Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group

Edited by

Michael L. Brodie and Joachim W. Schmidt

September 1981

Doc. No. SPARC-81-690

The Relational Database Task Group had the following members at the time that this report and the associated SD-3 was forwarded to the DBS-SG:

Michael L. Brodie.....University of Maryland
(co-chairman) and Computer Corporation of America
Joachim W. Schmidt.....University of Hamburg
(co-chairman) and Computer Corporation of America
Alex J. Arthur.....International Business Machines Corporation
Margret Ball.....Intel Systems Corporation and Burroughs Corporation
Charlie Bontempo (Alternate for Arthur).IBM Systems Research Institute
Harrison R. Burris.....TRW Inc.
Beth Driver.....PRC Information Sciences Co. and Technology Service Corporation
Roy Hammond.....Statistics Canada
Peter Hitchcock.....University of Victoria and British Columbia Systems Corporation
Kate Kinsley.....University of Central Florida
Nancy McDonald.....University of South Florida
Alain Pirotte.....Philips Research Laboratory, Brussels, and Computer Corporation of America
Daniel R. Ries.....Lawrence Livermore Laboratory and Computer Corporation of America
Edgar H. Sibley.....Alpha Omega Group, Inc.

Acknowledgments

Several RTG members gratefully acknowledge the support of their participation by research agencies:

Michael L. Brodie.....Army Research Office and National Bureau of Standards
Peter Hitchcock.National Scientific and Engineering Research Council of Canada
Joachim W. Schmidt.....Federal Ministry of Research and Technology of W. Germany
Edgar H. Sibley.....Environmental Protection Agency and National Bureau of Standards

The RTG thanks the Computer Corporation of America for its generous support of production of this report. Special thanks are due to David Darcy who carefully formatted the entire document.

Others who contributed to the work of the Relational Database Task Group are:

Kjell Bratbergsengen.....Sperry Univac and University of Trondheim
Edgar F. Codd.....International Business Machines Corporation
Chris Date.....International Business Machines Corporation
Wolfgang Dotzek.....University of Hamburg
Jim Driscoll.....University of Central Florida
Oris D. Friesen.....Honeywell Information Systems
K. Grammel.....University of Central Florida
Andrew Hutt.....International Computers Limited
H. Randall Johnson.....Boeing Computer Services
J.Z. Kornatowski.....University of Toronto
William Kent.....International Business Machines Corporation
Winfried Lamersdorf.....University of Hamburg
Michael Lacroix.....Philips Research Laboratory, Brussels
Jim Larson.....Sperry Univac and Olivetti, Italy
Manuel Mall.....University of Hamburg
John McNally.....University of South Florida
T. William Olle.....T. Wm. Olle, Assoc.
Ken Paris.....Peat, Marwick, Mitchell & Company
Dzenan Ridjanovic.....University of Maryland
C.M. Robertson.....University of Toronto
Michael Stonebraker.....University of California, Berkeley
Tom Wilson.....Sperry Univac

Preface

In May 1979, the Relational Task Group (RTG) was chartered to investigate the justifiability of proposing to ANSI/X3 that a project be initiated to develop a relational standard. Having concluded that such a proposal is justifiable, the RTG drafted a document entitled "Proposal for Standardizing Interfaces to Relational Database Management Systems" to be submitted to ANSI/X3. The RTG proposes that the functionality of the interfaces to Relational Database Management Systems (RDBMS) be standardized.

The current report is intended to support the proposal for a relational standard. It documents the results of the RTG's investigations which consisted of three tasks:

1. Identify the fundamental concepts of the Relational Data Model (RDM).

2. Characterize the features of existing and potential RDBMSs to determine the interface functions.

3. Investigate the role of the RDM and RDBMS in a DBMS architectural framework such as the ANSI/X3/SPARC prototypical architecture, and in a coherent family of DBMS standards.

The first two tasks were tractable. The results of the first task are found in Chapter 2. The results of the second task are found in Chapter 3*. These results form the technical basis for the proposal.

The third task addressed open research problems. Chapter 4 documents the problems by identifying issues and known alternatives. Chapter 5 summarizes the results and proposes guidelines for developing a relational standard.

---

*The results are based on detailed feature analyses of twelve individual RDBMSs. These twelve RTG working documents will be published in Brodie, M.L., and Schmidt, J.W. (Eds.), "Relational Database Management Systems: Analysis and Comparisons," Springer-Verlag, Heidelberg/New York, in press.

CONTENTS

	Page
1. THE RELATIONAL DATABASE TASK GROUP	1
1.1 Motivations for a Relational Standard	1
1.2 RTG Charter	1
1.3 RTG Products	1
2. THE RELATIONAL DATA MODEL (RDM)	3
2.1 Inherent Definitional Issues	3
2.1.1 RDM Core Concepts and Extensions	3
2.1.2 Definition Issues	3
2.2 Examples of RDM definitions	5
2.3 Concepts, Terminology, and Examples	6
2.3.1 Basic Concepts of the RDM	7
2.3.2 Relations and Tuples	7
2.3.3 Queries and Altering Operations	8
3. ANALYSIS OF RELATIONAL SYSTEMS	15
3.1 Development of the Feature Catalogue	15
3.2 Database Constituents	16
3.2.1 Definitions	16
3.2.2 Results	16
3.2.3 Observations on Database Constituents	16
3.3 Functional Capabilities	16
3.4 Schema Definition	16
3.4.1 Descriptions	16
3.4.2 Results	18
3.4.3 Observations	19
3.5 Additional Definition, Generation, and Administration Facilities	20
3.5.1 Background	20
3.5.2 Results	23
3.5.3 Observations	23
3.6 Functional Classes	23
3.6.1 Definitions	23
3.6.2 Results	25
3.6.3 Observations	25
3.7 Interface Flavors	26
3.7.1 Definitions	26
3.7.2 Results	26
3.7.3 Observations	26
3.8 System Architectures	26
3.8.1 Results	26
3.8.2 Observations	26
3.9 Operational Aspects	27
3.9.1 Definitions	27
3.9.2 Results	28
3.9.3 Observations	28
4. ARCHITECTURAL ISSUES AND THE RDM	31
4.1 The Importance of Architecture	31
4.2 Alternative DBMS Architecture Frameworks	31
4.3 Model Related Architectural Issues	32
4.3.1 The Schema Architecture	32
4.3.2 Multiple Models in User Interfaces	32
4.3.3 Separate or Unified Components of an Architecture	33
4.3.4 Granularity of Building Blocks in the Architecture	33
4.4 Internal Architecture of Relational Systems	33
4.4.1 The INGRES Architecture	33
4.4.2 The System R Architecture	34
4.4.3 Observations	35
4.5 Candidate RDBMS Architecture	35
4.5.1 A Proposed Architecture	35
4.5.2 The CCA Strawman Architecture	36
4.5.3 The ANSI/SPARC Architecture	37
5. DEVELOPING A RELATIONAL STANDARD	39
5.1 Guidelines and Issues	39
5.2 Recommendations for a Relational Standard	40
6. REFERENCES	41
6.1 Relational Bibliography	41
6.2 Reference Papers	44
APPENDICES	
A. NATURE OF A RELATIONAL STANDARD INTERFACE	47
A.1 Interface Functions for Database Definition	47
A.2 Interface Functions for Database Operations	47
A.3 Interface Functions for Database Schema Access	48
B. A FORMAL DEFINITION OF THE RELATIONAL MODEL	49
B.1 Semantic Domains for Relational Databases	49
B.1.1 Abstract Syntax	49
B.1.2 Static Consistency Constraints	49
B.2 Syntactic Domains for Relational Databases	49
B.2.1 Abstract Syntax	50
B.2.2 Dynamic Consistency Constraints	50
B.3 Elaboration Functions for Relational Databases	51
C. RELATIONAL DBMS SURVEY	53
C.1 Performing the Survey	53
C.2 RDBMS Survey Results	54

1. THE RELATIONAL DATABASE TASK GROUP

For more than a decade the relational database model (RDM) has received considerable attention in the research community and is now receiving attention by database practitioners. The RDM's simplicity and formality continue to inspire a large amount of research and development. Many practical and theoretical advantages have been demonstrated or claimed, e.g., high level relational languages have strongly influenced the development of database query languages and theoretical results have provided database design aids.

Over the past ten years the relational approach to databases has become a dominant factor in database education. It is used for database design, analysis, and query formulation. Currently there are over 70 DBMSs that are said to support relational concepts and languages. Extensive interest in these systems indicates that they soon will come into widespread use.

Although they are similar, the 70 RDBMSs do not provide the same "relational" functionality. For these and other reasons, the ANSI/X3/SPARC Database Systems Study Group (DBS-SG) established the RTG to investigate the justifiability of proposing to ANSI/X3 that a relational standard be developed. This chapter outlines the motivations, charter, and products of the RTG.

1.1 Motivations for a Relational Standard

Three motives led to the investigation of a relational standard. One motive concerned timing. Since its introduction over a decade ago [CODD70], the core RDM concepts have provided a stable basis for database research and development. For some years, the relational approach has been widely taught and understood in both commercial and academic settings. Currently, significant interest is being expressed in the DBMS marketplace. Over 70 DBMSs, languages, and hardware devices said to be based on the RDM have been developed, many of which are or soon will be commercially available. Keen user interest has resulted in many projections of their widespread use in the near future. The effectiveness of relational technology has been demonstrated for a wide range of user and application needs.

A second motive is the desire for alternative approaches to data description, access, manipulation, and integrity control. This was recognized at the outset of the CODASYL approach to databases. More recently, the desire for coexisting, heterogeneous databases, motivated (in part) the ANSI/X3/SPARC prototypical architecture [TSIC78]. To date, only the CODASYL approach has been considered for standards development. The desire for alternatives can be seen in the variety of DBMS types (e.g., CODASYL, relational, hierarchic, primitive, hybrid) in today's DBMS marketplace. In September of 1980, the National Bureau of Standards made a firm commitment to support three primary alternatives (i.e., CODASYL, relational, and hierarchical). The appropriateness of an alternative depends not only on the nature of the application requirements and user needs but also on matters of taste and style. Since the relationship between the alternatives (e.g., one a subset of another) is an open research topic, they must currently be viewed as distinct.

A third motive involved the claims made, and in some cases demonstrated, for the inherent benefits of the relational approach. Four of these claims are: The RDM and its related languages are simple and uniform and are based on a small number of well understood concepts. The RDM permits a high degree of data independence -- freedom of relational schemata and languages from representation details. The RDM is based on a sound theoretical foundation which permits the analysis of languages and applications, and provides a basis for further research. The RDM provides high level, nonprocedural qualification primitives (e.g., the relational calculus) for the design of high level query and manipulation languages.

1.2 RTG Charter

At its May 29, 1979 meeting in Boston, the ANSI/X3/SPARC Database System Study Group (DBS-SG) unanimously approved the establishment of and charter for the Relational Database Task Group (RTG). The charter reads as follows:

"The Relational Database Task Group (RTG) will establish whether there is sufficient justification for proposing to ANSI/X3 that a standards project be initiated for the relational approach to databases. To do so the RTG will identify or establish:

1. Relational Data Model (RDM) concepts and terminology including an RDM definition;
2. aspects of the RDM and of Relational DBMSs (RDBMSs) that might be appropriate for standards development;
3. issues concerning the role of the RDM and RDBMS features in a DBMS architecture such as the ANSI/X3/SPARC prototypical architecture;
4. the relationship of existing and potential software standards to potential relational standards;
5. other guidelines for standards development in this area, as appropriate."

1.3 RTG Products

The RTG had its first meeting, under the chairmanship of Michael L. Brodie, on July 22-23, 1979 together with DBS-SG in Minneapolis. At that time, it was proposed that the RTG respond to its charter by producing two documents for the DBS-SG to be considered for presentation to SPARC.

The primary product of RTG will be a recommendation for or against the initiation of an ANSI/X3 standards project for the relational approach to databases. The format of the recommendation will follow the one defined in the ANSI/X3/SD-3 document. The resulting document will be referred to as "the SD-3". Typically, if X3 approves an SD-3, it establishes a technical committee responsible for the proposed standards project.

The RTG's main objective for the SD-3 was to provide sufficient and precise information to aid

the DBS-SG, X3, and SPARC in deciding whether to initiate a relational standards project. This involved a clear description of the nature of the standard (i.e., definitions of the RDM and features of an RDBMS) and a thorough evaluation of the state of the art. The SD-3 should also provide adequate guidance for a subsequent technical committee by outlining an unambiguous scope and effective program of work. The SD-3 should not overly constrain the technical committee's work, e.g., issues and alternatives should be raised but not resolved. The SD-3 has been completed and is entitled "Proposal for Standardizing Interfaces to Relational Database Management Systems".

The current document, "The Relational Database Task Group Final Report", has three purposes: it provides technical support for the SD-3; it records the results of the RTG's work; and it provides a base document for subsequent technical committee(s).

During its two year existence, the RTG consisted, on the average, of 18 people. Each of its 13 meetings was attended by at least 12 RTG members.

The RTG established liaison with the following groups and countries:

ANSI/X3 (U.S.A.): H2, H4, J1, J3.1, J4, DBS-SG
CODASYL (U.S.A.): DDLC, FDBLC
National Bureau of Standards (U.S.A.)
International Standards Organization: TC97/SC5/WG3, TC97/SC5/GTS
Canadian Standards Association (Canada)
British Standards Institute (Great Britain)
British Computer Society (Great Britain): DDSWP, DBORG, DDAWG
Deutsches Institut fuer Normung, DIN (Germany)
Gesellschaft fuer Mathematik und Datenverarbeitung: GMD (Germany)
Association Francaise de Normalisation, AFNOR (France)
Bureau d'Orientation de la Normalisation en Informatique (France)
Institute National de Recherche en Informatique et en Automatique, INRIA (France)
Consigla Nazionale delle Ricerche (Italy)
Finland
Norway
Sweden

2. THE RELATIONAL DATA MODEL (RDM)

From their initial formulation to their precise definition and their final realization in a Relational Database Management System, the concepts that constitute the RDM are defined in different ways. There are three distinct steps in this progression:

- The Relational Data Model is the collection of basic concepts that underlies the relational approach to data definition, manipulation, query, and integrity.
- A Relational Data Model Definition provides a notation (syntax) to identify the concepts of the RDM and an unambiguous specification of the meaning (semantics) associated with this notation.
- Relational Database Management Systems (RDBMS) are software systems that provide interfaces with the functionality specified by an RDM definition. Languages (syntax and semantics) at these interfaces are tailored to meet user's needs.

The purpose and relative importance of the steps are clear. RDM concepts are fundamental. An RDM definition is one of many possible expressions of the RDM concepts. A RDM definition is needed to guide the specification and implementation of the relational features of an RDBMS.

This chapter outlines issues in determining the constituent concepts of the RDM and in formulating definitions of RDM concepts. References to the literature are used to illustrate a variety of RDM definitions. The basic concepts are discussed and some relational terminology is introduced.

2.1 Inherent Definitional Issues

This section addresses two central definition issues: first, which concepts should be included in a basic RDM definition and which might be considered as extensions; second, what is the relationship between the purpose of a definition and its form. Issues such as precision, level of detail, and formality of RDM definitions are also raised.

2.1.1 RDM Core Concepts and Extensions

Of the many concepts associated with the RDM, which of them are fundamental, and which could be defined in terms of (or as extensions to) fundamental concepts? For the purpose of a DBMS standard, a major criterion for distinguishing core RDM concepts from extensions concerns the state of the art. The core should contain those concepts needed to account for current RDBMSs as opposed to other DBMSs. Further, the core concepts should be demonstrably practical. Hence, some desirable and useful concepts that are prohibitively hard to implement should remain, for the moment, extensions.

There are other criteria for distinguishing core concepts. Fundamental concepts need not be minimal. They should provide a complete basis for data definition, altering, querying, and integrity of relational databases. They should also provide a basis for distinguishing the RDM from other data models as well as a consistent basis for extending the RDM (e.g., so that a standard based on the fundamental concepts could evolve naturally to take advantage of new research results).

Concepts could be excluded from a basic RDM definition if they:
- conflict with fundamental concepts, e.g., duplicate tuples, row ordering;
- concern representation issues, e.g., representation of relations, detailed syntax of DDLs and DMLs;
- are nonrelational in nature, e.g., library functions, security;
- concern the use of the RDM, e.g., dependency and normalization theory for database design and data integrity;
- concern implementation or language issues, e.g., create and drop relation; store, name, or maintain derived relation values (hence dynamic and static derived relations are excluded);
- concern open research issues, e.g., null values, surrogates.

The RTG considers the following concepts to be fundamental to the RDM. These concepts form the basis of the RDM description given in Section 2.3.

Basic Background:
set theory, propositional logic, first order predicate calculus;

Structure Concepts:
relation, relation schema, relation value, tuple, attribute, domain, relational database, relational database schema;

Operators:
(a) Retrieval and Derivation:
(the relational algebra operations are given; however, the relational calculus could also be used)
Cartesian product, union, intersection, set difference, selection, projection, join, division;
(b) Altering:
insert, delete, replace;

Integrity Concept:
Key;

Examples of concepts excluded by the RTG from the basic RDM definition are:

Structure Concepts:
derived relation (static and dynamic), derived relation scheme, null values;

Operators:
(a) Retrieval and Derivation:
library functions, derivation over null values;

(b) Altering:
altering concerned with null values.

Integrity Concepts: candidate key, referential integrity (foreign key);

2.1.2 Definition Issues

The RDM concepts can be formulated in many substantially equivalent definitions. Formulations may vary in: terminology, representation, notation (formalism), level of detail, precision, and degree of formality. Ideally, each definition takes advantage of the possible variations to meet the particular needs of the intended audience, to emphasize certain RDM aspects, or to express a

particular perception of the RDM.

Possible audiences for an RDM definition differ substantially in interests, needs, and backgrounds. Typical audiences include: DBMS designers and implementors, application development language designers and implementors, applications designers and implementors, application users, students, and researchers. A different RDM definition may be appropriate for each audience and its intended purpose for the definition. For DBMS standards, the most important use of one or more RDM definitions would be as a basis for specifying a relational standard. The definitions must be sufficiently precise, formal, and detailed to permit analysis for consistency and completeness of the standard and for conformity of a particular DBMS with the standard.

Terminology

There is no single generally accepted terminology for DBMSs or for discussing the RDM. While the existence of synonyms is often cited as a terminology problem, the RTG has found pseudo-synonyms (e.g., words that may refer to similar objects but whose definitions differ, such as "record", "tuple", and "row"), and words that have multiple meanings (e.g., "key") to pose even greater problems. The use of a definition and term depends on one's perspective and the topic at hand; for example, "attribute", "data item", and "field" may be used in discussions of different issues.

The relational terminology used throughout this report is given above in the list of fundamental RDM concepts (Section 2.1.1). The terms "table", "row", "column", etc. (see Section 2.2), are not used because they are less precise and are intended for a wider audience.

Representation

The RDM can be represented in many ways. Hence, the RDM should not be biased towards a particular representation. Relations can be perceived or represented in a variety of ways, e.g., sets of structured tuples, n-ary predicates on elementary values, arrays of tuples, or as n-ary functions (in general multi-valued) from elementary values to elementary values. Relational operators can be expressed in several forms, e.g., in relational algebra, tuple-oriented relational calculus, domain-oriented relational calculus, or in n-ary functions.

Only certain combinations of structure representation and operator form are appropriate, e.g., sets with algebra, tuple or domain calculus with predicates, functions for both, and sets and calculus. Some languages mix both sets and arrays, and calculus and predicates.

Notations

Many notations can be used to define the RDM. English can be used for informal definitions and many formalisms can be used for a formal definition, e.g., set theory [CODD70], extended set theory [CHILDS77], Positional Set Notation [HARD80], Meta-IV of the Vienna Development Method [BJOR78], the Vienna Definition Language [LUCAS71], Floyd-Hoare axiomatics [HOARE72], functional specifications [MILLS79], algebraic specifications [GUTT78], initial algebras [GOGU77], and graph theory [FURT78] (see also Section 2.3).

Level of Detail

In comparison with RDBMS implementors, users of relational query languages and users of applications implemented on an RDBMS need to know only relatively few RDM details. For example, the RDM description in Section 2.3 may be adequate for users but inadequate for RDBMS implementors.

Precision

A definition at any level of detail would be precise if there were no question as to the understanding of the details given and the scope of what is described. In addition, a precise definition would remain consistent as details were added. For example, the footnotes in Section 2.3 concerning first order predicate logic as a basis for selection criteria in the relational calculus are precise but not detailed.

Formality

Informal definitions are appropriate for education, end user guidance, and some aspects of database design. Formal definitions facilitate analysis for consistency and completeness of the definition itself, specification of relational languages and systems, some aspects of database design, conformity of a DBMS implementation with a standard.

This report contains several examples of descriptions of the RDM. Section 2.2 lists several others. Table 2.1 compares the descriptions in this report based on the above criteria.

TABLE 2.1 COMPARISON OF SEVERAL RDM DESCRIPTIONS

RDM description	Section 2.2 explicit	Feature Catalogue implicit	Appendix A implicit	Appendix B explicit
purpose	essence of RDM, how RDM concepts go together, education	(implicit) basis for features of an RDBMS	functions constituting RDBMS interface	formal definition for implementation
audience	introduction for those knowledgeable in databases, e.g., students, DBSSG	RDBMS comparers, designers, and implementors	those concerned with DBMS standards, e.g., DBSSG, ANSI/X3	RDBMS designers and implementors, theoreticians
terminology	domain, relation,...	domain, relation,...	domain, relation,...	domain, relation,...
representation	sets; algebra and tuple calculus	sets/predicates/arrays; algebra/tuple calculus	sets; algebra and tuple calculus	sets; tuple calculus and algebra
notation	English	English	functions and English	VDM
Level of detail	essence and high level details	more detail than 2.2	little detail	complete detail
Precision	precise	imprecise	imprecise	precise
Formality	informal	informal	informal	formal

2.2 Examples of RDM definitions

Although there is considerable relational literature, relatively few papers give a complete definition of relational concepts. In the absence of a basic reference document and universally agreed upon concepts, different perceptions of the RDM have been developed, together with specific terminology, assumptions, and concerns, and often a particular data manipulation language. Some of these issues are discussed in Section 2.1.2.

The task of explicitly analyzing the variety of definitions, concepts, and languages as different perceptions of one relational model remains to be done. This section is a contribution to that effort. A number of papers are listed that have contributed significantly to the establishment of some relational concepts, or that provide typical perception of the RDM.

Beeri C., Bernstein P., Goodman N., "A Sophisticate's Introduction to Database Normalization Theory". VLDB Berlin, 1978.

This paper is a tutorial on a number of topics concerning relational theory: dependency constraints (such as functional and multivalued dependencies), normal forms, and relational schema design. It summarizes results and mentions a number of open issues (e.g., the importance of the universal relation assumption).

Bjørner D., "Formalization of Database Models". In: Abstract Software Specifications, Springer Lecture Notes in Computer Science 86, 1980.

Using the Vienna Definition Method (VDM) of denotational semantics definition, this paper gives formal descriptions of relations, the relational algebra, and the tuple relational calculus.

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

The paper that started everything! Unlike scores of others, it ages very well. It presents a set-theoretic definition of relations, mentions the table view (called array representation), and describes the algebraic operations.

Codd E.F., "A Database Sublanguage Founded on the Relational Calculus". Proc. ACM SIGFIDET workshop, San Diego (November 1971).

This is the definition of ALPHA, derived directly from the tuple relational calculus, which has directly or indirectly influenced the design of many relational query languages.

Codd E.F., "Relational Completeness of Database Sublanguages". In: Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall (1972).

A standard reference for a definition of the relational algebra, the tuple relational calculus, and relational completeness. The paper also discusses the translation of calculus formulas into algebraic expressions.

Codd E.F., "Extending the Database Relational Model to Capture More Meaning". TODS, Vol. 4, No. 4 1979.

This paper contains a definition of basic concepts of the RDM, using set theory and rules for a tabular view of relations. The definition includes a version of the algebraic operations. The paper suggests two rules which control the representation of entities by relations and constrain the alteration operations of the RDM. The terms "fully relational" and "semirelational" are given a definition which was referenced in Section 1.5 of the Feature Catalogue [BROD80c]. The paper also proposes a number of extensions to the relational model (representations of entities, associations, treatment of null values, etc.).

Date, C.J., "An Introduction to Database Systems", Third Edition, Addison-Wesley, 1981.

The part of this book that deals with the RDM has been revised and expanded in the third edition. It includes among other things a set-oriented definition of relations, illustrated by table representations, a definition of relational algebra, the tuple, and domain relational calculi. Codd's definitions of "fully relational" and "semirelational" are given, and the observation is made that no currently existing systems are fully relational, or even semirelational.

