in the POLICY data base.

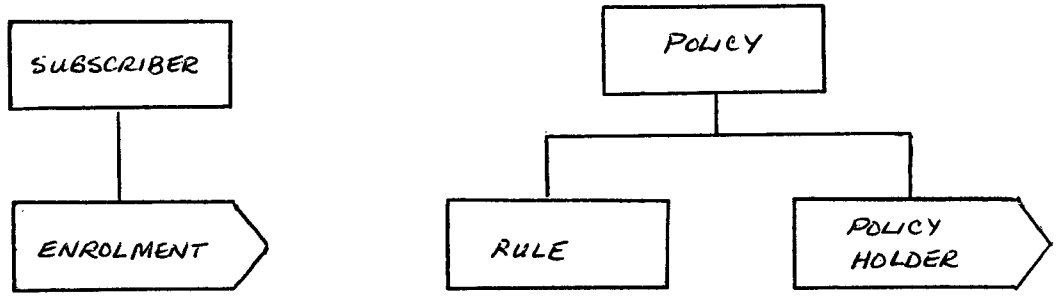
There are some problems. One is that duplicate information exists; the POLICY-NUMBER must be maintained in two separate places. A second problem is that the programmer must be aware of how the data is physically separated in order to access the correct information. The programmer must know:

1. that the policy-number is the key in the policy data base.
2. that the different IMS calls are issued to two different physical data bases. This is required in the IMS command.
3. that an implied relationship exists between the SUBSCRIBER data base and the POLICY data base through the ENROLMENT segment.

Now suppose information is required about a specific policy and the various people who subscribe to that policy. The data bases

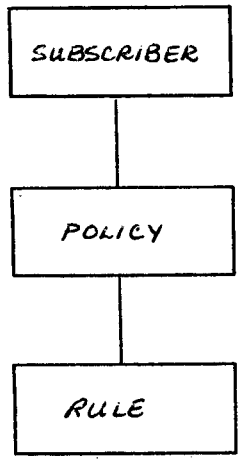
are not organized to suit this query, thus presenting a third problem. In fact to satisfy such a query the entire SUBSCRIBER data base would have to be scanned!

Because there were varied and conflicting requirements for access to information in the insurance application, the data bases were redesigned using logical data bases instead of physical data bases. In IMS logical data bases are constructed through the use of pointers. Consider the previous example with SUBSCRIBERS and POLICIES.

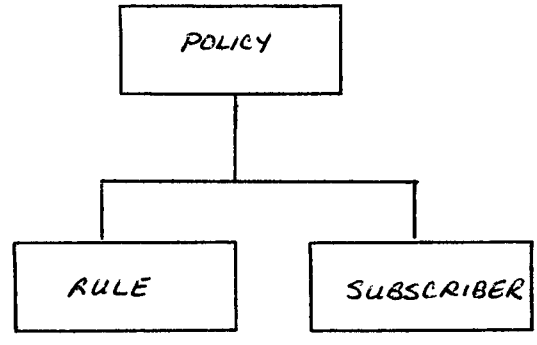


The "arrowed" segments contain pointers from one physical data base to another. Contrast this with strictly physical data bases in which data (the POLICY-NUMBER)is used to represent a relationship. Here the SUBSCRIBER data base is logically connected via the ENROLMENT segment to the POLICY data base and the POLICY data base is logically connected via the POLICY HOLDER segment to the SUBSCRIBER data base.

Through the use of logical connections a programmer can now equally readily find all policies for a given subscriber or all subscribers for a given policy. This is possible because logical data bases in IMS provide more tailored views of the data. One programmer can work with view A and another with view B, without knowing that physical separation of data exists.



VIEW A



VIEW B

Someone, such as a data base administrator (DBA), will describe the physical data bases to IMS and include in that description information about the different pointer segments (what physical data base they reside in and what physical data base segment they point to for example). A DBA will also describe the logical data bases as represented in view A and view B so that IMS can handle the physical data separation and appropriate pointer updating.

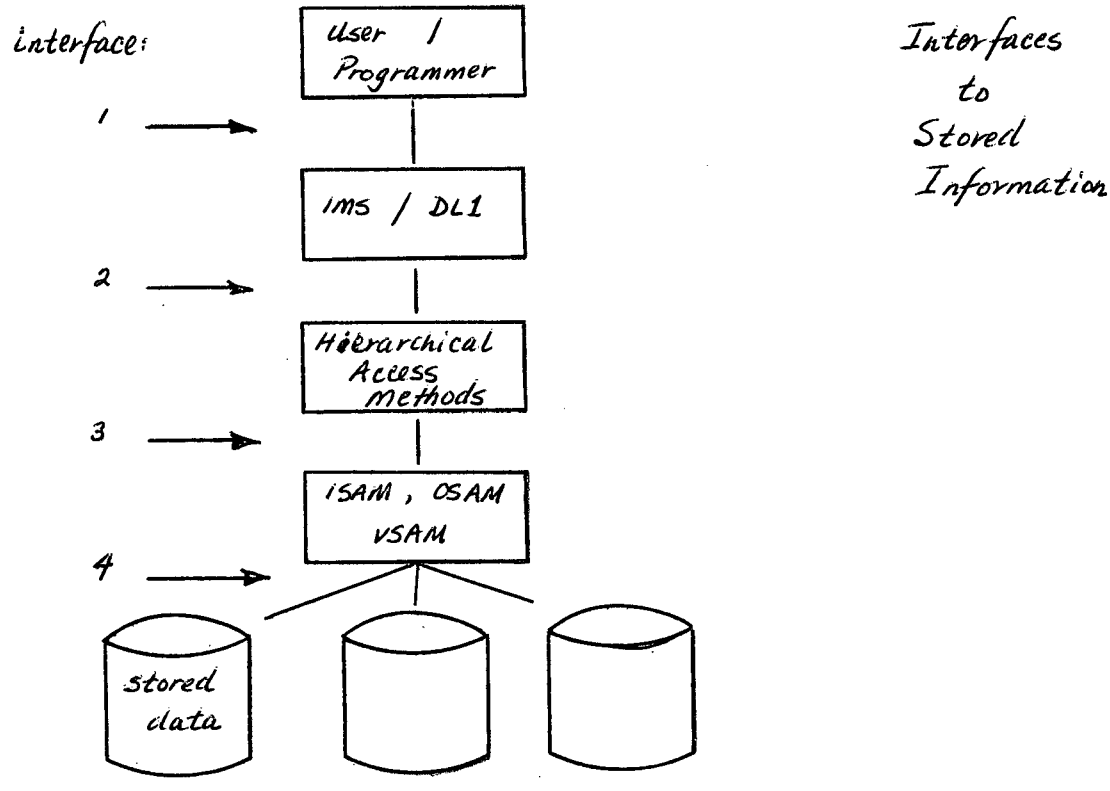
When a new subscriber is added with a given policy IMS manages the pointers and can automatically create a POLICY HOLDER segment once an ENROLMENT segment is created. Thus improved data integrity is provided.

In implementing logical data bases one must be aware that there is overhead involved on the part of IMS in maintaining pointers. Each proposed pointer segment must be examined with particular emphasis on how often it is used and the mode of use, batch or online. Information that is required frequently for online use is a prime candidate for a pointer segment.

3.3 IMS STORAGE STRUCTURES

When using a data base management system such as IMS the programmer need not be concerned with the physical storage structure and the access to such storage. Instead the interface is through the data management language that has been provided. This language may consist of a host language such as PL/1 and some predefined subroutine calls or it may be a specialized self-contained language designed only to support data access (often these languages are as english-like as possible).

The data management language must then interface with a specific access method which in turn interfaces with the physical records. For IMS this may be described with the following diagram:

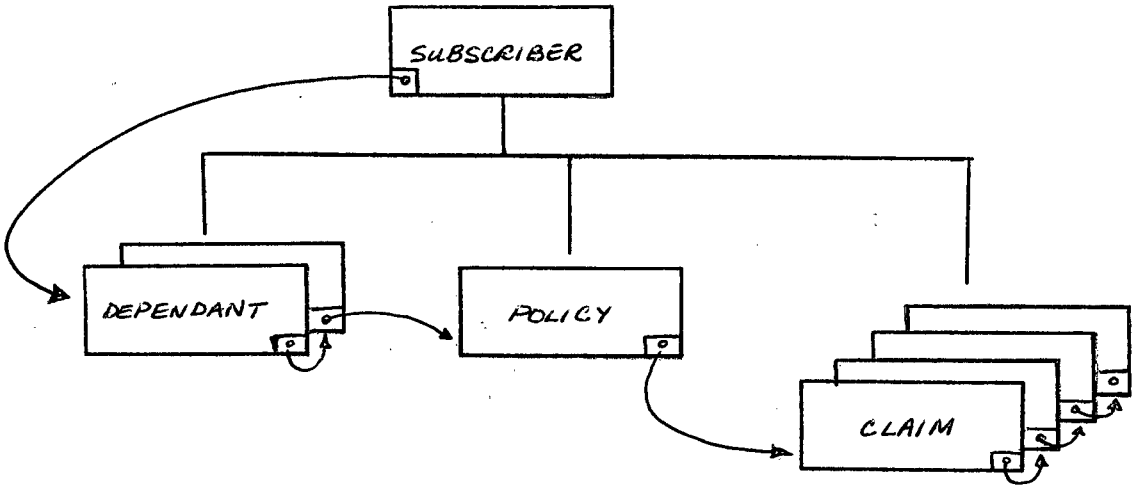


The first interface was generally considered in the explanation of the various IMS calls and the use of logical data bases.

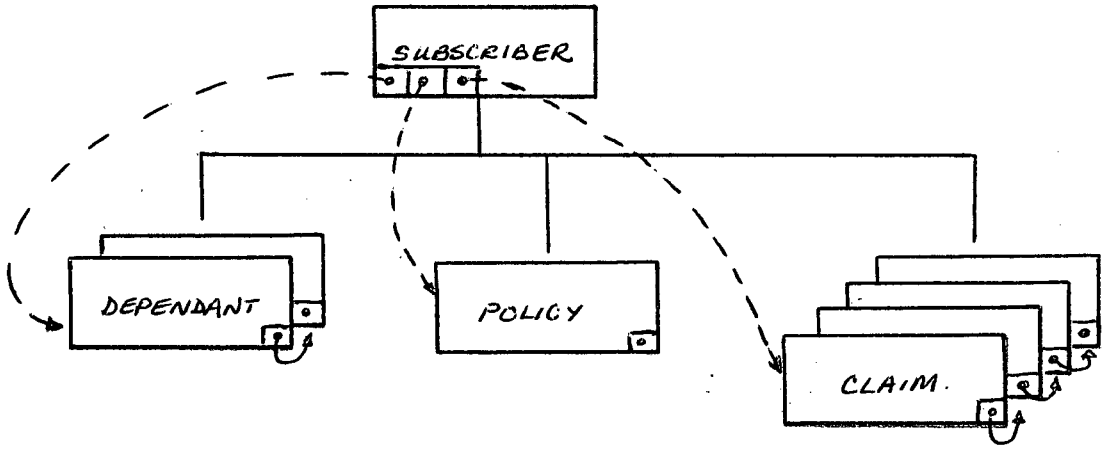
Through interfaces 2 and 3, the program and the data base language function without any knowledge of device dependent details and with a hierarchical picture of the stored segments. A program "does not know (a) anything about physical records (blocks); (b) how stored fields are associated to form stored records ...; (c) how sequencing is performed (eg. it may be by means of physical continguity, an index, or a pointer chain); or (d) how direct access is performed". [4] Knowledge of such details is provided at interface 4.

There are four hierarchical access methods: HSAM, HISAM, HDAM and HIDAM. HSAM and HISAM are strongly sequential in nature, in fact HSAM can exist on tape. For direct access, most existing IMS data base implementations use either HDAM, hierarchical direct access method or HIDAM, hierarchical indexed direct access method. HDAM and HIDAM use hierarchical and/or child-twin pointers to link segments together. These pointers provide a mechanism for linking all segments within a root occurrence and for linking root occurrences.

The subscriber data base will be used to indicate how these two types of pointers differ, and then HDAM and HIDAM are described in more detail.



hierarchical pointers in SUBSCRIBER data base



child - twin pointers in SUBSCRIBER data base.

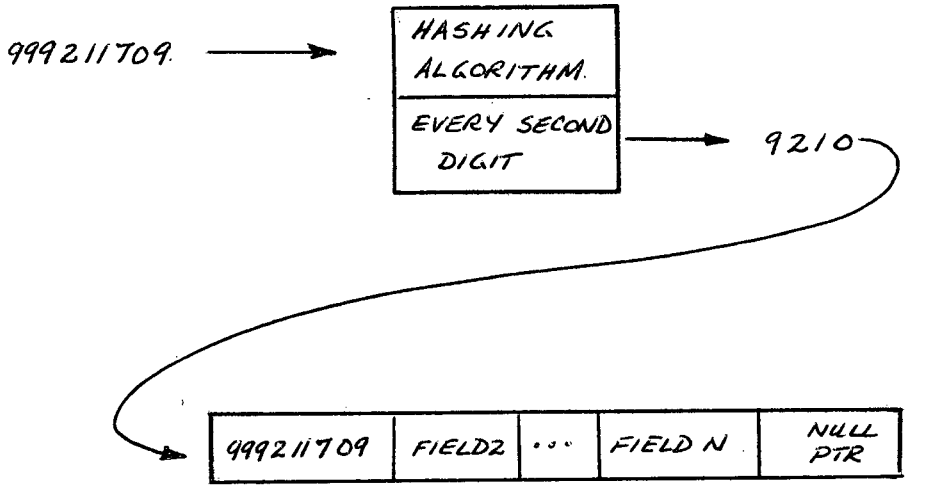
Hierarchical pointers begin at the root and follow the segments from top to bottom and left to right through the hierarchy. Only one pointer exists in the root segment pointing to the first occurrence of the left-most child segment. Each child segment points either to its first left-most child or to the next occurrence of that segment type. When there are no more segments of that type the pointer crosses to the next segment type in sequence; for example, the last DEPENDANT segment points to the first POLICY segment.

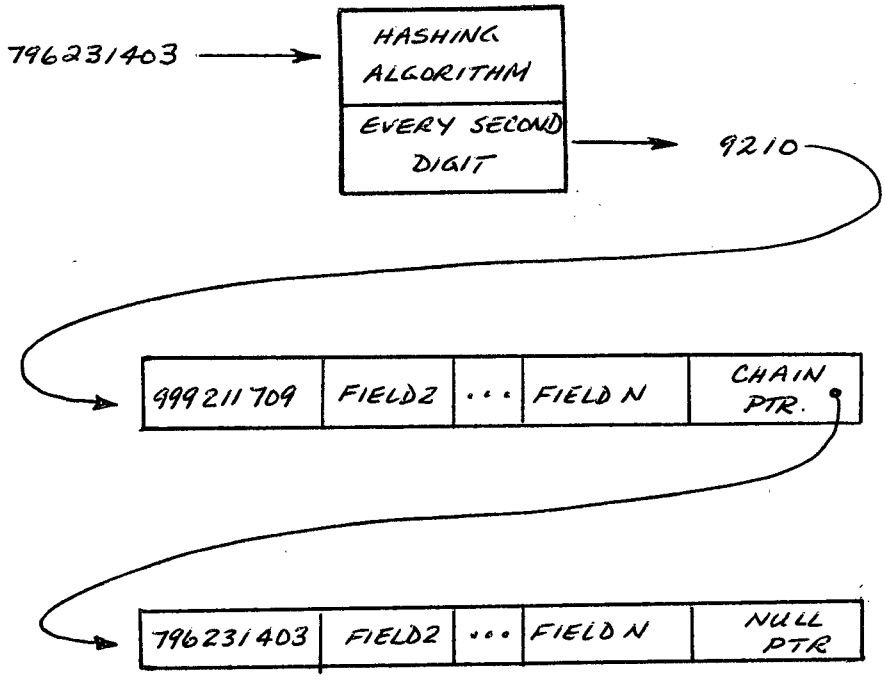
The second diagram illustrates child-twin pointers. In this example there are three pointers in the root (SUBSCRIBER) segment, one for each type of child segment. If the SUBSCRIBER segment had more children there would be more pointers, always one pointer for each child type. A child segment then points to its next segment occurrence. If the child segment itself had children there would be more pointers, again, one for each child type.

The hierarchical pointer scheme requires less storage while the child-twin pointer scheme provides faster access to the different points in the hierarchy.

