DESIGN

OF A

RELATIONAL

DATA BASE MANAGEMENT SYSTEM

by

John M. Vujcich

A thesis presented in partial fulfilment
of the requirement for the degree of

Master of Science in Computer Science

at

MASSEY UNIVERSITY

February 1980

ABSTRACT

This thesis explains the design of a relational data base management system. In an effort to achieve a system which is shared, on-line, easy to use, responsive, capable of growth, capable of change, and having extensive security facilities, many innovations have been introduced. All data needed for enterprise operation and DBMS operation are stored in data base relations; administrators are considered as users; and one language is given with facilities for defining data, declaring mappings, defining comprehensive security/integrity constraints, and declaring new DBMS operations. Finally, a primitive language is given, which allows for a practical implementation of these innovations with a result of increased overall system performance, greater flexibility, and use of modern micro-processor technology.

ACKNOWLEDGEMENTS

In presenting this thesis I would like to express my thanks to the following good people:

To my supervisor, Peter J. Melhuish, without whose initial guidance and encouragement this thesis may not have reached fruition;

To Tom Docker and Ian Gillespie for their invaluable criticism in the final stages of its preparation;

To Professor Graham Tate for his patience and understanding;

Finally to all the members in the Department of Computer Science for their part in sustaining a pleasant and stimulating study environment.

Massey University

John Vujcich

February 1980

PREFACE

It is the contention of this thesis that the relational DBMS offers many practical advantages in both simplifying problems and extending the capabilities of modern DBMSs. Particular emphasis is placed upon the following two aspects.

1) The potential of a relational data base language in achieving user objectives.

2) The feasibility of implementing such a language in a fashion that lends itself to data base processor technology.

The task is handled by presenting an example design of a relational DBMS in which these objectives and their resulting innovations are given particular emphasis.

Chapter 1 gives a brief overview of the proposed DBMS. Major components are outlined and the various features that result from the above aspects are described.

Chapters 2 and 3 consider the detail of the relational Calculus. They cover the problems of implementing the proposals in this one language. Chapter 2 concentrates on the relation manipulating features of the language while Chapter 3 describes the definition and controlling features of the language.

Chapters 4 and 5 consider the detail of a primitive language and the problems associated with parsing the Calculus into this primitive language. Chapter 4 defines the primitive language and examines the problem of parsing the Calculus GET statement. Chapter 5 considers how the other Calculus constructs can be expressed as a set of primitives.

Particularly it shows how mappings and constraints can easily be implemented.

As yet, no standardisation has occurred to any great extent in the DBMS environment. Thus, a degree of variation in the meaning of definitions often occurs which sometimes obscures and complicates even simple concepts. The terminology used in this thesis follows a generally accepted norm, and all significant variations from this norm are clearly indicated. No attempt is made to introduce the reader to DBMS concepts, instead, the reader is referred to the excellent books of Date (26, 27) and Martin (50, 51). However, throughout the thesis attempts have been made to sustain a general perspective of the subject by including definitions where it is felt that they would be of particular importance in highlighting design decisions.

TABLE OF CONTENTS

CHAPTER 2 (continued)

CHAPTER 4 (continued)

LIST OF FIGURES

THE PROPOSED

DATA BASE MANAGEMENT SYSTEM

1

A modern DBMS must achieve many different objectives*. These may be very general in nature such as "data availability", or limited such as "good response".

Different enterprises assign a relative priority or weight to each of these objectives. In a specialised application some may be considered as being only minor while others may be considered as being most essential. Thus there is in effect a grouping of objectives into primary and secondary classes depending upon the particular implementation. One major aspect complicating such a grouping is the inter-relationships and dependencies that exist between various objectives. Many of these aid the development of some other objectives, but many also hinder the development of yet others. So in general cases, it is necessary to obtain some optimum compromise between the various objectives. Typically this entails the development of a general, well designed DBMS with good data availability, good security and integrity facilities, evolvability, one that is shared, and has acceptable development and running costs. It also follows that no one DBMS design can be considered as "the best" for all possible applications. Thus the proposals given here are not intended to present an ideal DBMS but rather one that explores the possibilities of the following five major innovations:

---

* *Everest (35)*

(1) Inclusion of all data necessary for the operation of the DBMS in the one data base.

(2) Similar treatment of all users from casual users to administrators.

(3) Use of one relational calculus language for all users.

(4) The ability to define through this language extensive security and integrity facilities as well as new DBMS operations.

(5) Use of a primitive language which allows greater flexibility and performance.

