IBM

International Systems Centers

IBM DATABASE 2
RELATIONAL CONCEPTS

GG24-1581

IBM DATABASE 2 Relational Concepts

Document Number GG24-1581-00

Goran Sandberg, IBM Sweden

International Systems Center - Santa Teresa
San Jose, California

This publication was produced using the
IBM Document Composition Facility
(Program number 5748-XX9) and
the master was printed on the IBM 3800 Printing Subsystem

First Edition (June 1983)

This edition applies to IBM DATABASE 2 (DB2) Release 1 (Program Number 5740-XYR), Query Management Facility (QMF) Release 1 (5668-972), and Data Extract (DXT) Release 1 (5668-973).

Requests for copies should be made to the IBM branch office that serves you.

Forms for reader's comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to:

IBM Corporation
International Systems Center - Santa Teresa
Department 471
P.O. Box 50020
San Jose, California 95150, U.S.A.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

ACKNOWLEDGEMENT

This document is the result of a residency conducted at the International Systems Center - Santa Teresa.

We would like to acknowledge the excellent work done by the author:

- Goran Sandberg, IBM Sweden

We would also like to thank the many people who have reviewed and commented on this document.

Peter Backlund
Wes Dayton
Colin White

International Systems Center - Santa Teresa
June 1983

[illegible] [illegible]
GG24-1594 DB2 [illegible]
GG24-1600 DB2 [illegible] + Tuning Guide

This document:

- **GG24-1581 IBM DATABASE 2 Relational Concepts**

  describes the relational approach to data base systems in general and to IBM DATABASE 2 (DB2) in particular. The relationship between IMS/VS DL/I and DB2 applications is also covered. The intended audience is DP Professionals who wish to understand the relational model of data and how it is implemented in DB2.

It is one in a series produced by the International Systems Center - Santa Teresa. Other documents in the series are:

- **GG24-1582 IBM DATABASE 2 Concepts and Facilities Guide**

  which gives a functional overview of the IBM DATABASE 2 relational data base management system. It is intended to be read by all DP Professionals who wish to obtain a good functional knowledge of the product.

- **GG24-1583 IBM DATABASE 2 SQL Usage Guide**

  which demonstrates the power of Structured Query Language (SQL), a data base management language which permits IBM DATABASE 2 users to access and manipulate data in relational data bases. The document is intended for DP Professionals who wish to obtain a good functional knowledge of SQL. It covers SQL by using a series of examples starting with the very basic and becoming increasingly complex.

LIST OF ILLUSTRATIONS

CHAPTER 1. INTRODUCTION

This document describes the relational approach to data base systems for those with a general understanding of data base concepts. IBM and customer personnel, familiar with the hierarchical data base approach used in IMS/VS, should find the document easy to read. However, knowledge of IMS/VS is not a prerequisite. The document contains the following chapters:

Chapter 2. **What is a Relational Data Base?**
Gives an overview of the most important ideas in relational data base systems.

Chapter 3. **The Relational Data Base Model**
Describes the basic components of a relational data base system, starting with the theory of the relational data base model.

Chapter 4. **How DB2 Implements Relational Concepts**
Describes how the relational data base model that was introduced in chapter 3 is implemented in DB2.

Chapter 5. **Relational and Hierarchical**
Compares key characteristics of the relational approach with a hierarchical approach. Comparisons are also made here between the IMS/VS implementation of the hierarchical model and the DB2 implementation of relational model.

Chapter 6. **Advantages of a Relational Data Base System**
Identifies the main advantages that could be achieved with the relational approach. Comparable advantages of the hierarchical approach are also summarized.

Chapter 7. **Relational Applications**
Describes typical characteristics of applications that are most suited to be implemented with a relational data base system because of the advantages identified in Chapter 6.

THE MOTIVATION FOR RELATIONAL DATA BASE SYSTEMS

What is the motivation for a new data base system? What is the driving force that leads to new alternative ways to structure and to manipulate a data base? For the development of relational data base systems the most important factor has been the striving for **simplification** in the ways a data base system is used.

Relational data base systems have been developed with the following main objectives

- Simple concepts

  Basic data base concepts must be simplified so that many and different users will find them more easy to understand.