HDAM

"HDAM provides direct access (by sequence field value) to the root segments, via a hashing and chaining technique, together with pointer access to the subordinate segments." [5] In the customer account data base the key of the root segment is the SIN, this key is supplied with a value, given to the hashing algorithm and an address is generated. If this is the first key value to hash to the calculated address then no subsequent chaining is required. However if the SIN collides with an existing SIN at the same address, the SIN is chained to the first SIN via a pointer. All child segments of the root (and their child segments, etc.) are held together through the use of hierarchical or twin-child pointers.


999211709
HASHING ALGORITHM
EVERY SECOND DIGIT
9210
999211709
FIELD2
...
FIELD N
NULL PTR

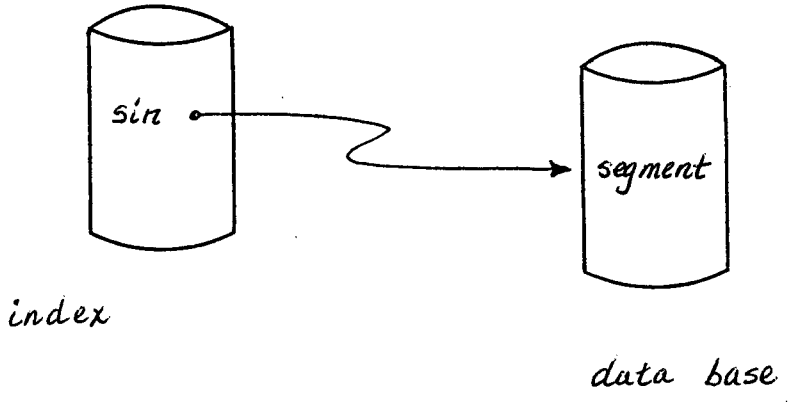

796231403
HASHING ALGORITHM
EVERY SECOND DIGIT
9210
999211709
FIELD2
...
FIELD N
CHAIN PTR.
796231403
FIELD2
...
FIELD N
NULL PTR

So, as long as there are a small number of root segments that hash to the same location the access to a given root segment is fast. Dependent segments are placed as physically close as possible to the root segment in order to minimize disk seek time when looking at a root and all its children.

HIDAM

Instead of direct access to the roots, HIDAM provides indexed access via the sequence field of the root to the root segments and then pointer access to any dependent segments. "A HIDAM data base actually consists of two data bases: a "data" data base, which contains the actual data, and an associated INDEX data base, which provides the (dense) index. There is an entry in the INDEX data base for each root segment in the "data" data base." [6]

In the insurance example an index would be established for the key field of the SUBSCRIBER segment, social insurance number. When a social insurance number is provided in an IMS call, IMS uses that value to search the index data base. From the index data base IMS retrieves physical location information so that the actual data can be found.



4.0 STRUCTURE HAS ITS ADVANTAGES

A data base management system such as IMS can be criticized for its lack of data independence or for the artificial and confining nature of the hierarchical structures it imposes on information. Another disadvantage is the host-language interface through which a programmer must work in order to communicate with data. Such a host-language requires a programmer to know too much about the system; how data is stored, how it is accessed, exact record layouts are examples. These criticisms are warranted if we could ignore the incredible growth in data bases and slow disk access times. As it is, the structure inherent in IMS (and other hierarchic or network DBMS) is the mechanism that permits relatively fast access to such data bases.

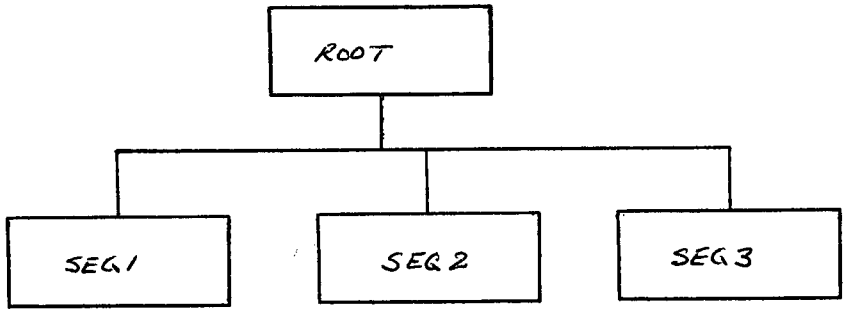
There are two major factors to be considered in designing data base applications: one is the amount of storage space required (eg. the size of the data base) and the second is the speed with which information can be assembled from the data base. The next section deals with methods or organizing an IMS data base and processing data from the data base that will improve response. In general these methods are related to the data base structure (the hierarchy).

We will first examine what a data base administrator can do and go on to consider what each individual programmer can do. All of these techniques have been employed in the insurance application mentioned earlier.

4.1 THE DATA BASE ADMINISTRATOR AFFECTS PERFORMANCE

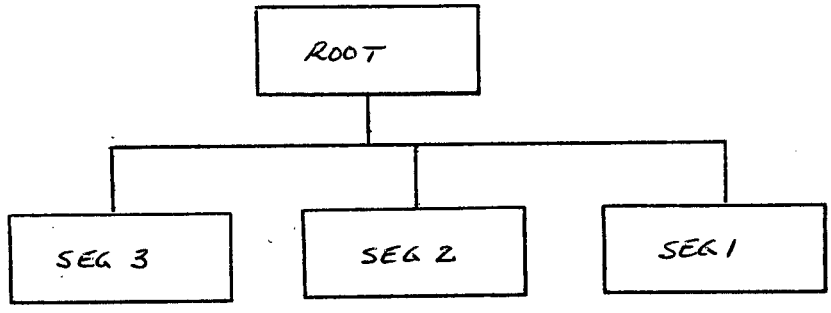
Segment Position

We begin with a general purpose statement about IMS that has bearing on segment positioning as well as on several other areas in our discussion. "IMS/VS manages the space within the records, and allocates and reclaims space as segments are inserted and deleted. In allocating space, an attempt is made to keep adjoining segments in the hierarchic sequence as close together in storage as possible." [7] (Emphasis added).



When IMS positions segments for this data base in physical storage SEG1 is "closest" to the ROOT segment and SEG3 is "furthest" from the ROOT segment. When segments are close together it is likely that one physical access to the disk will bring these segments into core together. Thus a retrieval of ROOT followed by one or more retrievals

of SEG1 segments will involve only one access. If however, access to this data base record typically involves ROOT and SEG3 then the hierarchy should be:



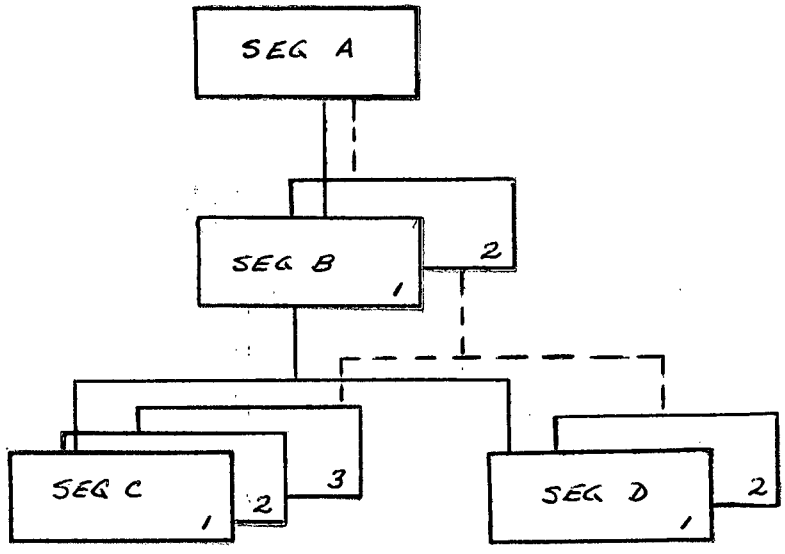
As a general rule segments that are accessed frequently should be close to the top-left in the hierarchy. Segments that are accessed infrequently and/or in batch-only processing (where the time, constraints are not so severe) should be at the bottom-right.

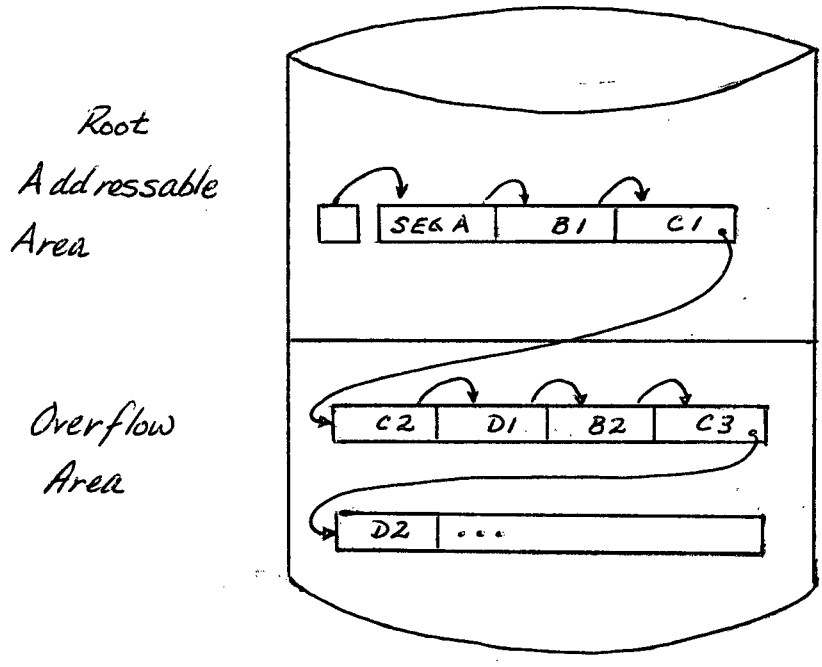
HDAM vs HIDAM

In the insurance application, because most functions were done online, direct access to root segments was required but the choice had to be made between HDAM and HIDAM data bases. "HIDAM is useful when both sequential and direct access by root key are required." [8] HDAM on the other hand is better for direct access than HIDAM because the index data base is removed. HDAM was chosen for the insurance data bases to facilitate online processing since the majority of online processes required direct access. It was decided that although some batch programs required sequential processing of data, a sort step would provide the necessary ordering at a small additional cost. The human factor involved in online systems demanded fast response.

ROOT ADDRESSABLE AREA, SYNONYM CHAINS

An HDAM data base consists of one primary area called the root addressable area (RAA) and an overflow area.





Root segments are stored in the root addressable area their address being determined by a hashing algorithm chosen by the data base administrator (DBA). In addition a limited number of dependent segments are stored in the RAA, this limit is set by the DBA. The length of the RAA is also controlled by the DBA. How can the DBA choose the most appropriate (1) size for the root addressable area (2) hashing algorithm and (3) number of dependent segments stored in the RAA?

In the insurance application statistics were gathered to predict the average, minimum and maximum number of child segments per parent segment for all data base segments.

Armed with these figures the data base administrator could choose suitable parameters for each physical data base. In fact these statistics led to splitting one data base into three data bases for better performance.

Given the typical number of child segments per parent the number of dependent segments stored in the RAA was determined. It is important to store as many segments as possible, preferably all, in the RAA so as to minimize accesses. This is valid given the statement that IMS keeps segments in a data base record together if at all possible. Hence one access to the root segment will get all child segments most of the time. "At least root segments should be stored in the root addressable area. In addition, active dependent segments should be placed in the root addressable area

since this will provide fast access to them because of their physical proximity to the root segment." [9]

The total size of the root addressable area can be determined knowing the average size of a data base record and the number of roots. But because a data base is not static an additional factor is the amount of free space necessary so that insertions can be done within the RAA instead of the overflow area.

$$MS = \frac{N \times E}{NB \times .8}$$

$$S = MS + MS \times F$$
$$= MS\ (1 + F)$$
$$= MS \times F^1$$

where

N = number of bytes in the average data base record (including user and IMS maintained data).

E = expected number of data base records.

NB = number of bytes per block, multiplied by .8 to keep chaining at a reasonable level.

MS = minimum size of RAA in blocks.

F, F^1 = a growth factor, $F^1 > 1$

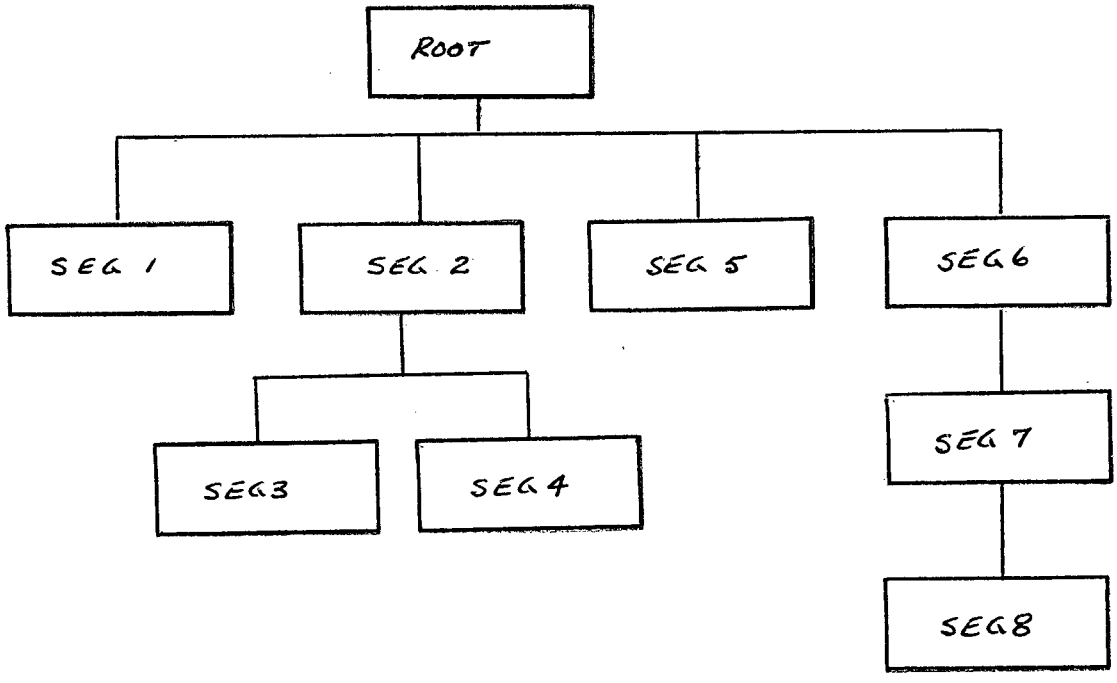
S = optimum size of RAA.

Note that F^1, the growth factor must be a function of data base activity. [10]

In choosing a hashing algorithm it is important to choose one that minimizes the number of synonyms produced and distributes the data fairly evenly over the root addressable area. To do this an analysis of root segment keys must be done to determine their characteristics. Provided with IMS are several hashing modules and a utility which takes as input the root segment keys and produces a map for each algorithm showing key distribution.

SECONDARY INDEXES

If a segment is frequently accessed through a non-key field (this is particularly applicable to root segments), regardless of the root key field, then it is a candidate for a secondary index data base. In this situation an index data base is created, as in HIDAM data bases, containing one record for each key of the secondary index data base.



A secondary index set up for SEG7 will allow access directly to this segment without first qualifying the ROOT segment and SEG6.

4.2 DESIGNER AND PROGRAMMER AFFECT PERFORMANCE

Having dealt with how a data base administrator can optimize an IMS application, we turn to the role of designers and programmers.