These proposals and their ramifications result from the emphasis on user simplicity and DBMS flexibility. A DBMS is desired which is easy to use, simple in concept, powerful in operation and yet still flexible enough to be tailored to specific enterprise requirements.

It is recognised that other important variables not given much attention here also have a major effect on the architecture of a practical DBMS. Three such variables are:

(1) Size of the data base.

(2) Hardware resources available - particularly storage space.

(3) The degree of data base distribution.

1.1 What Data Model?

One of the first things that must be decided upon when designing a particular DBMS is the data model, or models , it is to support. The hierarchical, the network and the relational models are the three most common models in existence today. Of these three data models the relational model comes closest to achieving the desired objectives.

This is clearly evident from the advantages seen in the relational approach as outlined by Date (26,27). The main areas of concern are as follows.

(1) Simplicity. One of the most simple representations for data is in the form of flat files. The system becomes easier to use and maintain as well as having greater clarity and precision. Users are no longer confronted with a mass of pointers, nor are they mislead by ambiguous directed links. Therefore integration and sharing is easier to implement. Finally the full power and precision of the mathematical nature of relations can be reaped.

(2) Flexibility. By using relations it becomes easier for the user to retrieve, modify, add, and delete data in a generalised manner. That is, the manipulation language need not be so procedural in nature. It is possible to express complex security and integrity constraints with ease. New domains and relations together with the complex relationships between them can also be easily added, modified or removed.

(3) Ease of Implementation. Many of the desirable objectives can be implemented as the physical issues are independent of the logical. Thus it is possible to have a structured approach to implementation resulting in greater "inter-system" compatibility and a higher degree of data independence. In this way the system is more general, capable of being manually or automatically tuned, has greater data availability, extensibility and evolvability. Also there is the added advantage of it being easier to physically store flat files rather than tree or plex structures.

1.2 System Components

Overall, the proposed architecture does not differ significantly from the currently accepted "standard" view. The only difference is one of

simplification. Here the problem of distributed data bases and multiple storage schemas is not considered in any great depth. Rather the internal schema and conceptual schema of ANSI/X3/SPARC (2) are replaced by a single schema. See Figure 1.2:1. The proposed extensions therefore exist in the methods of implementation and operation of DBMS components; particularly the schema, mappings, definitions, constraints, and the language.

1.2.1 The Schemas

The data base is generally visualised as consisting only of the operational data. And so the resulting schemas are considerably biased in their contents. Unfortunately such a view tends to disagree with the need for a considerable amount of "other"* data necessary to support each datum of operational data. Thus difficulty is often experienced when attempts are made to include this data in the data base, as typically seen in the problems associated with data dictionary implementations. Such a distinction forces the DBMS to handle the control data differently. So users requiring access to it often find they have to use special language facilities or special operations. These can be quite different and often more complex than those normally associated with operational data manipulation. This problem also applies to the DBMS itself as it frequently requires access to control data. Hopefully it will only be a matter of time before the necessity for such a distinction is seriously questioned.

It is proposed here that all the data necessary for the operation of

---

* *The "other" data will be referred to as "control data", or "system data", so that there is no confusion with the operational data. It will normally consist of all the data needed for the operation of the DBMS. For example, security constraints, user profiles, time of day and the like.*