- Few, well-defined objects

  The number of data base terms that various users must understand should be kept as low as possible. More important, logically separate objects should be kept separate so that the same object is not used for different purposes in different contexts.

- High level language

  Ease-of-use and simplicity should be accomplished by providing a high level language to operate on data as compared to today's quite detailed data base languages. It should be possible to express data base requests more in terms of what the end-result should be rather than how this result technically should be produced.

- Well defined operations on data which have a theoretical foundation

  The possible operations on data should be based on a set of basic common principles that apply to all operations. Such a theoretical foundation will make it possible to predict the outcome of all potential operations.

- Simplified data definition and design

  The process of designing and defining new data bases should be simplified. There should be means to iteratively correct, change and complete earlier design in an on-going gradual process.

- Simplified installation and operation

  The basic principle of simplified user interfaces to data should also be carried over to installation and operation of the data base management system itself.

These are the overall objectives that have been of the greatest importance in the development of relational data base systems. These general objectives may not always be fully realized in all individual implementations. In fact, implementations of relational data base systems will most probably differ with respect to how they succeed in providing such improved ease-of-use facilities. However, the overall principle of increased simplicity is the single most important objective for relational data base systems.

RELATIONS - TABLES, COLUMNS AND ROWS

The most fundamental property of a relational data base system is that data is presented to the user as **tables** - the mathematical name for a table with unique rows is a **relation** - and that the system provides suitable operators for the manipulation of these tables.

An example of a relational data structure is:

TELEPHONE DIRECTORY

LASTNAME	WORKDEPT	PHONENO
Smith	E11	2095
Spenser	E21	0972
Geyer	E01	6789
Perez	D21	9001
Haas	A00	3978
Johnson	D21	8953
Nicholls	C01	1793
Thompson	B01	3476
Lutz	D11	0672
Pulaski	D21	7831
Setright	E11	3332
Stern	D11	6423

Figure 1. A telephone directory

This example shows only a single table. The table is a telephone directory containing name, workdepartment and telephone number. There is one row for each employee. Each row has three values - one for each column in the table.

A real relational data base is composed of many different tables, not just one. Thus, a relational data base appears as a collection of tables, each consisting of columns and rows. The rows in a table correspond to records in a file and the columns correspond to the fields within such records. In order for the analogy with a file to be meaningful such a file should have records of one single type only, all occurrences having the same record structure and layout.

The alternatives to presenting data as tables are to present data in the form of hierarchies (as DL/I) or in the form of networks (as DBTG[1]). A main difference between these alternatives is that in the case of network or hierarchical data structures, the relationships between data are to some extent represented by the structure itself - in a network through Owner-Member links, in a hierarchy through Parent-Child links. In relational data structures all relationships between tables are represented purely by the values in the tables.

OPERATIONS ON TABLES

In addition to providing a structure for data in the form of tables with rows and columns, a relational data base system also includes a language with a set of operators to access and update such tables. These relational data base operators deal exclusively with rows and columns. Operations on tables will directly reference fields by their values as the only means to access the data in the tables. Operations on tables do not depend on relationships that are implied in the data structure - like one segment type being dependent on another segment type.

A relational data base language provides facilities for at least the following types of operations

[1] DBTG is a generic name used for the specifications of a network view of data as defined by committees within the CODASYL organisation. DBTG stands for Data Base Task Group - the original working group within CODASYL that came up with a set of specifications for network data base systems in 1971.

- Retrieval from a single table

  A subset of rows from a given table or a subset of columns from a given table, or both, is selected.

- Combining multiple tables

  Rows from different tables are combined into new tables based on common fields. This is a crucial facility of a relational data base system.

- Modification of tables

  Individual rows or a group of rows are inserted, deleted or updated.

- Definition of tables

  A data definition language is provided for defining new tables, and changing or deleting existing tables. This part of the language also includes the functions necessary for the data base administrator to define and maintain the storage oriented aspects of the data bases.

DISTINCTION BETWEEN DATA BASE DESIGN AND RELATIONAL DATA BASE SYSTEMS

Quite often the following two separate areas of data base management are confused:

- Data base design

- Relational data base management system implementations

A relational data base management system is the software package that will provide facilities to access and update tables. As such it includes a language to define, access and update the databases, storage methods to maintain data on disk, utility functions, concurrency control facilities and various service functions.