BATCH vs ONLINE

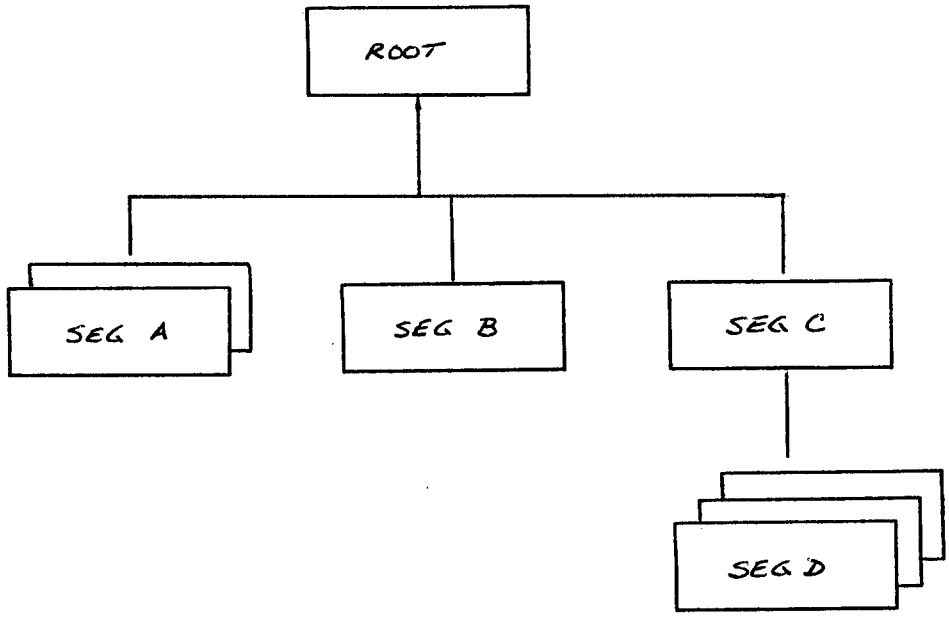
Programs within the insurance application were split into batch and online transactions. Batch transactions (report writers and statistics gatherers, for example) usually examine all roots of a particular data base and report on these in order. Hence using HDAM data bases a sort step is required. This requires additional time and thus such transactions are not scheduled online where response time is so important. Rather they are scheduled as overnight jobs and do not interfere with the online system.

Some insurance programs originally designed to be online and yet work sequentially through one data base were modified so that only one root of the data base is used. For example, an inquiry transaction designed originally to give information on all insurance claims for a particular subscriber now gives information on either the most recent insurance claim or a unique insurance claim, as specified by the user, for a particular subscriber. In another situation a program was split into an online portion and a batch portion to minimize data base activity during an online session.

Further, batch programs are not run concurrently with online programs since batch programs tend to make heavy use of buffers which would in turn slow online performance. If an online program requests a piece of information it is brought into the IMS buffer pool, some actions are then performed and information is displayed to the terminal. While awaiting the user's response, if a batch program is working away chances are that the buffers in the pool will be completely changed before the online program is able to continue. When the online program does continue it will cause another access to be executed, probably for the same information. If the batch program runs at a different time additional disk accesses can be minimized.

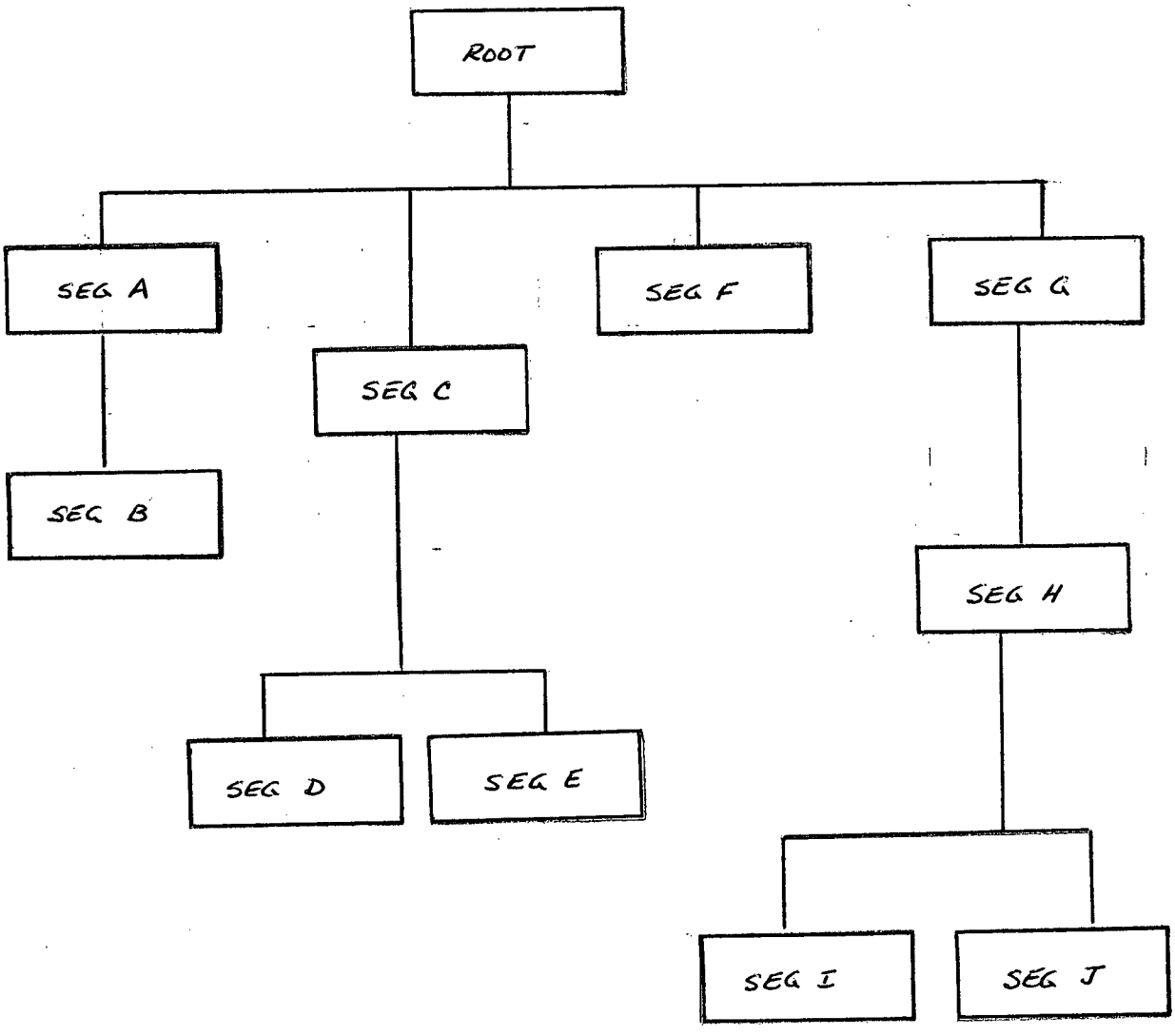
LOCALITY OF INFORMATION

Suppose the following data base structure exists:



It is always adviseable to retrieve a root segment and process all child segments of that root (two SegA's, one SegB, one SegC and three SegD's) before continuing to examine another root segment either in the same or a different data base. This will minimize accesses since a root and its child segments can typically be retrieved with one access.

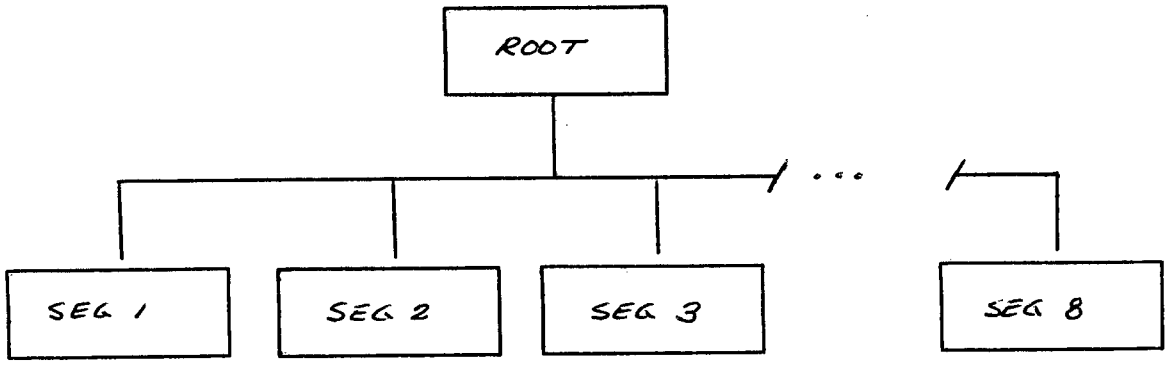
As another example of locality consider:



There are too many segments to retrieve with one access in the above example. SEGG, SEGH, SEGI, SEGJ are stored in a separate block. However, IMS tends to keep segments together according to structure as much as possible. A programmer should therefore try to process these segments in order (top to bottom and left to right) and thus minimize disk access time.

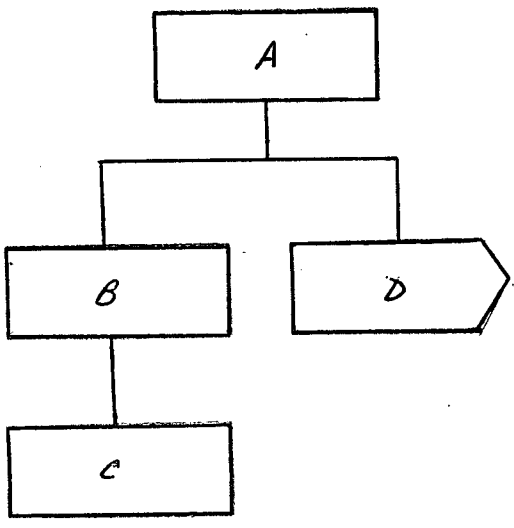
HIERARCHICAL POINTERS

A programmer need not be aware of hierarchical or child/twin pointers in the data base. However, if hierarchical pointers are in use, performance can be enhanced if the program accesses each segment in order according to the hierarchical pointers.

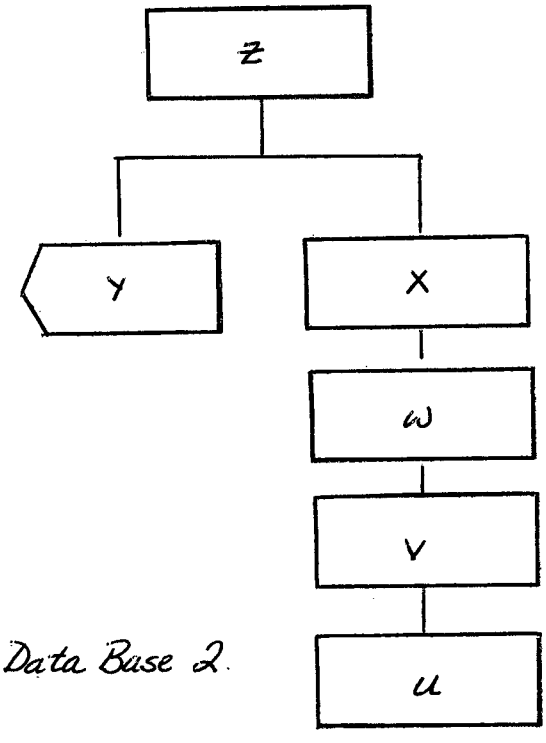


In addition to extra disk accesses as discussed in the section on locality, IMS will work much harder following pointers if a program asks for SEG1 to SEG8 in any order other than that shown above. This is not so with child/twin pointers since the ROOT segment would then contain a separate pointer for each child segment type instead of one pointer pointing to SEG1.

LOGICAL DATA BASES

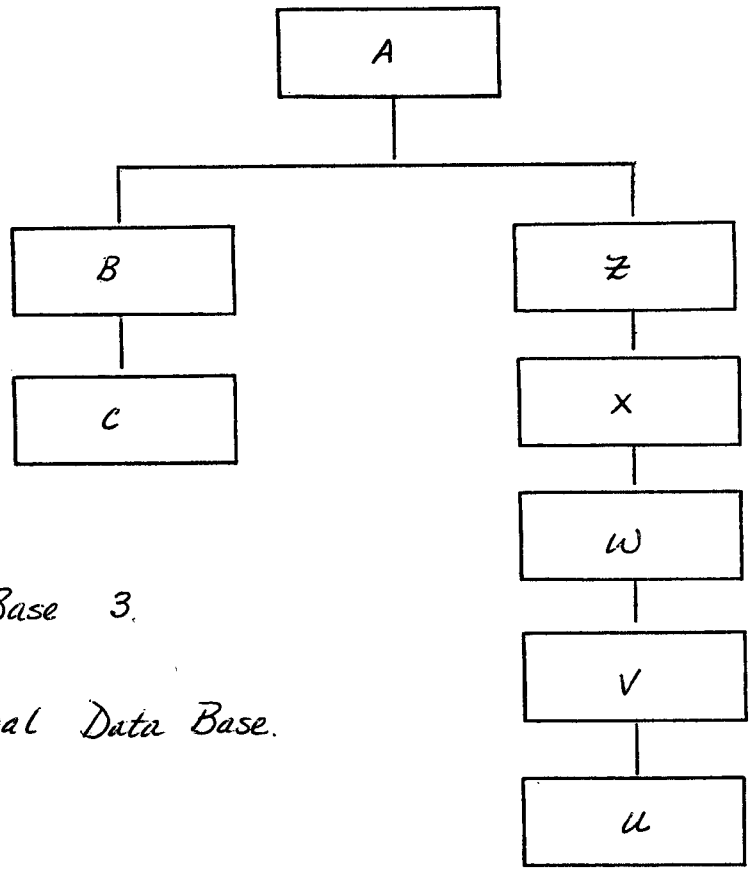


Data Base 1



Data Base 2

Segment D in DATA BASE 1 points to segment Z in DATA BASE 2 and segment Y points to segment A. Thus a logical data base could be constructed joining the two data bases together (data base 3). A program that needs information from DATA BASE 1 and DATA BASE 2 could use this logical view however each time segment Z is accessed that program is crossing from one physical data base to another and in doing so causes additional accesses.



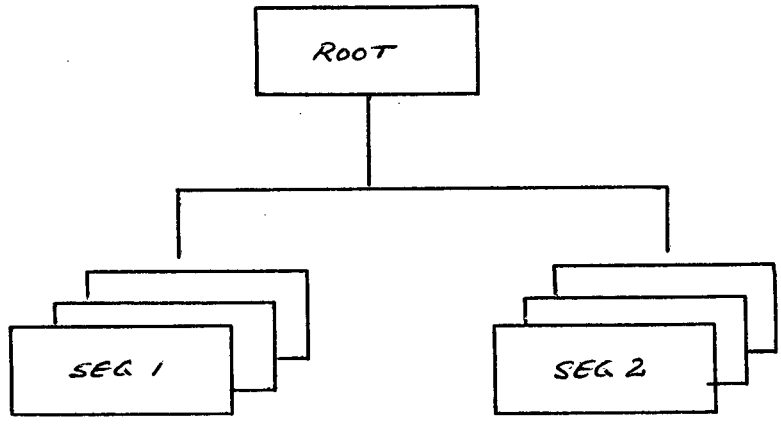
Data Base 3.

A Logical Data Base.

To reduce this cost, one program in the insurance application uses two separate physical data bases, extracts information from each then sorts and merges the resultant information to achieve the same effect. It is important to know when physical boundaries are crossed.

MULTIPLE POSITIONING

For each data base that an application program accesses IMS maintains a pointer to the current position within the data base. In most situations this is sufficient. However, on occasion a programmer must take advantage of multiple positioning in IMS. For example:



Here it is necessary to access the first occurrence of SEG1, then the first occurrence of SEG2, the second occurrence of SEG1 then the second occurrence of SEG2, etc. With multiple positioning IMS will maintain a pointer to the current position in segments of type SEG1 and a second pointer for segments of type SEG2. The programmer must specify a request for multiple positioning in a situation of this type.

OTHER CONSIDERATIONS

A programmer must be aware that a GU call in IMS is more expensive than a GN call. Hence once position is established via a get unique it is advantageous to use as much information as possible from that parent segment via GN calls. This encourages a programmer to follow the hierarchical structure whenever possible.

Insert and delete operations are expensive in IMS due to pointer and space management activities that take place. If at all possible a replace (REPL) should be done instead of a delete followed by an insert. In the insurance application one data base was extensively redesigned to use replace calls instead of delete/insert calls.

All calls to IMS should be as fully qualified as possible thus eliminating much overhead in IMS and potentially reducing the search time.

4.3 CONCLUSIONS

An attempt has been made to show that the structure of a system such as IMS is one of the features allowing current DBMS to cope with increasing data base sizes and response demands of online systems.

In the next section relational data base systems will be examined and the question of whether these systems can indeed perform as well as hierarchic or network systems will be addressed.

5.0 RELATIONAL DATA BASE MANAGEMENT SYSTEMS

There are several criticisms of hierarchical and network data base management systems. One criticism is the lack of data independence that these systems provide, and a second concerns the cumbersome interface that a programmer must deal with. Relational data base management systems provide a much higher degree of data independence and a simpler interface.

Because of the growth in data base size, an emphasis on integrated data and thus shared data, the need for readily available information and the dynamic nature of such information, it has become increasingly important for application programs and query languages to be independent of changes in the stored representation of information. The term for this family is data independence. Systems like IMS have provided some data independence; logical data bases and schemas are examples. However as E.F. Codd has written, "Three of the principle kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence and access path dependence." [11]

IMS can be used to illustrate these different dependencies.