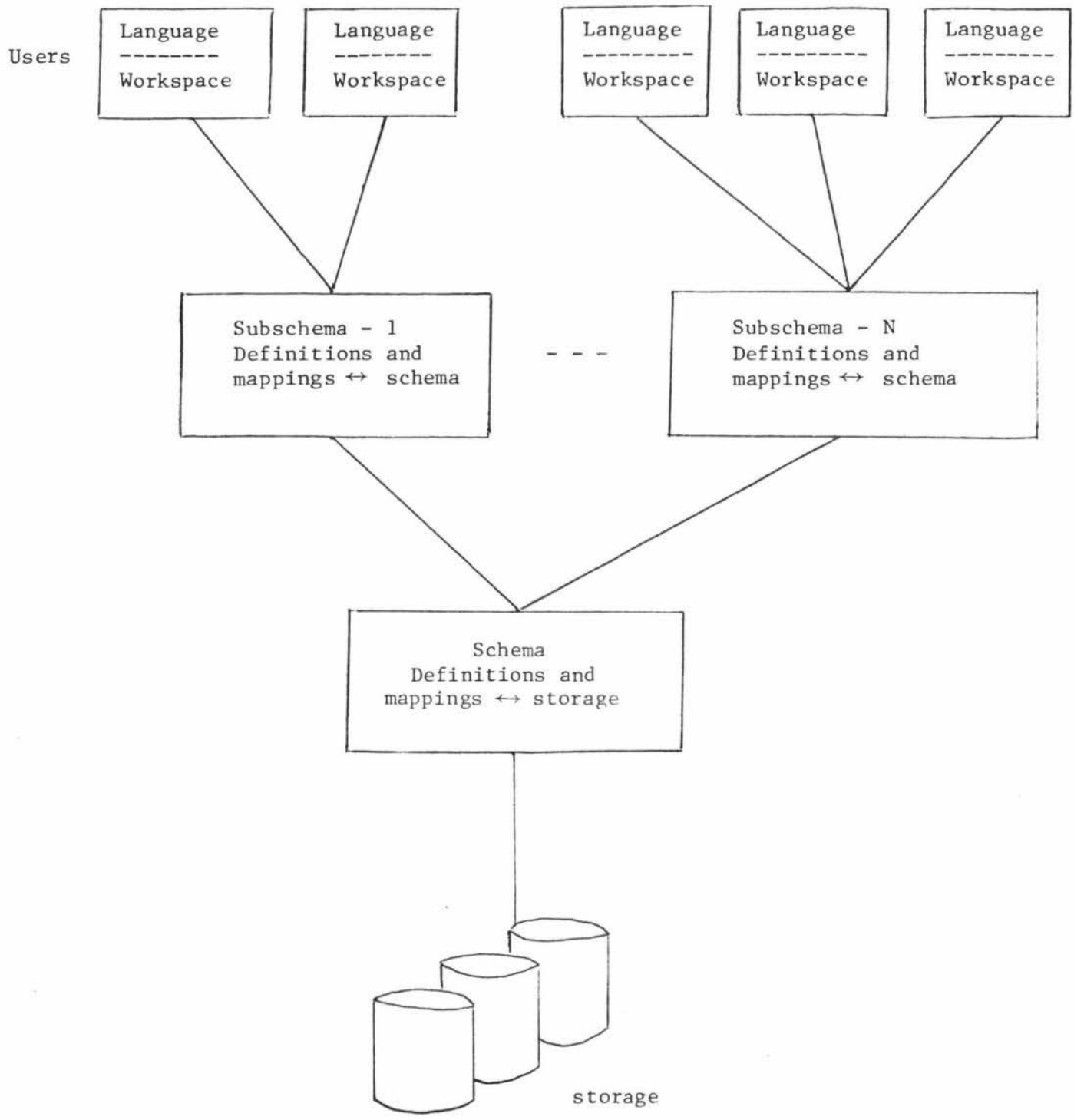

Figure 1.2:1

Components of the
Proposed DBMS

both the enterprise and the DBMS are stored in the data base. It also follows that all this data should be stored, in the logical data base, in one form. That is, the schema consists of a set of relations together with mappings, definitions, and security/integrity constraints for both the operational data and the control data. Henceforth all reference to the schema refers to a schema of this form.

Considerable advantages can be gained from such a perspective. Some such advantages are as follows - by no means are the possibilities exhausted; more will be seen in later chapters.

a) A relation defining all domains available to a given user may be accessed as simply as any other relation which consists of operational data. In short data dictionaries can easily be established, accessed, extended and modified.

b) Administrators will have simplified access to the data they need for DBMS control. Also DBMS tuning and management will be much easier for administrators. For example, new users can be included by simply adding a tuple to the user profile relation.

c) Data dictionaries, user profiles, audit trails and all such control data can easily be protected by extensive security and integrity constraints in exactly the same manner as operational data is protected.

d) It may even be an advantage to include parsing information in the data base. For example, suppose a symbol table containing the symbols of allowed language constructs is kept for each user in the data base, then the parser will simply not recognise any unauthorised user statement and so will treat it as if it were just any other nonsense symbol. Also, such a relation can then easily be assessed by an administrator whenever it is

necessary to extend the language facilities of a particular user.

1.2.1.1 Subschema

The subschema is simply a subset of the above extended schema. Thus, not only is it possible for some users to view a subset of the operational data, but now it is also possible for users to view a subset of the control data. So a user's view may include a subset of the data dictionary, storage data, audit relations, or may even exist entirely of dummy relations containing training data. See figure 1.2:2.