Delobel C., "An Overview of the Relational Data Theory". IFIP Congress 1980.

This paper summarizes a number of results from the theory of relational databases and their application to database design. Among other topics, the following are described: decomposition of relations; the universal relation assumption; functional, multivalued, hierarchical and join dependencies, and their axiomatization; tableaux and their use to formulate problems of dependencies. The paper also contains a definition of relations which mentions both the set and the predicate aspects of a relation.

Gallaire H., Minker J. (Eds.), Logic and Data Bases, Plenum Press 1978.

This book contains a number of papers which explicitly deal with the relational model. The typical logician's view of a relation is that it represents a predicate on domains, and the typical calculus language is some version of domain relational calculus. The notion of a tuple (or record) is a natural one in the database field, but much less so in logic, artificial intelligence, and natural language studies.

Lacroix M., Pirotte A., "Domain Oriented Relational Languages". VLDB, Tokyo 1977.

This paper defines the domain relational calculus, where variables range over domain values rather than over relation tuples as in the tuple relational calculus. The paper also discusses the design of domain-oriented languages.

Sandberg G., "A Primer on Relational Database Concepts". IBM Systems Journal, Vol. 20, No. 1, 1981.

This paper presents a tabular view of the RDM. One advantage of the relational model is that its data structures can be informally described as tables. The following description is an informal definition of the table view of relations (other characterizations of similar descriptions are the "record view" or the "flat file view" of relations).

"The data are structured in the form of tables consisting of columns and rows, with the rows corresponding to records or segments, and the columns representing fields within the records. The following rules must be followed:

- Each table contains only one record type.
- Each record (row) has a fixed number of fields, all of which are explicitly named.
- Fields are distinct (atomic) so that repeating groups are not allowed.
- Each record is unique -- duplicates are not allowed.
- Records may come in any order; there is no predetermined sequence.
- Fields take their value from a domain of possible field values.
- The same domain may be used for many different field types, thus becoming the source of field values in different columns in the same or different tables."

This definition can be derived from the definition given in Section 2.3 if the following correspondence of terms is introduced:

- relation value = table
- relation scheme = record type (or table type)
- attribute = column name
- tuple = row = record
- N-ary relation = table with N columns

Ullman J.D., Principles of Database Systems, Computer Science Press 1980.

An important part of this book is devoted to relational concepts, with a set-theoretic definition of relations and a presentation of the basis of several query languages based on a thorough definition of tuple and domain relational calculus, relational algebra, and their correspondences.

2.3 Concepts, Terminology, and Examples

Disclaimer

Due to the pressure of time in completing the work of the RTG, it was not possible for this section of the Final Report to be reviewed by the full task group. This section was developed in substance by Joachim Schmidt, with contributions from Michael Brodie and Alain Pirotte, and with additional comments from other members of the group. Although this material does not necessarily represent the consensus of the group, it is included as an example of a relational data model description.

Essentially, a data model provides three kinds of mechanisms to support data management: structures, operations, and constraints. This section discusses the basic mechanisms with emphasis on the Relational Data Model (RDM). Terminology associated with relations and relational operations (such as relational algebra and calculus) are described.

2.3.1 Basic Concepts of the RDM

This section presents one view of the fundamental RDM concepts. Other views are possible as is indicated by general issues given in Section 2.1.2, by different definitions listed in Section 2.2, and by the following specific issues:

Can a relation schema have associated with it exactly one relation value or more than one relation value?

Are attribute names part of a relation value or not?

Should application-independent value sets, such as integers or strings, be considered to be domains?

Is it more appropriate to model a relation value as a time varying set of tuple values or as a time varying relation predicate? (A relation predicate defines set membership.)

The structuring mechanisms of a data model provide means for grouping elementary data items into compound data items. A fixed number of atomic data items can be composed to form structured data objects (such as tuples or records) and a variable number of data objects can be grouped into partitions (such as relations or DBTG sets). In the RDM, the partitions are defined so that:

- members of the same partition have the same structure, i.e., each member has the same number of components and corresponding components of different members have comparable values, i.e., values of the same kind;
- members of the same partition are distinct, i.e., a data object can exist only once in one partition; and
- members of different partitions may have comparable component values (this provides the basis for relating partitions and for constructing new partitions through operational means).

The operational mechanisms of a data model serve two purposes: first, to derive data from one or more partitions and, second, to alter data in a partition. In the RDM, the result of a derivation operation can be viewed as a partition. Derived data objects depend on the derivation operation in two ways:

- their values fulfill a selection criterion provided by the derivation operation; and
- their structure consists of components denoted explicitly or implicitly by the derivation operation.

As a result of an altering operation, the altered partition contains either more data objects, fewer data objects, or data objects with different component values.

Constraint mechanisms of a data model can be used to define and maintain time invariant properties of partitions that cannot be expressed directly by the structural and operational means. Constraints control the altering operations to maintain data integrity. Typically, RDM constraints are based on the value of data objects. The presence of some data objects can be made dependent on the presence or absence of other data objects. Constraints can be associated with individual relations, with collections of relations, and with transitions between database states.

2.3.2 Relations and Tuples

In the RDM, partitions are called relations. A relation has various distinct properties.

A relation value is the set of values of the data objects that are currently members of the relation. Derivation operations refer to the relation value. It is the relation value that is changed by an altering operation, thus, a relation value is a time varying property of a relation.

A relation schema defines the time invariant properties of a relation. Relation structure is one such property; a relation has the structure of a set of identically structured data objects.

Additional time invariant properties can be defined by relation constraints. One class of relation constraints enforces uniqueness of values of distinguished components (keys) throughout the data objects of a relation.

Example 1: Definition of a Relation and a Relation Schema; Display of a Relation Value

definition of a relation and a relation schema:

```
parts = relation pnr : partnumbers ;
                 pname : partnames ;
                 color : colortype ;
                 weight : weightunits ;
                 city : citynames
                 key <pnr>
        end ;
```

display of a value of relation "parts" :

pnr	pname	color	weight	city
p1	nut	red	12	Vienna
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	Vienna
p5	cam	blue	12	Paris
p6	cog	red	19	Vienna

Remark:

The key constraint defined for the relation "parts" requires uniqueness of pnr values.

Data objects in a relation are called tuples. Each tuple in a relation consists of the same number of components called attributes.

The time invariant properties of a tuple are defined by its tuple structure that associates with each attribute an attribute name and a domain. Domains are sets of atomic values that are used to define, for each attribute, the legal attribute values. Domains are also used to define attribute compatibility and thus tuple and relation

compatibility. Compatibility is required in derivation operations where attributes of one tuple are compared with attributes of another tuple. Compatibility is also required in altering operations where attribute values are altered or tuple values are inserted or removed.

Example 2: Definitions of User Defined Domains

```
partnumbers = string of [5] char ;
partnames = string of [20] char ;
colortype = (red, blue, yellow, green, black) ;
weightunits = 0..9999 ;
citynames = string of [25] char .
```

The time varying properties of a tuple are given by its value. A tuple value is defined by its attribute values just as a relation value is given by its tuple values. While the number of tuple values that define a relation value may vary in time, the number of attribute values that define a tuple value is fixed. Tuple values can be decomposed into attribute values; for that purpose attribute values are defined as pairs of attribute names and values. The values have to be elements of the domain associated with the attribute identifier by the tuple structure definition.

A relational database schema is an extension of a relation schema. It defines the time invariant properties of a number of relations that constitute a relational database.

A relational database schema defines three properties:

1. Database structure: the relations that constitute the database; each relation specified by its schema and relation name.
2. Database domains: the domains used for relation schema definition.
3. Database constraints: additional constraints can be defined. While relation constraints refer to one relation only, database constraints may affect more than one relation.

Example 3: Definition of a Relational Database Schema

```
partnumbers = string of [5] char ;
partnames = string of [20] char ;
colortype = (red, blue, yellow, green, black) ;
weightunits = 0..9999 ;
citynames = string of [25] char ;
suppliernumbers = string of [5] char ;
suppliernames = string of [20] char ;
quantity = integer ;
creditstatus = 0..999 ;

businessdatabase =
database parts = relation
                     pnr : partnumbers ;
                     pname : partnames ;
                     color : colortype ;
                     weight : weightunits ;
                     city : citynames
                     key <pnr>
                 end ;
     shipments = relation
                     snr : suppliernumbers ;
                     pnr : partnumbers ;
                     qty : quantity
                     key <snr,pnr>
                 end ;
     suppliers = relation
                     snr : suppliernumbers ;
                     sname : suppliernames ;
                     status : creditstatus ;
                     city : citynames
                     key <snr>
                 end
end ;
```

2.3.3 Queries and Altering Operations

Database operations that derive relations are often called queries. The result of a query is a relation. This is the basis for a "closure" property of relational operations. A query definition has to provide the following information:

1. the "source relations", i.e., relations from which tuple values are selected to be transformed into the values of the result tuples;
2. the "selection criterion", i.e., a condition which source tuples have to fulfill to be selected; and
3. the "target transformation", i.e., rules that define how result tuple values are derived from selected tuple values.

Conceptually, the process of query evaluation consists of three steps:

1. provide access to tuples of the source relations;
2. evaluate the selection criterion for any tuple combination that can be formed according to 1); and
3. apply the target transformation to any tuple combination that fulfills the selection criterion, and store the result in a tuple of the result relation.

Example 4: The Three Steps of Query Evaluation

Query: Name the parts that are manufactured in Vienna and weigh less than 17 units.

Step 1: provide access to the tuples of the source relation, i.e.,

"parts" relation :

pnr	pname	color	weight	city
p1	nut	red	12	Vienna
p2	bolt	green	17	Paris
p3	screw	blue	17	Rome
p4	screw	red	14	Vienna
p5	cam	blue	12	Paris
p6	cog	red	19	Vienna

Step 2: evaluate the selection criterion, i.e., (city = "Vienna") and (weight < 17);

Criterion is :

	pnr	pname	color	weight	city
true	p1	nut	red	12	Vienna
false	p2	bolt	green	17	Paris
false	p3	screw	blue	17	Rome
true	p4	screw	red	14	Vienna
false	p5	cam	blue	12	Paris
false	p6	cog	red	19	Vienna

Step 3: apply target transformation to qualified tuples, i.e., choose pname values, and store the result in the result relation;

Result relation:

	pnr	pname	color	weight	city
-->	p1	nut	red	12	Vienna
	p2	bolt	green	17	Paris
	p3	screw	blue	17	Rome
-->	p4	screw	red	14	Vienna
	p5	cam	blue	12	Paris
	p6	cog	red	19	Vienna

derive result relation

pname
nut
screw

In the RDM, the result of a query evaluation is always a relation. This fact is the basis for what is called the "closure property" of the relational operators.

Dependent on the specific approach to relational query languages, the functionality required for querying a relational database is provided by different sets of operators.

The basic operations of the relational algebra approach can be characterized as follows:

1. Algebraic operators refer to a fixed number of relations; selection and projection are unary operators, i.e., they operate on one relation. Binary operators are union, intersection, difference, join, and division and operate on two relations.

2. Algebraic operators apply simple selection criteria: the selection operator applies a criterion that refers to constants and attribute values of individual tuples in one relation only.+ The join operator applies a selection criterion that refers to two attribute values of individual tuples in two relations. The division operator tests sets of attribute values in two relations for set inclusion.

3. Algebraic operators provide target transformations for tuple restructuring. Projection and division operators choose distinguished attribute values to form derived relations.

+This class of selection criteria can be defined formally by the expressions of the propositional logic.

Example 5: A Select Operation in the Relational Algebra Notation

Query: Derive data on suppliers that are either located in Paris or have a status different from 20.

Query definition (in PRTV Relational Algebra [TODD76]):
suppliers : (city = "Paris" or status ≠ 20)

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Query evaluation: Select tuples of the "suppliers" relation that have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the entire value of each qualified tuple, i.e., all its attribute values, into the result relation.

Result relation :

snr	sname	status	city
s2	Jones	10	Paris
s3	Blake	30	Paris
s5	Adams	30	Athens

Remarks:

The select operator refers to only one relation, e.g., "suppliers". The selection criterion refers only to tuple attributes and constants; it contains comparison operators, such as =, <, >, ≤, ≥, ≠, and logical operators, such as and, or, not. The relation "suppliers" is only read, not altered.

Example 6: A Join Operation in the Relational Algebra Notation

Query: Derive data on the suppliers and on the shipments from these suppliers.

Query definition (in PRTV Relational Algebra [TODD76]):
suppliers * shipments

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

"shipments" relation :

snr	pnr	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Query evaluation: Select tuple pairs from relations "suppliers" and "shipments" that have the same value in the snr attribute, i.e., the same supplier number. Copy the value of the qualified tuple pairs into the result relation (copy only one of the two identical snr values).

Result relation :

snr	sname	status	city	pnr	qty
s1	Smith	20	Vienna	p1	300
s1	Smith	20	Vienna	p2	200
s1	Smith	20	Vienna	p3	400
s1	Smith	20	Vienna	p4	200
s1	Smith	20	Vienna	p5	100
s1	Smith	20	Vienna	p6	100
s2	Jones	10	Paris	p1	300
s2	Jones	10	Paris	p2	400
s3	Blake	30	Paris	p2	200
s4	Clark	20	Vienna	p2	200
s4	Clark	20	Vienna	p2	200
s4	Clark	20	Vienna	p4	300
s4	Clark	20	Vienna	p5	400

Remarks:

In PRTV, attributes over which the join operation is formed are implicitly defined by the attribute names the joined relations have in common. In other languages, attributes to be compared must be named explicitly;

The above example shows an "equi-" join operation, i.e., compares attribute value for equality. In some languages, comparison operations other than equality can be applied.

Example 7: A Project Operation in the Relational Algebra Notation

Query: Derive the supplier names

Query definition (in PRTV Relational Algebra [TODD76]):

suppliers % sname

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Query evaluation: For each tuple in the "suppliers" relation, select the attribute "sname", i.e., the supplier's name. Copy the value of the attribute "sname" into the result relation.

Result relation :

sname
Smith
Jones
Blake
Clark
Adams

Remarks:

A relation may have more than one tuple with the same value for a given set of (non-key) attributes. However, the result of a project operation on those attributes contains a certain value only once.

In the relational algebra approach, complex queries are expressed by nesting simple queries. Thus, algebraic query languages rely heavily on the closure property of its operators.

Example 8: A Join/Select/Project Operation in the Relational Algebra Notation

Query: Derive the names of those suppliers that shipped some part and either are located in Paris or have a status different from 20.

Query definition (in PRTV Relational Algebra [TODD76]):

suppliers * shipments : (city = "Paris" or status ≠ 20) % sname

"suppliers" relation :

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

"shipments" relation :

snr	pnr	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Query evaluation: Select tuple pairs from relations "suppliers" and "shipments" that have the same value in the attribute "snr" and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Query languages based on the relational calculus support the definition of complex queries by expressions. The basic expression of the tuple relational calculus approach can be characterized as follows:

1. Calculus expressions may refer to any number of source relations.
2. Calculus expressions may apply complex selection criteria that refer to constants, to tuples in source or any other relations. References to tuples are established through so-called tuple variables, free variables for the tuple to be selected, bound variables for the tuples that form the selection criterion.+
3. Calculus expressions may provide transformations to choose distinguished attribute values from the selected tuples or to apply standard computations such as average, maximum, or minimum.

Example 9: A Join/Select/Project Operation in the Tuple Relational Calculus Notation

Query: Derive the names of those suppliers that shipped some part and are either located in Paris or have a status different from 20.

Query definition (in PASCAL/R Relational Calculus [SCHM77]):

```
[<s.sname> of each s in suppliers :
      ((s.city = "Paris") or (s.status ≠ 20))
      and some sp in shipments (s.snr = sp.snr)]
```

"suppliers" and "shipments" relation: see Ex. 8

Query evaluation: Select tuples of the "suppliers" relations that have the same value in the attribute "snr" as some tuple in the "shipments" relation and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of example query 9 is supposed to be identical to that of example query 8.

+This class of selection criteria can be defined formally by the expressions of the first-order predicate logic.

In the domain relational calculus, a query requests a set of tuples of values from "target" database domains. Only those tuples of values which satisfy a "selection criterion" are selected. This criterion is a property expressed in terms of values and additional simpler properties. In the domain calculus, relations express properties of values.

Conceptually, the process of query evaluation in the domain relational calculus consists of three steps:

1. access the elements of the source or target domains;
2. evaluate the selection criterion for any tuple of values that can be formed by taking one value from each target domain. Relations (more precisely, relational predicates) are accessed during that process. References are expressed through so-called domain variables, which range on database domains or on relation columns (i.e., single-attribute projections);
3. add each tuple of values that fulfills the selection criterion to the result relation.

Example 10: A Join/Select/Project Operation in the Domain Relational Calculus

Query: Derive the names of those suppliers that shipped some part and are either located in Paris or have a status different from 20.

Query definition (in DRC [LACR77b]):

```
{sn: ∔ s : suppliers (sname : sn, snr : s)
     and (suppliers (snr : s, city : "Paris")
        or suppliers (snr : s, status : ≠ 20))
     and ∔ part : shipments (pnr : part, snr : s)}
```

Note that "sn", "s", and "part" are domain variables.

Query evaluation: Consider in turn the values of the "sname" attribute of relation "suppliers". Keep as an element of the result each value which occurs in relation "suppliers" with a supplier number "s", that occurs in "suppliers" either with "Paris" as a value of the "city" attribute or with a value different from 20 as a value of the "status" attribute, and that occurs in relation "shipments" with some part number "part" as the value of the "pnr" attribute.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of examples 8, 9, and 10 are the same.

Since the evaluation of a relational calculus expression results in a relation, calculus expressions may be nested. The nesting structure of calculus expressions is that of the predicate calculus; it is the nesting of ranges of bound variables in the selection criterion. Nesting of queries is necessary in the algebraic approach to express more complex queries. Nesting in the algebra is more obvious than in the calculus since it is operations that are nested not variable ranges as in the calculus.

Relational algebra, tuple relational calculus and domain relational calculus mark three positions in the spectrum of relational query languages. Many query languages designed to improve language usability have characteristics of all three approaches.

Example 11: A Join/Select/Project Operation in a notation that combines algebra and calculus features

Query: Derive the names of those suppliers that shipped some part and either are located in Paris or have a status different from 20.

Query definition (in SQL [CHAM76b]):

```
select sname
   from suppliers
      where (city = "Paris" or status ≠ 20)
      and snr in (select snr
                  from shipments)
```

"suppliers" and "shipments" relation: see Ex. 8

Query evaluation: Select tuples of the "suppliers" relation whose values in the attribute "snr" are in the set of "snr" values of the "shipments" relation and have either the value "Paris" in the attribute "city" or a value different from 20 in the attribute "status". Copy the value of the attribute "sname" of each qualified tuple into the result relation.

Result relation :

sname
Jones
Blake

Remarks:

The meaning and result of example query 11 is identical to that of example queries 8, 9, and 10.

In order to alter the value of a relation, its name and its new value have to be provided. In principle, a single altering operator that assigns the new value to the named relation would suffice. There are, however, notational and implementational advantages in distinguishing three altering operators.

1. The insert operator alters a relation value by inserting new tuples into the relation. An insert operation has to provide the name of the relation and the values for the new tuple.

   If none of the relational constraints are violated, the new relation value is equal to the value defined by the union of the old tuples and the new tuple values.

2. The delete operator alters a relation value by deleting tuples. A delete operator has to identify the relation to be altered and the tuples to be deleted.

   If none of the relational constraints are violated, the new relation value is equal to the set difference of the old tuple values and the new tuple values.

   Since the tuples to be deleted are necessarily elements of the relation to be altered, the query mechanisms can be used to select them.

3. The replace operator alters a relation value by replacing the value of distinguished tuples. A replacement operation must name the relation to be altered, identify the tuples that will get new values, and provide new values for these tuples. The effect of a replacement operation can also be achieved by a combination of delete and insert operations.

Example 12: An Insert Operation

Operation: Insert tuples with the values
<s7, Green, 7.5, Zurich>,
<s3, Smith, 15, Paris>,
<s6, Maier, 1, Vienna>,
<s8, Brown, 13, 1982, Toronto>
into "suppliers" relation.

Insert statement (in PASCAL/R [SCHM77]):

```
suppliers :+ [<s7, Green, 7.5, Zurich>,
              <s3, Smith, 15, Paris>,
              <s6, Maier, 1, Vienna>,
              <s8, Brown, 13, 1982, Toronto>]
```

"suppliers" relation (before operation):

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens

Insert operation: new tuples with the above values are supposed to be inserted into the relation "suppliers." The RDBMS has to make sure that none of the structural or integrity rules defined for the relation "suppliers" are violated by the insert operation.

"suppliers" relation (after) operation):

snr	sname	status	city
s1	Smith	20	Vienna
s2	Jones	10	Paris
s3	Blake	30	Paris
s4	Clark	20	Vienna
s5	Adams	30	Athens
s6	Maier	1	Vienna

Remarks:

Three out of the four tuple values to be inserted

violate structural, domain, or integrity constraints defined for the relation "suppliers": tuple value <s7, Green, 7.5, Zurich> has status value 7.5 which is not an integer (domain violation). Tuple value <s3, Smith, 15, Paris> has key value s3 which is already present in the relation "suppliers" (key violation). Tuple value <s8, Brown, 13, 1982, Toronto> has five attributes instead of four as defined for relation "suppliers" (structure violation).

There are different ways of handling operations that would violate a structural or integrity rule. Depending on its nature, a violation can be detected at parsing time (e.g., the violations caused by the tuples with key values s7 and s8) or at execution time (e.g., the violation caused by the tuple with key value s3). Execution time violations may be "resolved" by, e.g., program abortion, trigger invocation, or by eliminating the tuple values that cause the violation (as in the example above).

3. ANALYSIS OF RELATIONAL SYSTEMS

In order to "identify or establish aspects of the RDM and RDBMSs that might be appropriate for standards development", the RTG developed a "Feature Catalogue" of Relational Concepts, Languages, and Systems". The catalogue was intended as an abstract (implementation independent) characterization of RDM and RDBMS concepts. It was used as a common basis for a detailed feature analysis of a number of existing DBMSs that were claimed to support aspects of the RDM. Due to limited resources, only a few DBMSs could be examined.

The objective of this chapter is to identify potential features of a relational standard. The feature analyses of 12 DBMSs were used to prepare this chapter.(1) Different interpretations of the base documents(2) made it necessary in some cases to go back to the originators of the feature analysis. Their answers were not always reflected in the original feature analysis. Furthermore, this also led to some inevitable overlap between the different sections of this chapter.

The structure of this chapter follows closely that of the feature catalogue. Features from each section of the catalogue are summarized in tabular form.

3.1 Development of the Feature Catalogue

A feature catalogue(2) was developed to guide the analyses of selected relational database management systems. The purpose of using a feature catalogue was to ensure that all systems were described in the same format and that all features of interest to the RTG were considered in the analyses.

The object of the analyses was to get insight into:

1. RDM:
   a. essential or core concepts, and
   b. additional concepts or extensions.
2. Systems issues concerning the support and use of the RMD:
   a. aspects closely tied to the model, such as the design of interfaces, and
   b. general system features, e.g., locking, concurrency, security, access control.

Nonrelational features were included in the feature catalogue for two reasons:

1. The analyses were required to decide whether a particular common DBMS feature was related to the RDM or not. One objective of performing the analyses was to determine what DBMS features were dependent on the RDM.
2. One reason for supporting the development of a relational standard is the task group's finding that there are a significant number of complete, useable DBMSs supporting aspects of the RDM. If the only implementations of the RDMs were research vehicles then a standard might be premature. Therefore it was important to determine the full capabilities of the systems being analyzed.

The intent of the task group was to categorize the features on the basis of the analyses according to the following taxonomy [ARTH80]:

1. Features that appear in every system which are claimed to be relational.
2. Features that do not appear in every system but are rapidly being added to many systems.
3. Features that are felt to be important but do not yet appear except in the literature or in research systems.
4. Features that have been in some systems for some time but do not appear to be spreading to other systems.
5. Features that distinguish the relational approach from other approaches.

Features in categories 1 and 2 would form the basis for a standard, while items from categories 3 and 4 could be used as a basis for future developments. Features in Section 5 would be used to exclude nonrelational systems.

As well as supporting a standard, the analyses based on the feature catalogue are expected to be useful in the following ways:

1. To relate specific knowledge of the state of the art to research and development results.
2. To demonstrate the amount and nature of work in implementations, applications, and literature.
3. To determine to what extent RDM concepts are constrained by implementation concerns.