ORDERING DEPENDENCY:

The roots of an IMS data base have some order associated with them; for example an employee data base could be ordered by employee number. If the ordering of this file were altered so that

all employees were ordered by social insurance number, then any programs depending on an ordering according to employee number would have to change.

INDEXING DEPENDENCY:

IMS provides indexing facilities at the non-root level, these are called secondary indexes. Programs can then be written that will access data based on the index. If the index is removed these programs would require modification.

ACCESS PATH DEPENDENCY:

IMS presents a hierarchic picture of data in a data base. If this structure, which in effect defines the access path, changes then the programs using the structure may also have to change. In particular, if the hierarchy is being used efficiently as explained in the previous chapter the programs would have to change.

Data manipulation languages designed to work with hierarchic or network DBMS depend heavily on the underlying structure of the data model. For example, IMS has the commands GN and GNP that are specifically designed to guide a programmer through a hierarchy from top to bottom and left to right. In addition the languages that define structure, logical connections and access path sensitivity in IMS are not at all consistent with the data manipulation language. Hence a user must learn several interfaces.

To provide a comparison relational data base management systems will now be discussed. System R, a prototype under development

at IBM is used as an example. Particular attention will be paid to the underlying storage techniques and basic retrieval mechanisms that System R provides.

5.1 THE RELATIONAL MODEL

A paper by E.F. Codd provides a definition of the term relation:

"Given sets S_1, S_2, ... S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on. We shall refer to S_j as the jth domain of R. As defined above, R is said to have degree n." [12]

The most convenient way to represent a relation is with a table. Suppose we consider an insurance application as mentioned in the previous section, the following relations could occur:

SUBSCRIBER RELATION

SIN	STATUS	NAME	LOCATION	BD	SEX
S1	ACTIVE	JONES	VANCOUVER	31 01 50	M
S2	ACTIVE	FORD	VICTORIA	15 06 46	F
S3	CANCELLED	KING	VANCOUVER	03 03 35	F
S4	ACTIVE	BOOTH	KAMLOOPS	27 11 42	M

ENROLMENT RELATION

GROUP NO.	SIN	DATE	STATUS	COMPANY
G1	S4	1977	ACTIVE	A
G2	S1	1976	ACTIVE	B
G2	S3	1976	UNPAID	B
G3	S2	1978	ACTIVE	C
G4	S4	1975	CANCELLED	D

CLAIM RELATION

CLAIM NO.	SIN	DATE	STATUS	AMOUNT
C1	S1	06 06 76	PAID	$100
C2	S1	20 09 77	PAID	$ 30
C3	S3	04 04 78	REJECTED	$ 0
C4	S3	15 03 77	PAID	$ 12
C5	S3	31 01 78	PAID	$ 39
C6	S4	12 05 75	PAID	$ 52

The SUBSCRIBER relation shows all subscribers that have contracts with the insurance company. ENROLMENT states that a particular subscriber belongs to a particular group (GROUP NO.), was enrolled in a given year (DATE) and has a certain status on that group. All claims that have been received are listed in the CLAIM relation. For those claims that are paid (STATUS) the amount of payment is listed (AMOUNT).

Consider the definition of a relation. This system contains three relations (more relations will be added later), the domains of the SUBSCRIBER relation are SIN, STATUS, NAME, LOCATION, BD and SEX and the degree of the CLAIM relation is five. Furthermore, each row in a table represents a tuple of the relation, the ordering of rows is insignificant, all rows are distinct and the ordering of columns is significant. Each relation has a primary key, one or more domains which uniquely identify each tuple in the relation. The keys in the above relations are SIN, GROUP NO. and SIN, and CLAIM NO respectively. Notice that in the enrolment relation the two domains

SIN and COMPANY could also be a primary key. It is also possible that a domain of one relation is a primary key of another relation, for example SIN in the CLAIM relation.

There are two basic retrieval languages for relational data base management systems: relational algebra and relational calculus. The relational algebra provides basic operations that are performed on one or more relations to produce a relation as a result. These basic operations are permutation, projection and join. Permutation results in the interchange of two or more columns of a relation and is necessary because ordering of columns in a relation is significant. Thus one permutation of the CLAIM relation would produce columns ordered as CLAIM NO., STATUS, DATE, SIN, AMOUNT. Note that the set of queries answerable by a relation is the same as the set answerable by any permutation of that relation.

A projection consists of selecting certain columns from a relation, forming a second relation and removing any duplicates from that second relation. Using the notation of Date [13]:

ENROLMENT [GROUP NO., COMPANY]

results in a relation

GROUP NO.	COMPANY
G1	A
G2	B
G3	C
G4	D

Note that the tuple G2, B exists only once.

C.J. Date describes the join operation "Two relations with a common domain, D, can be joined over that domain. The result is a relation in which each tuple consists of a tuple from the first relation concatenated with a tuple from the second relation which contains the same D-value". [14] To illustrate a join operation a fourth relation is added, the CONTRACT relation.

CONTRACT RELATION

GROUP NO.	RULE NO.
G1	R1
G1	R5
G2	R2
G3	R2
G3	R3
G4	R4
G4	R1

A join of ENROLMENT and CONTRACT over GROUP NO., ENROLMENT * CONTRACT would give:

GROUP NO.	SIN	DATE	STATUS	COMPANY	RULE NO.
G1	S4	1977	ACTIVE	A	R1
G1	S4	1977	ACTIVE	A	R5
G2	S1	1976	ACTIVE	B	R2
G2	S3	1976	UNPAID	B	R2
G3	S2	1978	ACTIVE	C	R2
G3	S2	1978	ACTIVE	C	R3
G4	S4	1975	CANCELLED	D	R4
G4	S4	1975	CANCELLED	D	R1

Operations in the relational algebra can, of course, be combined: ENROLMENT [SIN, COMPANY] * CONTRACT would result in:

SIN	COMPANY	RULE NO.
S4	A	R1
S4	A	R5
S1	B	R2
S3	B	R2
S2	C	R2
S2	C	R3
S4	D	R4
S4	D	R1

The relational algebra is more complex and more procedural than relational calculus. It demands that a user know precisely what tables are in the system, what common domains are in these tables and the order of domains within a table. Most implementations of relational DBMS use a high-level relational calculus as a user interface.

Relational calculus allows a user to define the desired result of a query and the underlying data management system translates that definition into a series of relational operations, such as projection and join.

Specifying a projection in relational calculus differs little from relational algebra:

{(ENROLMENT. GROUP NO., ENROLMENT. COMPANY)}

this is equivalent to

ENROLMENT [GROUP NO., COMPANY] .

A join operation is easier to specify using relational calculus:

{(ENROLMENT. SIN, ENROLMENT . COMPANY, CONTRACT. RULE NO.):
ENROLMENT . GROUP NO. = CONTRACT . GROUP NO. }

this is equivalent to

ENROLMENT [SIN, COMPANY] * CONTRACT .

The relational calculus allows a user to specify what information is required not how to get it. It is important to note that the two methods are equivalent in retrieval power and that an algorithon exists for converting any calculus expression into an algebra expression. [15] An example of relational calculus will be given when we deal with System R's retrieval language SQL.

5.2 SQL

System R provides a language called SQL, formerly SEQUEL, which can be used as a stand alone user language or interfaced with PL/1. SQL is a high-level language used for data retrieval, data manipulation, data definition and data control. This is in contrast with IMS since IMS provides separate and distinct facilities for data definition and data control.

SQL QUERY FACILITIES

All queries that can be expressed in SQL have the same basic format.

```
SELECT      ( some type of data )
FROM        ( a table containing the data )
WHERE       ( some condition describing the choice of
              data)
```

Using the tables developed earlier we will demonstrate the various query facilities.

```
SELECT      SIN
FROM        SUBSCRIBER
WHERE       SEX  =  ' F '
```

This query might be 'read' as "Select all social insurance numbers for subscribers that are female". Note that the domain name SIN has been provided and the name of the SUBSCRIBER relation.

Several domains may be selected:

```
SELECT      SIN, NAME
FROM        SUBSCRIBER
WHERE       SEX  =  ' F '
```

All domains of a given relation may be selected:

```
SELECT      *
FROM        SUBSCRIBER
WHERE       SEX = ' F '
```

All entries in one domain may be selected:

```
SELECT      SIN
FROM        SUBSCRIBER
```

Multiple conditions may be specified:

```
SELECT      SIN
FROM        SUBSCRIBER
WHERE       SEX = ' F '
AND         STATUS = ' ACTIVE '
```

The condition can contain an arithmetic expression:

```
SELECT      CLAIM NO.
FROM        CLAIM
WHERE       AMOUNT > 2 * DEDUCTIBLE
```

(This assumes that a value is assigned to DEDUCTIBLE.)

Built-in functions are available:

```
SELECT      AVG (AMOUNT)
FROM        CLAIM
```

This would give the average amount paid out for a claim.

Multiple relations may be specified:

```
SELECT      CLAIM NO.
FROM        CLAIM
WHERE SIN = SIN =
                SELECT    SIN
                FROM      SUBSCRIBER
                WHERE     SEX = ' F '
```

Other facilities include specifying the order in which queries are returned -- ascending or descending by some domain, a SELECT UNIQUE facility that will eliminate duplicates, reordering of domains so that they are not in the same order as in the table, and the ability to retrieve more than one set of information and perform set operations on the result (UNION, INTERSECTION, MINUS). A more complete description of SQL is given by G.H. Denny. [16]

It should be noted that in order to form the queries in SQL a user must know (1) the exact format of the tables, the domain names and (2) how information is stored in the tables, for example the domain SEX is stored as 'F' for female and 'M' for male.

DATA MANIPULATION

In SQL a user may UPDATE a tuple or tuples in a relation, DELETE tuples from a relation and INSERT new tuples in a relation. The format of these commands is very similar to the format of the SELECT command.

```
DELETE          CLAIM
WHERE           DATE < 01 01 77
```

'Delete all claims prior to January 1, 1977.'

```
UPDATE          ENROLMENT
SET             STATUS = ACTIVE
WHERE           SIN = 'S3'
```

'Update the status of S3 to active.'

```
INSERT INTO CLAIM
<C7, S4, 030077, PAID, $35>
```

'Insert a paid claim into the CLAIM relation.'

Note that the values being inserted must be in domain order.

DATA DEFINITION

Definition of new relations is done via a CREATE TABLE command and relations can be destroyed through a DROP TABLE command. When creating a table an ordering may be specified (an ORDER BY clause is used). This ordering has a bearing on the physical proximity of table entries.

A user who has need of specific tables and specific table ordering can use the DEFINE VIEW command. This is similar to a schema as discussed in Date. [17]

Another command, EXPAND TABLE, allows for dynamic modification of relations by adding new fields.

DATA CONTROL

There are four areas concerned with data control: transactions, authorization, integrity and triggers. A transaction in System R consists of a series of SQL commands that perform a function. The importance of transactions is their relationship to backup and recovery. The beginning of a transaction defines a point to which a user may backup.

Authorization refers to the ability of a user to read, insert, delete and update tables within the data base.

Assertions help maintain data integrity. For example, ASSERT ON INSERT TO CLAIM: AMOUNT $\geq$ \$0 means that no claim can have an amount less than zero.

Triggers allow a user to define other integrity information. For example a trigger could be defined so that whenever the STATUS in the ENROLMENT relation is set to ACTIVE the STATUS for the same SIN in the SUBSCRIBER relation is also set to ACTIVE.

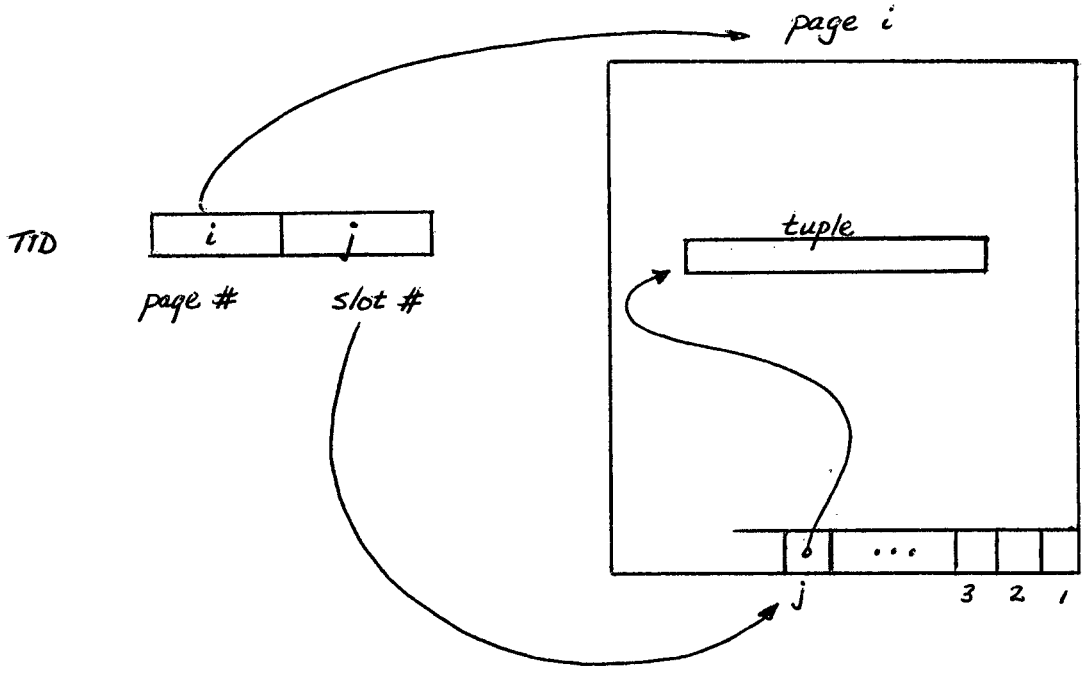
This brief discussion of data management facilities using SQL leads to a description of how System R stores information and finally to an outline of the optimizer that determines the optional way to satisfy a data request in SQL.

5.3 RELATIONAL STORAGE SYSTEM -- RSS

All data necessary for data base management, including user data, access path information, catalog information and intermediate results, is stored in a collection of logical address spaces called segments. (This is an unfortunate choice of terminology since IMS uses the term 'segment' for its record level of information.)

The storage component of RSS manages physical disk space and to this end maps the segments described above to physical locations on disk. Each segment in fact consists of a set of equal sized pages and the storage component deals with slots on disk the size of one page. Thus physical page slots are allocated and freed as required and RSS maintains a page map for each segment.