All the usual rules and advantages gained from using a subschema also exist here. For example, subschemas are particularly useful for achieving logical data independence and aiding system security. Each user has a library of subschemas from which one is usually chosen during log-on. But in the majority of cases this library will consist of only one subschema. Also this subschema will often be shared by a number of users who, preferably, require similar data facilities. Each subschema contains its own set of security/integrity constraints and control instructions which can either apply to specific users within the subschema, or to the subschema in general. These contraints are intended to further restrict the possible use and values of the data over and above those already given in the schema. This is because all schema constraints have the highest priority, so it is meaningless to include subschema constraints which allow a wider range of possibilities. Finally the subschema is supported by administrator written definitions and mappings which can be easily modified to suit changing user requirements.

1.2.2 Administrators

To date, considerable effort has been spent on the problem of simplifying casual user access. But, by comparison, little has been expended on the task of simplifying administrator access. If users are to be considered as enterprise personnel who require access to the data base in the course of completing their task, then, administrators would be one of the most frequent users. Their task consists of maintaining the enterprise's data base. In this thesis administrators will be considered as simply advanced users. It therefore will be possible to subject them to any necessary integrity of security constraints. Also available to them, and any other user , are all the advantages in simplification the system can offer. For example, the power and flexibility of the language, the capability of viewing data through a subschema, ease of access, and so on. This is achieved by placing all the data required for DBMS operation into the data base in much the same way all data required for enterprise operation is stored. In addition just as the operational data models the enterprise's operation, so too should the system data model the DBMS's operation.

Therefore administrators can select, through subschemas, their own limited model of DBMS operation, just as other users can select, through subschemas, a limited subset of the operational data. So administrators can now access relations containing performance data, or audit data as easily as accessing the operational data. See Figure 1.2:2. But one other major consideration remains. To the enterprise the computerised data base is just a handy storage medium. As the enterprise functions, data concerning its operation is continually fed into the DBMS. From there this data can be quickly and easily accessed and analysed by users who directly support the enterprise operation. As a result these users can then make minor alterations