The feature catalogue on which the analyses are based is the third version developed by the RTG. The initial version was based on the Generalized DBMS Survey completed by the CODASYL System Committee in 1969. This version was tested by asking members to analyze selected systems. The catalogue was found to be incomplete because some hierarchical DBMSs could be described as "relational" following the catalogue. A second version of the catalogue placed more weight on purely relational features, but was too detailed for our purposes and somewhat repetitive. In the third version of the feature catalogue this problem is not completely solved. However, the overall organization of information is clear and easy to follow.

The relative importance of relational concepts, languages, and systems is reflected in the structure of the catalogue. First, RDM concepts are considered, followed by associated functional capabilities and ending with DBMS features. Within the functional capabilities, query and manipulation

---

(1) These analyses will be published in Brodie, M.L., and Schmidt, J.W. (Eds.), "Relational Database Management Systems: Analysis and Comparisons," Springer-Verlag, Heidelberg/New York, in press.

(2) "RTG 80-81 Feature Catalogue of Relational Concepts, Languages, and Systems" and "RTG 80-90 Notes on Completing an RTG Feature Analysis."

are considered before definition, generation, and administration. This ordering indicates the distinction between RDM concepts and RDBMS features in response to the RTG charter.

The following systems were analyzed using the final feature catalogue:

IDM (Britton-Lee)
INGRES (University of California, Berkeley)
IPIP (Boeing Computer Services) -- incomplete feature analysis
MRDS (Honeywell)
MRS (University of Toronto)
NOMAD (National CSS, Inc.)
ORACLE (Relational Software Incorporated)
PASCAL-R (Hamburg University)
PRTV (IBM, United Kingdom)
QBE (IBM, Thomas J. Watson)
RAPID (Statistics Canada)
RAPPORT (LOGICA, United Kingdom)
SYSTEM R (IBM San Jose)

3.2 Database Constituents

3.2.1 Definitions

Analysis of the Feature Catalogue indicated that the following seven constituents appeared in a majority of the systems evaluated:

1. Database. The relational database is a collection of relations. Depending on the system, relations can be base relations, views, and/or snapshots.

2. Relation. A relation is a set of relation elements of the same type. A relation may be presented to the user as a table of rows and columns.

3. Tuple. A relation element is called a tuple. A tuple is a row in the tabular presentation of a relation.

4. Attribute. The attributes of a tuple are defined by the tuple type. The legal values of an attribute are chosen from a specific set of values, called domains.

5. User defined domain. The set of values over which an attribute can range is called a domain. A user defined domain allows the user to describe specific domains, such as the domain COLOR where COLOR can be red, yellow, blue, or green.

6. View. A view is a relation derived from existing relations. After its declaration, modifications to its defining relations are reflected in the view, according to its definition.

7. Snapshot. A snapshot is a relation derived from existing relations. After its declaration, it is independent of its defining relations and does not reflect updates made to its defining relations.

The following concepts were listed as constituents for systems that were analyzed.

8. Transaction/Lock. A transaction/lock is a set of queries to be executed without interference from any other query. A lock is normally an interactive command, while a transaction is usually batch. In some systems, however, a transaction can be issued in either batch or interactive mode.

9. Trigger. A trigger is a set of user specified commands that are executed automatically after a specific situation has occurred in the database. For example, modification of a specific relation may cause a trigger to be activated.

10. Authority/Assertion. Authority and assertion allow control privileges over a relation.

11. Relational Dictionary. A data dictionary which is presented to the user in relational form and can be queried using relational operators.

12. Relation Logging. A user can request that a log be maintained of all atomic changes to a specified relation.

13. Workspace. A workspace is an area where a relation can be manipulated and changed without affecting the database.

14. Image. An image defines the output picture for the data.

15. Index. An index is an access mechanism that can be defined for faster access.

16. Macro. A macro is a set of queries defined by a name. When the name is issued, the macro is automatically executed.

3.2.2 Results

The results are displayed in Table 3.1.

3.2.3 Observations on Database Constituents

The twelve feature analyses indicate uniformity in the following database constituents: database, relation, tuple, and attribute. Snapshots and views, while not supported by all of the systems, are well represented with 9 systems supporting snapshots and 8 systems supporting views. PASCAL/R supports snapshots as a temporary relation only. The user defined domain was supported by only half of the systems. Under the heading of other database constituents it was found that the transaction, relational dictionary, and macro were the most prevalent.

3.3 Functional Capabilities

Thirty two features of the functional capabilities of an RDBMS were considered. They involved qualification, retrieval and presentation, alteration, and additional operations. Table 3.2 characterizes these features for the 12 DBMSs and presents a summary.

3.4 Schema Definition

3.4.1 Descriptions

Schema definition features of a system can be put into 11 categories. A description of each follows:

TABLE 3.1 DATABASE CONSTITUENTS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Database	Yes	Yes	Yes	Yes	Yes	Yes
2. Relation	Yes	Yes	Yes	Yes	Yes	Yes
3. View	Yes	Yes	Yes	No	Yes	Yes
4. Snapshot	Yes	Yes	Yes	No	Yes	Yes
5. Tuple	Yes	Yes	Yes	Yes	Yes	Yes
6. Attribute	Yes	Yes	Yes	Yes	Yes	Yes
7. Domain (user defined)	No	No	Yes	No	No	No
8. Other	Transaction, Macro, Relation Logging, Index	Index Macro, Authority	Transaction, Macro	Index Transaction		Macro, Transaction

TABLE 3.1 DATABASE CONSTITUENTS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Database	Yes	Yes	Yes	Yes	Yes	Yes
2. Relation	Yes	Yes	Yes	Yes	Yes	Yes
3. View	No	Yes	No	Yes	No	Yes
4. Snapshot	Yes	Yes	No	Yes	Yes	No
5. Tuple	Yes	Yes	Yes	Yes	Yes	Yes
6. Attribute	Yes	Yes	Yes	Yes	Yes	Yes
7. Domain (user defined)	Yes	Yes	No	No	Yes	Yes
8. Other	Relational Dictionary	Workspace Macro	Trigger, Transaction, Index	Index Transaction, Dictionary	Macro Authority, Image, Index Transaction, Dictionary	Index

TABLE 3.1 DATABASE CONSTITUENTS SUMMARY

Database	Supported by 12 Systems
Relation	Supported by 12 Systems
View	Supported by 8 Systems
Snapshot	Supported by 9 Systems
Tuple	Supported by 12 Systems
Attribute	Supported by 12 Systems
User Defined Domain	Supported by 5 Systems

1. Define Database. An explicit command allows a user to define an entire database. Defining one or more relations is not considered to be an explicit command for defining an entire database.

2. Generate Database. An explicit command exists for populating a database. In addition to this, the generation facility can dump data from the database and into the database, i.e., a fast load/unload facility.

3. Destroy Database. An explicit command exists that will destroy an entire database. Destroying one relation at a time is not considered to be an explicit command to destroy a database.

4. Define Relation. An explicit command defines a relation.

5. Destroy Relation. An explicit command destroys a relation.

6. Reorganize Relation. An explicit command reorganizes a relation.

7. Define Snapshot. An explicit command defines a snapshot.

8. Define View. An explicit command defines a view.

9. Drop Attribute. An explicit command causes an attribute to be deleted from a relation if the original relation still exists. A projection command that chooses all but the unwanted attribute (and forms this into a relation) is not considered to be an explicit command to delete an attribute.

10. Add Attribute. An explicit command will add an attribute to a relation. A join command that forms another relation (which includes all attributes of the original relation and an additional attribute) is not considered to be an explicit command to add an attribute.

TABLE 3.2 OPERATIONS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Retrieval tuple handling	Duplicates removed under user request	Duplicates removed under user request	Duplicates removed under user request	Duplicates removed under user request	Duplicates never removed	Duplicates removed under user request
2. Nesting	Fully	Fully	Fully	Restricted	Restricted	Fully
3. Closure	Fully closed	Fully closed	Restricted		Fully closed	Fully closed
4. Set membership operator	No	No	No		Yes	Yes
5. Set operators (union, intersect, difference, equality)	No	No	Yes	No	No	No
6. Group by operator	Yes	Yes	No	Restricted	Yes	Yes
7. Arithmetic operators	Yes	Yes	Yes	No	Yes	Yes
8. String operators	Yes	No	Yes	No	Yes	No
9. Transaction operators	Yes	No	No	No	No	Yes
10. Existential quantifier	Explicit operation	Implied	Implied	Implied	Implied	Implied
11. Universal quantifier	Implied	Implied	Implied		Implied	Implied
12. Boolean expressions	Yes	Yes	Yes	Yes	Yes	Yes
13. Query language environment	Stand alone	Stand alone and host			Stand alone	Stand alone
14. Self-join capability	Full	Full	Full	None	Full	Full
15. Equi-join capability	Full	Full	Full	Full	Full	Restricted
16. Natural join capability	Full	Full	Full	Full	Full	Restricted
17. Projection capability	Implicit	Implicit	Implicit	Implicit	Implicit	Implicit
18. Sort capability	Explicit	No	No	No	Explicit	Explicit
19. Insert	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
20. Delete	Explicit	Explicit	Restricted	Explicit	Explicit	Explicit
21. Modify	Explicit	Explicit	Explicit and restricted	Explicit	Explicit	Explicit
22. Update over more than one relation	No	Yes			No	No
23. Commit/undo	Explicit	No	No		Implicit	No
24. Triggers	No	No	No	No	Yes	No
25. Aggregate functions	Yes	Yes	Yes	Yes	Yes	Yes
26. User-defined functions	No	No	Yes	No	Yes	No
27. Library functions	No	Yes			Yes	Yes
28. User-defined error conditions (exit capabilities)	Yes	Yes	No	No	Yes	No
29. Format of schema information	Relational	Relational	Other	Other	Other	Relational
30. System performance data available	Yes	No	No	No	Yes	No
31. Report generator	No	Yes	No	No	Yes	Yes
32. Relationally complete	Yes (self-contained)	Yes (self-contained)			Yes (self-contained)	No

11. Rename Attribute. An explicit command will rename an attribute. A command which allows a synonym for an attribute name is not considered to be a command which renames an attribute. The original name of the attribute must not exist after this command is issued.

12. Define Tuple. An explicit command defines a tuple.

Other schema definitions listed for the systems include displaying and modifying a directory, specifying backup, defining a secondary index, checking a domain, grouping a set of relations, encoding and decoding data, and opening and closing a relation.

3.4.2 Results

The results are displayed in Table 3.3.

TABLE 3.2 OPERATIONS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Retrieval tuple handling	Duplicates always removed	Duplicates always removed	Duplicates never removed	Duplicates removed under user request	Duplicates not allowed	Duplicates removed under user request
2. Nesting	Fully	Fully	None	Fully	Fully	Restricted
3. Closure	Fully closed	Fully closed	No	Fully closed	Fully closed	Fully closed
4. Set membership operator	Yes	No	Yes	Yes	No	
5. Set operators (union, intersect, difference, equality)	Yes	Yes	No	Union-supported	Yes	Yes
6. Group by operator	No	Yes	Restricted	Yes	Yes	No
7. Arithmetic operators	Yes	Yes	Yes	Yes	Yes	No
8. String operators	Yes	Yes	No	No	No	No
9. Transaction operators	No	No	Yes	Yes	No	Yes
10. Existential quantifier	Explicit operation	Implied	Implied	Implied	Implied	Implied
11. Universal quantifier	Explicit operation	Implied	No	Implied	Implied	Implied
12. Boolean expressions	Yes	Yes	Limited	Yes	Yes	Yes
13. Query language environment	Stand alone and host	Stand alone	Stand alone and host	Stand alone and host	Stand alone and host	Stand alone
14. Self-join capability	Full	Full	No	Full	Full	Full
15. Equi-join capability	Full	Full	No	Full	Full	Full
16. Natural join capability	Full	Full	No	Full	Full	Full
17. Projection capability	Explicit	Explicit	Restricted (only on output)	Implicit	Explicit	Restricted (only on output)
18. Sort capability	Restricted	No	Explicit	Explicit	Explicit	No
19. Insert	Explicit	Implicit	Explicit	Explicit	Explicit	Explicit
20. Delete	Explicit	Implicit	Explicit	Explicit	Explicit	Explicit
21. Modify	Explicit	Implicit	Restricted	Explicit	Explicit	Explicit
22. Update over more than one relation	No	No	No	Yes	Yes	No
23. Commit/undo	No	No	Explicit	Explicit	Explicit	No
24. Triggers	No	No	No	No	No	No
25. Aggregate functions	No	Yes	Yes	Yes	Yes	No
26. User-defined functions	Yes	Yes	Yes	No	Yes	No
27. Library functions	No	Yes	No	Yes	Yes	No
28. User-defined error conditions (exit capabilities)	No	No	Yes	Yes	No	Yes
29. Format of schema information	Relational	Other	Non-existent	Relational	Relational	Other
30. System performance data available	Yes	No	Yes	No	No	Yes
31. Report generator	Yes	No	No	No	No	No
32. Relationally complete	Yes (self-contained)	Yes (self-contained)	No contained)	Yes (self-contained)	Yes (self-	No

3.4.3 Observations

In general, the systems reviewed dealt with relations rather than with the database itself. All systems had the ability to define a relation, and a great majority allowed the user to destroy and reorganize relations and define snapshots and views. Only eight systems had an explicit command to define a database, and of those only six had an explicit command to populate the database. This is because in several systems, the database is defined and populated implicitly when the relations are defined and generated. Most systems had explicit commands to drop, add, and rename attributes. Only two systems explicitly defined tuples. In the other systems, a tuple is defined implicitly when the relation of which it is an element is defined.

TABLE 3.2 OPERATIONS SUMMARY

1. Retrieval tuple handling	Duplicates removed under user request:	7
	Duplicates always removed:	2
	Duplicates never removed:	2
	Duplicates not allowed:	1
2. Nesting	Full nesting:	8
	Restricted nesting:	3
	Nesting not allowed:	1
3. Closure	Full closure:	7
	Restricted closure:	2
	Not closed:	1
4. Set membership operator	Supported by 5 systems	
	Not supported by 5 systems	
5. Set operators	Supported by 5 systems	
6. Group by operator	Full:	7
	Restricted:	2
	Not supported:	3
7. Arithmetic operators	Supported by 10 systems	
8. String operators	Supported by 5 systems	
9. Transaction operators	Supported by 5 systems	
10. Existential Quantifier	Explicit:	2
	Function provided by other operator combinations:	10
11. Universal quantifier	Explicit:	1
	Function provided by other operator combinations:	9
	Function not supported:	1
12. Boolean expressions	Full:	11
	Limited:	1
13. Query language environment	Stand alone:	4
	Stand alone and host:	5
14. Self-join capability	Supported by 10 systems	
15. Equi-join capability	Full:	10
	Restricted:	1
	Not supported:	1
16. Natural join capability	Supported by 10 systems	
17. Projection capability	Explicit:	3
	Implicit:	7
	Restricted to output:	2
18. Sort capability	Explicit:	6
	Restricted:	1
	Not supported:	5
19. Insert	Explicit:	11
	Implicit:	1
20. Delete	Explicit:	10
	Implicit:	1
	Restricted:	2
21. Modify	Explicit:	9
	Restricted:	2
	Implicit:	1
22. Update over more than 1 relation	Supported by 3 systems	
23. Commit/undo	Explicit:	4
	Implicit:	1
	Not supported:	6
24. Triggers	Supported by 1 system	
25. Aggregate functions	Supported by 10 systems	
26. User defined functions	Supported by 6 systems	
27. Library functions	Supported by 5 systems	
28. User defined error conditions	Supported by 6 systems	
29. Format of schema information	Relational:	6
	Other:	5
30. System performance data available	Yes:	5
	No:	7
31. Report generator	Yes:	4
	No:	8
32. Relationally complete	Yes:	7
	No:	2

3.5 Additional Definition, Generation, and Administration Facilities

3.5.1 Background

Additional definition, generation, and administration facilities are shown in Table 3.4. These facilities were chosen based on the importance of the relational model as determined by the RTG. Facilities are considered to be not applicable (N/A) if the constituent about which the facility is defined is not supported in the system.

1. Attribute Synonyms. An explicit command exists to define a synonym for an attribute name of a relation.

2. Relation Synonyms. An explicit command exists to define a synonym for a relation name.

TABLE 3.3 SCHEMA DEFINITIONS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
Define database	Yes	Yes	Yes	No	Yes	No
Generate Database	Yes	Yes	No		Yes	No
Fast load/unload	Yes	Yes	No		Yes	No
Destroy database	Yes	Yes	No	Yes	Yes	No
Define relation	Yes	Yes	Yes	Yes	Yes	Yes
Destroy relation	Yes	Yes	No	Yes	Yes	Yes
Reorganize relation	Yes	No	No	Yes	Yes	No
Define snapshot	Yes	Yes	Yes		Yes	Yes
Define view	Yes	Yes	Yes		Yes	Yes
Drop attribute	No	No	No	Yes	Yes	No
Add attribute	No	No	No	Yes	Yes	Yes
Rename attribute	Yes	No	Yes	Yes	Yes	No
Define tuple	No	No	No		No	No
Other	Open, close		Index, check domain	Display database, backup	Group a set of relations, encode/decode	

TABLE 3.3 SCHEMA DEFINITIONS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
Define database	Yes	Yes	Yes	Yes	Yes	Yes
Generate Database	Yes	No	Yes	Yes	Yes	
Fast load/unload	Yes	No	Yes	Yes	Yes	
Destroy database	Yes	Yes		Yes	Yes	
Define relation	Yes	Yes	Yes	Yes	Yes	Yes
Destroy relation	No	Yes		Yes	Yes	Yes
Reorganize relation	Yes	Yes	Yes	No	Yes	No
Define snapshot	Yes	Yes		Yes	Yes	No
Define view	No	Yes	No	Yes	No	Yes
Drop attribute	No	Yes		Yes	Yes	Yes
Add attribute	No	Yes		Yes	Yes	Yes
Rename attribute	Yes	Yes	Yes	No	Yes	Yes
Define tuple	Yes	No	No	No	No	Yes
Other						Open, close, status

TABLE 3.3 SCHEMA DEFINITIONS SUMMARY

Define database	Supported by 10 systems
Generate database	Supported by 7 systems
Fast load/unload	Supported by 7 systems
Destroy database	Supported by 8 systems
Define relation	Supported by 12 systems
Destroy relation	Supported by 9 systems
Reorganize relation	Supported by 7 systems
Define snapshot	Supported by 9 systems
Define view	Supported by 8 systems
Drop attribute	Supported by 6 systems
Add attribute	Supported by 7 systems
Rename attribute	Supported by 9 systems
Define tuple	Supported by 2 systems

TABLE 3.4 ADDITIONAL FACILITIES

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
Attribute synonyms	Yes	No	Yes	No	Yes	Yes
Relation synonyms	Yes	No	Yes	No	No	Yes
Duplicate tuples	Yes	No	Yes (view) No (snpsht)	Yes	Yes	Yes
Can view be defined over more than one relation	Yes	Yes	Yes	N/A*	Yes	Yes
Can view be defined from other views	Yes	No	Yes	N/A	No	Yes
User/DBA can define mapping to explicit update view	No	No	No	N/A	No	No
Selection criterion	Yes	Yes	Yes	Yes	Yes	Yes
Aggregate function	Yes	Yes	Yes	Yes	Yes	Yes
Primary keys required	No	No	Yes	No	No	No
Key can be modified	Yes	Yes	No	N/A	Yes	Yes
Key can be declared unique	Yes	No	Yes	N/A	Yes	Yes
Concatenated primary key	Yes	No	Yes	N/A	Yes	No
Views inherit keys	No	Yes	Yes	No	No	N/A
Null Valued keys allowed	No	No	Yes	No	Yes	No

*N/A = not applicable

TABLE 3.4 ADDITIONAL FACILITIES (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
Attribute synonyms	Yes	No	Yes	No		No
Relation synonyms	Yes	Yes	No	Yes		No
Duplicate tuples	No	No	No	Yes	No	Yes
Can view be defined over more than one relation	N/A	Yes	N/A	Yes	N/A	No
Can view be defined from other views	N/A	Yes	N/A	Yes	N/A	
User/DBA can define mapping to explicit update view	N/A	No	N/A	Yes	N/A	
Selection criterion	Yes	Yes	Yes	Yes	Yes	Yes
Aggregate function	No	Yes	Yes	Yes	Yes	No
Primary keys required	Yes	No	Yes	No	No	Yes
Key can be modified	No	N/A		Yes		Yes
Key can be declared unique	Yes	N/A	Yes	Yes	Yes	No
Concatenated primary key	Yes	N/A	Yes	Yes	Yes	Yes
Views inherit keys	N/A	N/A	N/A	Yes	N/A	No
Null Valued keys allowed	No	N/A	No	Yes	No	Yes

3. Duplicate Tuples. The system supports storage of duplicate tuples. This may be due to the way the system has been implemented or it may be a user option.

4. Can a view be defined over more than one relation, e.g., in the case of join, union, intersection, and relative complement operations?

5. Can a view be defined from other views, i.e., can a dynamic derived relation be used in the definition of another dynamic derived relation?

6. User/DBA can define a mapping to update a view, i.e., a view can be explicitly updated, causing updates to be mapped down to its defining relations?

7. Selection Criterion. Tuples of a relation can be selected based on meeting a user specified criterion.

8. Aggregate Function. The system supports aggregate functions such as MIN, MAX, and COUNT.

9. Primary Keys. The system requires primary keys as part of the logical data definition.

10. Keys can be modified. A user can modify a key value.

11. Keys can be declared unique. Once a key is declared, its value must be unique for each tuple. Note that this feature allows, at the

TABLE 3.4 ADDITIONAL DEFINITION, GENERATION, AND ADMINISTRATION FACILITIES SUMMARY	
Attribute synonyms	Supported by 6 systems
Relation synonyms	Supported by 6 systems
Duplicate tuples	Supported by 8 systems
Can view be defined over more than one relation	Supported by 7 systems
Can view be defined from other views	Supported by 5 systems
User/DBA can define mapping to update view	Supported by 1 systems
Selection criterion	Supported by 12 systems
Aggregate function	Supported by 10 systems
Primary keys required	Supported by 4 systems
Key can be modified	Supported by 6 systems
Key can be declared unique	Supported by 9 systems
Concatenated primary key	Supported by 8 systems
Views inherit keys	Supported by 3 systems
Null valued keys allowed	Supported by 4 systems

user option, definitions of "primary" and "candidate" keys for certain relations.

12. Concatenated Key. A key can consist of more than one attribute of a relation.

13. Views inherit keys. When a view is declared, the key is inherited from its underlying relations.

14. Null valued keys are supported. A primary key may have a null value.

3.5.2 Results

The results are displayed in Table 3.4.

3.5.3 Observations

The selection criterion facility is supported by all of the systems. Often used with selection is the aggregate function, which is found in ten of the systems. Eight of the systems support duplicate tuples; however, in most of these systems, the user can specify if he/she wishes to see unique tuples only. Synonyms are allowed for both attributes and the relations half of the systems. The domain of an attribute can be operated on in only five of the systems. An example of such an operation might be limiting the domain of an attribute SALARY such that the fixed decimal value is greater than zero and less than $100,000. In most systems the domain is an implicit entity used in the definition of attributes.

Primary keys are supported in 10 of the systems. In four systems -- SYSTEM R, ORACLE, IDM, and NOMAD -- the key is unique only if the user specifies. A majority of the systems that supported keys allowed modification of key values and allowed a key to be a set of concatenated attribute values.

Views can be defined in terms of more than one relation in seven of the nine systems which support views. Less than half of the systems allow views to be defined in terms of other views, to inherit keys of their defining relations, and to be defined prior to run time. Only SYSTEM R allows the explicit update of views.

3.6 Functional Classes

3.6.1 Definitions

In analyzing the feature catalogue, 12 different functional classes were defined. These classes were used to group logically relational functions and are described below.

1. Database Schema Definition. These functions are included in interfaces 1, 4 of ANSI/SPARC DBMS framework and include the conceptual and external schema source definitions.

2. Database Retrieval. These functions include the set oriented and record-at-a-time oriented retrievals from the database. They include functions found in interfaces 7, 8, 9, 10, 11 of the ANSI/SPARC DBMS framework.

3. Database Altering. These functions include the set oriented and record-at-a-time oriented inserts, modifications, and deletes in the database. They include functions found in interfaces 7, 8, 9, 10, 11 of the ANSI/SPARC DBMS framework.

4. Integrity Constraints. These functions include the definitions of constraints on data values and their relationships. They include functions that would be in interfaces 1, 4 of the ANSI/SPARC DBMS framework.

5. Database Generation and Regeneration. These functions include facilities, separate from the database altering functions, for logical bulk loading, unloading, and reloading of relations. These functions can be found in interface 39 of the ANSI/SPARC DBMS framework.