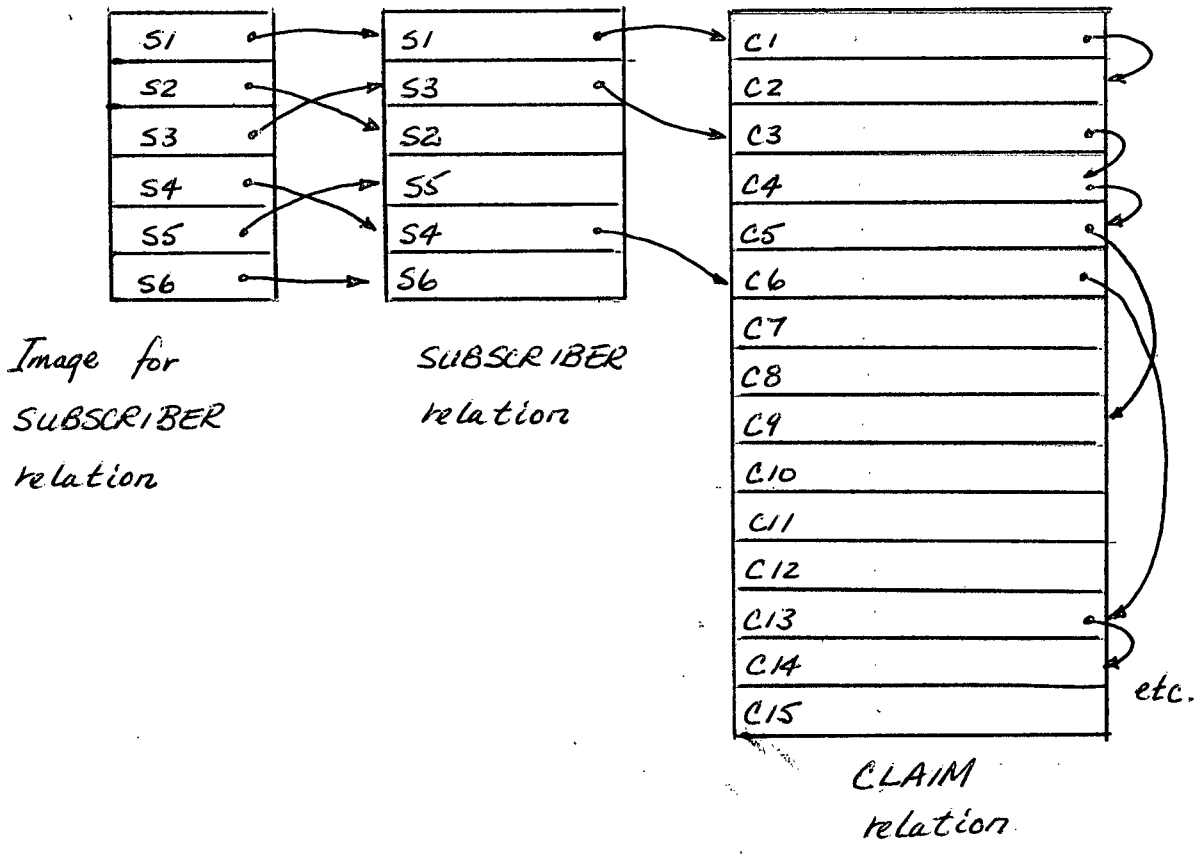
Relations which consistsof a number of tuples are stored within segments. "Associated with every tuple of a relation is a tuple identifier or TID." [18] The RSS is responsible for storing and accessing tuples and for maintaining the various pointer structures that link tuples of the same or different relations together. A tuple identifier is implemented as shown in this diagram:



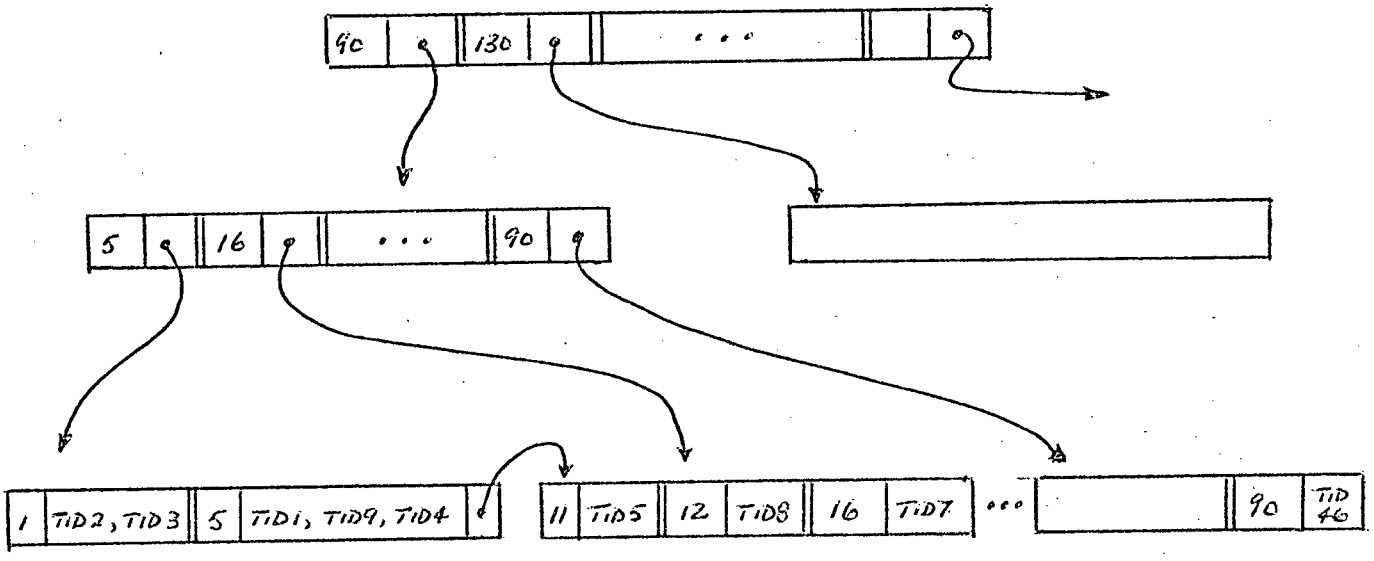
A tuple identifier contains a page number and a slot number. The page number references a particular page within a segment while the slot number directs the RSS to an area always maintained at the bottom of the page. From that slot an additional pointer that has been stored via RSS points to the actual tuple. Since data is accessed in pages this scheme retrieves the requested information in a single page access, in most cases, once the TID itself is accessed.

Tuples are associated with each other either through the use of images or links which are defined by the user and maintained through RSS. An image is a logical ordering of an n-ary relation, with respect to values in one or more sort fields. Images are maintained through the use of a multi-page index structure and the index information is stored on one or more pages within the segment containing the relation.

While images are used to order tuples within a relation, links are used to connect tuples in different relations. The links perform a similar function to child/twin or hierarchical pointers in IMS. RSS maintains these links and adds to them as new tuples are added to the data base.



In this diagram an image exists for the SUBSCRIBER relation ordering tuples according to subscriber number. Note that the image is stored separately from SUBSCRIBER information. A binary link exists from SUBSCRIBER to claim associating all claims with the subscriber that issued them. A unary link could be added to order claims on date if that was a requirement. (Not shown in the diagram.) Thus a unary link is used to order tuples within one relation without using an image. Note that an image requires extra storage space and additional IO processing. Binary links always connect tuples from two different relations. The following diagram illustrates how images are implemented using a VSAM-like tree which is based on the B-tree concept.



5.4 THE OPTIMIZER

"The objective of the optimizer is to find a low cost means of executing a SEQUEL statement, given the data structures and access paths available." [19] The optimizer has two main criteria in determining the best access path: minimizing page fetches and minimizing CPU instructions. In addition the optimizer can weight CPU cost or 1/O cost depending on whether the system is compute bound or 1/O bound.

As discussed earlier, System R provides for physical clustering of tuples and relations based on images and links. It is the function of the optimizer to take advantage of these properties whenever possible in determining the execution of a given query. The optimizer goes through several steps in determining the best access.

(1) Classify the statement according to certain language features such as the user of a join operator or a simple restriction operator.

(2) Check the system catalogues to determine what pertinent images and links are available. For example:

```
SELECT   A, B
FROM     R
WHERE    C IS LESS THAN X
```

In this query an image on the field C would be pertinent to efficient execution.

(3) For each language feature mentioned in (1) above the optimizer can choose from a set of predetermined methods of execution. One or more methods will be reasonable for the query. Two methods that would be reasonable in the above situation are:

Method A: Use a clustering image which matches a predicate whose comparison operator is not ' = '.

Method B: Use a non-clustering image which matches a predicate whose comparison operator is not " = ".

(4) Given more than one reasonable method, calculate an expected cost and choose the minimum cost method. For method A, assuming half the tuples in the relation satisfy the predicate, the expected cost is R/ (2 x T) where R is the relation cardinality and T is the average number of tuples per data page. For method B, the expected cost to retrieve all tuples is R/2.

If an image or link exists that will directly facilitate the query, a method will be chosen that uses it. In addition, if more than one link or image is available, the method which uses the link or image with a clustering property will be chosen since it will reduce IO significantly.

As a further illustration, consider another query.

```
SELECT   A,B
FROM     R
WHERE    C = X
AND      D = Y
```

A clustering image exists on C and a non-clustering image exists on D. The optimizer will choose to use the clustering image existing on C. Note that in this case it would be best to be able to use both images and compare selected tuple identifiers before accessing any tuples from the relation. However, the optimizer as described does not provide this facility.

This section is intended to briefly describe the optimizer facility of System R. A discussion of its effectiveness is outside the scope of this paper since little detailed information is available.

6.0 PERFORMANCE CONSIDERATIONS IN SYSTEM R

Given a brief explanation of System R the insurance application will be examined in more detail to compare the performance of some typical tasks using IMS or System R.

The following relations exist:

SUBSCRIBER	SIN	NAME	ADDRESS	STATUS	AGE
ENROLMENT	GROUP NO.	SIN	DATE	STATUS	COMPANY
CLAIM	CLAIM NO.	SIN	DATE	STATUS	AMOUNT
RECEIPTS	CLAIM NO.	DEP	TYPE	DATE	AMOUNT
DEPENDANTS	DEP	SIN	STATUS	AGE	
CONTRACT	GROUP NO.	RULE NO.			
RULES	RULE NO.	CATEGORY	PERCENT	LIMITS	

SUBSCRIBER: describes all people who have contracts.

DEPENDANTS: all dependants of a given subscriber.

CLAIM: all claims that have been processed.

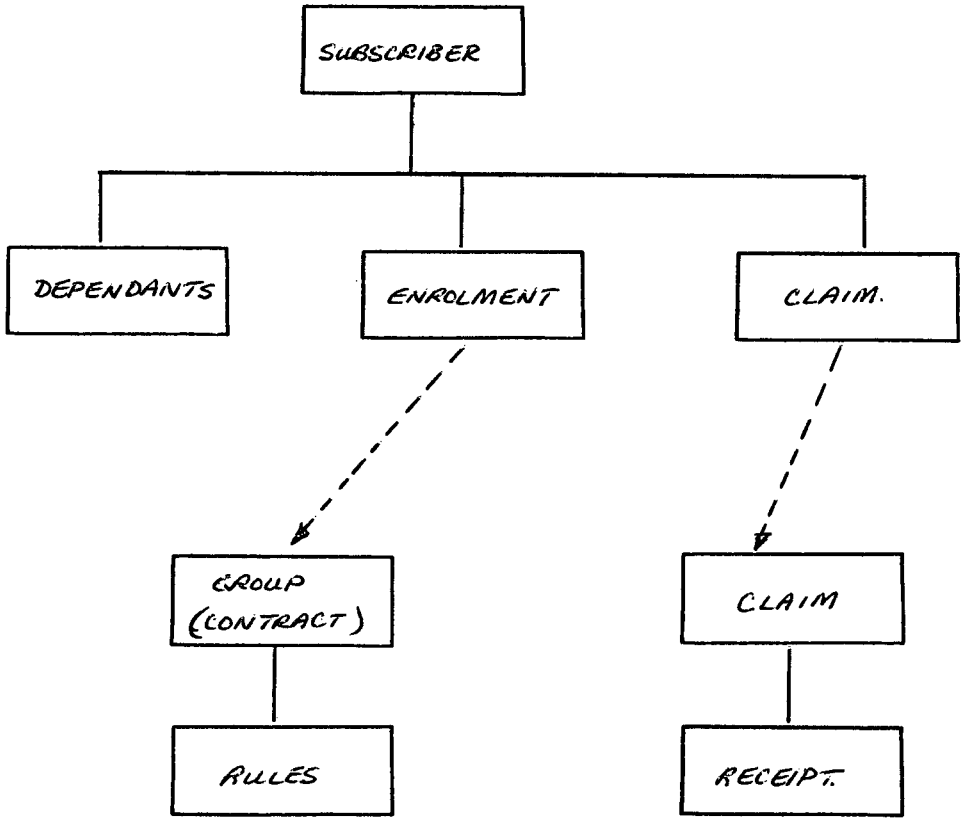
ENROLMENT: the group/company to which a subscriber belongs.

RECEIPTS: details of the claim.

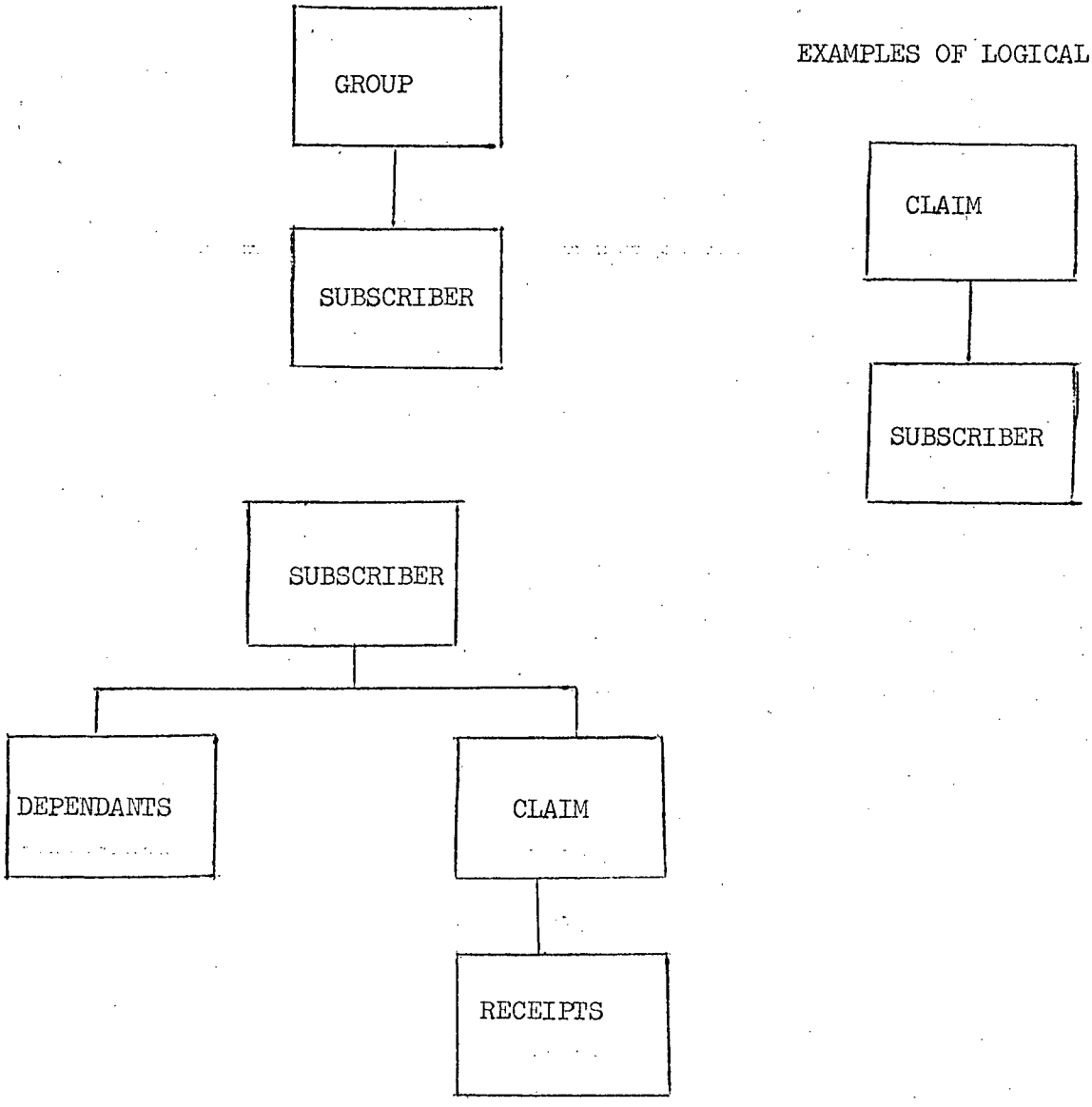
CONTRACT: the contract rules for a particular company.

RULES: details of specific rules.

An IMS version of this data base might be:



There are three data bases: one with subscriber related information, one with group/contract information and one describing all claims that have been processed. Relationships between the subscriber and contract information as well as the subscriber and claim information exist, thus various "logical views" of the data base can be constructed.



The first typical task is to evaluate a new claim for a given subscriber. Such a process consists of several steps (note that this is a simple view of claim evaluation):

1. Does the claim already exist?
2. Does the subscriber exist?
3. For each receipt in the claim
    a) does the dependant exist?
    b) is the item covered for the date?
    c) determine amount to be paid.
4. Insert the claim in the data base.
5. For each receipt, insert the receipt information.

It is assumed that all IMS data bases are HDAM, (See Section 3), which facilitates random processing at the expense of sequential processing. In System R, it is assumed that links exist providing clustering so that the data bases' underlying structure closely resembles that in IMS. The critical element in this analysis is the number of disk accesses (IOs) that are required to perform the task.