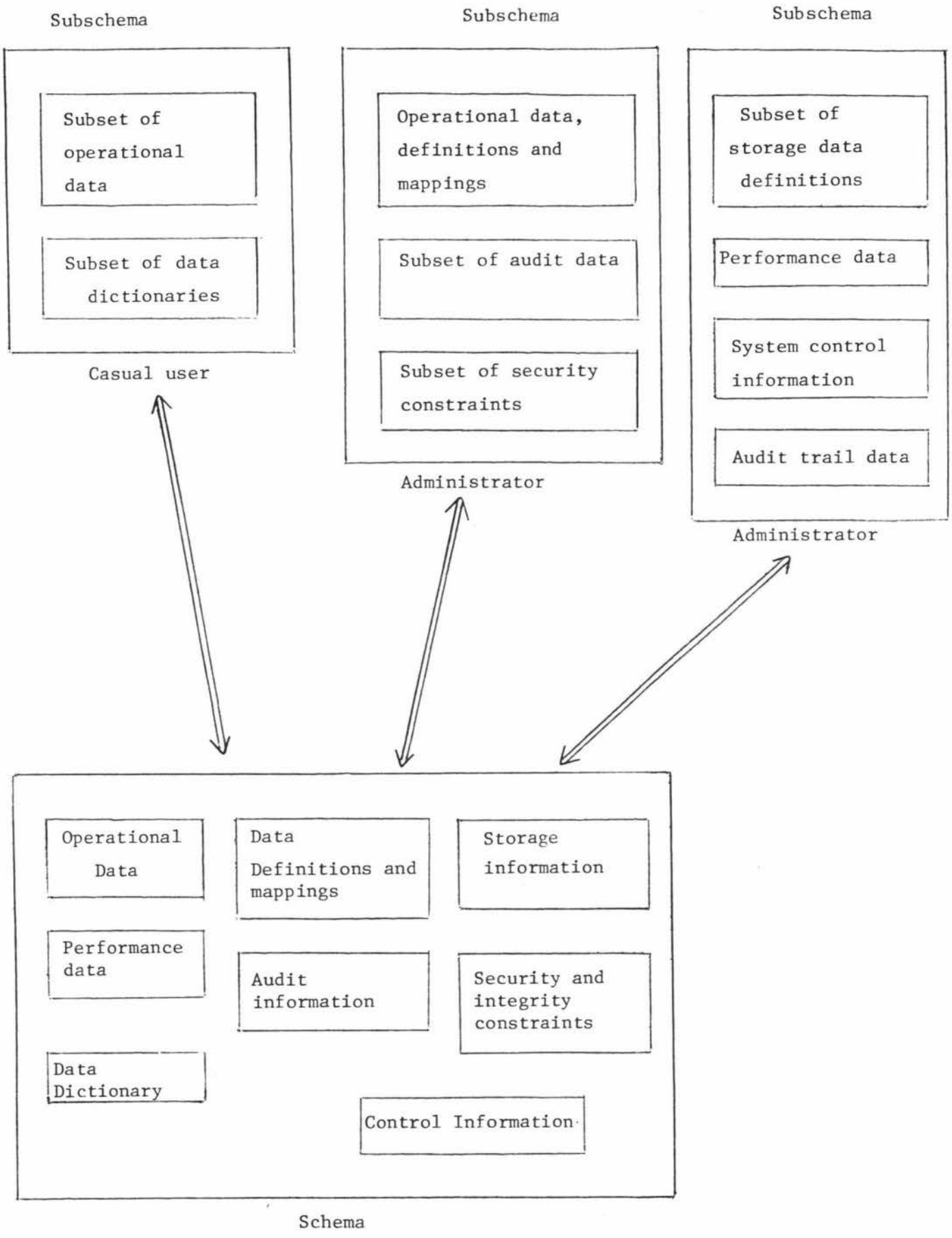

Figure 1.2:2

Example Schema and Subschemas, showing possible data content.

to the functioning of the enterprise. Likewise, if the DBMS were to collect its own data concerning its operation and store this system data in the data base, then, from there it can be easily and quickly analysed by administrators. Further, if the DBMS were to monitor these relations it could then change its physical operation to suit the modelled operation. Thus an administrator can change a physical parameter of the DBMS, say block-size, by simply modifying a tuple value in some system relation which specifies that block size. These and other possibilities will be seen in Chapters 4 and 5.

1.2.3 The Proposed Language

Clearly a DBMS language directly affects the users. It is perhaps one of the most important aspects of a DBMS. A good language enables many of the desired objectives to be achieved, in particular, data independence, simplicity, sharing, and data availability. The two most common mechanisms used in the relational DBMS context are the relational algebra and relational calculus. Both of these offer a high degree of data independence and powerful manipulation facilities. Unfortunately, both also have their disadvantages that must be overcome by a practical DBMS.

1.2.3.1 Relational Algebra

A relational algebra consists of a set of operators which operate on a relation, or a number of relations, and produce from them a resultant relation according to some criterion. A user must define a sequence of such operators which operate on data base relations in a way that will construct the desired resultant relation. Actually, this is not as difficult as it may seem. There is a wide selection of these generalised operators and they can be used to achieve any desired tabular representation of the data. As an example, consider

only two such operators: projection and join.

Projection

Projection operates on one relation, selecting from this relation a set of attributes, or columns , which are then ordered as specified to form a resultant relation. All redundant tuples that arise in this resultant relation are automatically removed. See Codd (21) for a full definition.

Consider the following examples showing how a projection can be used to answer simple queries on the PART relation of Figure 1.2:3. Note that the notation used is similar to that of Codd (21).

1) "What are the different colours that a part may have?"

This query can be answered by executing a projection of PART on attribute colour as symbolically represented below.

PART [COLOUR] =

COLOUR
RED
GREEN
BLUE

2) "What parts have the different colours?"

PART [COLOUR, P#] =

In this example notice that the order of the attributes is important.

COLOUR	P#
RED	P1
GREEN	P2
BLUE	P3
RED	P4
BLUE	P5
RED	P6

Join

If Ø represents any of the mathematical relations =, <, >, etc., then the join of a relation R on attribute A with relation S on attribute B is simply a resultant relation consisting of concatenated tuples, from the respective relations, whose specified attribute values satisfy the particular mathematical relation. Again see Codd (21) for a more precise and complete definition. Such a join can be represented as follows by using a notation similar to Codd's:

R (A Ø B) S

As an example, suppose that a user wishes to know the part numbers of all red parts. This can be extracted from the PART relation by joining it with the constant relation W shown below.

W

COLOUR
RED

PART (COLOUR = COLOUR) W

= W2

P#	PNAME	COLOUR	WEIGHT	QOH	COLOUR.W
P1	NUT	RED	12	26	RED
P4	SCREW	RED	14	24	RED
P6	COG	RED	19	3	RED

Now the resultant relation W2 can be reduced to only the relevant information by a further projection of W2 on P#.

That is, W2 [P#] =

P#
P1
P4
P6

Notice that the above two expressions can be combined into the following:

PART (COLOUR = COLOUR) W[P#]

Finally, while the algebra offers great flexibility, it is sadly lacking in the other necessities for a good language. That is, easy to use facilities for adding, modifying and deleting data, as well as facilities for writing security and integrity constraints.