TABLE 3.5 FUNCTIONAL CLASSES

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Database schema definition	QL commands dynamic	QL commands dynamic	Utility static	QL commands dynamic	QL commands dynamic	QL dynamic
2. Database retrieval Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time		Yes	Yes		Yes	Yes
3a. Database insert Set-oriented	Yes	Yes				Yes
Record-at-a-time		Yes	Yes	Yes	Yes	Yes
3b. Database modify Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time			Yes	Yes	Yes	Yes
3c. Database delete Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time			Yes	Yes	Yes	Yes
4. Integrity constraints	QL commands	QL commands	DD utility		DD command	
5. Database generation, regeneration	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load	Logical bulk load
6. Database schema redef. & renaming	Indirect	Indirect	Indirect	Indirect	Indirect	Indirect
7. Report generation		Print command	Print command		Print command	Utility
8. Special data entry			Prompting			Prompting
9. Security, monitoring	QL commands	QL commands	DD utility	QL commands	QL commands	DD utility
10. LOAD/DUMP/recovery	Yes	Yes	Yes	Yes	Yes	
11. Definition of access paths	QL commands	QL commands	DD utility	QL commands	QL commands	QL commands
12. Database dictionary	QL access	QL access	Special commands	QL access	Special commands	QL access

TABLE 3.5 FUNCTIONAL CLASSES (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Database schema definition	Utility static	QL command dynamic	Utility static	QL command dynamic	QL command dynamic	User program dynamic
2. Database retrieval Set-oriented	Yes	Yes	Yes	Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3a) Database insert Set-oriented	Yes	Yes		Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3b) Database modify Set-oriented	Yes			Yes	Yes	
Record-at-a-time	Yes	Yes	Yes	Yes	Yes	Yes
3c) Database delete Set-oriented	Yes	Yes	Yes	Yes	Yes	Yes
Record-at-a-time	Yes	Yes		Yes	Yes	Yes
4. Integrity constraints	DD utility	none	DD utility	QL command	QL command	DD utility
5. Database generation, regeneration	Logical bulk load	Logical bulk load			Logical bulk load/unload	Logical Bulk load/unload
6. Database schema redef. & renaming	Indirect	QL commands	Indirect	QL command	QL command	Utility
7. Report generation	PRINT command	Utility		PRINT command	PRINT command	Utility
8. Special data entry		Utility				Forms
9. Security, monitoring				QL command	QL command	
10. LOAD/DUMP/recovery	Yes	Yes	Yes	Yes	Yes	Yes
11. Definition of access path	DD utility		DD utility	QL command	QL command	Utility
12. Database dictionary	QL access	QL access	QL access		QL access	Utility

TABLE 3.5 FUNCTIONAL CLASSES SUMMARY	
1. Database schema definition	Separate utility/static : 3 systems QL command/dynamic : 8 systems User program/dynamic : 1 system
2. Database retrieval	
Set-oriented	Supported by 11 systems
Record-at-a-time	Supported by 10 systems
3a) Database insert	
Set-oriented	Supported by 7 systems
Record-at-a-time	Supported by 11 systems
3b) Database modify	
Set-oriented	Supported by 9 systems
Record-at-a-time	Supported by 10 systems
3c) Database delete	
Set-oriented	Supported by 12 systems
Record-at-a-time	Supported by 9 systems
4. Integrity constraints	QL commands : 4 systems DD utility : 5 systems None : 2 systems
5. Database generation, regeneration	Bulk load supported by 10 systems
6. Database schema redef. & renaming	indirect (unload, destroy, create, load) : 8 utility : 1 separate command : 3
7. Report generation	PRINT command : 7 systems utility : 3 systems
8. Special data entry	Prompting of att. names : 3 systems Forms approach : 3 systems
9. Security, monitoring	QL commands : 6 systems Part of data def. : 2 systems
10. LOAD/DUMP/recovery	Supported in 11 systems These & others depend on O.S.
11. Definition of Acess Paths	QL commands : 7 systems Part of data def. : 4 systems
12. Database dictionary	QL access : 8 systems Special commands : 3 systems

6. Database Schema Redefinition and Renaming. These functions include facilities for renaming attributes and relations and adding or deleting attributes to already existing relations and adding or deleting entire relations. These functions would be supported through functions 1 and 4 of the ANSI/SPARC framework.

7. Report Generation. These functions include facilities that would be found in interface 8 of the ANSI/SPARC framework and are used for formatting and producing printed reports.

8. Special Data Entry. These functions would include screen entry or form entry processors and correspond possibly to function 11 of the ANSI/SPARC framework.

9. Security Definition and Monitoring. These functions are used to define protection and access rights and to indicate whether and how security violations are to be housed. They would be included in functions 1 and 4 of the ANSI/SPARC framework.

10. Database Utilities. These functions would support physical dumping and loading of databases and rollback and rollforward from database logs. They could correspond to function 24 of the ANSI/SPARC framework.

11. Definition of Storage Structure and Access Paths. These functions are used to define the static internal schema of the database and as such are similar to function 13 of the ANSI/SPARC framework.

12. Database Dictionary. These functions are used for retrieval of schema data and auxiliary user supplied textual descriptions, and can include functions found in functions 34, 35, 36, 37, 38, 2, 3, 5, 6, 14 and 15 of the ANSI SPARC architecture.

3.6.2 Results

The results are displayed in Table 3.5.

3.6.3 Observations

The completed feature catalogs of 12 relational systems were analyzed in order to locate the existence of these functional classes. Note that this analysis was concerned with the existence of these classes and the functional similarity rather than with their power or completeness.

There appears to be uniformity in many of the systems on some of the classes. These classes, in decreasing order of uniformity, include set oriented retrieval operations (ignoring duplicate removal), set oriented tuple deletion and modification operations, record-at-a-time insertion operations, query language style commands to define the database, and query language style commands to retrieve those definitions. These classes of functions appeared somewhat similar in 8 or more than 7 of the systems.

Other function classes were either not present in more than 7 of the systems or exhibited considerable variability in this style.

3.7 Interface Flavors

3.7.1 Definitions

The functions to the various systems can be characterized by expected uses of the function. Three such categories include:

1. Interactive End User Interface. In this function the user issues 'query' language commands through a keyboard (CRT or hard copy) terminal to access the database.

2. Host Programming Language Interface. In this function, a programming language such as COBOL, PL/1, or FORTRAN is used to access the database. These function can be implemented with the use of a preprocessor which maps database commands to host language subroutine calls, with direct (user controlled) subroutine calls, or with actual extensions to an existing (or new) language and compiler.

3. Special Interfaces. These functions generally require special hardware, such as graphics terminals, for accessing the database.

3.7.2 Results

The results are displayed in Table 3.6.

3.7.3 Observations

The availability of these interfaces in the systems analyzed are shown in Table 3.6. Most systems supported an interactive end user interface and a host programming language interface. The realization of the programming language functions varied considerably, as did the special functions.

3.8 System Architectures

3.8.1 Results

The systems analyzed can be loosely grouped into two types of architectures. Those architectures are shown in Figure 3.1. Systems of the Type A architecture have separate utilities for defining the database and for manipulating the data items. In those systems the data definitions are stored in an internal form where they are accessed (retrieval only) in order to process data manipulation commands. Some systems in this architecture also store the data definitions in relations for convenient and powerful access by database users. Systems of the Type A architecture include MRDS, NOMAD, PASCAL/R and RAPPORT.

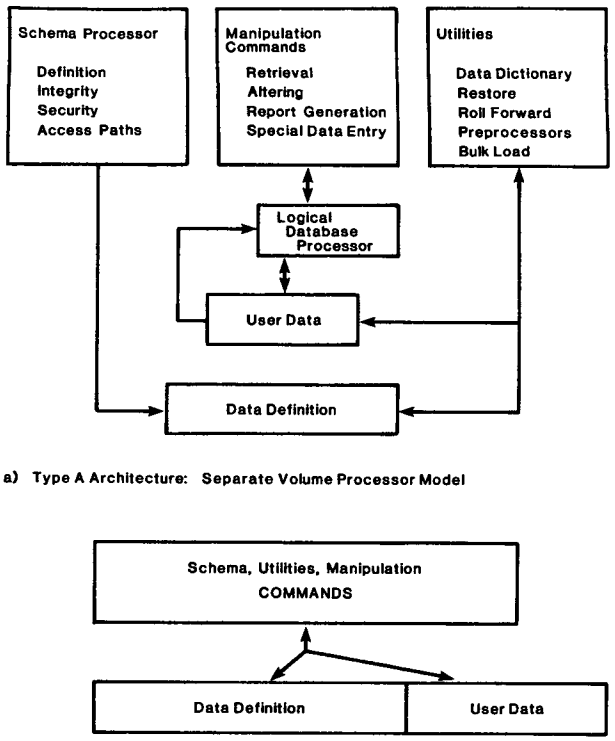

Figure 3.1 Relational Architectural Styles

Systems of the Type B architecture, on other hand, generally have one query level type interface (for both retrieval and update) through which schema commands and data manipulation commands can be freely intermixed. Systems with this type of architecture include IDM, INGRES, MRS, ORACLE, PRTV, SYSTEM R and QBE.

3.8.2 Observations

A few observations can be made about the architectures and functions of the systems. First, many of the systems store system data in relations. Those relations, protection permitting, can be accessed by the same retrieval commands that access user relations. However, special create or define commands are required to populate the system data. In a few systems it may be possible to apply 'user relation' update commands to the system relations.

Secondly, in most of the systems, the record-at-a-time operators appear to go through the same level of function as do the set-at-a-time operators. In no system is it made explicit to the user that the 'user set oriented operators' are mapped to the 'user individual record oriented operators.' In another system, IPAD (for which the architecture part of the feature analysis was not completed), that type of 'set operator' to 'record operators' mapping is suggested.

TABLE 3.6 INTERFACE FLAVORS

	Host Programming Language				
	Interactive End User	Pre-processor	Calls	New Language	Special Interface
1. IDM					Yes - Back End Machine
2. INGRES	Yes	Yes			CUPID
3. MRDS	Yes		Yes		
4. MRS	Yes				Interactive Subsystem (tuple editor)
5. NOMAD	Yes		Yes	Yes	SCREEN Editor for Data
6. ORACLE	Yes		Yes		Forms for Data, Editor for Commands
7. PASCAL/R	Yes			Yes	
8. PRTV	Yes		Yes		
9. RAPPORT	Yes	Yes (?)			
10. SYSTEM R	Yes	Yes			
11. QBE	Yes		Yes		QBE Tabular Form
12. RAPID			Yes		Yes - various utilities
	10 systems	3 systems	6 systems	2 systems	7 Special Interfaces have been defined
	10 systems				

3.9 Operational Aspects

3.9.1 Definitions

In section 6 of the feature analysis, the security, concurrency control, and crash recovery mechanisms of the 12 relational database systems were reviewed. Information was provided in the following areas:

1. Security -- User Related. This section, labeled access control, was concerned with how (and if) users were identified to the DBMS, Three types of identification were identified:

   a. User Id from Operating System. The DBMS, through a system call, identifies the current user in order to determine the users capabilities.

   b. Passwords. The DBMS requires a user to supply a password in order to determine the user's capabilities.

   c. File Access Control. The Operating System controlled access to the database files. For some DBMSs, this method is the only form of access control. For others, the O.S. control is used in conjunction with user identification of type a or b. For the latter type of system, the DBMS itself is the owner of the database files.

2. Security -- Data Related. This section, labeled capabilities, was concerned with the granularity and/or level of protection.

   a. Item protected for most systems was a relation, view, or both, but had to be explicitly named. Some systems allowed protection on specific attributes of relations.

   b. Value based protection indicated that the data value can be used to define the items to be protected. Note this feature is distinct from the value dependent view mechanism in that the value based granules need not be 'named'.

3. Concurrency Control. This section was concerned with the consistency of the database during multiple user update. The particular items noted were:

   a. Multiple User Updates. Could users simultaneously (at least from their perspective) update the database?

   b. Multiple Update Commands. If so, could the user group multiple update commands into a transaction?

   c. Degree of Consistency. If multiple users could simultaneously update the database, what degree of consistency was permitted? Degree 3 consistency implies that no user sees the update of another user until the other user has completed a transaction (possibly a one command transaction). Some DBMSs achieve Degree 3 consistency by locking everything that a transaction reads or writes and holding those locks until the end of the transaction. With Degree 1 consistency, one user can see updates of other user transactions that have not completed. Some of the DBMSs achieve Degree 1 consistency by locking records (or pages) while they are in main memory but not when the records are returned to the disk.

   d. Explicit Locks. If multiple users could simultaneously update the database, can they explicitly preclaim resources by requesting locks for those resources.

4. Crash Recovery. Does the system make any provisions for protecting the consistency of the database in the presence of system and disk crashes?

a. Database Back-up and Restore: Does the Database System have special functions for backing up the disk copy of the database and restoring the database from that back-up? If the database system does not have those functions, some feature analyses suggested that operating system file dump and restore utilities could be used on the database files.

b. Logging. Is logging of record changes or page changes supported? By supported, we mean that the log can be 'rolled forward' together with the DB dump/restore to return the database to a consistent state in the case of a disk crash.

c. Recovery from System Crash. Does the system have any provisions for protecting the database from inconsistencies due to partially complete transactions and soft (non-disk damaging) crashes?

3.9.2 Results

The results are displayed in Table 3.7.

3.9.3 Observations

The results shown in Table 3.7 indicate that there are a variety of approaches to these operational aspects of DBMSs and that approaches do not seem related to the relational model.

The results are important, however, establishing the fact that some of the relational systems are in some sense 'complete' DBMSs; they are suitable for shared databases used by multiple users simultaneously updating the database, and they allow for environmental realities such as soft and hard crashes.

TABLE 3.7 OPERATIONAL ASPECTS

	IDM	INGRES	MRDS	MRS	NOMAD	ORACLE
1. Security (user related)						
a. User ID from OS	Yes	Yes			Yes	Yes
b. Passwords						Yes
c. File access by OS		Yes	Yes	Yes	Yes	
2. Security (data related)						
a. Item of protection	Views/ relations/ attributes	Relations/ attributes	View/ relations	Relation	Relation/ Attributes	View/ Relations
b. Based on data value		Yes			Yes	
3. Concurrency control						
a. Multiple user update	Yes	Yes	Yes		Yes	Yes
b. Multiple commands per transaction	Yes	No	Yes		Yes	Yes
c. Degrees of consistency	3	3	3		1	3 if explicit preclaim 1 otherwise
d. Explicit locks	No	No	Yes			Yes
4. Crash recovery						
a. DB back-up/restore	Yes	OS	OS	Yes	OS	OS
b. Logging	Record					
c. Recovery from system crashes	Yes	Yes	Yes		Yes	

TABLE 3.7 OPERATIONAL ASPECTS (continued)

	PASCAL/R	PRTV	RAPPORT	SYSTEM R	QBE	RAPID
1. Security (user related)						
a. User ID from OS				Yes	Yes	
b. Passwords	Yes					
c. File access by OS	Yes	Yes		Yes	Yes	Yes
2. Security (data related)						
a. Item of protection				Views/ relations/ attributes	Relation/ Attribute	View
b. Based on data value					Yes	
3. Concurrency control						
a. Multiple user update	No	No	Yes	Yes	No	Yes
b. Multiple commands per transaction				Yes		Yes
c. Degrees of consistency				3,2 or 1 user selects		
d. Explicit locks			Yes	No		Yes
4. Crash recovery						
a. DB back-up/restore	OS	OS	Yes	Yes	Yes	Yes
b. Logging				Page logging	Page logging	Page logging
c. Recovery from system crashes			Record level	Yes	Yes	

TABLE 3.7 OPERATIONAL ASPECTS SUMMARY

1. Security (user related)	
a. User ID from OS	6 systems
b. Passwords	2 systems
c. File access by OS	9 systems
2. Security (data related)	
a. Item of protection	Views: 5 systems Relations: 8 systems Attribute(R,W): 5 systems
b. Based on data value	3 systems
3. Concurrency control	
a. Multiple user update	Yes: 8 systems
b. Multiple commands per transaction	Yes: 6 systems
c. Degrees of consistency	3 : 5 systems
d. Explicit locks	Yes: 4 systems
4. Crash recovery	
a. DB back-up/restore	6 systems (OS for others)
b. Logging	Record : 3 systems Page : 2 systems
c. Recovery from system crashes	6 systems

4. ARCHITECTURAL ISSUES AND THE RDM

4.1 The Importance of Architecture

Database management systems are complex hardware and software systems used for a wide variety of applications. This complexity and variety has resulted in DBMSs with different functions and architectures. The differences in DBMSs are further complicated by the fact that DBMSs interface to various programming languages and to end user query languages. The functions and architectures of database management systems also differ in terms of the data models they support.

An architectural framework for DBMSs is frequently proposed in order to unify the diversity in systems and approaches. Such a framework can serve many purposes.

First, the framework can be used to classify objects in a database and identify and define the operations on those objects independent of any particular implementation. Within such a framework, related functions can be grouped, the semantics of those functions defined, and uniform terminology established.

In the presence of such a 'semi-formal' database system definition, the difference and commonalities of different models can be determined. This is the second purpose of an architectural framework. Some of the objects and functions will be the same, regardless of the database model. Other objects and functions will only be defined for specific data models or definitions will vary depending on the data model. In addition to identifying the model dependent components (objects and functions), the framework may suggest mappings between the different models.

The third purpose of an architectural framework is to guide the development of DBMS standards. An architecture will allow for the standardization of both internal and external interfaces. The standardization of an internal interface such as 'logical database processor', could result in one DBMS product that is used to support external user interfaces provided by different DBMS vendors. Similarly, a competitor could develop a different logical database processor that conformed to the standards, and market that product without developing any end user interfaces. The designers of such internal and external standards use the framework definitions to determine which interfaces can be the same for all data models and which interfaces must be model dependent.

Additional purposes for an architectural framework have been proposed. The architecture could be used to guide the first stages of designing and developing a database management system. The framework could also serve as a basis for DBMS research efforts for identifying and formally specifying interfaces, defining data models, etc.

In this section, the impact of the relational model on architectural frameworks is analyzed. In Section 4.2, five approaches to architectural frameworks are reviewed. In Section 4.3, several open issues concerning relational standards are discussed. In Section 4.4, the architectures of two of the best known relational systems, INGRES and System R, are reviewed. Finally, in Section 4.5, a potential RDBMS architecture is presented and compared to several of the proposed architecture frameworks. The central goal of this section is to examine the relationship between potential RDBMS standards and more general DBMS standards.

4.2 Alternative DBMS Architecture Frameworks

Several different approaches to DBMS architecture frameworks are currently being developed. Some of the more prominent of these architectures are briefly described below. Note that it is not the jurisdiction or intention of the RTG to select or evaluate these different approaches. Rather, the RTG considered whether or not some or all of the approaches are compatible with a relational data model standard.

A. "ANSI/X3/SPARC DBMS FRAMEWORK" by An ANSI/X3/SPARC Study Group of Database Management Systems. [TSIC78].

The study group presents a generalized framework for the description of database management systems. In particular, the group recommends 42 interfaces for standardization. In addition, a schematic of processors for interrelating those interfaces is proposed. Those processors can be thought of as internal processing functions, program preparation and executive subsystems, and people acting in specific roles.

The study group's report emphasizes three levels of a schema corresponding to the external, conceptual, and internal view of the database. The external view of the database is used to describe the data as seen by the programmer. The conceptual view of the database describes how the information of an enterprise is modelled in the database. Finally, the internal view of the database is used to describe the data to the system. One of the main goals and contributions of the ANSI/SPARC framework was to allow data independence in the presence of evolving changes in an organization at the external levels without requiring extensive changes at the internal level.

The ANSI/SPARC framework is the basis upon which the DBS-SG must develop and maintain a Reference Model for standardization. That Reference Model is designed to serve as a single global map of a generic DBS onto which specific standardization efforts can be projected. The DBS-SG Reference Model can be classified into two categories dealing with the direct and indirect properties of DBMSs. The direct properties include functional components, the connections between the components, and the interfaces. A function supported by an interface is directly processed by its immediate component. However, the processing of that function may require indirect interactions with schema definitions, system logs, user authentication, etc. These activities are referred to as indirect properties of the database architecture and include the placement of the data declarations, and the propagation of those declarations through the Reference Model.

B. "On a Functional Framework for Database Management Systems" by Michael L. Brodie [BROD80a].

A DBMS framework is defined as "An implementation-free characterization of a DBMS in terms of its functional components". The programming language concept of data abstractions is used

to encapsulate the different components of a database together with the DBMS operations on those components. In particular, an abstract, functional DBMS component is defined. It consists of a language or syntax for initiating functions, the functions, and the objects that the functions act on.

As an example of the framework abstractions, thirteen types of objects and ten basic functions are introduced. The potential functional components can then be defined as an entry in the matrix (13 by 10) which represents the application of each basic function to each object.

C. "NBS Strawman Architecture for a Family of DBMS Standards" by the Computer Corporation of America for an NBS sponsored project [CCA80].

The Computer Corporation of America project is to develop a framework architecture which can be used to guide the evolutionary development of database standards. There are four key concepts in the CCA architecture:

1. a comprehensive list of DBMS functions are grouped into components that could be marketed as independent database products;
2. a family of components are defined for allowing different standards for different data models;
3. a standard schema of schemas and its associated functions are defined; and
4. both internal and external interfaces are defined.

D. "Union of Existing Data Models" as proposed by Johnson (IPAD) [JOHN80], Date (UDL) [DATE 76; 79a; 79b; 79c; 79d; 80], and others.

This approach emphasizes the commonality between the hierarchical, network, and relational data models. Rather than develop a separate standard for the relational model, it could be considered as just a subset of a unified model. Under this approach, the database architecture could closely follow the existing CODASYL architectures.

4.3 Model Related Architectural Issues

4.3.1 The Schema Architecture

A database model can be described by its data structure, the operations on that structure, and the integrity constraints on these operations and structures. The relationship between the different models and how that relationship is encapsulated can affect DBMS standards in at least two important ways. First, the components of an architecture can be arranged in such a way that the different models could share some of the same internal components. For example, the following three approaches could be used to integrate the models.

1. The ANSI/SPARC DBMS framework introduced the concept of external, conceptual, and internal schemas. It has been suggested that a "semantically rich" model for the conceptual schema, such as an entity relationship model, generalization hierarchies, or functional data models be defined. In this approach, the conceptual schema would be considered "rich", i.e., define many problem oriented properties. The external schemas could consist of relational, hierarchical, or CODASYL objects and use their respective operations and integrity constraints to map to a common, rich conceptual schema.
2. The relational model itself is considered a conceptual schema independent of CODASYL, or hierarchical conceptual schemas. Those independent conceptual schemas could be mapped to common access methods (internal schema).
3. The different models are mapped to a common data model such as a set-theoretic model. Such models can be called sparse because they consist of a small number of concepts (i.e., a set and an element of a set). These concepts are augmented by extremely powerful and general operators. Such models are not usually considered as "conceptual models" with which to represent the information in an enterprise.

The first approach would allow different external schemas (i.e., based on relational, hierarchical, or network models) to be defined over the same database. However, users of a relational external schema may not be aware of multiple entity interdependencies. Either the user will have to understand the underlying conceptual schema, or considerable research will be needed to adequately specify and implement the mappings between conceptual and relational objects. Furthermore, new data models may be difficult to map onto the rich model used for the conceptual schema.

An advantage of the second approach is the 'simpler' view: that what a user sees is realized in the conceptual model. Any multiple entity constraints are value dependent and expressed in a manner similar to the user's query language. Within the second approach an individual database would have to be declared as a relational or a network model, but not both.

The third approach has the advantage of being more extendible to new data models. However, there may be performance penalties related to both mapping overhead and lost optimization opportunities between the semantically weak schema and the access methods.

The above discussion concerns the effects of data models on the component architecture for DBMSs. The second effect of data models on DBMS standards concerns the "exact" definitions of the data models. DBMS standards need to specify both the syntax and semantics of the model operations on the model objects in the presence or absence of model integrity constraints. In order to specify the model semantics and make comparisons between the relative power of the models, a formal and consistent definition of the models may be appropriate. Note that approaches 1) or 3) mentioned above could be used for model definitions independently of any component architecture.

4.3.2 Multiple Models in User Interfaces

There are two approaches to supporting multiple models in query languages and host language interfaces in one DBMS system:

1. One approach to the different models takes the union of the functions in the different models. This approach, as suggested in UDL and IPAD, would then allow a relational or hierarchical system to be restricted to a subset of the language constraints.

2. An alternate approach to the different models takes the intersection of the functions in the different models. This approach, as suggested by the CCA architecture, would then allow a network, hierarchical, or relational system to be an independent extension of the base set of functions.