STEP	IMS	IOS	SYSTEM R	IOs
1.	GU (claim)	1	SELECT	1 (or more) for index page. 1 for data page.
2.	GU (subscriber)	1	SELECT	1 (or more) for index page. 1 for data page.
3a.	GNP (dependant)	0	SELECT	0 if link exists from subscriber to dependant.
3b.	GNP (enrolment)	0	SELECT	0 if link exists to cluster information.
3c.	GNP (rules)	1	SELECT	0 for contract information if link exists. 1 (or more) for index to rule. 1 for rule information.
4.	ISRT (claim)	1	INSERT	1 (or more) for index. 1 for data.
5.	ISRT (receipts)	0	INSERT	0 if link between claim and receipts exists.
TOTAL		4		8 minimum

In this example even when System R has a similar underlying structure to that of IMS, additional access time is required : 8 accesses versus 4 accesses. This analysis is generous to System R since an index access will typically require as many accesses as the level of the index. If the index level were two for each relation then the number of accesses in System R would become 12 and an index level of three would force the number of accesses to 16. The prime reason for additional access time

required by System R is the indexing scheme used to implement images for relations, as discussed earlier. If such an index scheme were replaced by a hashing scheme as in HDAM, System R would lose the flexibility and relative speed of scanning a relation effectively which it must be able to do for proper performance of join operators, a cornerstone of relational data base management.

In an article called "Computing Joins of Relations", L.R. Gotlieb has analyzed different algorithms used in computing relation joins. [20] He has determined that I/O activity must be minimized and that ordered key lists when kept on both domains being joined are the most effective way of minimizing I/Os. These ordered key lists are similar to images in System R. The concept of relation joining is not necessary in IMS since relations are already joined through the hierarchical data base structure of IMS. In an IMS data base it is important therefore to structure the hierarchy to join those relations that are in fact most frequently joined in the application. Thus for example DEPENDANTS is a child segment of SUBSCRIBER since it is often necessary to join the SUBSCRIBER and DEPENDANTS relations, similarly RECEIPTS is a child segment of CLAIM.

As a second example suppose a user asked the system to "list all paid claims for a subscriber". In System R a SELECT statement is used:

```
SELECT          CLAIM NO.
FROM            CLAIM
WHERE           STATUS = PAID
AND             SIN = S1
```

Assuming that a link is provided between the SUBSCRIBER relation and all claims in the CLAIM relation, the above statement would require one access to index the subscriber and one access to retrieve the page containing that subscriber's information. In contrast an IMS programmer would code:

```
        .
        .
        .
CALL  PLITDLI     ( FOUR,
                    GU
                    SUBSCRIBER - POINTER,
                    RETURN - AREA,
                    KEY = S1 ) ;
DO WHILE (SUBSCRIBER - STATUS = BLANK) ;
     CALL PLITDLI ( FOUR,
                    GNP,
                    SUBSCRIBER - POINTER,
                    RETURN - AREA,
                    NO KEY ) ;
        .
        .
        .
END ;
```

While this is obviously not as concise and straight forward as an equivalent System R statement it does result in a single access, if symbolic pointers are used to implement the logical relationship between SUBSCRIBER and CLAIMS.

Another typical task involves enrolling a new subscriber. To do this both systems will have to:

1. create a SUBSCRIBER entry.
2. create dependants for this subscriber.
3. link subscriber to an appropriate group.

STEP	IMS	I/O	SYSTEM R	I/O
1.	ISRT (subscriber)	1	INSERT	1 (or more) for index. 1 for data.
2.	ISRT (dependants)	0	INSERT	0 if link is available.
3.	ISRT (enrolment)	0	INSERT	0 if link is available.
TOTAL		1		2 (or more)

The above analysis assumes that no images exist for either SUBSCRIBER or DEPENDANTS relation. Further IOs are required if images do exist.

These few examples show situations where System R is not as efficient as IMS. Assuming for a moment that some other examples can be constructed showing System R to be more efficient that IMS there are other aspects of System R that also degrade performance. These will be discussed in the next section.

6.1 OTHER PERFORMANCE CONSIDERATIONS USING SYSTEM R

The functions of System R's relational storage system include concurrency control, facilities for checkpoint and restart, transaction management, transaction recovery, physical storage management, maintenance of links and images as well asddata access. At another level within System R, the Relational Data System (RDS) "provides high level, data independent facilities for data retrieval, manipulation, definition and control." [21] What overhead is involved in providing these functions?

F.H. Lochovsky and D.C. Tsichritzis have examined relational, network and hierarchical data base management systems with user performance in mind. [22] Three factors were considered:

1. proportion of correctly coded application programs.
2. coding time.
3. debugging time.

In their study users were chosen and assigned to one of the three data base management systems and given time to familiarize themselves with the user interfaces. Following this, a set of programs were devised and the users implemented these programs. Results of this study showed that the relational DBMS provided a better user interface than the hierarchical DBMS in all three areas; correctness, coding time and debugging time. One of the major problems in using the hierarchical DBMS was setting the data base position pointer before

"navigating" through the hierarchy, while another was using the get next within parent (GNP) command correctly. In addition, some problems were associated with the unnaturalness of hierarchical DBMS.

The reason for introducing this study is to emphasize that the advantages associated with using a relational data base management system result to a large degree from the high level user interface provided -- for example SQL in System R. However queries expressed in SQL must be analyzed by the optimizer. This involves the overhead of parsing the SQL statement, classifying the statement according to certain rules, examining various system catalogs to find suitable links and images and then selecting the best method of satisfying such a query. All this must be done before any relevant data can be retrieved. A possible solution would be to perform the above actions at compile time instead of execution time, however, this is impossible if System R is to maintain the flexibility of dynamic link and image definition, dynamic table definition and dynamic table extension.

Arguments can be made to show that a relational data base management system, such as System R, requires additional storage space. Thus, over a wide range of accesses to the data base more physical IOs will be required. Consider for example such features as backup page maps used to handle segment recovery. Extra storage is also required for intermediate relations that are created by various relational operators. Page maps are maintained for each

segment, a segment consists of physical pages and the page map show the physical location of each page of a segment on disk. For recoverable segments (intermediate relations are not recoverable segments) a backup page map initially has entries identical to the current page map. As information in the segment changes the current page map also changes but the backup page map maintains an old version for possible recovery. Relations require more storage because the key information has to be stored in both "parent" and " child" segments. Consider, for example, the various relations in the insurance application: SIN is maintained in SUBSCRIBER and DEPENDANTS relations, CLAIM NO. exists in CLAIM and RECEIPTS whereas in ahhierarchical data base SIN occurs once only in the SUBSCRIBER segment, CLAIM NO. occurs once only in the CLAIM segment.

As mentioned in the previous discussion, all relations in System R are accessed through an indexing scheme. Thus access to any tuple or relation directly involves one or more accesses to an index page followed by one access to a data page. This is a distinct disadvantage to System R.

In System R there is no well defined concept of locality of information, no equivalent to the "top - left" in a hierarchy which is known by a programmer to be close (i.e.: in the same physical page) to the root segment. A programmer can not therefore take advantage of segment ordering in order to minimize IOs.

Because links and images may be defined with a clustering property in System R and this clustering property can be defined dynamically there is some danger that these will be used indiscriminately, resulting in serious fragmentation of relations and clusters.

These are some areas for concern in System R and indeed in any relational DBMS. A summary is provided by an article in Computing Surveys. [23]

> "The user of a procedural DSL, like the DBTG DML, can select for his particular interaction the most efficient access strategy, corresponding to a given schema definition. He can give the system directions for traversing the ... model to locate the desired information rather than letting the system choose a route of its own. In situations where efficiency is critical, such a system might be able to out-perform higher level, less procedural interactions."

6.2 ANOTHER STUDY

In a similar study M. Stonebraker and G. Held [42] have compared network, hierarchical and relational data base management systems. This study concentrated on the different language levels; high level non-procedural languages and low level procedural languages. It was concluded in the study that non-procedural languages result in increased programmer efficiency, increased data independence and better protection and integrity at the expense of machine efficiency.

To support the statement concerning machine efficiency three situations are discussed in which a procedural system can be more effective.

1. In some situations it is possible to express a particular query in more than one fashion. It is argued that a programmer using a procedural language will determine the most efficient way of implementing the request before carrying out the programming task. On the other hand, using a non-procedural language the user is unaware of the fastest method and may not choose appropriately.

2. Stonebraker and Held demonstrate that a non-procedural language may be inappropriate for certain requests. The example they use is to find the member of a particular department who has the second highest salary. In a procedural

system such as IMS if the segment containing salary is ordered on salary value then it is simple to satisfy this request. Expressing such a request in SQL would require:

```
SELECT   X. SALARY
FROM     EMP  X,  EMP  Y
WHERE
COUNT    (X. SALARY  BY X. SALARY
          WHERE  X. DEPARTMENT = 12
           AND  Y. SALARY  >  X. SALARY )  =  2
```

This is very difficult to formulate and there is no guarantee that the optimizer could determine an efficient way to execute this request.

3. The third situation deals with an example that is similar to those discussed in Section 6.0.

6.3 SUMMARY

Despite the relative efficiencies of hierarchical systems, such as IMS, and relational data base management systems, such as SYSTEM R, there is still a basic bottleneck to overcome -- the cost in terms of performance associated with current mass storage hardware. A query such as "retrieve all claims less than one year old" should not require a DBMS to bring each set of claim information into core, check whether the condition is satisfied and reject or accept the claim accordingly. Nor should the DBMS require some specialized set of pointers based on the claim date in order to answer such a query. Instead "smart hardware" should be available that will search the data based on its content rather than its physical address. Content addressable memories were proposed as early as the 1950's, however, due to the high cost associated with them, they have only been implemented on a very small scale. In the next section some trends in data base management research will be discussed, associative memories will be described along with some specific systems. A more detailed examination of RAP, a system developed at the University of Toronto will also be given.

7.0 ASSOCIATIVE PROCESSORS

The 1977 Conference on Very Large Data Bases included a session on directions in data base research. Some quotes from that session are significant.

"A number of storage techniques and search algorithms in use in current data management systems are impractical for very large data bases. We are interested in new hardware, probably exploiting LSI technology, that would make those techniques and algorithms feasible for very large data base systems. An example would be a novel implementation of various 'key word' algorithms for searching free text in a 'smart memory'. [24]

"In order to realize an operational data base system it is substantial to achieve a reasonable response time. There are two kinds of very time consuming processes in the system. One is the problem solving process which appears in [the] natural language understanding part and [the] deductive translation part. We need special purpose hardware such as a LISP machine or more sophisticated machine. The other is the data base manipulation process. We need a very efficient data base machine, especially when we have a very large data base. The most important and difficult target is to improve the performance of n^2 - type operations such as join and projection in relational algebra." [25]

"Areas of particular interest in very large data base systems include the following ... specialized hardware. In addition to back-end data base management systems, we are interested in associative memories, intelligent disks, intelligent terminals, and graphics systems." [26]

"Management of very large data bases will be heavily dependent on new hardware technologies supporting new storage and retrieval methods. Especially associative memories and parallel access algorithms seem to be a promising approach." [27]

These quotes highlight certain ideas. It is evident that there is concern over the ability of current data base management systems to handle very large data bases. Improved hardware seems to be the answer considering the comments on associative memories, LISP machines and data base machines. The join and projection operations of relational systems are specifically noted to be problems.

7.1 ASSOCIATIVE HARDWARE

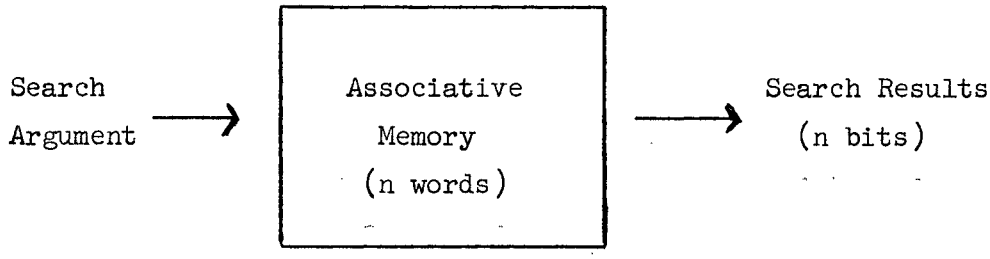
"Currently, the microprocessor/computer-on-a-chip revolution is providing the potential for production of very cost-effective high-performance computers through utilization of a large number of these processors in parallel or in a network." [28] An associative processor, as mentioned earlier, can retrieve stored data using the content of the data rather than its physical address. In addition, an operation such as retrieval can be performed in an associative memory on many pieces of data at the same time. Using Flynn's terminology [29] to classify computer architecture an associative processor falls into the SIMD class (parallel processors also fall into this class). SIMD means that there is a single instruction stream operation on a multiple data stream. This leads to the observation that due to this parallelism and the ability to retrieve information based on content, an associative processor will have a faster data processing rate than traditional devices. Given a faster data processing rate an associative processor will be "more effective in handling many types of information processing problems such as information storage and retrieval of rapidly changing data bases, fast search of a large data base, arithmetic and logical operations on large sets of data" [30] and others.

Associative processors are being used at present for some highly specialized data processing functions. Because of cost considerations these processors are small and are used for tasks

such as virtual memory management, resource allocation, interrupt processing and scheduling tasks. Use of associative processors will grow in the future but they will remain special purpose to be used in applications having a large number of independent data sets, that can be processed in parallel, a need for fast response and a need for addressing based on content.

It is beyond the scope of this paper to describe details of the various hardware techniques used to implement associative memories and processors however, the main features will be discussed.

One way to understand an associative memory is to consider that the input consists of a search argument of x bits and the output consists of a bit for each word in the memory indicating success or failure on matching the search argument.



Not all systems work in this fashion, some systems output the contents of those words in memory that matched the original search argument.

On a more detailed level, an associative processor contains a data register which is loaded with the data to be compared with

the data in memory. A mask register is used to mask off those fields in the memory that are not included in the search, a word select register indicates which words are to be searched, a results register contains one bit for each word and the bit is turned on if its corresponding word matches the search criteria. The number of matches is contained in the match indicator and the multiple match resolver points to the first word that matched. A control unit is used to specify the operation (eg. equals, greater than, less than) that is to be performed.

Consider the query introduced earlier: retrieve all claims less than one year old. In this example the data register contains the date, all fields except the date field are masked off via the mask register. The word select register would indicate that all words are to be searched and the operation less than is put into the control register.

Data Register

0	0	01 06 77	0	0

Mask Register

0	0	1	0	0

Data

C1	S1	06 06 76	PAID	$100	1	0
C2	S1	20 09 77	PAID	$ 30	1	1
C3	S3	04 04 78	REJECTED	$ 0	1	0
C4	S3	15 03 77	PAID	$ 12	1	0
C5	S3	31 01 78	PAID	$ 39	1	1
C6	S4	12 05 75	PAID	$ 52	1	0
					word select register	search results register

As a result the match indicator would have a value of two and the multiple match resolver would point to word two.

To be used effectively for data manipulation an associative processor consists of a number of identical memory cells. In addition, the retrieval time should be largely independent of the number of cells and the memory should be modularly expandable. There are four classifications of associative processors; fully parallel, bit serial, word serial and block oriented. Those that are fully parallel contain comparison logic within the associative memory for every bit-cell of every word. These are very expensive to implement. In a bit-serial associative processor one bit-column of all the words in associative memory is operated on at a time. A word-serial associative processor is essentially a hardware implemention of a program loop to search for a special value. These machines have the advantage, over standard sequential processors, of reduced instruction decoding time since only a single instruction is required to execute the search. However word-serial associative processors are slow in comparison with the other classes. Block-oriented associative processors use a mass rotating storage device such as a disk that has some logic associated with each track. Thus tracks can be search in parallel. Some example of associative processors will be discussed in the next section.