1.2.3.2 Calculus Versus Algebra

With the relational algebra the user must specify the individual operations required to produce the desired data. However, with the relational calculus the user has only to define the result needed. This is a more natural approach, and is therefore helpful in simplifying the user interface. Also it leaves the DBMS free to decide which operations can best produce the result; thus it is possible to "optimise" the request. But perhaps the greatest advantage of the calculus is that it permits easy definition of security and integrity constraints, this is because the constraints can be based on a definition of the properties of the data. In Chapters 2 and 3 it will be seen how a calculus based on Codd's ALPHA can be extended to include mappings, definitions, and constraints. Perhaps the biggest disadvantage with the calculus is the difficulties it presents in implementation. A possible solution will be given in Chapters 4 and 5.

1.2.3.3 Proposals for the Language

If the above proposals are consistently applied, then, immediately a problem will be seen to exist with DBMSs that require different languages for different functions. That is, it is inconsistent to have one language for casual users, another for administrators, another for defining new data structures and so on. Surely even casual users may wish to add their own relations or define new

SUPPLIER

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

PART

P#	PNAME	COLOUR	WEIGHT	QOH
P1	NUT	RED	12	26
P2	BOLT	GREEN	17	8
P3	SCREW	BLUE	17	10
P4	SCREW	RED	14	24
P5	CAN	BLUE	12	35
P6	COG	RED	19	3

PROJECT

J#	JNAME	MGR-NO
J1	SORTER	M4
J2	PUNCH	M1
J3	READER	M3
J4	CONSOLE	M1
J5	COLLATOR	M4
J6	TERMINAL	M2
J7	TAPE	M5

SUPPLY

S#	P#	J#	QTY
S1	P1	J1	2
S1	P1	J4	7
S2	P3	J1	4
S2	P3	J2	2
S2	P3	J3	2
S2	P3	J4	5
S2	P3	J5	6
S2	P3	J6	4
S2	P3	J7	8
S2	P5	J2	1
S3	P3	J1	2
S3	P4	J2	5
S4	P6	J3	3
S4	P6	J7	3
S5	P2	J2	2
S5	P5	J5	5
S5	P5	J7	1
S5	P6	J2	2
S5	P2	J4	1

Figure 1.2:3

Operational Data

domains. Here it is proposed that a single language be used which is capable of satisfying all user needs over the whole spectrum, from casual users through to sophisticated administrators. This requires a language through which new data structures can be defined, security and integrity constraints written, mappings declared, DBMS control instructions given, as well as its being capable of the usual data manipulation feats. It need not be a completely new language. Indeed, if tasks can be adequately accomplished by using already existing language constructs then it would be wasteful if new constructs were defined for the same purpose. Therefore, an already existing language has been chosen as a base, and this language has subsequently been modified and extended in an orthogonal fashion. Thus the goal of Chapters 2 and 3 is to identify the modifications and extensions as well as show how the language now accomplishes the desired goals in a realistic way. A syntax is also given, as a short yet complete way of identifying all possible constructs and demonstrating their full power and flexibility. It is certainly not intended to be used in a particular implementation as it stands.

In the language, particular emphasis has been placed on the writing of constraints and DBMS control instructions. Consider these two aspects in more detail.

Security and Integrity Constraints

Maintaining security and integrity is a highly complex problem as there are so many varied events that can cause security and integrity violations. Martin (50) gives a list indicating some of the more common and well understood events. This list is by no means complete. In fact there is a real danger that a designer may concentrate on one aspect alone so causing integrity and security to suffer in other areas. Ideally the data base must be protected from every possible

event that can cause illegal alteration, destruction, disclosure or addition. Clearly this is an impossibility and therefore it is unreasonable to expect a DBMS to be designed which is capable of offering complete protection in the current environment, let alone the future environment.