An advantage of the first approach would be that relational and hierarchical systems could be defined as just a subset of the CODASYL like systems. Unfortunately, if the models do not suitably interact, the definitions of the data models may have to be changed. An advantage of the second approach would be that features that are easier to implement in one of the models (i.e., query languages and their optimizers are easier to implement in a relational system) could be standardized without waiting for the feature to be designed and implemented for all of the models. A disadvantage of the second approach is that some interfaces for different models could needlessly evolve in different directions.

4.3.3 Separate or Unified Components of an Architecture

Approaches to incorporating different models into database framework architecture can be grouped into three classes:

1. One architecture with multiple components and interfaces can be defined. Each component and interface in the framework would have to be defined to accommodate all of the models. This architecture approach is similar to the first approach for supporting multiple data models in one user interface.

2. The architecture can be divided into those components and interfaces which are model dependent and those which are not. The components that are model independent could be standardized on their own. Separate standards for each model dependent component could be developed for each model. This approach is similar to the alternate approach for supporting multiple data models.

3. Separate architectures could be defined for each model. There would be no architectural relationship between standards developed for different data models. This approach is not compatible with the current ANSI/SPARC DBS-SG charter.

Approach three might be the 'easiest' to implement because it requires no coordination between committees working on standards for different data models. Note, however, that there would be no uniformity between the resultant standards. Approach one, on the other hand, would be the most difficult to define but would result in the most uniformity between standards for different models. Approach one is perhaps the theoretical ultimate goal of the DBS-SG reference model. Approach two allows for more flexibility and would serve as a more practical reference model to support today's technology.

4.3.4 Granularity of Building Blocks in the Architecture

Software components and their interfaces are the key constituents of an architecture framework. The issue addressed in this section is the granularity or size of the components. For example, is a DBMS considered to be one large software component, or should some grouping of the functions supported by the DBMS be identified? Identifying the components and their interfaces of a DBMS produces an architecture. The interfaces between components will be called internal. DBMS users deal only with external interfaces.

Three approaches to partitioning the functions supported by a DBMS into components are:

1. Components could be defined along the lines of external interfaces. For example, one component for a DBA interface could support all functions (e.g., creating and maintaining a database). Alternately, DBA functions could be supported by more than one component (e.g., two: one for logical database definition and one for physical database definition). These components and their external interfaces constitute the architecture.

2. One component could be defined for each function supported by a DBMS. In this approach each function that is identified would result in a separate component in the architectural framework. Separate internal components that perform a function for other components (and not directly for end users) could also be identified. For example, file read and file write components could be identified in the architecture.

3. One component could be defined for a class of external or internal interfaces. This approach differs from the first approach because internal components and their interface are also identified in the architecture. An internal interface could be used by multiple components that provide different external interfaces. For example, an internal file access component could be used by components interfacing to DBAs and by components interfacing to end users. This approach differs from the second approach because logically related functions are grouped into one component.

The first partitioning, based on external interfaces only, is normally considered in DBMS standardization activities. The second approach is useful for identifying DBMS functions and understanding their interrelationships in a DBMS architecture. The third approach modularizes those functions into potentially standardizable components that include internal interfaces. The standardization of internal interfaces may become increasingly important with the development of database machines and distributed databases and would allow for the integration of separate DBMS products.

4.4 Internal Architecture of Relational Systems

The internal architecture of two of the better known relational DBMSs, INGRES and the System R are reviewed and the commonality and differences in the architectures are discussed. INGRES and System R were considered due to the availability of published documentation of their architectures.

4.4.1 The INGRES Architecture

The INGRES architecture can be broken into six logical components as shown in Figure 4.1 [STON76, STON80]. The user interface is responsible for interacting directly with the end user and could be

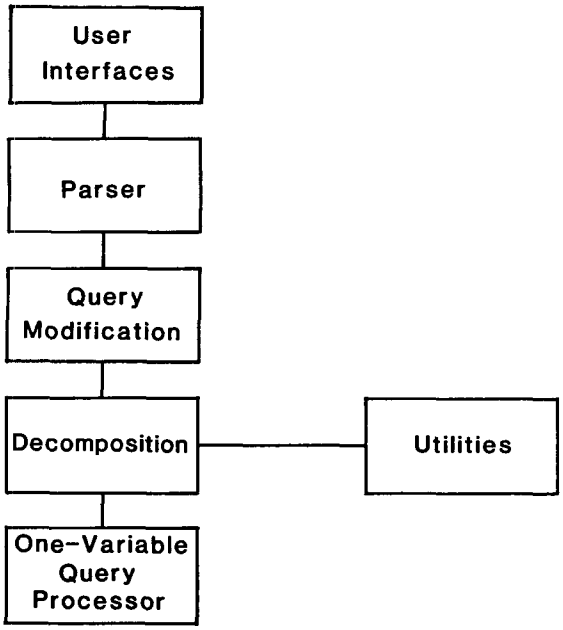

Figure 4.1 The INGRES Architecture

the INGRES terminal monitor, an Equal program, or a CUPID interface. Each of those interfaces sends commands to the parser.

The parser is responsible for syntax checking, the binding of relation and attribute names, and the construction of the parse tree. Note that since relation and attribute names are stored as part of the user database, and the parser must access this information; the parser includes the database access modules.

The Query Modification component modifies the parse tree as required in order to incorporate a view mechanism, integrity constraints, and protection constraints. The query modification component also establishes user identification, date, and time of execution and physical terminal identification. Note that access to the view, integrity, and protection definitions also requires access to the database.

The Decomposition component of the INGRES architecture is responsible for the concurrency control, query optimization, and executes the joining operation. This component invokes the needed utilities to create, sort, or destroy temporary or permanent relations as required for the execution of the query. Note that the decomposition component must read the schema data to determine size characteristics, and must read the database to obtain values for performing a join operation.

The one variable query processor is responsible for access path selection on one relation, retrieval of data from that relation, application of a predicate, updating a relation, and returning data values to the user interface.

The utilities modify the schema, reorganize the access paths on relations, provide logical bulk load and unloading, and provide some data dictionary facilities. The utilities thus read and write both the schema and user data.

4.4.2 The System R Architecture

The System R architecture is generally described as consisting of three components as shown in Figure 4.2 [ASTR76], [BLAS79], [CHAM81]. The user interface could be a general purpose query language, SQL embedded in a PL/1 program or other user interfaces.

The Relational Data System (RDS) is responsible for parsing, authorization, integrity constraints, catalogue management, and optimization. Note that the parsing, authorization and integrity constraints require access to the schema. Optimization requires access to the physical schema. Catalog management requires the retrieval and update of the system data which is also stored in relations. The Relational Storage System (RSS) provides a "record-at-a-time" interface for the relational data system. RSS is responsible for searching indexes, retrieving and storing relation occurrences in the database, concurrency control and recovery. RSS supports both clustered and non-clustered indexes and links between entity types. However, RSS makes no decisions about which access paths to select, but instead is explicitly directed by higher level directives. Note that RSS is thus different than the INGRES one-variable query processor.

A major difference between the System R and INGRES architectures is that System R compiles SQL commands that are embedded in a PL/1 program. These commands are compiled into specific access modules which are stored in the database. As shown

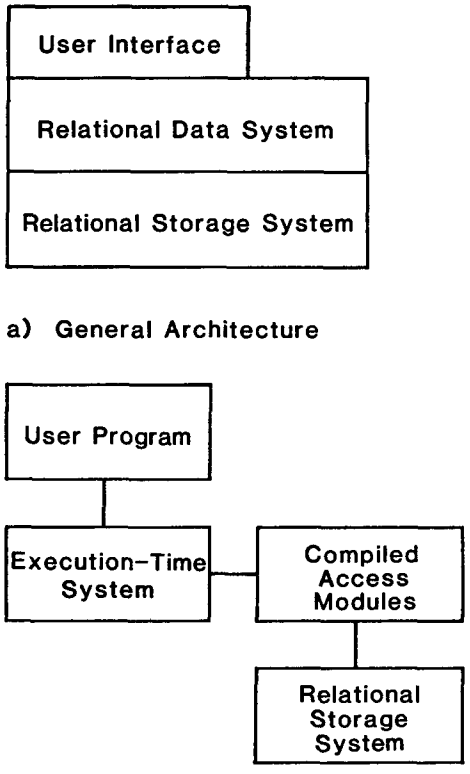

Figure 4.2 The System R Architecture

in Figure 4.2, a user program then makes calls on a special execution time interface which retrieves the access modules from the database. Those access modules make direct calls on RSS.

4.4.3 Observations

Several observations are in order regarding these architectures and the uses of these systems.

1. Nonprocedural set oriented relational interfaces make a wide variety of end-user interfaces feasible. System R can support ad hoc queries and PL/1 programs through the Relational Data Interface to the Relational Data System. INGRES provides similar support through its parser. That INGRES interface has been used to implement a diverse set of End User interfaces such as CUPID and Geoquel at the University of California at Berkeley, a menu selection, retrieval and reporting system at Lawrence Livermore Laboratory and a graphics oriented database browsing system (SDMS) at the Computer Corporation of America.

2. Existing low level DBMSs do not provide sufficient physical schema and actual database size information to support the optimization of a high level non-procedural interface. RDS of System R and the Parser, Decomposition, etc., of INGRES requires extensive knowledge of the physical and logical schemas and various instance specific cardinalities of the user data in order to optimize user queries. In addition, both systems require the ability to modify the schema internally. Most Codasyl systems, IBM's IMS, or Cincom's Total, in general do not provide the needed physical schema knowledge at runtime and do not allow the dynamic updating of such schemas. The record at a time interface that INGRES and System R use are tailored to support their specific implementations.

3. Existing DBMS interfaces to programming languages may not provide sufficient logical schema information to support the general purpose interpretive interface typically associated with "relational" systems. For example, consider the problem of supporting a user interface which simply accepts a record type and item(s) name(s) and performs a selection on the record type or simply prints all of the record occurrences. The needed logical schema information on the existence, type, and size of each of the named objects cannot be readily obtained from most CODASYL schemas. Furthermore, the CODASYL COBOL DML used to retrieve the record occurrences could not be readily compiled (or precompiled) at runtime. Some CODASYL and other low level implementations may provide the schema information through subroutine calls and may also allow database navigation through subroutine calls. Such calls, however, are not part of the CODASYL standard and are, in fact, implementation specific. Thus while a vendor might extend its system to support the relational model, a third party vendor or user group could not.

4. Beneath the Parser interface of INGRES and the Relational Data Interface of System R, the architecture of the systems vary widely. Both systems depend on direct knowledge of their schemas, i.e., they do not use a "purely" relational interface to the schema data. Perhaps one of the most dramatic differences between the system architectures is that System R compiles its programming language embedded SQL commands while INGRES is fully interpretive. The significance of this discussion to architecture is that no internal components of the system could be easily interchanged.

4.5 Candidate RDBMS Architecture

The architecture shown in Figure 4.3 illustrates the components and interfaces of a Relational DBMS which may be suitable for standardization. Those components and interfaces are discussed in Section 4.5.1. That architecture is compared to the CCA architecture in Section 4.5.2 and to the ANSI/SPARC DBMS framework in Section 4.5.3.

4.5.1 A Proposed Architecture

In Figure 4.3, an architecture or grouping of interfaces are presented. Each of the components is discussed below. Note that the lists of interfaces for the various components are for illustrative purposes only. It is not claimed that the lists are complete or that all of the elements of the list are necessary. The Class 1 DBA interfaces are those that deal with a database in its entirety. These could include:

1. The creation of a database
2. The destruction of a database
3. Physical Database Dump, Restore

There is nothing particularly relational about these functions. They probably should not be part of a relational standard.

The Class 2 DBA functions are those that define, destroy, or modify parts of a schema. These functions include:

1. Relation definition, destruction or modification
2. View definition, destruction or modification
3. Access paths for or between relations

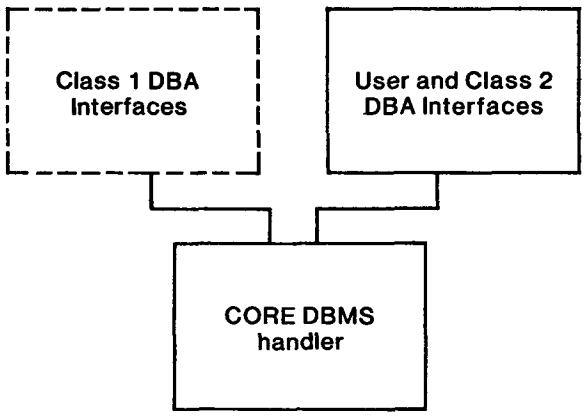

Figure 4.3 A RDBMS Architecture

4. Integrity constraints
5. Security constraints

Since these functions are model dependent, they should be addressed in a Relational DBMS standard.

The User interfaces that could be addressed in a Relational DBMS Standard include:

1. User Query Languages (Retrieval)
2. Set-oriented Update Languages
3. Logical Bulk Load and Unload of Relations
4. Report Generator
5. User Query Language applied to schema data

Note that these interfaces are model dependent because the operations they specify are on the objects in the model. Also note that these interfaces are prevalent in existing relational systems. Other interfaces, such as specific programming language interfaces, perhaps should not be considered for standardization since the individual implementations differ widely.

Class 2 DBA functions are combined with the user interfaces because in existing relational systems the query language is used in the definition of views, integrity constraints, and view constraints. It would seem appropriate, therefore, that if an algebraic (calculus, etc.) query language were standardized, that same standard should be used in the DBA functions.

The core database handler could also be standardized. The core database handler would perform the functions of mapping logical access to physical access for queries and updates, view mapping, integrity control, security control, optimization, and schema management. Note that this interface is an "internal" interface and as such its standardization should be in the form of subroutine or procedure calls. Note also that the retrieval and update interfaces to the database schema are also standardizable. In comparison to the INGRES architecture, this interface would be at a level similar to the query modification component. In comparison to the System R architecture, this interface would be a level inside the RDS, immediately following the parser.

There are two major purposes in standardizing the core database handler internal interface. First, a standard interface at this level could be used to uniformly define the functionality of a Relational DBMS without dependence on a particular query language interface. Multiple user interfaces based on different query languages could be separately standardized (and implemented) on top of the CORE database handler interface.

The second purpose in standardizing the CORE database handler interface is to provide extensibility for innovative and nonstandard interfaces. The availability of such a high level interface in INGRES has resulted in a wide variety of graphics interfaces, programming language interfaces, and menu selection forms interfaces being built "on top of" INGRES. By standardizing the internal interface, these special purpose user interfaces could be designed to be marketed for standard relational DBMSs from a variety of vendors. As previously mentioned, it is the lack of a standard internal interface and inaccessibility of the schema data through such an interface that has hindered users from developing higher level interfaces on top of the existing CODASYL systems.

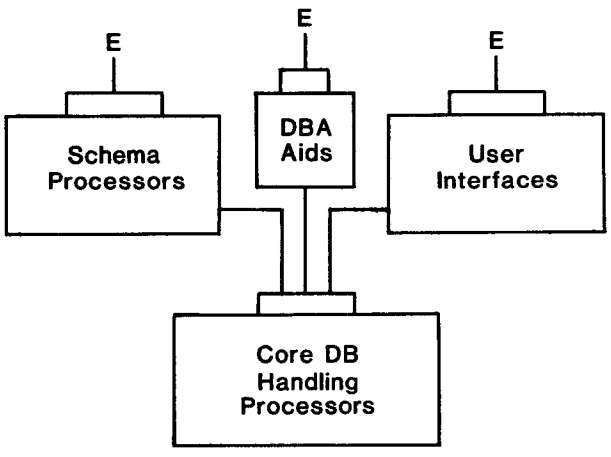

Figure 4.4 The CCA Strawman Architecture

4.5.2 The CCA Strawman Architecture

The CCA Strawman Architecture consists of 38 components which can be grouped into four major blocks as shown in Figure 4.4. This architecture has influenced the design of the proposed RDBMS architecture and in this section we review the differences.

The Strawman Architecture separates schema processors, DBA aids, and user interfaces into separate logical groupings. In the RDBMS architecture, the schema processors are combined in the same grouping as the user interfaces in order to reflect the fact that the user query languages in many relational systems are used in certain types of schema definitions.

The RDBMS architecture includes a subset of the user interfaces described in the strawman architecture. The subset represents those interfaces common to many of the relational systems analyzed. Most notably lacking are the programming languages' interfaces.

The Relational DBMS architecture does not include DBA aids such as physical and logical database design tools as defined in the strawman architecture mainly because no such relational DBA aids were found in the feature analysis. Such DBA aids are not prevented in DBMSs for relational or other models.

The strawman architecture and the RDBMS architecture both include a Core DB handler for standardizing both the functionality of a given data model and the schema management of the model. The interfaces differ in two important ways. Firstly, the strawman architecture contains a rich set of record-at-a-time operators at the CORE DB handler level. Relational systems that were analyzed supported a much weaker set of record-at-a-time operations and the use of those operations varied widely from system to system. Thus, record-at-a-time operations are not suggested for relational standardization at this time.

The second difference is that the strawman architecture interface does not support special commands for database management. Rather, the schema is to be stored in a database of its own and the usual data manipulation operators are to be used for both retrieval and updating of the schema

The ANSI/SPARC external data manipulation language interfaces (8, 9, 10) correspond to the report generator, query language, and update specifics of the RDBMS architecture respectively. Note that the ANSI/SPARC framework does not suggest a logical bulk load and unload utility for end users but instead includes it as part of an internal database utility (17).

Some of the ANSI/SPARC internal interfaces (2, 5, 14, and 12) specify object format interfaces and correspond to the internal RDBMS interface to the CORE Database handler.

The remaining ANSI/SPARC interfaces either are not prevalent or uniform in existing relational systems or are not particularly model dependent.

The two major concepts in the ANSI/SPARC architecture that are not emphasized in the proposed RDBMS architecture are the conceptual model and role definitions for individuals. The conceptual model is not explicitly emphasized in the RDBMS architecture due to the still open issues regarding the conceptual model and that research into conceptual models has not stabilized. The role definitions for individuals is also not explicitly emphasized in the RDBMS architecture. In INGRES, database users can create their own relations, views, and access paths. In System R the users can create those objects and share them with other users. In general, with dynamic schemas, the roles can become blurred and were not considered for standardization.

data. While the RDBMS architecture does not preclude this implementation, it does not suggest it for standardization at this time. Many of the relational systems analyzed supported retrieval from relations containing schema information. However, very few systems would allow updating of these system relations through the normal DML update commands.

The strawman architecture describes additional internal components including a physical database processor and a file access processor. These interfaces were not suggested for a relational DBMS standard since these interfaces vary widely in the analyzed systems and may not be particularly relational in nature.

4.5.3 The ANSI/SPARC Architecture

The ANSI/SPARC architecture specifies 42 interfaces. The proposed RDBMS architecture suggests standardization of some of those interfaces and introduces other interfaces not in the ANSI/SPARC architecture. The source schema interfaces (numbers 1, 4, 13 in the ANSI/SPARC architecture) are identical to the RDBMS interfaces in the DBA Class 2 grouping. The conceptual interface (1) could correspond to the relation schema and integrity constraints interfaces. The external interface (2) could correspond to the view schema interfaces. The internal schema interface (13) could correspond to the physical schema interfaces.

5. DEVELOPING A RELATIONAL STANDARD

The RTG has concluded that the development of a relational standard is justifiable on a number of grounds. Section 1 of this report discusses three motives for a relational standard, namely: 1) the appropriate time for introducing a standard; 2) the desire for an alternative approach to data definition, manipulation, querying, and integrity; and 3) the inherent benefits of the relational approach. Section 2 illustates the adequacy of relational theory, i.e., the soundness of the RDM. Section 3 gives a detailed demonstration of the practicality of relational technology and describes the features of important research and commercially available RDBMSs. Finally, Appendix C indicates the strength of interest in the DBMS marketplace for RDBMSs by listing over 75 systems that are said to support significant aspects of the RDM. Although the 75 DBMSs have similarities they do not all provide the same "relational" functionality.

The RTG recommends that a standard be developed for interfaces to RDBMSs. This section presents guidelines, issues, and recommendations for developing such a standard.

5.1 Guidelines and Issues

The RTG's primary objective was to provide clear guidance to a subsequent technical committee while not unnecessarily restricting the possible alternatives. To do this, the RTG attempted to identify issues that arise in developing a relational standard, to determine the possible alternatives, and to outline their consequences when possible. The RTG did not attempt to resolve issues.

The RTG applied the following two simple guidelines in its investigation and strongly recommends their application in the development of the relational standard.

1. Concentrate on RDM concepts and on the RDM per se.
2. Concentrate on the semantics or functionality of RDM concepts and of interfaces to RDBMSs.

These guidelines are elaborated and supported in the remainder of this section.

Following the first guideline, the RTG emphasized RDM concepts and the RDM per se over more general topics such as:

- The role of the RDM and RDBMSs in a DBMS architectural framework such as the ANSI/X3/SPARC prototypical architecture.
- The role of a relational standard in a coherent family of DBMS standards.
- The relationship of RDM and RDMBS concepts and terms with general data model and DBMS concepts and terms.

However, the RTG identified issues concerning these topics whenever it was possible.

The general topics received less emphasis for several reasons. First, the general topics are all open research problems, hence currently inappropriate for standardization. Second, architectural issues are, for the most part, independent of data models. The RDM can be implemented following many DBMS architectural frameworks and one such framework can be used to implement different data models (see Section 4). Third, dealing with more general topics may result in an overlap and possible incompatibilities with the work of other standards projects (e.g., ANSI/X3H2, ANSI/X3H4, DBSSG task groups addressing DBMS architectural, concepts and terminology, etc.). Finally, the RDM and its concepts were emphasized since a DBMS standard should not be based solely on the current state of DBMS technology in this rapidly changing field. There should be some emphasis on the conceptual basis for the standard, the concepts that shape the common understanding of the RDM. A standard with such a basis will accommodate the evolution of both the technology and the standard itself.

Following the second guideline, the semantics or functionality of RDM concepts and of interfaces to RDBMSs were emphasized over topics such as:

- The grouping of functions (e.g., for data definition, manipulation, query, and integrity control) into one or more languages.
- One or more of the many possible syntactic variations of languages that support RDM concepts.
- The embedding of RDM data definition, manipulation, and integrity functions in host languages.

Again, whenever it was reasonable to do so, issues concerning these topics were identified and discussed.

RDM semantics were emphasized over languages and syntactic issues for a number of reasons. First, the RDM provides a basis for a wide range of high level languages [CODD70] [PIRO79], a number of which are equally important to different classes of users. These languages may be self-contained or embedded in host languages. They may provide different representations of relations (e.g., as sets, arrays, predicates, or functions) and different forms of operators (e.g., based on relational algebra, tuple-oriented or domain-oriented relational calculus). Second, based on a RDM semantics specification, languages can be designed (and possibly standardized) for particular user or application needs. Since the languages are based on the same RDM semantics they can be readily, perhaps automatically, translated from one to the other. Third, the choice of languages and their relationship is an architectural issue which, following the first guideline, is of less consequence to the investigation.

Standardizing syntax without an adequate semantic definition would not produce an adequate standard. Currently, the SQL syntax [ASTR76] is provided by four systems: SYSTEM/R, ORACLE, MRS, and MRDS. Each system implements different semantics. For example, each imposes different restrictions on the join operation.

The nature and development proposed for a relational standard should be considered in light of the CODASYL proposal, the only comparable DBMS standards development project. The CODASYL development began over 15 years ago. At that time database management system concepts were not widely known or developed. Few DBMSs existed; hence there was only limited user or application experience. The CODASYL approach was based on one such system, Bachman's IDS. Currently, the relational approach is widely understood. Over 70 RDBMSs have been developed but are not yet in widespread use.

Three important aspects of the CODASYL development are in marked contrast to the two guidelines proposed for the development of the relational standard. First, the DBTG committee of

CODASYL defined a specific syntax using a metalanguage and gave the semantics in English prose. Second, from 1971 to 1981 the data definition and data manipulation languages (DDL and DML) were developed independently by separate committees. There is no high level query language. Third, although several DMLs were considered (both self-contained and host-language embedded), only the COBOL embedded version has been developed. The recent development of a FORTRAN embedded DML has posed consistency problems with respect to the COBOL DML. The two guidelines are proposed for the relational standard to avoid these and other problems in developing a relational standard and subsequent interfaces to RDBMSs.

Issues concerning the development of a relational standard can be classified into those concerning RDM concepts and the RDM itself, and those concerning RDBMSs. RDM issues are discussed in Sections 2.1 and 2.3. This section concludes by outlining several RDBMS issues.

First there is the issue of the possible spectrum of support of the RDM by a DBMS. The spectrum is based on DBMS features, on the supported RDM concepts, and on the restrictions placed on them. Two points on the spectrum are of interest for a relational standard: one indicating fully relational, the other indicating nonrelational. The spectrum between the two indicates semirelational systems or perhaps those languages or DBMSs that conform to the standard. How are these points to be defined? Fully relational could be defined as support of all core RDM concepts (see Chapter 2) without restriction. Nonrelational may include concepts that conflict with or preclude some core RDM concepts. If so, what restricted support would still be considered in conformance with the standard? Could a DBMS be considered relational if it supported relational manipulation or query functions but not data description functions? These issues remain unresolved.