7.2 CASSM -- CONTEXT ADDRESSED SEGMENT SEQUENTIAL MEMORY

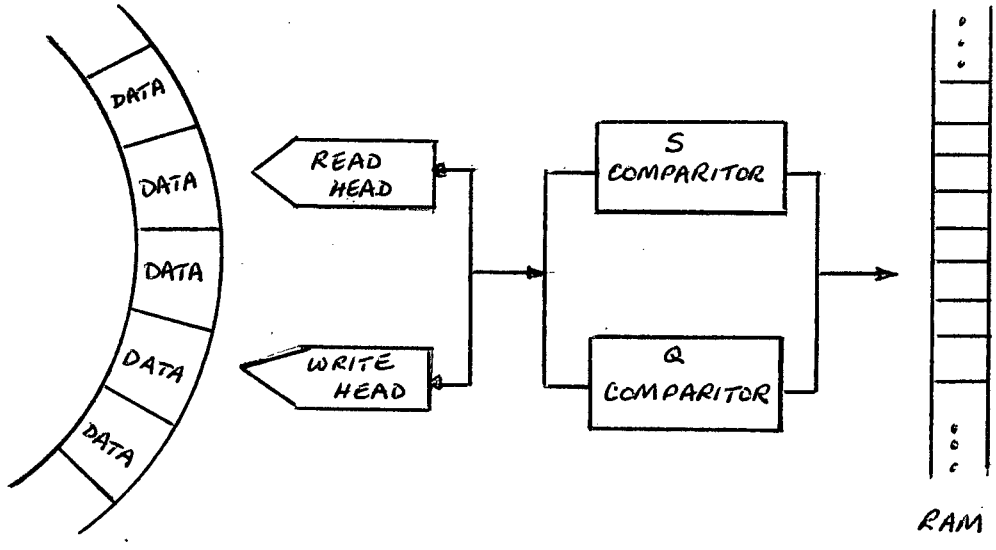
The basic concept behind CASSM is that all operations to be performed on the data base are done directly in disk memory. This eliminates the need to schedule paging of data between disk and main memory of the processor. In addition, "parallelism is used to make the time to search the data base independent=of the data base size." [31] Thus the entire data base can be searched by hardware for each search instruction. Most high level retrieval languages allow the user to express parallelism in their queries, an architecture that provides parallelism in retrieval eliminates the need to translate the query into a long complicated set of procedures (consider IMS for example).

CASSM consists of identical cells, connected through an IO bus. In addition each cell can connect to two neighboring cells. A cell consists of a segment of memory (for example, a track) and a large section. "All segments of memory circulate concurrently and in synchronization, while each logic section reads, searches, modifies and rewrites its segment of memory from one end to the other. Thus, all segments of memory are operated on in one circulationoof memory." [32]

A query against a data base can be characterized as having a specification part and a qualification part. For example, SUBSCRIBER . SIN : SUBSCRIBER . SEX = M, specifies that all SINs for SUBSCRIBERS are to be retrieved, given the qualification

that the SEX of the SUBSCRIBER is male. This query implemention in CASSM involves searching and marking all occurrences that satisfy the above condition. There are two comparator registers within each cell, one for the specification and one for the qualification thus allowing the query to be satisfied in one sweep of memory.

A RAM (one for each cell) that is one bit wide is used to mark the data items satisfying a given query. One bit is maintained for each data item in the cell and the association is maintained by relative position.



Consider how the SUBSCRIBER relation might be represented in CASSM.

SET TYPE	LEVEL NO.	INFORMATION FIELDS					
A	1	SUBSCRIBER					
AA	2	SIN	STATUS	NAME	LOCATION	BD	SEX
V	2	S1	ACTIVE	JONES	VANCOUVER	31 01 50	M
V	2	S2	ACTIVE	FORD	VICTORIA	15 06 46	F
V	2	S3	CANCELLED	KING	VANCOUVER	03 03 35	F
V	2	S4	ACTIVE	BOOTH	KAMLOOPS	27 11 42	M

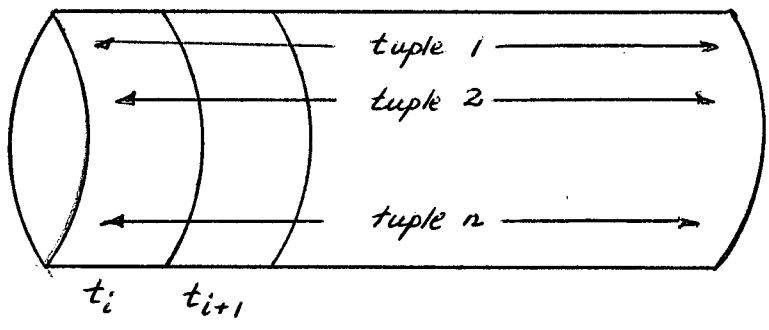
A set type of A indicates an attribute, whereas V indicates a value. The level numbers show how the rows of information fit in with their corresponding table. If a second table were included the level number would be reset to 1 and the attributes of that table shown before values would be given. The query SUBSCRIBER . SIN : SUBSCRIBER . SEX = M would be answered in three revolutions of the memory.

SEG REV.	SET TYPE	LEVEL NO.	S INFO.	Q. INFO.
1.	A	1	SUBSCRIBER	SUBSCRIBER
2.	A	2	SIN	SEX
3.	V	2	DON'T CARE	M

7.3 RARES : ROTATING ASSOCIATIVE RELATIONAL STORE

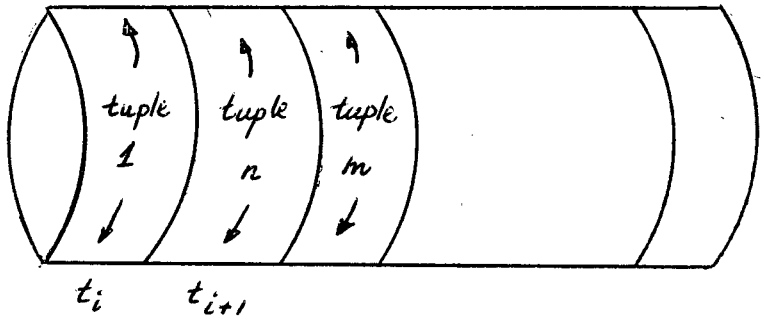
RARES can be implemented on a computer system that contains a CPU, random-access high speed main memory, and head-per-track rotating secondary memories. Data is transferred from main to secondary memories via channels. An associative memory is constructed by adding content-addressing hardware to the secondary memories. Selection of tuples in response to a query is performed at the secondary storage device; thus only correct tuples are sent via the channel to main memory. RARES searches all tracks on the storage device simultaneously. "The net result is that RARES can decrease the average utilization of CPU, main storage, channels and secondary storage devices by a query. In many cases this will allow the interface to assume a heavier query load without degrading response time, or alternatively, to offer a reduced response time with the same query load." [33]

RARES is implemented in much the same fashion of CASSM. However, the method of storing tuples on the secondary storage device is different. Tuples can be read from storage concurrently, however, the channel can only receive information (tuples) sequentially. In CASSM tuples are output one at a time from the device thus requiring several revolutions before all marked tuples are output. This is an inexpensive solution to the problem. RARES takes the approach of laying out tuples on the storage device across tracks rather than along tracks. When a search operation is complete tuples are then already in a form suitable for fast output.



RARES layout

t_i = track i



CASSM layout

t_i = track i

Relations are often stored in insort order because this allows queries to be process more efficiently in many cases. RARES because of its orthogonal layout can preserve this sort order and process against it more efficiently than CASSM. In fact the only way for CASSM to preserve a sort order is to search one track at a time. Using this technique the output rate for CASSM would be much slower than RARES.

7.4 RAP : A RELATIONAL ASSOCIATIVE PROCESSOR

This particular system will be described in more detail than CASSM or RARES. A virtual memory system for RAP which allows large relational data bases will be examined. Then some performance evaluation statistics comparing RAP to a conventional relational DBMS will be shown.

Certain features are essential to data base management and RAP has been implemented with these features in mind:

" (a) A large capacity and modular storage with low cost per bit.

(b) Ability to directly map logical data structures into physical data structures without using auxiliary storage structures.

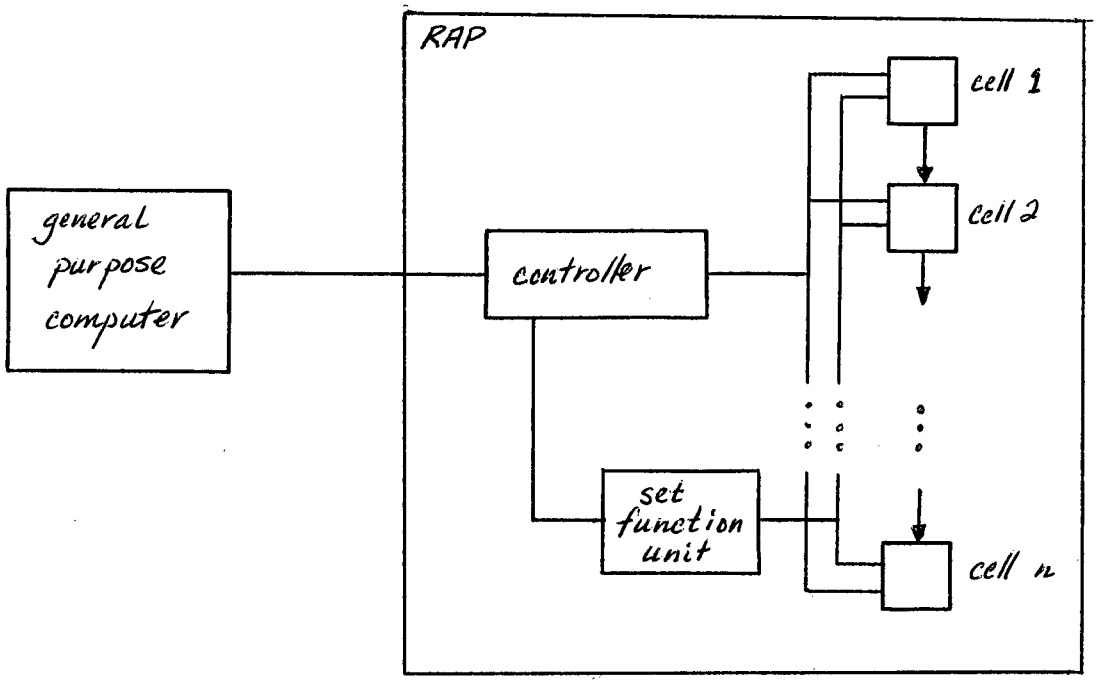
(c) Variable length data formats.

(d) Fast retrieval and update suitable for on-line concurrent environment.

(e) Context . . . search operations assisted by total associativity.

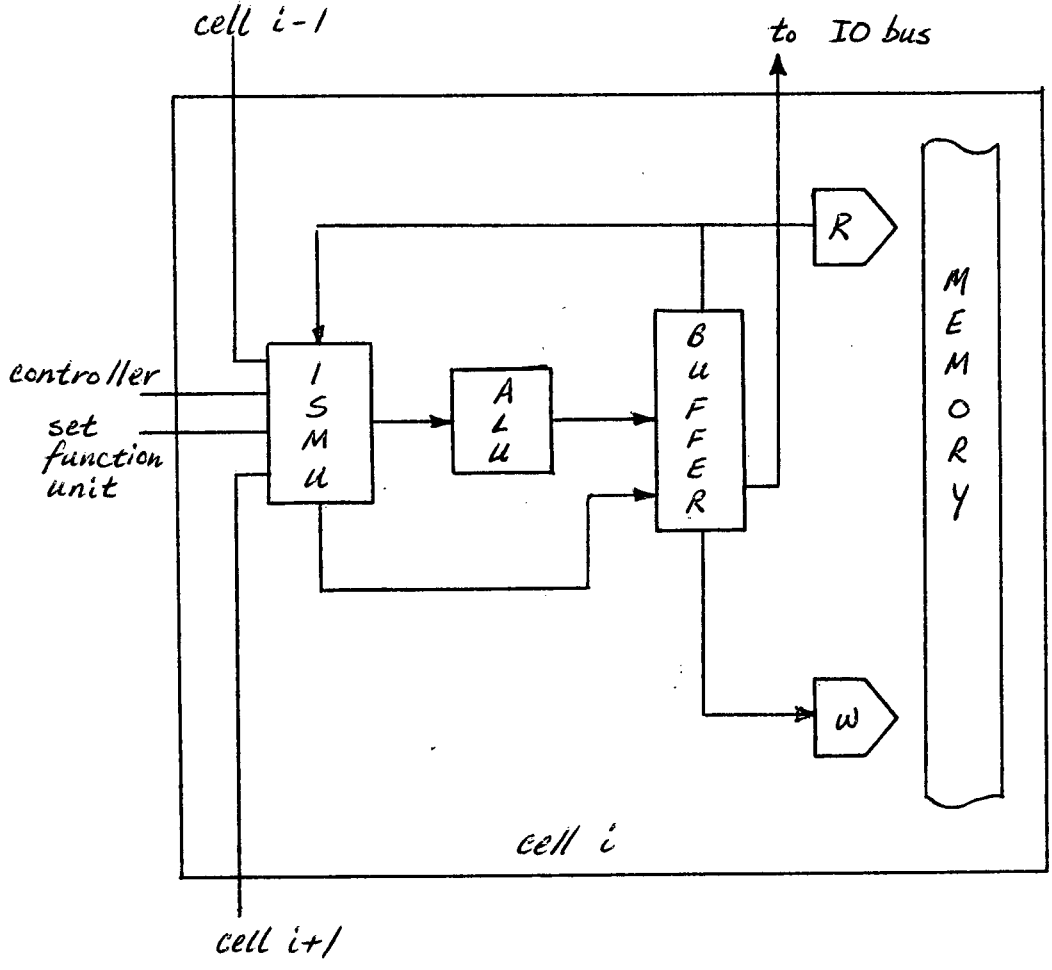
(f) Simple in-place arithmetic computations and update. " [34]

Consider the following overview diagram of RAP:



RAP is a special purpose processor that communicates with a general purpose computer. Each cell within RAP, as in the other architectures, contains a memory component and a logic component. The memory component is a track of a disk and the logic component "is a micro processor which acts as a 'search machine' on data, directs data manipulation, and performs limited numeric computations required by data base processing." [35] The set function unit provides logic to combine results of a search, for example COUNT, SUM, MAXIMUM, MINIMUM, AVERAGE. The controller co-ordinates cell searches and initiates the set function unit if required.

As in the other associative systems, each cell is searched simultaneously. Thus response time is largely independent of data base size. A typical query consists of one or more RAP instructions and each instruction is executed with one rotation of the RAP memory. A cell is organized as follows: [36]



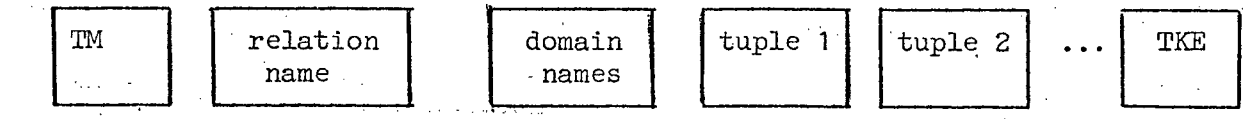
Data is stored in the memory area and is read and written (R and W) via fixed heads. Within one revolution the entire contents of memory can be read. A buffer with a length of 1024 bits is used to provide a time delay between reading and writing information to memory. This time delay allows the ISMU, information search and manipulation unit, to perform the necessary logic on the data. The ISMU is also responsible for inter-cell communication, command decoding, I/O data transfers and ALU control.

RAP instructions are provided for retrieval, update, insertion, deletion, data base create and destroy, control (eg. branching) and set functions such as SUM, COUNT, etc. These are implemented through the controller. Each RAP instruction is executed in parallel by each cell within the processor. Thus a data base operation is executed against every piece of data in the data base in one revolution of the disk.

A general purpose computer interfacing with RAP must provide data communication facilities for users, compilation of user queries into basic RAP instructions, transfer of instructions to the RAP controller, support of a concurrent processing environment, and maintenance of data base structure information.