It would be most desirable, from a designer's point of view, if the details of the various possible security infringements can be ignored. That is, a designer would not have the need to design software for DBMS protection against possible events that can cause security or integrity violations. Instead, it is better to design a general mechanism that is capable of being instructed on how best to handle each specific event. Thus the system becomes flexible, capable of introducing new security checks on unforeseen future requirements and capable of dropping unnecessary security checks. No longer need the designer attempt to predict future security needs in future environments, instead the responsibility falls on administrators as the needs arise. The design problem now becomes one of introducing such a general mechanism. There are a number of possible alternatives as to how this can be done, but in each case it must be possible to write constraints for any relation, domain or attribute value in the data base, and, for any other resource of the DBMS. Here it will be possible to write constraints in a declarative fashion for operational data, system data and the language constructs. These constraints are very flexible in nature, offering a wide and almost unlimited choice of possible security checks that can be made, and almost unlimited choice of actions that can be taken on any detected violation. For example, it is possible to apply security and integrity constraints to administrators so limiting their access to data and use of language constructs. It is possible to apply security constraints to data dictionaries, audit data, even control instructions. It is possible

to allow user access to data items only during certain time intervals of the day or only after some other user has granted permission. In fact the other user may not be permitted to access that data item. Finally it was found that the highest level of security attainable for a resource exists when an action using the resource requires authority from a group of administrators, or enterprise officials. That is, no one person has ultimate authority, instead, the group controls each other.

DBMS Control Instructions

Clearly it is impossible to predict all the functions that a DBMS might be called upon to do. So for the same reasons given above it is proposed here that the DBMS be limited to a set of fundamental operations, such as searching, retrieval and storage. Further, new DBMS functions are included as required by defining the new functions in terms of the basic set. Again, this is done in a declarative fashion by some advanced user. Therefore, the DBMS has the flexibility to meet continually changing enterprise and user demands. For example, through the language an administrator can instruct the DBMS to maintain a record of all additions, deletions and/or modifications to a particular relation, domain or attribute value. The DBMS may also be instructed to dump data concerning any security breach on tape. See Chapters 2, 3 and Appendix II.

Miscellaneous Features

Still other important features of the language that need to be remembered when considering the language are as given below:

1. The language will depend heavily on a host language for its syntax details and other additional processing requirements.
2. For convenience it will be called the Calculus. There should not be any confusion with the calculus mentioned by Codd (22).

3. The host language and Calculus constructs are thoroughly intermixed. For example, host language statements may be found in a Calculus ON statement and Calculus statements may be found within host language constructs.
4. It is assumed that the Calculus is used through a terminal. This introduces the added complexity of real time processing and the simplifying aspect of imagining each Calculus statement to be executed immediately. It is not intended that the back-up problem associated with batch processing in the data base environment is to be ignored, nor is it intended that batch processing should be precluded. Many different back-up mechanisms are available within this design. See section 5.4.
5. The language describes and manipulates logical structures. So there is no need to mention physical parameters or provide constructs for a physical description of actual stored relations.

1.3 Operation of the DBMS

There are two aspects of particular concern which affect the overall DBMS operation. These stem from the implementation problems associated with the language. Firstly there is the problem of how the language should be parsed and executed. Secondly, the problem of how best to utilise the high degree of physical data independence offered by the relational DBMS. Both of these aspects have a considerable influence on performance of the DBMS. In fact, the more powerful the language facilities and the greater the data independence then greater also is the response times and running costs.

1.3.1 Implementing the Language

The decision as to whether the language should be compiled or interpreted is perhaps one of the first considerations. Both techniques have their advantages and disadvantages. A compiler greatly improves

execution performance, whereas an interpreter has the capability of adapting to any change in the storage structure. Unfortunately the major problem associated with compilers is that they "bind" the compiled program to the existing storage structure. Interpreters, on the other hand, are generally too slow, particularly if optimisation of user statements is required. Many other advantages and disadvantages remain, but for the sake of brevity consider briefly the problem of binding, and how the proposals here handle this problem as well as retaining some desirable compiler features.

1.3.1.1 Binding

Binding occurs whenever one representation of the data is associated with another. It occurs when a subschema is bound into a schema or when a user's view of a schema is bound to the physical storage. There can be both logical and physical binding just as there is both logical and physical data independence. Once binding occurs a user program no longer has data independence. Therefore any change to the data structures before this program is executed will produce errors. So a compiled program will have a very short life expectancy. For this reason binding should be done only when the data is to be accessed rather than when it is first compiled. If this is done, then, "dynamic binding" is achieved; that is, there is dynamic data independence. This is proposed here as the structures will change frequently through user modifications and automatic tuning. It is therefore intended that the Calculus is first compiled into a high level, data independent, primitive language (instruction set). During this compilation all the benefits of a compiler can be reaped. The primitive language can then be interpreted. Thus the advantages of an interpreter are achieved as well as any other advantage that might be offered by a simple procedure-like instruction set. For example, the primitive set may be executed by a data base processor.