A DBMS that places restrictions on the join operation may be considered semirelational or even nonrelational. In a fully relational system a join is permissible between any two relations based on union compatibility of the attributes on which the join is based. As a consequence, a relation can be joined with itself and relations can be joined in several ways if there is more than one pair of compatible attributes.

Second, there are issues concerning the embedding of sublanguages in a host language: Are RDM concepts supported directly in the host language or are they mediated with host language concepts through workspaces? How are general computation operations (e.g., arithmetic) integrated with relational operations? Other issues include: Are the operators set-oriented or tuple-oriented? What are the restrictions placed on defining, accessing, and manipulating derived relations? Are static or dynamic derived relations supported? What are the mechanisms for handling constraint violations? Are nonrelational DBMS features such as concurrency or security visible in the language? Finally, is the language relationally complete without relying on iteration and recursion constructs in the language? Some of these issues and others were considered in Chapters 2 and 3.

Third, there are architectural issues that should also be considered but excluded from a relational standard. Architectural issues were discussed with the first guideline. There are also basic questions such as: What is an architectural framework? What are the alternatives? There is the issue of mappings between relational schemas and between a relational schema and a nonrelational schema. There is the question of a conceptual data model. These and other issues are addressed in Chapter 4.

The RTG did not consider in detail capabilities that are not directly related to the RDM, e.g., general computational operations, privacy control, concurrent update, rollback, and recovery. The RTG does not regard standards activities in such areas as necessarily tied to a data model.

5.2 Recommendations for a Relational Standard

1. A Relational Standard is justifiable because of the:
   - Stability of the relational approach
   - Marketplace and educational interest: 70 systems, considerable user interest

2. An initial Relational Standard should concentrate on:
   - RDM concepts and the RDM *per se*
   - The functionality or semantics of RDM concepts and of interfaces to RDBMSs

3. An initial Relational Standard should exclude:
   - Architectural aspects
   - Specific language design issues, e.g., syntax
   - Aspects likely to be treated by other standards, e.g., database dictionary
   - Non-data model aspects of DBMSs, e.g., concurrency, recovery, distribution, security, physical storage aspects

Examples of functions that might define a Relational Standard Interface are given in Appendix A.

6. REFERENCES

We distinguish between two kinds of documents:

1. Relational Bibliography, i.e., partial list of publications of importance to the relational approach.

2. Reference Papers, i.e., additional documents referenced in this report.

6.1 Relational Bibliography

[AMBLE79]
Amble, T. "ASTRAL: A Structured and Unified Approach to Data Base Design and Manipulation". IFIP TC-2 Working Conf. on Data Base Architecture, June 1979.

[ASTR80a]
Astrahan, M.M., W. Kim and M. Schkolnick. "Evaluation of System R Access Path Selection Mechanism". Research Report RJ2797, IBM Research Laboratory, San Jose, California, April 1980.

[ASTR80b]
Astrahan, M.M. et al. "A History and Evaluation of System R". Research Report RJ2843, June 12, 1980.

[ASTR76]
Astrahan, M.M. et al. "System R: Relational Approach to Database Management". ACM TODS, Vol. 1, No. 20, June 1976.

[BJOR73]
Bjørner, D. et al. "The Gamma-0 n-ary Relational Data Base Interface Specifications of Objects and Operations". Research Report RJ1300, IBM Research Laboratory, San Jose, California, April 1973.

[BLAS79]
Blasgen M.W. et al. "System R: An Architectural Update". Research Report RJ2581, July 1979.

[BLAS77]
Blasgen, M.W. and K.P. Eswaran. "Storage and Access in Relational Data Bases". IBM Systems Jour., Vol. 16, No. 4, 1977.

[BOYCE73]
Boyce, R.F. and D.D. Chamberlin. "Using a Structured English Query Language as a Data Definition Facility". Research Report RJ1318, December 1973.

[BRAT80]
Bratbergsengen, K. et al. "A Neighbor Connected Processor Network for Performing Relational Algebra Operations". 5th Workshop on Computer Architecture for Non-Numeric Processing, March 1980.

[BRAT79]
Bratbergsengen, K. and R. Larsen. "Design of a Relational Data Base System". Computer Manufacturing International Inc. Data Base Seminar, March 1979.

[BROD80b]
Brodie, M.L. Data Abstraction, Database and Conceptual Modeling: An Annotated Bibliography. NBS Special Pub. 500-59, May 1980.

[BROD80d]
Brodie M.L. and J.W. Schmidt. "Issues in Investigating a Standard for the Relational Approach to Databases". Proc. NBS Conf. of DBMS Standards, Gaithersburg, MD, September 1980.

[BUNE79]
Buneman, P. and R.E. Frankel. "FQL -- A Functional Query Language". Proc. 1979 SIGMOD, Boston, May 1979.

[CHAM81]
Chamberlin et al. "Support for Repetitive Transactions and Ad Hoc Queries in System R". ACM TODS, Vol. 6., No. 1, March 1981.

[CHAM79]
Chamberlin, D.D. et al. "Support for Repetitive Transactions and Ad-Hoc Query in System R". Research Report RJ2551, May 1979.

[CHAM77]
Chamberlin, D.D. et al. "Data Base System Authorization". Research Report RJ2041, July 1977.

[CHAM76a]
Chamberlin, D.D. "Relational Data-Base Management Systems". Computing Surveys, Vol. 8, No. 1, March 1976.

[CHAM76b]
Chamberlin, D.D. et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control". IBM Jour. of Research And Development, November 1976, pp. 560-575.

[CHAM75]
Chamberlin, D.D., J.N. Gray and I.L. Traiger. "Views, Authorization and Locking in a Relational Data Base System". Proc. 1975 AFIPS National Computer Conf., pp. 425-430.

[CHANG79]
Chang, C.L. "On Evaluation of Queries Containing Derived Relations in a Relational Data Base". Research Report RJ2667, IBM Research Laboratory, San Jose, California, October 1979.

[CHILDS77]
Childs, D.L. "Extended Set Theory". Proc. Third Int'l. Conf. on Very Large Data Bases, Tokoyo, October 1977.

[CODD80]
Codd, E.F. "Data Models in Database Management". in Brodie, M.L. and S.N. Zilles (Eds.) Proc. Workshop on Data Abstraction, Databases and Conceptual Modeling, to appear, 1980.

[CODD79a]
Codd, E.F. "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks". Research Report RJ2599, IBM Research Laboratory, San Jose, California, August 1979.

[CODD74]
Codd, E.F. "Recent Investigations in Relational Data Base Systems". Information Processing '74, Amsterdam, Holland.

[DATE80]
Date, C.J. An Introduction to the Unified Database Language (UDL). Proc. 6th Conf. on Very Large Databases, October, 1980, Montreal, Canada.

[DATE79a]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) COBOL Version. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79b]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) COBOL Version, Relational Subset. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79c]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) PL/I Version. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DATE79d]
Date, C.J. An Architecture for HIGH LEVEL Database Extensions: (UDL) PL/I Version, Relational Subset. IBM Santa Teresa Laboratory, San Jose, California, April 1979.

[DAYAL78]
Dayal, U. and P.A. Bernstein. "On the Updatability of Relational Views". IEEE, June 1978.

[DRIS78]
Driscoll, J.R., B.A. Dutton and K.C. Kinsley. "A Relational Storage Scheme Suitable for Derived Views". Proc. of the 1978 ACM Annual Conf. (ACM '78).

[EPST80a]
Epstein, R. and P. Hawthorn. "Aids in the '80s". DATAMATION, February 1980.

[EPST80b]
Epstein, R. and P. Hawthorn. "Design Decisions for the Intelligent Database Machine". Proc. of the 1980 NCC.

[FAGIN79]
Fagin, R. "Normal Forms and Relational Database Operators". Research Report RJ2471, IBM Research Laboratory, San Jose, California, February 1979.

[FURT78]
Furtado, A.L. "Formal Aspects of the Relational Model". Information Systems 3, 2 (1978), pp. 131-140.

[FRY76]
Fry, J.P. and E.H. Sibley. "Evolution of Database Management Systems". Computing Surveys, Vol. 8, No. 1, March 1976.

[HALL75a]
Hall, P.A.V. "Optimization of a Single Relational Expression in a Relational Data Base System". Research Report UKSC 76, June 1975.

[HALL75b]
Hall, P.A.V., P. Hitchcock and S.P.J. Todd. "An Algebra of Relations for Machine Computation". Conf. Record of 2nd ACM Symposium on Principles of Programming Languages, Palo Alto, California, January 1975.

[HANS76]
Hansel, A. "A Formal Definition of a Relational Database System". Research Report UKSC 80, 1976.

[HARD78]
Hardgrave, W.T. "The Relational Model: A Reformulation of some Mathematical Aspects Using Positional Set Notation". IFSM T.R. NO. 25, Dept. of Information Systems Management, University of Maryland, March 1978.

[HEATH71]
Heath, I.J. "Unacceptable File Operations in a Relational Data Base". ACM SIGFIDET Workshop, 1971, pp. 19-33.

[HELD75]
Held, C.D., M. Stonebraker and E. Wong. "INGRES: A Relational Database System". Proc. ACM Pacific 75 Regional Conf., pp. 409-416.

[HITC76]
Hitchcock, P. "User Extensions to the Peterlee Test Vehicle". Systems for Large Data Bases. Amsterdam, Holland, 1976, pp. 169-180.

[HOUS79]
Housel, B.C. "QUEST: A HIGH LEVEL Data Manipulation Language for Network, Hierarchical, and Relational Databases". Research Report RJ2588, July 1979.

[JARD79]
Jardine, C. and J. Owlett. "Applying a Relational Database System to Historical Data". IBM UK Technical Note 74, 1979.

[JOHN79a]
Johnson, H.R., J.A. Larson and J.D. Lawrence. Data Description Language for Network and Relational Modelling. Sperry Univac, Roseville, Minnesota, 1979.

[JOHN79b]
Johnson, H.R., J.A. Larson and J.D. Lawrence. "Network and Relational Modelling in a Common Data Base Architecture Environment". Research Report TMA00720, Sperry Univac, Roseville, Minnesota, March 1979.

[JONES78]
Jones, S.E. and D.R. Ries. A Relational Data Base Management System for Scientific Data. Lawrence Livermore Laboratory, February 1978.

[KIM79]
Kim, W. "Relational Database Systems". ACM Computing Surveys, Vol. 11, No. 3, September 1979.

[LACR76]
Lacroix, M. and A. Pirotte. "Generalized Joins". ACM SIGMOD Record, Vol. 8, No. 3, 1976, pp. 14-15.

[LACR77a]
Lacroix, M. and A. Pirotte. "ILL: An English Structured Query Language for Relational Databases". Proc. IFIP TC-2 Working Conference on Modeling in Database Management Systems, Nice, France, January 1977, Nijssen (Ed.), North-Holland (1977), pp. 23-260.

[LACR78]
Lacroix, M. and A. Pirotte. Example Queries in Relational Languages. Tech. Note 107, MBLE Research Laboratory, Brussels, Belgium, revised April 1978 (see also Research Reports 351 and 367).

[LACR80]
Lacroix, M. and A. Pirotte. "User interfaces for Database Application Programming". Proc. 74th Infotech State of the Art Conference on Databases, London, October 1980.

[LACR81]
Lacroix, M. and A. Pirotte. "Associating Types with Domains of Relational Databases". Proc. NBS-ACM Workshop on Data Abstraction, Databases, and Conceptual Modeling, Pingree Park, Colorado, June 1980, in: SIGART Newsletter 74 (January 1981), SIGMOD Record, Vol. 11, No. 2 (February 1981), SIGPLAN Notices, Vol. 16, No. 1 (January 1981), pp. 144-146.

[LAME80]
Lamersdorf, W. and J.W. Schmidt. "Semantic Definition of PASCAL/R". Berichte Nr. 73 and 74, University of Hamburg, Federal Republic of West Germany, July 1980.

[LORIE79]
Lorie, R.A., R. Casajuana and J.L. Becerril. "GSYSR: A Relational Database Interface for Graphics". Research Report RJ2511, April 1979.

[McDO75]
McDonald, N. and M. Stonebraker. "CUPID: The Friendly Query Language". Proc. ACM Pacific 75 Regional Conf., April 1975.

[MYLO75]
Mylopoulos, J., S. Schuster and D. Tsichritzis. "A Multi-Level Relational System". Proc. 1975 AFIPS National Computer Conf.

[OSBO79]
Osborn, Sylvia L. "Towards a Universal Relational Interface". IEEE, August 1979, pp. 52-60.

[OWENS71]
Owens, R.C. "Evaluation of Access Authorization Characteristics of Derived Data Sets". Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, 1971, pp. 263-278.

[OWLE77]
Owlett, J. "Deferring and Defining in Databases". A section of: "Architecture and Models in Data Base Management Systems". Proc. of the IFIP Working Conf. on Modelling in Data Base Management Systems, November 1977.

[PIRO79]
Pirotte, A. "Fundamental and Secondary Issues in the Design of Non-Procedural Languages". Proc. 5th VLDB, Rio de Janiero, Brazil, August 1979, pp. 239-250.

[PIRO78]
Pirotte, A. "Linguistic Aspects of High Level Relational Languages". Infotech State of the Art Report on Data Bases Technology, Vol. 2, pp. 271-300, 1978.

[PIRO77a]
Pirotte, A. "The Entity-Property-Association Model: An Information-Oriented Data Base Model". Proc. ACM International Computing Symposium (ICS77), North-Holland (1977), pp. 581-597.

[PIRO80]
Pirotte, A. "High Level Database Query Languages". in "Logic and Databases", Gallaire and Minker (Eds.), Plenum (1978), pp. 409-435.

[SCHM80a]
Schmidt, J.W. and M. Mall. "PASCAL/R Report". Bericht Nr. 66, University of Hamburg, Federal Republic of Germany, January 1980.

[SCHM79]
Schmidt, J.W. "Parallel Processing of Relations: A Single-Assignment Approach". VLDB5, Rio de Janeiro, 1975.

[SCHM77]
Schmidt, J.W. "Some High Level Language Constructs for Data of Type Relation". ACM TODS, Vol. 2, No. 3, September 1977.

[SELI79]
Selinger. P.G. et al. "Access Path Selection in a Relational Database System". Research Report RJ 2429, January 1979.

[SHIP81]
Shipman, D. "The Functional Data Model and the Data Language DAPLEX". ACM TODS, Vol. 6, No. 1, March 1981.

[SMITH77]
Smith, J.M. and D.C.P. Smith. "Database Abstraction: Aggregation and Generalization". ACM TODS 2, 2 (June 1977).

[STON80b]
Stonebraker, M. "Retrospection on a Database System". ACM TODS, Vol. 5, No. 2, June 1980.

[STON78]
Stonebraker, M. "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES". ERL Technical Memorandum Reprint 1702, University of California, August 1978.

[STON76]
Stonebraker, M. et al. "The Design and Implementation of INGRES". ERL Technical Memorandum Reprint 1468, University of California, September 1976.

[STON75]
Stonebraker, M. "Implementation of Integrity Constraints and Views by Query Modification". Proc. Int'l. Conf. on Management of Data, 1975, pp. 65-78.

[STRN71]
Strand, A.J. "The Relational Approach to the Management of Data Bases". Proc. of IFIP 1971 Congress, 1971, pp. 901-904.

[TIBU76]
Tibuya, S. "Practice of Noun Phrase Model". Proc. of IBM Int'l. Conf. in Relational Database, Bari, 1976.

[TODD77a]
Todd, S.P.J. "Automatic Constraint Maintenance and Updating Defined Relations". Information Processing '77, IFIP Congress Series, pp. 145-148.

[TODD77b]
Todd, S.P.J. "Database Research at the IBM UK Scientific Centre, Peterlee: A Survey". Research Report UKSC 93, 1977.

[TODD76]
Todd, S.P.J. "The Peterlee Relational Test Vehicle - A System Overview". IBM Systems Jour., No. 4, 1976.

[TODD74]
Todd, S.P.J. "Implementation of the Join Operator in Relational Data Bases". IEEE/IERE Colloquium on Information Structure and Storage Organisation. London, April 1974.

[TSIC78]
Tsichritzis, D. and Klug, A. "The ANSI/X3/SPARC DBMS Framework". Report of the Study Group on a Database Management System". Information Systems, Vol. 3, No. 4, 1978.

[VERH79]
Verhofstad, J.S.M. "An Evaluation of the PRTV Optimiser". Research Report UKSC 91, preparation.

[VERH76]
Verhofstad, J.S.M. "The PRTV Optimiser: The Current State". Research Report UKSC 83, May 1976.

[WONG77]
Wong, H.K.T. and J. Mylopoulos. "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management". INFOR, Vol. 15, No. 3, October 1977.

[ZLOOF78a]
Zloof, M.M. "Query-By-Example: Operations on the Transitive Closure". IBM Research Report RC5526, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, February 7, 1978.

[ZLOOF78b]
Zloof, M.M. "Security and Integrity Within the Query-By-Example Data Base Management Language". IBM Research Report RC6982, February 7, 1978.

[ZLOOF77]
Zloof, M.M. "Query-By-Example: A Data Base Language". IBM Systems Jour., No. 4, 1977.

[ZLOOF76]
Zloof, M.M. "Query-By-Example: Operation on Hierarchical Data Bases". AFIPS Conf. Proc., NCC 45, 1976, pp. 845-853.

[ZLOOF75a]
Zloof, M.M. "Query-By-Example". AFIPS Conf. Proc., Vol. 44, AFIPS Press, Montvale, N.J., 1975.

[ZLOOF75b]
Zloof, M.M. "Query-By-Example: The Invocation and Definition of Tables and Forms". Proc. of the Int'l. Conf. on Very Large Data Bases. Boston, Massachusetts, September 22-24, 1975, pp. 1-24.

6.2 Reference Papers

[BEER78]
Beeri, C., P. Bernstein, and N. Goodman. "A Sophisticate's Introduction to Database Normalization Theory". VLDB, Berlin, 1978.

[BEKIC79]
Bekic, H., D. Bjørner, W. Henhapl and C.B. Jones. "A Formal Definition of PL/1 Subset, Parts I and II". Tech. Report TR25.139, IBM Laboratory, Vienna, December 1979.

[BJOR78]
Bjørner, D. and C.B. Jones. The Vienna Development Method: The Meta Language. Springer-Verlag, New York, 1978.

[BJOR80]
Bjørner, D. "Formalization of Database Models". in "Abstract Software Specifications", Springer Lecture Notes in Computer Science 86, 1980.

[BROD80a]
Brodie, M.L. "A Functional Framework for Database Management Systems". TR-78, Dept. of Computer Science, University of Maryland, February 1980.

[BROD80c]
Brodie, M.L. and J.W. Schmidt. "Feature Catalogue of Relational Concepts, Languages and Systems." RTG Working Doc. 80-81, May 1980.

[BROD80e]
Brodie, M.L. "Standardization and the Relational Approach to Databases". Proc. 6th Int'l Conf. on Very Large Databases, October 1980, Montreal, Canada.

[CCA80]
Computer Corporation of America. "Overview of NBS Strawman Architecture for Family of DBMS Standards". Presented at NBS Conference, September, 1980.

[CODA73,78]
CODASYL Data Description Language Committee, Journal of Development, 1973, 1978.

[CODA69,71]
CODASYL Database Task Group Reports 1969, 1971.

[CODD70]
Codd, E.F. "A Relational Model of Data for Large Shared Data Banks". Comm. of ACM, Vol. 13, No. 6, June 6, 1970.

[CODD71a]
Codd, E.F. "Relational Completeness of Data Base Sublanguages". Data Base Systems, Courant Computer Science Symposia, Vol. 6, May 1971.

[CODD71b]
Codd, E.F. "A Database Sublanguage Founded on the Relational Calculus". Proc. ACM SIGFIDET Workshop, San Diego, Calif., November 1971.

[CODD72]
Codd, E.F. "Relational Completeness of Database Sublanguages". in Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall (1972).

[CODD79b]
Codd, E.F. "Extending the Relational Data Base Model to Capture More Meaning". ACM TODS, Vol. 4, No. 4, 1979.

[CODD81a]
Codd, E.F. "The Significance of the SQL/Data System Announcement". Computerworld, February 16, 1981.

[CODD81b]
Codd, E.F. "The Capabilities of Relational Database Management Systems". IBM Research Laboratory Report, RJ3132, May 11, 1981.

[DATE81]
Date, C.J. "An Introduction to Database Systems", Third Edition, Addison-Wesley, 1981.

[DELO80]
Delobel, C. "An Overview of the Relational Data Theory". IFIP Congress, 1980.

[DONA76]
Donahue. "Complementary Definitions of Programming Language Semantics". Lecture Note 42, Springer Verlag, New York, 1976.

[EARL71]
Earley, J. "Towards an Understanding of Data Structures". Comm. of ACM 14, 10 (October 1971), pp. 617-627.

[EGLI74]
Egli, H. "Programming Language Semantics Using Extensional Lambda-Calculus Models". TR-206, Cornell University, April 1974.

[GALL78]
Gallaire, H. and J. Minker (Eds.) Logic and Databases. Plenum Press, New York, 1978.

[GOGU77]
Goguen, J.A., J.W. Thatcher and E.G. Wagner. "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types". RC 6487, IBM Yorktown Heights, New York, April 1977.

[GORD79]
Gordon, M.J.C. The Denotational Description of Programming Languages. Springer Verlag, New York, 1979.

[GUTT78]
Guttag, J.V. and J.J. Horning. "The Algebraic Specification of Abstract Data Types". Acta Informatica 10, pp. 27-52 (1978).

[HARD80]
Hardgrave, W.T. "Positional Set Notation". Draft Paper, National Bureau of Standards, February 1980.

[HARD76]
Hardgrave, W.T. "Set Processing: A Tool for Data Management". IFSM T.R. NO. 6, Dept. of Information Systems Management, University of Maryland, April 1976.

[HOARE74]
Hoare, C.A.P. and P.E. Lauer. "Consistent and Complementary Formal Theories of the Semantics of Programming Languages". Acta Informatica 3, pp. 135-153 (1974).

[HOARE73]
Hoare, C.A.P. and N. Wirth. "An Axiomatic Definition of the Programming Language PASCAL". Acta Informatica 2, pp. 335-355 (1973).

[KINS80]
Kinsley, K. and J. Driscoll. "Preliminary Feature Analysis". Working Paper of RTG, 1980.

[LACR77b]
Lacroix, M. and A. Pirotte. "Domain Oriented Relational Languages". Proc. of Third VLDB, October 1977, pp. 370-378.

[LARS79]
Larson, J.A. and T.B. Wilson. Bridging the Gap between Network and Relational Data Bases. Sperry Univac, Roseville, Minnesota.

[LUCAS71]
Lucas, P. "Formal Definition of Programming Languages and Systems". IFIP Congress 71, August 1971.

[MILLS79]
Mills, H.D., R.C. Linger and B.I. Witt. "Structured Programming Theory and Practice". Addison-Wesley, 1979.

[SAND81]
Sandberg, G. "A Primer on Relational Data Base Concepts". IBM Systems Jour., Vol. 21, No. 1, 1981.

[SCOTT72]
Scott, D. "Lattice Theory, Data Types, and Formal Semantics". NYU Symposium on Formal Semantics, Prentice Hall, 1972.

[STON80a]
Stonebraker, M. "A Tale of Three Standards". Proc. 1980 NBS Workshop on a Family of DBMS Standards, NBS Publication, October 1980.

[ULLM80]
Ullman, J.D. "Principles of Database Systems". Computer Science Press, 1980.

APPENDIX A. NATURE OF A RELATIONAL STANDARD INTERFACE

The SD-3 proposes the standardization of RDBMS interfaces. The functions that will constitute the interfaces will support:

1. the definition of structures, domains, and integrity constraints;
2. the execution of query and altering operations over relational databases in accordance with 1); and
3. access to the definition of structures, domains, and constraints.

This appendix describes, briefly, examples of functions that might constitute the interface. Other sets of functions may serve equally well. According to their purpose the functions are divided into three groups:

1. database schema definition -- structure, domain, and constraint definition;
2. database operations -- deriving and altering relation values; and
3. schema access -- deriving information that defines a relational database schema.

The syntax used in the examples is "abstract." Many different forms could be used equally well.

A.1 Interface Functions for Database Definition

This section presents examples of functions that have to be provided in one form or another, by an RDBMS in order to accomplish a relational database schema definition.