"RAP accomplishes relational data base management without complex data structures and software aids such as inverted lists and hashing for multi-key searching required in conventional systems. This is especially important for applications which have extensive update activity." [37]

To understand more about RAP it is useful to see how relations are stored within memory. [38.7]



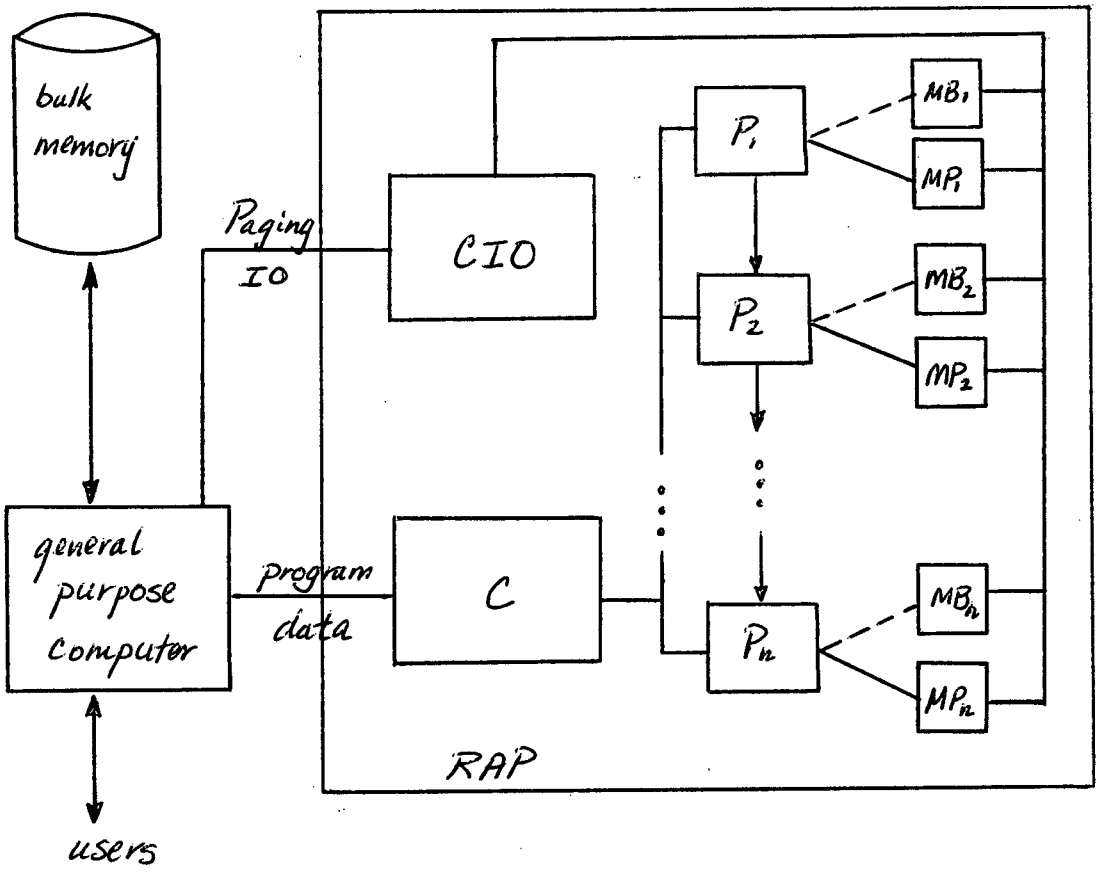
In contrast with RARES, RAP stores tuples in a linear fashion along a track rather than across several tracks. TM marks the beginning of the track, following this are stored the relation name, the domain names and the tuples of that relation. One tuple, TKE, delimits the end of a relation. Relation and domain names are repeated once on each track of a relation. Also, no cell can have tuples from more than one relation although a relation could easily spread over more than one track.

Each tuple has a delete flag, and four mark bits A, B, C and D. A tuple with a delete flag on is liable for garbage collection. The four mark bits are used to further qualify sets of tuples and to allow the results of one instruction to be used by another instruction. For example, if a query is given with multiple qualification phrases then the first qualification is implemented with one RAP instruction and appropriate mark bits are set, the second qualification tests for those mark bits as well as the second boolean condition of the query.

RAP with VIRTUAL MEMORY

One of the problems with current associative memories is their size, for example, a RAP processor that can be realistically implemented given current LSI technology would contain 10^8 to 10^9 bits of memory. Large data bases require much more than this. To solve this problem a virtual memory system was designed for RAP.

The overall architecture of a virtual memory for RAP is shown below [39]:



Bulk memory contains the data base and is divided into fixed size pages equal to the capacity of one cell. Each page contains information from one relation only so that when a page is transferred to a cell, the cell then contains data from one relation only, as mentioned in the previous section. Data is transferred from bulk memory to RAP under control of the general purpose computer. The basic concept is that enough data to answer a query is paged from bulk memory into RAP cells. However, in order to make this as efficient as possible paging is overlapped with query processing, and to this end each cell contains two memories, MB_i, buffer memory for $cell_i$, that is being loaded with information and MP_i, processor memory for $cell_i$, which is being used to execute the currently active query. When a query is complete buffer memory becomes processor memory and the next query can be executed.

A controller, C, functions as previously explained, receiving programs from GPC and transferring query results to GPC. CIO, the IO controller is connected to GPC via a separate channel.

Overlapping of paging and query processing can be achieved relatively easily. Each page holds information for one relation only and each page has a unique identification. When a query is specified the relation must also be specified thus it is a simple matter to know, based on a relation name, what pages are to be sent to RAP for processing. Note that there is an assumption that RAP

can contain all information required to process any given query although it is not able to contain the entire data base.

A virtual memory system is significant to the feasibility of a design such as RAP especially when paging can be overlapped with query processing to improve efficiency. Results of a simulation study done for this virtual memory system [40] showed that response time for an average query is directly proportional to the size of relations. Considering that all pages of a relation must be in RAP memory before a query is executed this result is not surprising. Locality of information has some bearing on response time since it is assumed that the higher the locality the more chance there is that relevant relations will already be in RAP memory. If the data base is very large however, the effect of locality will diminish.

7.5 RAP PERFORMANCE

A study has been done to compare the performance of RAP with a conventional relational data base management system. Models developed for the conventional system assumed that inverted lists were provided for selected attributes of each relation. (The selection would be made based on frequency of access via any particular attribute.) The models developed quantified the times required to perform basic operations such as simple retrieval, update, qualified retrieval and implicit join.

On retrieval "RAP's advantage grows in almost direct proportion to the number of records satisfying the qualification when inverted lists are used in the conventional system." [41] The advantage ranges from 40% to 90% depending on the level of qualification. For select and update operations RAP also has an advantage. The operation modelled in this case involves selecting a set of tuples based on some qualification and then updating the selected tuples. Use of inverted lists in the conventional system causes problems because when an attribute that is modified (on update) is an inverted attribute the appropriate inverted list must also be modified. In contrast, RAP can perform selection and update (remember the read write heads) within one revolution of RAP memory in most cases. Selection followed by deletion involves the same problem with inverted lists in a conventional system (remember links and images in System R).

Another operation studied consists of a select, computation of some value based on the tuples selected and then retrieval of other tuples based on that value. Because of the need to access data in the conventional system through inverted lists RAP has the advantage in this operation as well. In fact, the advantage is particularly significant when the number of tuples involved in the computation is large.

As associative processor such as RAP has a significant performance advantage over conventional hardware in supporting data base management. The characteristics of associative processors that lead to such an advantage are:

1. an ability to search data fields in parallel.
2. search logic imbedded in the storage device which allows data comparison to take place without passing data back to the central processor.
3. retrieval based on content which means that complicated pointer schemes and index data bases and their maintenance are not required.
4. reduced interface problems because the physical storage of data more closely resembles the users view of data.

8.0 CONCLUSIONS

Data bases are becoming increasing important to today's business environment. As companies become aware that data is a resource to be developed, shared and efficiently managed, data bases are increasing in size and access to them is more varied. It is essential to find efficient ways of accessing data both in terms of people time (eg. how long does it take a person to get access to required information) and in terms of machine time.

This paper has described a hierarchical data base management system, IMS and a relational data base management system, System R. The focus has been on showing that IMS, because of its structured approach to modelling data, is more efficient than System R as far as machine time is concerned. System R, on the other hand, with its high level data management language SQL is more efficient in terms of people time.

In discussing associative processors it is shown that implementing a relational data base management system using associative processors allows both efficient machine time and people time.

REFERENCES

1. McGee, W.C., "Generalization: Key to Successful Electronic Data Processing", JACM 6, January 1959, pp. 1 - 23

2. Senko, M.E. et al, "Data Structures and Accessing Database Systems", IBM Systems Journal, Vol. 12, No. 1, 1973, page 217

3. "Feature Analysis of Generalized Data Base Management Systems", CODASYL Systems Committee, May 1971

4. Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Co., 1975, page 21

5. Ibid, page 180

6. Ibid, page 183

7. McGee, W.C., "The Information Management System IMS/VS, Part I and Part II", IBM Systems Journal, Vol. 16, No. 2, 1977, page 109

8. Ibid, page 110

9. "IMS/VS Version I System/Application Design Guide", IBM Form No. SH20 - 9025-4, page 4 - 40

10. Ibid, page 4.51

11. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", CACM Vol. 13, No. 6, June 1970, page 377

12. Ibid, page 379

13. Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Co., 1975

14. Ibid, page 54

15. Codd, E.F., "Relational Completeness of Data Base Sublanguages", Data Base Systems, Courant Computer Science Symposia Series, Vol. 6, Prentice-Hall, 1972

16. Denny, G.H., "An Introduction to SQL, A Structured Query Language", IBM Research Laboratory, San Jose, California, RA93, 1977

17. Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Co., 1975

18. Blasgen, M.W. et al, "System R : A Relational Approach to Data Base Management, Part 3 : The Relational Storage System", IBM Technical Report, 1977

19. Astrahan, M.M. et al, "System R : Relational Approach to Database Management", ACM TODS, Vol. 1, No. 2, June 1976, page 110

20. Gotlieb, L.R., "Computing Joins of Relations", ACM SIGMOD Conference, San Jose, California, May 1975

21. Astrahan, M.M. et al, "System R : Relational Approach to Database Management", ACM TODS, Vol. 1, No. 2, June 1976, page 100

22. Lochovsky, F.H. and Tsichritzis, D.C., "User Performance Considerations in DBMS Selection", ACM SIGMOD International Conference on Management of Data, Toronto 1977

23. Michaels, A.S. et al, "A Comparison of the Relational and CODASYL Approaches to Data-Base Management", Computing Surveys, Vol. 8, No. 1, March 1976, page 146

24. Proceedings of the Third International Conference on Very Large Data Bases, Tokyo, Japan, 1977, page 195

25. Ibid, page 196

26. Ibid, page 197

27. Ibid, page 198

28. Thurber, K.J. and Wald, L.D., "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, December 1975, page 215

29. Flynn, M.J., "Some Computer Organizations and their Effectiveness", IEEE Trans. Computers C-21, No. 9, September 1972

30. Yau, S.S. and Fung, H.S., "Associative Processor Architecture -- A Survey", Computing Surveys, Vol. 9, No. 1, March 1977, page 3

31. Copeland G.P. et al, "The Architecture of CASSM : A Cellular System for Non-Numeric Processing", Proc. of the First Annual Symposium on Computer Architecture, 1973, page 121

32. Ibid, page 123

33. Lin, C.S. et al, "The Design of a Rotating Associative Memory for Relational Database Applications", ACM TODS, Vol. 1, No. 1, March 1976, page 54

34. Ozkarahan, E.A. et al, "A Data Base Processor", Computer Systems Research Group, University of Toronto, Technical Report CSRG - 43, page 8

35. Ibid, page 10

36. Ibid, page 14

37. Schuster, S.A. et al, "A Virtual Memory System for a Relational Associative Processor", Computer Systems Research Group, University of Toronto, Technical Report CSRG - 64, page 6

38. Ozkarahan, E.A. et al, "A Data Base Processor", Computer Systems Research Group, University of Toronto, Technical Report CSRG - 43, page 28

39. Schuster, S.A. et al, "A Virtual Memory System for a Relational Associative Processor", Computer Systems Research Group, University of Toronto, Technical Report CSRG - 64, page 12

40. Ibid, general reference

41. Ozkarahan, E.A. and Schuster, S.A., "A High-Level Machine Oriented Language for a Data Base Machine", Computer Systems Research Group, University of Toronto, Technical Report CSRG - 65, page 20

BIBLIOGRAPHY

1. Aron, J.D., "Information Systems in Perspective", Computing Surveys, Vol. 1, No. 4, December 1969

2. Astrahan, M.M. et al, "System R: Relational Approach to Database Management", ACM TODS, Vol. 1, No. 2, June 1976

3. Bachman, C.W., "The Evolution of Storage Structures", CACM, Vol. 15, No. 7, July 1972

4. Bachman, C.W., "The Programmer as Navigator", CACM, Vol. 16, No. 11, November 1973

5. Blasgen, M.W. et al, "System R : A Relational Approach to Database Management, Part 3 : The Relational Storage System", IBM Research Laboratory, San Jose, California

6. Blasgen, M.W. and Eswaran, K.P., "Storage and Access in Relational Data Bases", IBM Systems Journal, Vol. 16, No. 4, 1977

7. Chamberlin, D.D. et al, "SEQUEL 2 : A Unified Approach to data definition, Manipulation and Control", IBM Journal of Research and Development, Vol. 20, No. 6, 1977

8. Chamberlin, D.D., "Relational Data Base Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976

9. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", CACM, Vol. 13, No. 6, June 1970

10. Codd, E.F., "Relational Completeness of Data Base Sublanguages", Data Base Systems, Courant Computer Science Sumposia Series, Vol. 6, Prentice-Hall, 1972

11. Copeland, G.P. et al, "The Architecture of CASSM : A Cellular System for Non-numeric Processing", Proceedings of the first annual Sumposium on Computer Archietecture, 1973

12. Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Co., 1975

13. Denny, G.H., "An Introduction to SQL, A Structured Query Language", IBM Research Laboratory, San Jose, California, RA93, 1977

14. Dodd, G.C., "Elements of Data Management Systems", Computing Surveys, Vol. 1, No. 2, June 1969

15. Flynn, M.J., "Some Computer Organization and their Effectiveness", IEEE Trans, Computers, C-21, No. 9, September 1972

16. Fry, J.P. and Sibley, E.H., "Evolution of Data Base Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976

17. Gotlieb, L.R., "Computing Joins of Relations", ACM SIGMOD Conference, San Jose, California, May 1975

18. Hollander, G.L., "Architecture for Large Computer Systems", Proc. AFIPS, SJCC, 1967

19. Lin, C.S. et al, "The design of a Rotating Associative Memory for Relational Database Applications", ACM TODS, Vol. 1, No. 1, March 1976

20. Lochovsky, F.H. and Tsichritzis, D.C., "User Performance Considerations in DBMS Selection", ACM SIGMOD International Conference on Management of Data, Toronto, 1977

21. Lorie, R.A. and Wade, B.W., "The Compilation of a Very High Level Data Language", IBM Research Laboratory, San Jose, California, RJ2008, 1977

22. Martin, J.A., "Computer Data Base Organization", Prentice-Hall Inc., New Jersey, 1975

23. McGee, W.C., "Generalization : Key to Successful Electronic Data Processing", JACM, Vol. 6, January 1959

24. McGee, W.C. "The Information Management System IMS/VS", IBM Systems Journal, Vol. 16, No. 2, 1977

25. Michaels, A.S. et al, "A Comparison of Relational and CODASYL Approaches to Data Base Management", Computing Surveys, Vol, 8, No. 1, March 1976

26. Minker, J., "An Overview of Associative or Context-Addressable Memory Systems and a KWIC Index to the Literature : 1956-1970", Computing Reviews, October 1971

27. Mowshowitz, A., "The Conquest of Will : Information Processing in Human Affairs", Addison-Wesley Publishing Co., 1976

28. Ozkarahan, E.A. and Sereik, K.C., "Analysis of Archetectural Features for Enhancing the Performance of a Database Machine", ACM TODS, Vol. 2, No. 4, Decefber 1977

29. Ozkarahan, E.A. et al, "A Data Base Processor", University of Toronto, Technical Report, CSRG-43

30. Ozkarahan, E.A. and Schuster, S.A., "A High Level Machine - Oriented Assembler Language for a Data Base Machine", University of Toronto, Technical Report, CSRG-74

31. Ozkarahan, E.A. et al, "Performance Evaluation of a Relational Associative Processor", University of Toronto, Technical Report, CSRG-65

32. Rosen, S., "Electronic Computers : A Historical Survey", Computing Surveys, Vol. 1, No. 1, March 1969

33. Schuster, S.A. et al, "A Virtual Memory System for a Relational Associative Processor", University of Toronto, Technical Report, CSRG-64

34. Senko, M.E. et al, "Data Structures and Accessing in Database Systems", IBM Systems Journal, Vol. 12, No. 1, 1973

35. Senko, M.E., "Data Structures and Data Accessing in Database Systems, Past, Present, Future", IBM Systems Journal, Vol. 16, No. 3, 1977

36. Stonebraker, M. and Held, G., "Networks, Hierarchies and Relations in Data Base Management Systems", Proceedings ACM Pacific Conference, San Francisco, April 1975

37. Thurber, K.J. and Wald, L.D., "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, December 1975

38. Yau, S.S. and Fung, H.S., "Associative Processor Architecture - A Survey", Computing Surveys, Vol. 9, No. 1, March 1977

39. "Feature Analysis of Generalized Data Base Management Systems", CODASYL Systems Committee, May 1971

40. "IMS/VS Version 1, System/Application Design Guides, IBM Form No. SH20-9025-4.

41. Third International Conference on Very Large Data Bases, Tokyo, Japan, 1977