1.3.2 The Data Base Processor

Unfortunately, the problem of performance remains even if such a primitive language exists. Therefore it is suggested that the primitive language should be as simple and machine-like as possible. Then it may be possible to execute it with a specialised data base processor in parallel with other DBMS functions. The data base processor, DBP, is a separate processor which handles all the storage and retrieval problems associated with mass storage of a data base. This is becoming more and more of a profitable objective, especially with the great advances made in cheaper and better hardware components, in particular, the development of micro-processors. There is a growing tendency to move away from the one single central processor performing all tasks and toward multi-processor systems - these systems being specially designed to operate in parellel. Thus there is greater emphasis on parallel processing as a means for increasing system performance. DBMS s have grown considerably in complexity and consist of many subtasks handling user needs as well as controlling mass storage devices. Many of these tasks are independent of each other. So it is only a matter of time before parallel processing techniques will be extensively used in DBMSs. For example, the manipulation and searching of files can be considered separately and handled by a DBP. These processors will be dedicated processors. That is, they have a specialised purpose just as array processors are specifically designed for fast array processing. It is therefore feasible to use a specialised instruction set and machine architecture to efficiently handle its assigned DBMS functions. Such an instruction set could form a base for a primitive assembler-like language yet leave it still capable of manipulating data at a fairly high level. Then again, this primitive language may be actually micro-programmed in the DBP.

1.3.2.1 Advantages of a DBP

Numerous advantages can be gained by using a DBP. In particular there is higher performance resulting from executing these access and data base control processes in parallel with other DBMS processes. Anderson (1) identifies four technological factors which support the development of specialised DBPs.

(1) Distributed processing

(2) Data base languages

(3) Micro-processors

(4) Mass memory technology.

A full utilisation of these will be needed if the desired objectives of modern DBMS are to be achieved.

1) The advances in network technology are forcing shared data bases to become distributed. A DBP can be used to help achieve a practical solution to data base distribution. It will be able to handle much of the added processing needed. But more important, by allowing a data independent communication, it eliminates any requirements for remote users to understand different storage mechanisms. Thus a host of different storage devices with their differing technologies can be added to a distributed data base. Notice, that each DBP is intended to have its own storage schema - (internal schema).

A DBP may be linked to a network system in one of two major ways.

(a) Directly linked to the network via its own communication lines. Thus it will have to handle communication protocols as well.

or (b) Through a host processor. Then the host processor will handle all communication problems and leave the DBP to handle its primitive language alone.

The last concept will be the one chosen here.

2) Modern data base languages, in particular the above mentioned Calculus, give very general and powerful expression facilities. However, the different commands can be reduced to a single set of commonly used processes. For example, retrieval on certain keys, storage, and ordering. The Calculus lends itself to a set of primitive algebra-like commands. These can be part of a DBP's micro-programmed instruction set.

3) Micro-processors can give cheap and powerful processing power. It is logical to expect the high speed micro-processor technology to be used in DBMSs as DBPs. Already different architectures are proposed for mass storage of data where hundreds of micro-processors are used. Each is intended to handle the data in a portion of memory. See Ozkarahan (57) for a description of RAP (an associative processor).

4) Mass storage consists of compromises between many different technologies. There is the slow tape storage through to the high speed disk. A few more years may see greater use of new developments such as the bubble and electron beam storage devices. How to structure data, how to find and retrieve it, how best to utilise the storage medium, and the time space considerations are all problems which must be considered when storing data on the different devices. It is an entirely separate problem in itself but one which can greatly affect the performance of a DBMS. So there is a need for constant tuning. Typically this requires selection of access methods, restructuring and re-allocation of data. One of the main purposes of the DBP is to handle these problems, so removing considerable load from the central processor.

1.4 Conclusion

The emphasis in Chapter 1 is on the differences between the proposed

system and the typical DBMS. There are other assumptions and concepts though of relative minor importance. Firstly it is assumed that all data base relations are in third normal form - this has the well accepted advantage of greatly simplifying DBMS operations, particularly those involving deletions and additions to the data base. It is also assumed that the system is an on-line system. This has the effect of high lighting the various requirements of a real time DBMS.

Finally the operation of the proposed DBMS will basically follow the sequence given below.

(1) A user requests data through some Calculus construct.

(2) This construct is compiled into a primitive code. All mappings and constraints relating to this user are also included in the code for run-time evaluation.

(3) The code is executed by some dedicated processor or program module.

(4) The resulting relation is returned to the user along with any other system relation. Note, since all information is stored in relations, the easiest way to inform the user of any failures is to also return the system relation containing this data.