Example A1:

The interface function

```
function dfn_domain (domain_id : string ;
                     value_set : {val})
```

defines a domain by a set of values, {val}, and identifies it by a string. The value set may be given extensionally, e.g., by {red, blue, yellow}, or intentionally, e.g., by the key word "integer" or by an interval {1..999}.

Example of function invocation:

```
dfn_domain ("creditstatus", {1..999}).
```

Example A2:

The interface function

```
function dfn_relation_structure (
     rel_structure_id : string ;
     attribute_set : {< attribute_id : string,
                        domain_id : string>} )
```

defines a relation structure by a set of attributes, i.e., by a set of ordered pairs of attribute and domain identifiers. Furthermore, it identifies the structure definition by a string.

Example of function invocation:

```
dfn_relation_structure("supplier_structure",
                       {<"snr", "suppliernumbers">,
                        <"sname", "suppliernames">,
                        <"status", "creditstatus">,
                        <"city", "citynames">}).
```

Example A3:

The interface function

```
function dfn_relation_schema (
                    rel_schema_id : string ;
                    rel_structure_id : string ;
                    key_attribute_set :
                        {attribute_id : string} )
```

defines a relation schema by a structure definition and by a set of attribute identifiers that are to constitute the relation key. Furthermore, it names the schema definition by a string. A more complete specification of this function would require that the key attribute set is a subset of the attribute identifiers specified in the corresponding relation structure definition.

Example of function invocation:

```
dfn_relation_schema ("supplier_schema",
                     "supplier_structure",
                     {"snr"}.
```

Additional functions are required for database schema definition and for the deletion and alteration of definitions.

If the data structures that store a schema definition are defined as a relational database (subsequently called a meta or schema database) the above functions represent specific insert procedures for the schema database.

A.2 Interface Functions for Database Operations

The interface functions for database operations are somewhat more complicated. We discuss some of the problems using a select operation as an example (an extension of Example 5, Section 2.3 of the report): select the suppliers that either are located in Paris but not named Smith, or have a status different from 20.

The selection criterion involved is

((city = "Paris") and (sname ≠ "Smith")) or
(status ≠ 20)

Here we face the problem of how to parameterize such criteria. One solution is to pass the selection criterion simply as a string. Another solution is to adopt, as a convention, a normal form in which any selection criterion is transformed into a sequence of or-connected predicates that are sequences of and-connected comparison terms* (our example is already in that normal form). With this convention a selection criterion can be represented as a set of elements where the elements are sets of ordered triples, e.g.,

{ {<"city", =, "Paris">, <"sname" ≠ "Smith">},
{<"status", ≠, 20>} }

The interpretation of this set is obvious:

- Each triple represents a comparison term.
- The inner sets represent predicates that are and-connections of these comparison terms.
- The outer set represents a selection criterion that is an or-connection of these predicates.

*In formal logic this normal form is called a disjunctive normal form.

The following interface function uses this type of parameter to pass selection criteria.

Example A4:

```
function select (
        source_relation_id : string ;
        selection_criterion :
            {{<lhs_id, comparison_op, rhs_id>}};
        result_relation_id : string    )

(lhs = left hand side, rhs = right hand side)
```

This function requires the name of a source relation, a selection criterion, and an identifier for a (temporary) result relation.

Example of function invocation:

```
select ("suppliers",
       {{<"city",=,"Paris">,<"sname",≠,"Smith">},
         {<"status",≠, 20>}},"temp_relation").
```

The definition of the above function can be generalized by extending

- the first parameter to be a set of source relation identifiers, and
- the second parameter to accept selection criteria that include quantifiers*.

Whereas the first definition of the select function directly supports the select operator of the relational algebra, the extended definition provides the basic support for a query language in the relational calculus approach.

Additional functions are required to support projection (for both the algebra and the calculus approach), join, union, difference, and division (these functions are required for the algebra approach only).

The interface functions for the altering operations are straightforward.

Example A5:

```
function insert (lhs_relation_id : string ;
                 rhs_relation_id : string)
```

The second parameter identifies a set of tuple values to be inserted into the relation that is identified by the first parameter. The tuple values to be inserted may be given by a temporary relation resulting from a database query or by a set of new tuple values provided by the user (see Example 11 in Section 2.3 of the report).

Delete and replace functions can be defined in a similar way.

A.3 Interface Functions for Database Schema Access

A third class of interface functions provides access to a database schema definition. Access to schema information is required for a variety of reasons. For example, schema information is required to check if an altering operation is legal (i.e., does not violate any of the structure, domain, and integrity constraints).

If the database schema information is stored in a relational meta database the interface functions defined in the previous section can be used to access the meta database. The subsequent example follows this approach.

Example A6:

By Example A2 we sketched the structure definition of a "suppliers" relation. If the meta database had a relation, "relation_structures", that holds all structure definitions this (meta) relation may have the following value:

(meta) relation "relation_structures":

attribute names and relation value:

relation_structure_id	attribute _id	domain_id
supplier_structure	snr	suppliernumbers
supplier_structure	sname	suppliernames
supplier_structure	status	creditstatus
supplier_structure	city	citynames
part_structure	pnr	...
...	...	...
shipment_structure	qty	...
...	...	...

Access to the structure definition of relation "suppliers" (as, e.g., required by the legality test for an insert operation, see Example 11, Section 2.3 of the report) can be achieved by the select function defined above.

```
select ("relation_structures",
       {<"relation_structure_id", =,
                        "supplier_structure">},
       "actual_structure")
```

The resulting value of the relation "actual_structure" will be

result relation value:

supplier_structure	snr	suppliernumbers
supplier_structure	sname	suppliernames
supplier_structure	status	creditstatus
supplier_structure	city	citynames

This result can be used to determine, for example, the number of attributes or the domains of the "suppliers" relation.

In this appendix we sketched the meaning (semantics) of some interface functions using the English language. The level of precision that can be achieved by informal or semi-formal definitions is considerable. There are, however, important arguments (compare Section 2.1 of the report) to supplement a nonformal definition of a standard by a formal and precise specification. The definition of the RDM provided in Appendix B can serve as an example of a formal specification.

*In formal logic a corresponding normal form is called prenex disjunctive normal form.

APPENDIX B. A FORMAL DEFINITION OF THE RELATIONAL MODEL

The 'Vienna Development Method' (VDM) [BJOR78, LAME80] is used to specify formally the semantics of the Relational Data Model (RDM). A VDM-Specification consists of three parts:

1. The Semantic Domains define a structural model for all 'objects of interest' and their components.
2. The Syntactic Domains define the names of all operations and a structural specification for each of their operands.
3. The Elaboration Functions define the semantics of all operations introduced by the syntactic domains.

Additional constraints may be introduced either by Static Consistency Constraints, defining whether a semantic object is "well-formed", or by Dynamic Consistency Constraints, stating whether a syntactic object is "well-applied".

Auxiliary Functions may be used to improve the conciseness and readability of a definition.

B.1 Semantic Domains for Relational Databases

The main object specified is a Relational Database, RELDATABASE. It consists of three parts: the domain definition, DOMAINS; the database schema, RDBSCHEMA; and the database value, RDBVALUE.

A Domain definition is modeled as a mapping from the set of domain identifiers, DOMID, to the associated domains, DOM. A domain is a set of atomic domain elements, DEM.

A Database Schema defines the 'structure' of the database, RDBSTR, and the constraints, RDBCNSTR, to be maintained on the database. The database structure defines for each relation of a database a relation schema, RELSCHEMA, and a relation identifier, RELID. A relation schema is given by the relation's structure, RELSTR, and by relation constraints, i.e., the key attribute identifiers.

A Database Value, RDBVALUE, is defined by the relation values. A relation value, RELVALUE, is represented in two ways: as a set of tuple values, TUPLEVAL-Set, and as a predicate defined over all possible tuple values, TUPLEVAL-Pred.

Identifiers are not further specified and therefore are considered as 'TOKENS'.

B.1.1 Abstract Syntax

```
RELDATABASE     ::  RDBSCHEMA RDBVALUE

RDBSCHEMA       ::   RDBSTR DOMAINS RDBCNSTR
RDBSTR          =   (RELID -m-> RELSCHEMA)
RELSCHEMA       ::   RELSTR RELCNSTR
RELSTR          ::   TUPLESTR
TUPLESTR        =   (ATTRID -m-> DOMID)
RELCNSTR        =    KEYATTRID-Set

DOMAINS         =   (DOMID -m-> DOM)
DOM             =    DEM-Set
DEM             =     ...

RDBCNSTR        =     ...

RDBVALUE        =   (RELID -m-> RELVALUE)
RELVALUE        ::  SETREP PREDREP
SETREP          =   TUPLEVAL-Set
PREDREP         =   TUPLEVAL-Pred
TUPLEVAL        =   (ATTRID -m-> DEM)

DOMID           =   TOKEN
RELID           =   TOKEN
ATTRID          =   TOKEN
KEYATTRID       =   TOKEN
```

B.1.2 Static Consistency Constraints

```
is-wf RELDATABASE(reldatabase) =

let mk-RELDATABASE(rdbschema,rdbvalue ) be reldatabase
    in let mk-RDBSCHEMA(vdbstr,domains) = rdbschema
      in dom rdbstr = dom rdbvalue
       and (∀ relid ∈ dom rdbvalue)
           (is-wf-Relation(rdbvalue(relid) ,
               rdbstr(relid) , domains ))
       and test-rdb-constraints(reldatabase)

type : RELDATABASE ---> BOOL


test-rdb-constraints(reldatabase) =

     ...

type : RELDATABASE ---> BOOL


is-wf-RELATION(relvalue,relschema,domains) =

    let mk-RELVALUE(setrep,predrep) be relvalue ,
        mk-RELSCHEMA(relstr,relcnstr) be relschema
    in let mk-TUPLESTR(tuplestr) be relstr
          in rng tuplestr ⊆ dom domains

             and relcnstr ⊆ dom tuplestr

             and dom predrep =
                 {[attrid ---> dem ] |
                  attrid ∈ dom tuplestr and
                  dem ∈ domains(tuplestr(attrid))}

             and setrep =
                 {tupleval | tupleval ∈ dom predrep
                             and predrep(tupleval) }

             and test-key-constraints(setrep,relcnstr)

type :  RELVALUE RELSCHEMA DOMAINS ---> BOOL


test-key-constraints(setrep,relcnstr) =

    let relkey be relcnstr
    in (∀ tupleval,tupleval' ∈ setrep)
       ((tupleval | relkey = tupleval' | relkey) ===>
                  ( tupleval = tupleval' ))

type :  SETREP RELCNSTR ---> BOOL
```

B.2 Syntactic Domains for Relational Databases

The basic database operations are queries, Query, and relation altering operations, Altop.

Queries can be database relations (given by relation identifiers), 'selections', 'joins', 'projections', 'unions', ... (in the algebraic approach), or 'relational calculus expressions' (in the calculus approach). Operands for these operations are either relation expressions, Rel, which are built up (recursively!) by other queries, or 'external' relations, Extrel, from outside the database. A 'rename' operation, Rename, is specified to achieve unique attribute identifiers for relational operands.

An altering operation, Altop, takes a relation, Rel, and alters the value of a database relation identified by its name, RELID.

B.2.1 Abstract Syntax

```
Op                =    Query | Altop

Query             =    RELID | Selection | Join |
                       Projection | Union
                       Division | ...   |
                       Rename | Calc-Expression

Selection         ::   Rel Selpred
Selpred           =    TUPLEVAL-Pred
Join              ::   Rel ATTRID Rel ATTRID
Projection        ::   Rel Attrids
Attrids           =    ATTRID-Set
Union             ::   Rel Rel
 ...
Rename            ::   Rel Bijection
Bijection         =    (ATTRID <-m->ATTRID)

Calc-Expression   ::   Targetattrids Sourcerels Selpred
Targetattrids     =    ATTRID-Set
Sourcerels        =    Rel-Set

Altop             =    Insert | Delete | Replace
Insert            ::   RELID  Rel
Delete            ::   RELID  Rel
Replace           ::   RELID  Rel

Rel               =    Query | Extrel
Extrel            ::   RELVALUE RELSTR
```

B.2.2 Dynamic Consistency Constraints

```
is-wf Op(op) =

    (is-Query(op) ---> is-wf-Query(op) ,

     is-Altop(op) ---> is-wf-Altop(op))

type :  Op ---> BOOL


is-wf-Altop(altop) =

    let relid be s-RELID(altop) ,
        rel be s-Rel(altop)
    in is-wf-Rel(rel) and
       compatible(relid,rel)

type : Altop ---> BOOL


is-wf Query(query) =

    cases query :
    (mk-RELID(relid) ---> relid ϵ dom RDBVALUE,

     mk-Selection(rel,selpred) ---> is-wf-Rel(rel) and
            dom selpred = dom extract-predrep(rel) ,

     mk-Join(rel,attrid,rel',attrid') --->
       let tplstr be tpl-structure(rel) ,
           tplstr' be tpl-structure(rel')
       in is-wf-Rel(rel) and is-wf-Rel(rel') and
          attrid ϵ dom tplstr and attrid'
                                  ϵ dom tplstr' and
          tplstr(attrid) = tplstr'(attrid') and
          dom tplstr ∩ dom tplstr' = { } ,

     mk-Projection(rel,attrids) ---> is-wf-Rel(rel)
               and attrids⊆ dom tpl-structure(rel) ,

     mk-Union(rel,rel') ---> is-wf-Rel(rel)
                             and is-wf-Rel(rel')
                             and compatible(rel,rel'),

     mk-Division ...

     ...

     mk-Rename(rel,bijection) --->
       is-wf-Rel(rel) and
       dom bijection = dom tpl-structure(rel) ,

mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
   (∀ rel ϵ sourcerels)(is-wf-Rel(rel)) and
   (∀ rel,rel' ϵ sourcerels)((rel ≠ rel') ===>
      (dom tpl-structure(rel)∩
       dom tpl-structure(rel') = { } ))
   and targetattrids ⊆ {attrid |
                        attrid ϵ dom tpl-structure(rel)
                        and rel ϵ sourcerels }
   and dom selpred =
       {tupleval1 u...u tuplevaln |
        tuplevali ϵ dom extract-predrep(reli)
        and reli ϵ sourcerels
        and 1 ≤ i ≤ n = card sourcerels }

type : Query---> BOOL


is-wf-Rel(rel) =

(is-Query(rel) ---> is-wf-Query(rel),
 is-Extrel(rel) --->
   let mk-Extrel(relvalue,relstr) be rel
   in let relcnstr be dom mk-TUPLESTR(relstr)
      in is-wf-Relation(relvalue,mk-RELSCHEMA(relstr,
                        relcnstr),DOMAINS))

type : Rel ---> BOOL


compatible (rel,rel') =

let tplstr  be tpl-structure(rel) ,
    tplstr' be tpl-structure(rel')
in  tplstr  =  tplstr'

type : Rel Rel --->BOOL


tpl-structure(rel) =

(is-Query(rel) --->  cases rel :
mk-RELID(relid) ---> mk-TUPLESTR(s-RELSTR(s-RDBSTR
                     (c RDBSCHEMA)(relid))) ,

mk-Selection(rel', ) ---> tpl-structure(rel') ,

mk-Join(rel', ,rel", ) ---> tpl-structure(rel') u
                            tpl-structure(rel") ,

mk-Projection(rel',attrids ) ---> tpl-structure(rel')
                                  | attrids ,

mk-Union(rel', ) ---> tpl-structure(rel') ,
```

```
mk-Division   ...

 ...

mk-Rename(rel',bijection) --->
  let tplstr be tpl-structure(rel')
  in [attrid' ---> domid |
                   attrid' = bijection(attrid) and
      attrid ∈ dom tplstr and domid = tplstr(attrid)] ,

mk-Calc-Expression(targetattrids,sourcerels, ) --->
  let tplstr be tpl-structure(rel_1) u ...
                              u tpl-structure(rel_n)
      s.t. rel_i ∈ sourcerels
           and 1 ≤ i ≤ n = card sourcerels
  in tplstr | targetattrids                ) ,

is-Extrel(rel) ---> mk-TUPLESTR(s-RELSTR(rel)))

type : Rel ---> TUPLESTR
```

B.3 Elaboration Functions for Relational Databases

The meaning of an operation is specified by:

- An evaluation function, eval-Query, if the operation is a query. This function states which relation value, RELVALUE, is computed as the result of a query.
- An interpretation function, int-Altop, if the operation is an altering operation. This function specifies the changes of the actual database value, RDBVALUE, caused by the interpretation of that relation altering operation.

```
elab-Op(op) =

  cases op:
  (is-Query(op) ---> eval-Query(op) ,
   is-Altop(op) ---> int-Altop(op))

type : Op ---> RELVALUE | (RDBVALUE --->RDBVALUE)


eval-Query(query)

    let setrep be extract-setrep(query) ,
        predrep be extract-predrep(query)
    in mk-REVALUE(setrep,predrep)

type : Query ---> RELVALUE


int-Altop(altop) =

    cases altop :
    (mk-Insert(relid,rel) --->
    let setrep be
        extract-setrep(relid u extract-setrep(rel) ,
       predrep be extract-predrep(relid)
                  or extract-predrep(rel)
    in let relvalue be mk-REVALUE(setrep,predrep)
       in RDBVALUE := c RDBVALUE
                    + [relid ---> relvalue],

    mk-Delete(relid,rel) --->
    let setrep be extract-setrep(relid)
                  \ extract-setrep(rel) ,
        predrep be extract-predrep(relid) and
                   not extract-predrep(rel)
    in let relvalue be mk-REVALUE(setrep,predrep)
       in RDBVALUE := c RDBVALUE
                    +[relid ---> relvalue],

    mk-Replace(relid,rel)  ---> ...        )

type : Altop ---> (RDBVALUE ---> RDBVALUE )


extract-setrep(rel) =

    (is-Query(rel) ---> cases rel :
    (mk-RELID(relid) ---> s-SETREP(c RDBVALUE(relid))

     mk-Selection(rel',selpred) --->
        {tupleval | tupleval ∈ extract-setrep(rel')
                and selpred(tupleval)} ,
     mk-Join(rel',attrid',rel",attrid") --->
        let setrep' be extract-setrep(rel') ,
            setrep" be extract-setrep(rel")
        in {tupleval' u tupleval" |
            tupleval' ∈ setrep' and
            tupleval" ∈ setrep" and
            tupleval'(attrid') = tupleval"(attrid")},

     mk-Projection(rel',attrids) --->
        {tupleval' | tupleval' = tupleval
                   | attrids and
                     tupleval ∈ extract-setrep(rel')}

     mk-Union(rel,rel") ---> extract-setrep(rel')
                             u extract-setrep(rel"),

     mk-Division ...

     ...

     mk-Rename(rel',bijection) --->
       {[attrid' ---> dem
                    | attrid' = bijection(attrid) an
                      attrid ∈ dom tupleval and
                      dem = tupleval(attrid)]
                    | tupleval ∈ extract-setrep(rel')


mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
    let prodset be {tupleval_1 u ... u tupleval_n |
                    tupleval_i ∈ dom extract-setrep(rel
                    and rel_i ∈ sourcerels
                    and 1 ≤ i ≤ n = card sourcerels }
    in {tupleval' | tupleval' = tupleval
                  | targetattrids and
                    tupleval ∈ prodset and
                    selpred(tupleval) } ) ,

is-Extrel(rel) ---> s-SETREP(s-RELVALUE(rel)))

type : REL ---> SETREP


extract-predrep(rel) =

(is-Query(rel) ---> cases rel :

(mk-RELID(relid) ---> s-PREDREP (c RDBVALUE(relid)),

 mk-Select(rel',selpred) ---> extract-predrep(rel')
                              and selpred,

mk-Join(rel',attrid',rel",attrid") --->
   let predrep' be extract-predrep(rel') ,
       predrep" be extract-predrep(rel")
   in [(tupleval' u tupleval") --->
      predrep'(tupleval') and predrep"(tupleval")
      and tupleval'(attrid') = tupleval"(attrid")|
      tupleval' ∈ dom predrep'
      and tupleval" ∈ dom predrep"],
```

```
mk-Projection(rel′,attrids) --->
   let predrep be extract-predrep(rel′)
   in [tupleval′ ---> predrep(tupleval) |
       tupleval′ = tupleval | attrids and
       tupleval ϵ dom predrep]      ,

mk-Union(rel′,rel") ---> extract-predrep(rel′) or
                         extract-predrep(rel"),

mk-Division ...

...

mk-Rename(rel′,bijection) --->
 let predrep be extract-predrep(rel′)
 in [tupleval′ ---> predrep(tupleval) |
     tupleval′ = [attrid′ ---> dem |
                  attrid′ = bijection(attrid) and
                  attrid ϵ dom tupleval and
                  dem = tupleval(attrid)]
     and tupleval ϵ dom predrep ] ,
```

```
mk-Calc-Expression(targetattrids,sourcerels,selpred)
--->
    let prodpred be [(tupleval_1 u ... u tupleval_n)
                     --->
                     extract-predrep(rel_1)(tupleval_1)
                     and ... and
                     extract-predrep(rel_n)(tupleval_n)|
                     tupleval_i ϵ dom extract-predrep(rel_i)
                     and rel_i ϵ sourcerels
                     and 1 ≤^i i ≤ n = card sourcerels]
    in [tupleval′ ---> prodpred(tupleval) |
                     tupleval′ = tupleval
                                 | targetattrids and
                     tupleval ϵ dom prodpred] ),

     is-Extrel(rel) ---> s-PREDREP(s-RELVALUE(rel)))

    type : Rel ---> PREDREP
```

APPENDIX C. RELATIONAL DBMS SURVEY

An objective of the RTG was to evaluate the state-of-the-art concerning DBMSs that are claimed to support aspects of the RDM. The RTG found that over 60 DBMSs were claimed to support significant aspects of the RDM. Due to limited resources, a detailed analysis of all such systems was not possible. The Relational DBMS Survey was conducted to evaluate the state-of-the-art in broad detail. The survey attempted to determine the extent of design, development, and use of Relational DBMSs (RDBMS). It was intended to:

1. identify all major RDBMS development efforts in both research and commercial settings, and

2. characterize each effort briefly in terms of interfaces, availability and extent of use.

C.1 Performing the Survey

The RTG solicited this information through announcements and press releases to trade and academic periodicals, by letter, and at database conferences. Shortly, after the RTG's formation, an announcement of the RTG's charter and a request for information on RDBMSs was placed in a variety of publications (e.g., SIGPLAN Notices Volume 14, Number 12, December 1979, and Performance Development Corporation Database Newsletter Volume 7, Number 6, November 6, 1979).

In January 1980, the RTG announced the survey and feature analysis and solicited contributions by means of the following SPARC press release:

ANSI Task Group Solicits Input
Towards Relational Standard

The Relational Database Task Group of the ANSI/X3/SPARC Database Systems Study Group is investigating the justifiability of a standard for relational data base management systems (RDBMS's). It will identify or establish:

- a working definition for the relational data base model;
- the role of the relational data base model appropriate for standards development (e.g., relational languages);
- terminology and concepts for the relational data base model; and
- the relationship of a RDBMS standard with other software standards that exist or are in development.

The task group solicits comments, position papers, and reports on related research and development projects, and on existing and proposed RDBMS's and relational languages. Responses received by January 31, 1980 will be used in the development of a survey and feature analysis of RDBMS's to be published in 1980.

Submissions should be sent to either of the task group co-chairmen:

Michael L. Brodie	Joachim W. Schmidt
Department of Computer Science	Fachbereich Informatik Universitaet Hamburg
University of Maryland	Schlueterstr. 70
College Park, Maryland	D-2000 Hamburg 13
20742 U.S.A	West Germany
(301) 454-2002	(040) 4123-4164

The press release was widely distributed and was included in: Performance Development Corp. Database Newsletter Volume 8, Number 1, January 1980; SIGPLAN Notices Volume 15, Number 2, February 1980; Communications of the ACM Volume 23, Number 2, February 1980; and many of the trade publications such as Computer World.

Representatives of twenty two systems responded to the press release. Twelve respondents ultimately completed a detailed feature analysis of their system (see Section 4). The RTG felt that many other systems existed for which no response was received. Hence, the survey became independent of the feature analysis. A one page questionnaire was created and the following press release was issued by SPARC in March 1981.

Survey of Relational DBMSs
by X3's Relational Database Task Group

Implementors of database management systems (DBMSs) supporting aspects of the relational data model are being invited to complete a brief, one page questionnaire to describe their systems. This survey is part of an investigation by X3's Relational Database Task Group (RTG) into the potential standardization of the relational approach to databases. The survey will be presented to X3 soon after it is compiled in early June 1981 in support of a future DBMS standardization development effort of the American National Standards Institute (ANSI).

The questionnaire must be obtained before

MAY 1, 1981

by writing to:

Harrison R. Burris
Neotechnic Industries Inc.
P.O. Box 277
Redondo Beach, CA 90277

Only those completed questionnaires received by

JUNE 1, 1981

will be included in the survey results. A copy of the survey will be sent to all contributors.

For further information telephone the RTG chairman, Prof. Michael L. Brodie at the University of Maryland, 301-454-2002 or Harrison R. Burris at TRW Defense and Space Systems Group, Redondo Beach, CA, 213-535-1047.

The press release was widely published in trade and academic periodicals (e.g., SIGPLAN Notices Volume 16, Number 5, May 1981).

The following cover letter and questionnaire was mailed to all respondents of both press releases, RTG members, and individual and organizations known to the RTG to be candidates for the survey.

American National Standards Committees
X3 -- Computers and Information Processing
X4 -- Office Machines Supplies

operating under the procedures of the
American National Standards Institute

March 24, 1981

Dear Colleague:

Implementors of relational database management systems are being invited to complete the attached

questionnaire. This survey is part of an investigation by the American National Standards Institute (ANSI) X3's Relational Database Task Group (RTG) into the potential standardization of the relational approach to databases.

The questionnaire must be returned to:

Harrison R. Burris
TRW DSSG
One Space Park, 114/2740
Redondo Beach, CA 90278

no later than 31 July in order to be included in the survey results. A copy of the survey will be sent to all contributors after presentation to X3 sometime in October 1981.

Sincerely,

Harrison R. Burris

QUESTIONNAIRE

The Relational Task Group will use this questionnaire to construct a roster of relational database systems. Your assistance will ensure a more thorough job; please complete as much of the questionnaire as possible.

1. Name of System:_____

2. Name of vendor/implementor:_____

3. Contact person (name, address, and phone number):

4. General description (250 words maximum):

5. Does the system have the following interfaces?

	Yes	No
Conceptual database description (schema)		
Relational external schema		
Interactive query/update language		
Data dictionary		
Relational sub-language adapted to host languages		

Specify programming languages:_____

6. Date system available:_____

7. Current Applications:

8. List published documentation

The 13 responses received by July 31, 1981, were incorporated into the survey results given in the remainder of the section. Additional information about RDBMSs has been included from [BROD80d]; [KIM79].

C.2 RDBMS Survey Results

Disclaimer

Whereas the RTG has attempted to identify systems supporting significant aspects of the RDM, the RTG does not explicitly or implicitly declare the systems named in the survey to be relational in the sense discussed in other parts of this report.

INTERFACE TYPE:
- CS = conceptual database description (schema) language
- ES = relational external schema language
- QL = interactive query/update language
- DD = data dictionary
- RSL = relational sub-language adapted to host languages
- "x" = indicates that the system supports such an interface. Generally, no "x" indicates that no such interface exists.

STATUS
- C = available for production use
- R = research prototype
- D = design stage
- E = education

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
ADAPLEX	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x	x	x	x	x	D	
ARCH:MODEL	Architectural Research Laboratory University of Michigan Ann Arbor, MI 48109	x		x		x	C	
ARTEMIS	Meitier Management Systems Inc. Derek Hardy 10175 Harwin Dr., Suite 100 Houston, TX 77036	x	x	x	x		C	6/77
ASTRAL	Department of Computer Science University of Trondheim N-7034 Trondheim-NTH Norway	x	x	x	x	x	E,R	
CAFS	International Computers Ltd. UK						C	
CASSM	University of Florida	x		x			E	
CREATE/3000	CRI, Inc. 2570 El Camino Real Mountain View, CA 94040	x		x		x	C	
CS4	KTH Sweden						C,R	
DTSS	DTSS Inc. (David Goldberg) 10 Allen Street Hannover, NH 03755	x	x	x	x	x	C	1/79
DAPLEX	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x	x	x	x	x	D	
DATACOM/DB	DATACOM Division Applied Data Research 8515 Greenville Ave., Suite 101 Dallas, TX 75243	x	x	x	x	x	C	2/73
DIRECT	Bancohio Corporation 770 W. Broad St. Columbus, OH 43222	?	?	?	?	?	C	?
DMS-170	Control Data Corporation 8100 34th Ave. South Minneapolis, MN 55440	x				x	C	?
ECOMPASS (ENFORM)	Tandem Computers Inc. 19330 Vallco Parkway Cupertino, CA 95014	x		x		x	C	1981
FADABS	Kernforschungszentrum Karlsruhe GmbH Institut fuer Datenverarbeitung Postfach 3640 D-7500 Karlsruhe, West Germany	x		x		x	C	1978
FQL	University of Philadelphia			x			R,E	
FRAMIS	Lawrence Livermore Laboratory	x		x	x		C	9/78
GAMMA-0	IBM, Yorktown Heights	x				x	R	1972
GMIS	MIT/IBM Donovan (MIT)			x			R	1976
GRAM	IBM	?	?	?	?	?		
IDBP (Intel Database Processor)	Intel Corporation 3065 Bowers Ave. Santa Clara, CA 95051	x	x	x	x	x	C	late 1981
IDM	Britton-Lee, Inc.	x	x	x	x		C	12/80

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
IMPS	IBM	x	x	x	x	x	C	1980
INFOSYSTEM	Henco Inc. Wellesley, MA	x	x	x		x	C	1975
INGRES	Relational Technology, Inc. 2855 Telegraph Ave., Suite 515 Berkeley, CA 94405	x	x	x	x	x	C	9/80
IPIP	Boeing Computer Services (IPAD Project)	x	x	x		x	D	
JANUS	Massachusetts Institute of Technology	x				x	C,E	
MADAM	Massachusetts Institute of Technology	?	?	?	?	?	E	?
MANAGE	CSC Infonet Division 650 N. Sepulveda Blvd E640 El Segundo, CA 90245	x	x	x	?	x	C	?
MAGNUM	Tymeshare Inc. Magnum Development Department 20705 Valley Green Dr. Cupertino, CA 95014	x	x		x		C	1975
MINISIS	Intrl Development Research Center 60 Queen St. Ottawa, Ontario Canada	?	?	?	?	?	C	?
MRDS/LINUS	Honeywell Inc. Corporate Technical Center 10701 Lyndale Ave. South Bloomington, MN 55420	x	x	x	x	x	C	3/78
MRS	Dennis Tsichritzis University of Toronto Toronto, Ontario Canada	x			x		C,R,E	
ODEX	Dr. Gianola Maurizio Olivetti OWPOA/TS ICO 2nd piano v. Jervis 77 Ivca, Italty	x	x		x	x	C	1981
NOMAD 2	NOMAD Development National CSS Inc. 187 Danbury Road Wilton, CN 06897	x	x	x	x	x	C	1975
ORACLE/SQL	Relational Systems Inc. 3000 Sand Hill Rd. Menlo Park, CA 94025	x	x	x	x	x	C	1979
OMEGA	University of Toronto	x	x			x	E,D	
PASCAL/R	Universitaet Hamburg Fachbereich Informatik Schlueterstrasse 70 D-2000 Hamburg 13 West Germany	x	x	x	x	x	R (D) E	1977
PHLOX	INRIA B.P. 105 F-78150 Roquencourt France	?	?	?	?	?	R	
PLAIN	Dept. of Medical Information Science University of California San Francisco, CA	x	x	x	x		R	
PLANES	University of Illinois			x			E,R	

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
POREL	Universitaet Stuttgart Azenbergstrasse 12 D-7 Stuttgart 1 Postfach 560 West Germany	x	x	?	?	?	E,R	
PRTV(IS/1)	IBM UK Scientific Center Athelston House St. Clement St. Winchester Hampshire S023 8UT UK	x	x	x	x	x	C,R	1973
Extended QBE	IBM Scientific Center Heidelberg West Germany	x	x	x	x	x	R	1979
QBE	IBM 1133 Westchester Ave. White Plains, NY 10604	x	x	x	x	x	C	1978
PARIOU	M. Schneider, J.C. Perraud CUST BP 63 170 Aubiere France	x	x	x	x		E	1981
PINDAR	H. Fergen GMD/IIG D-5205 St. Augustin 1 Postfach 1290 West Germany	x		x			C	11/81
QUESTOR	Comshare, Inc. 3001 South State St. Ann Arbor, MI 48104	x		x			C	?
RAP	University of Toronto					R		
RAPID	Database Management Systems 12/P R.H. Coats Bldg Tunney's Pasture Ottawa H1A 0T6	x		x		x	C	?
RAPPORT	Logica Ltd. 341 Madison Ave. New York, NY 10017	x		x		x	C	?
RARES	University of Utah						R	
RELIANCE	Perkin-Elmer Data Systems 277 Bath Road Slough, Berkshire SL1 4AX UK	x	x	x	x	x	D	6/82
RENDEZVOUS	IBM Research, San Jose	?	?	x	?	?	R	
REL*STOR (RELATE 80)	GTE Sylvania P.O. Box 188 Mountain View, CA 94042	x		x		x	C	3/81
RIGEL	University of California, Berkeley	x	x				R	
RIM	Boeing Computer Services (IPAD Project)	?	?	?	?	?	C	?
RISS	Forrest Hospital	x		x			C	
RPS1100	Sperry-Univac		x	x			C	?
RTFILE	Inter Project Inc. P.O. Box 13 Brentwood, MD 20722	x					C	?
RDBMS	International Computers, Ltd. UK	?	?	?	?	?	C	?

SYSTEM	DEVELOPER/VENDOR	INTERFACE TYPE					AVAILABILITY	
		CS	ES	QL	DD	RSL	STATUS	DATE
RDBMS	Open University Mr. M.A. Newton Academic Computing Service Milton Keynes MK7 6AA UK	x	x	x	x		R	
RDMS	Massachusetts Institute of Technology	?	?	?	?	?	R	
RDMS(REGIS)	IBM UK	x	x	?	?	x	C	?
SESAM	SIEMENS AG Holger Conradi D ST DB22 Otto-Hahn-Ring 6 8000 Munich 83 West Germany	x	x	x		C	10/80	(v12.1) 10/82 (v13)
SDD-1	Computer Corporation of America 575 Technology Square Cambridge, MA 02139	x		x	x	x	R	1979
SYNTEX-II	R. Demolombe, J.-M. Nicolas, M. Lemaitre ONEA-CERT 2 Avenue Edouard Belin B.P. 4025 31055 Toulouse France	x		x		x	R	1975
SYSTEM 38	IBM Highway 52 & NW 37th St. Rochester, MN 55901	x	x	x	x	x	C	9/80
SYSTEM 3000	MRI Systems Corporation						D	
SYSTEM-D	Hannu Kangassalo University of Tampere P.O. Box 607 SF-33101 Tampere 10 Finland	x	x	x	x	x	D	
SYSTEM-R/SQL	IBM Research, San Jose	x	x	x	x	x	R	1975
SQL/DS	IBM Armonk, NY 10504	x	x	x	x	x	C	1982
HIBE	STIS 3135 South State St. Suite 104 Ann Arbor, MI 43104	x	x	x	x	x	C	1982
VERSO	INRIA F. Bancilhon, M. Scholl B.P. 105 F-78153 Le Chesnay Cedex France	x		x	x		R	
ZETA/TORUS	Dennis Tsichritzis CSRG Toronto, Ontario Canada	x	x			x	D	

Textual Responses

The following information was supplied by those that responded to questions 4, 5, 7, and 8.

ARTEMIS

General Description:

ARTEMIS is a mobile, self-contained Project Management Information System based on a dedicated minicomputer. In addition to in-built procedures for planning, scheduling, cost reporting, and probabilistic analysis, ARTEMIS also offers:

- a relational database featuring total field independence with dynamic dataset structuring and linking
- a simple yet powerful command language which is identical for all modules
- a versatile report writer
- a wide range of system options in both hardware and software
- communication with large mainframe computers
- concurrent processing of user-developed applications written in other programming languages
- graphical output of "S" curves, barcharts, histograms, and pie charts, using an 8-color flatbed plotter onto paper or transparencies, or projecting from the terminal directly onto a screen
- automated network logic drawings, either linear, variable, or non-timescaled, onto a 36" 4-color sum plotter
- annotated barchart drawings on either drum or flatbed plotter
- expandable up to 32 terminals, local or remote, and online disc storage up to one billion bytes

Current Applications:

Industry types include offshore oil, civil construction, aerospace and weapons, petrochemicals, automobiles, pharmaceuticals, energy, and government agencies. Application examples include planning and scheduling, cost/schedule integration, performance measurement, management reporting and forecasting, cost management, materials management, records management, plant and equipment registers, and maintenance planning and control.

Documentation

1. Network Planning System User Guide
2. Information Processing and Reporting System User Guide
3. Systems Managers Manual (4 versions for different memory capacities)
4. Management Graphics User Guide
5. Network Graphics User Guide
6. 2780 Emulator User Guide
7. PAN User Guide

DTSS (DaTaSyS)

General Description:

All data is stored in flat files. All access to data is via "views" of the files. Each file has full access to a master view. Subviews with more limited access to selected records and fields may be created. Such subviews may have operation restrictions. Views may be combined using three operators, JOIN, INCLUDE, and INSERT. INCLUDE is similar to JOIN, but only zero or one matches are permitted. If view EMP INCLUDES view DEPT, each EMP record will occur in the result even if no matching DEPT record is found. INSERT creates "hierarchical" views. If view DEPT INSERTs view EMP, each DEPT record will appear once in the result, followed by the matching EMP records.

Interface Details:

Query Language: TELL*ME. Accessible from: COBOL, PL/1, BASIC, FORTRAN, assembly (GMAP).

Current Applications:

Treasury Management, Law Case Management, Executive Report Retrieval, DTSS Customer Billing, DTSS Sales Prospects, DTSS System Bugs and Proposals.

Documentation:

1. DaTaSyS, The DTSS Database Management System 5 MAR 80 (revision 2A).
2. DaTaSyS Reference Manual, Pub. 1311.
3. User's Manual for DaTaSyS***:GUIDE, Pub. 1331.
4. Introduction to DTSS Software, Pub. 5322.
5. User's Guide to the TELL*ME Query System.

ENCOMPASS

Interface Details:

Accessible from: COBOL, TAL, FORTRAN, MUMPS.

FADABS

General Description:

FADABS is a DBS for the use in minicomputer based technical, industrial, or scientific application systems. Its architecture makes it suited to this class of application systems: (1) proper modularization helps keeping programs small (application programs as well as the DBS tasks themselves); (2) the usually powerful process synchronization and communication features provided by minicomputers are utilized to achieve realtime behavior of DBS-application programs; (3) a low level (i.e., tuple-at-a-time) relation interface to FORTRAN, with binding of data structures at compile-time, as a means for the implementation of efficient, tailored user interfaces; and (4) two general, high level (i.e., set-oriented, descriptive) interfaces: interactive query facility, report generator.

FADABS is implemented mainly in FORTRAN, to achieve a high degree of portability, and available on SIEMENS 300 and PDP11/40 minicomputers.

Current efforts are directed at making FADABS a generable DBS.

Interface Details:

Accessible from FORTRAN.

Other Details:

Report generator, data entry facility.

Current Applications:

Nuclear materials information systems.

Documentation

1. Polster, F.J.: FADABS: Ein Datenbanksystem fuer die SIEMENS-Prozessrechner S330. Tagungsbericht, 9. Jahrestagung des SIEMENS-Anwenderkreises I, Kernforschungszentrum, 1978, KfK-Bericht 2642.

2. Polster F.J.: Using a Preprocessor to Implement a Data Manipulation Language for a Minicomuter Data Base System. Proceedings First Symposium on Small Systems. ACM SIGSMALL, N.Y., 1978, p. 40-44.

3. Stoeckle, D.; Polster, F.J.: The Report-Definition Language RDL and its Implementation in the Report-Generator FAREG. Report, KfK 2910, Kernforschungszentrum Karlsruhe, 1980. (in German)

4. Polster, F.J.; Tritschler, P.: The Database Query Language FAQUEL. Report KfK 3089, Kernforschungszentrum Karlsruhe, 1981. (in German)

Multics Relational Data Store (MRDS)

General Description:

The Multics Data Base Manager (MDBM) provides Multics users with a general database management facility that is callable from all Multics programming languages supporting a call interface and from Multics command level. A full range of database definition, retrieval, and update capabilities is available, together with facilities that provide a large measure of data independence and control of concurrent accesses to the database.

The MDBM is unique in that it provides an interface for both relational and network (CODASYL) database users. The relational interface, known as the Multics Relational Data Store (MRDS), supports network databases using a subset of the CODASYL standard specification.

Though related to each other at the implementation level, the user should view each interface as a separate and independent database management system.

Interface Details:

Query, manipulation: LINUS (The Logical Inquiry and Update System). Accessible from: FORTRAN, COBOL, PL/1, APL, and any language with a call facility supported on MULTICS.

HIBE (Highly Intelligent Back End)

Interface Details:

Accessible through only high level programming language with a call mechanism. Also has an interactive central language (e.g., self-contained high level programming language).

ODEX (Olivetti Database Executive)

General Description:

ODEX provides services of database definition and manipulation to application programmers; the data are defined and structured at the Logical Schema level and at the Storage Schema level by using the data and Storage Description, are organized at the logical level using a relation model and at the storage level using an indexed file organization. The data manipulation facilities are offered as extensions of the PLZ language, by the Data Manipulation Language. These extensions are processed by a precompiler which generates an extended PLZ source module and an access module, which will then be compiled by the PLZ compiler and linked to the run time support routines. The DML is very high level language based on the values of the data and is completely independent from the physical structures. ODEX provides mechanisms for controlling multiuser operations and facilities to recover a consistent database state in case of a system failure. In particular, actions performed on the database can be grouped in units called transactions, which are guaranteed to be executed as indivisible units. A log facility is provided.

Interface Details:

Accessible through PLZ.

Current Applications:

DBMS for a network monitor system.

ORACLE/SQL

Interface Details:

Query, data definition, manipulation, and control language: SQL. Accessible from: PL/1, COBOL, FORTRAN, C, assembly.

PARIOU

General Description:

PARIOU is a relational database management system which has been designed for educational and research purposes. He is presently available on PDP 10 but is entirely written in standard COBOL. The physical structure is based on inverted files. The system includes algorithms to optimize the process of queries. The actual query language is like ALPHA language. It is possible to implement easily other query languages since all queries in high level language are transformed in standard form by relational algebra operators.

Current Applications:

School attendance database.

Documentation:

1. J.C. Perraud. "Conception d'un système de gestion relationnelles PARIOU", Thesis.

2. B. Collardey. "Realisation du noyau du système PARIOU", Thesis.

3. R. Mpassi et J.L. Robillon. "L'algorithme de recherche du système PARIOU".

PINDAR (Adaptable Interactive Database and Report System)

General Description:

PINDAR is a form oriented interactive system with its own user language and includes functions for the definition of databases (Schema), input, update, deleting of data, comprehensive evaluation of databases and other files, and the generation of reports.

A PINDAR database consists fundamentally of a linear data file and an inverted file. The data file contains records made up of fields and subfields and the inverted file contains references to the data fields. Linkages may be defined between otherwise unrelated databases.

The user language consists of descriptive and procedural elements, and applications of various degrees of complexity may be programmed. A real time menu technique is used, where many options have predefined defaults.

Other Details:

KLDS (compatible database interface adapted to the linear data structure), i.e., host language interface to call PINDAR data management services as defined for the German public services.

Documentation:

PINDAR User Reference Manual (German Edition).

RDBMS (Open University)

General Description:

RDBMS was designed with a primary objective of demonstrating relational concepts for students. It supports all relational algebra operations and is perceived as a relational "calculator", with all operations expressed as a single command (e.g., JOIN) which operate on a workspace relation (using data from a second named relation as appropriate). It has integrated data dictionary facilities and commands to examine all types of data. This includes a domain concept, referred to as a "category" because it is considered more powerful than a mathematical domain, supporting join operations and acting as an integrity constraint for inserting new values for foreign keys (constraints not yet fully implemented). Tuple identification is based on explicitly specified keys, and if a project operation removes any key than it is necessary to make the new key(s) explicit using a KEY function. Aggregate functions are also supported during project operations when the number of tuples is reduced. The conceptual schema is expressed in terms of relations and each external schema is expressed a user view, specifying the relations, data items of each relation and procedures available to that user.

The system runs on a DEC-20, written largely in COBOL and using B*-trees for storage. The user friendly environment (help, recognition) is system dependent.

Current Applications:

University equipment database.

RELIANCE

General Description:

RELIANCE is a high performance transaction processing system with a relational database management capability. There is a Data Dictionary for the definition of data items, groups, and dataviews. The dataviews represent Join and Projection operations and are available both to the forms driven query/report generator and to the COBOL and FORTRAN programmer interface. The underlying disc manager is optimized for transaction processing with advanced data integrity features.

Interface Details:

Accessible (tuple-at-a-time) from COBOL and FORTRAN.

Documentation:

RELIANCE Manuals (5 volume set).

SESAM V 12.1, SESAM V 13

General Description:

SESAM is a commercially oriented Database System whose Data Structures are compatible with the Relational Modul. Its principle features are selection, (field-) projection, Join, update of Relations and Password Protection at Relation, Field, and Activity-level. The system can be accessed by Assembler, COBOL, FORTRAN, ALGOL, RPG and PLI user programs. User programs and DBMSs may be distributed at the users convenience within a computer network.

Single user program can work with several DBMSs simultaneously which may be local, remote, or both. A DBMS may also revive requests from both remote and local user programs).

Interfaces for TP monitors UTM and COSMOS are available. Several Report generators may also be used with SESAM.

Transactions of consistence level 3 with records (Primary Key) as the lock unit will be possible when Version 13 is released.

Current Applications: Approximately 200.

Documentation:

1. Software Product SESAM (BS2000/BS1000) V 12.1, Database Creation and Maintenance, Reference Manual Part 1, No. U 275-J-Z55-1-7600.

2. Software Product SESAM (BS2000/BS1000) V 12.1, Data Retrieval and Direct Update, Reference Manual Part 2, No. U276-J-Z55-1-7600.

3. SESAM auf dem Weg zum relationalen Datenbanksystem
   Part 1 Systemberater published 1.5.80
   Part 2 Systemberater published 1.3.80
   Part 3 Systemberater published 1.9.80
   (W. Schmittel, B. Nauer).

SYNTEX-II

General Description:

SYNTEX-2 is a relational DBMS whose DML is predicate calculus oriented with a mix of indivi-

dual and tuple variables. The DML can be used either in stand alone or embedded into the programming language COBOL. The internal mode relies upon two types of data structures: indexed files and rings. This internal model together with an efficient query optimizer enable good performance.

SYNTEX-2 is programmed into a low level language, LP 70 (very similar to PL 360) on a CII-HB IRIS 80 machine (of a capacity similar to IBM 370-158). The minimum size of core memory needed is 45 K-words (1 word = 32 bits).

Interface Details:

Accessible from COBOL.

Current Applications:

SYNTEX-2 is a prototype which was used for 5 different applications of various sizes, but it is not now under use.

Documentation:

1. "The language of SYNTEX-2 an implemented relational-like DBMS in: Information Technology" Ed.: J. Moneta, North-Holland, JCIT3, 1978.

2. "SYNTEX-2 un prototype de SGBD relationnel" Conf. CIPS-DPMA, Quebec, 1979.

SYSTEM-D

General Description:

SYSTEM D is an experimental, integrated tool for information systems design, implementation, and database management. It is under development at the University of Tampere. It can be used as a relational database management system but it can also be used as a tool for database design and data processing systems design. It provides a user an external view both on the conceptual level and on the relational level. It provides also a spectrum of different query languages. The database administrator (DBA) can see all four schema levels: conceptual, relational, data structure, and storage structure levels. The DBA has at its disposal a special language, called Data Administration Language, which can be used to design, define, and maintain the database. SYSTEM D has an Adaptive query optimizer which can use different sets of optimization rules in different situations. If information requirements of a user are formulated as a query, then SYSTEM D can be used as an automated design tool which produces a set of programs and optimized intermediate files which corresponds to the original query.

Interface Details:

Accessible from PASCAL AND FORTRAN.

Documentation:

H. Kangassalo, H. Jaakkola, K. Jarvelin, T. Lehtonen, and T. Niemi: SYSTEM D — An integrated tool for systems design, implementation, and database management. Report, University of Tampere, Department of Mathematical Sciences, June 1981.

VERSO

General Description:

This is a database machine on which we are implementing a relational system. The hardware is based on the idea of "on the fly" filtering of data. The DBMS offers a standard low level interface (Physical Management, Algebraic queries and update processing, concurrency control and recovery) but no protection on end user interface.

Other Details:

Supports concurrency and functional dependency checking.

Documentation:

1. "Design of an I/O Processor for a Relational Database Machine", SIGMOD80, Bancilhon/Scholl.

2. "The database Machine VERSO: Binary Operation", 6th Workshop on Nonnumeric Architecture, 1981 (Hyers).