Data base design, on the other hand, is the task of structuring data in the form of record structures. This process is independent from the selection of which data base management system will later be used to store the data. Data base design is therefore a separate area of data base concepts.

Data base design theories have been somewhat formalized in the so called **Normalization** process. This process deals with how fields should be grouped into records, when a record type should be split into multiple different record types, and when many record types could be combined into one. These theories are often presented with much mathematical formalism and therefore often appear unnecessarily complex.

One does not have to understand normalization theory in order to understand relational data base systems. Normalization applies also when the record structures are subsequently adopted to a particular data base management system which is not a relational data base system. Normalization is sound design practice that should reduce future update and maintenance problems and is **independent** of the data base management system itself. However, as will later be shown, the process of adopting a given data base design to a relational data base management system may be simpler than adopting it to other types of data base systems.

IMPLEMENTATIONS OF RELATIONAL DATA BASE SYSTEMS

Several implementations of relational data base systems are offered commercially at this time. The most significant offerings provided by IBM are:

- DB2

  The relational data base system available for IMS/VS, CICS/VS and TSO users under the MVS operating system. The query interface for end-user (as opposed to programmer, DBA etc.) access to DB2 is QMF (Query Management Facility).

- SQL/DS

  The relational data base system available for CICS/DOS/VSE users.

- QBE

  The relational data base system available as an IUP (Installed User Program) in the VM/CMS environment. QBE stands for Query-By-Example.

Many research prototypes have been developed at universities and research institutions. Among those the most widely known is probably System R developed by IBM at the San Jose Research Laboratory. This prototype has provided much of the experience and groundwork for both SQL/DS and DB2.

Another prototype is the Peterlee Relational Test Vehicle (PRTV), developed in Peterlee UK, which later became a main component of a product called Interactive Management and Planning System (IMPS).

TERMINOLOGY USED IN RELATIONAL DATA BASE SYSTEMS

Often quite different terminologies are used when relational data base systems are discussed in different environments. Roughly speaking the following approximate equivalences could be used.

Table = record type or segment type or relation

Row = record occurrence or segment occurrence or tuple

Column = field type or data element type or attribute

Relation is a term that is obtained from mathematical disciplines. In principle it has the same meaning as a table.

Other terms that are often found in the theoretical literature on relational data bases are:

- Binary relation = table with two columns.
- N-ary relation = table with N columns.
- N-tuple = a record from a table with N columns.
- Degree = number of columns in a table.
- Cardinality = number of rows in a table.
- Domain = the total collection of all values that may occur for a given column.

  There is really no alternative term for domain but it has the following meaning. All values that may occur for a specific field type come from a domain of all the possible values of this type. Many different field types may use the same domain. An example of a possible domain would be that of all possible telephone numbers. At any given time only some of these possible numbers will actually occur in tables.

Throughout this document the terms **table** and **relation** will be used interchangeably to mean the same thing - a data structure of columns and rows.

The following discussion is intended to give some background on the **theoretical basis** of the relational data base model, in order to better understand DB2. It is not intended to replace a more rigorous treatment of the foundations of the relational theory as can be found in other documents.

The relational database model has three main components

- A way to structure data in the form of tables
- The operators by which one can operate on these tables
- The facilities that will provide data consistency

Each of these three areas will be covered as a separate section in this chapter.

RELATIONAL DATA STRUCTURES - TABLES

The relational database structure is one where data is externally structured in tables. It should be remembered that this applies only to the external level - to programmers and interactive users that use the data directly from a terminal. Internally, the data base management system will have quite different methods for storing and accessing data. The storage methods and access techniques are not part of the relational data base model.

The basic rules for Tables

Table data structures are just what one could intuitively expect - data organized in rows, these rows having one value for each column. The fact that table structures are something with which most people are familiar, and which have the same meaning for different people, is very important for the ease-of-use aspect of relational data base systems.

An example of a table structure is:

EMPLOYEE

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SEX	SALARY
000260	Johnson	D21	8953	52	16	F	17250
000270	Perez	D21	9001	55	15	F	27380
000050	Geyer	E01	6789	58	16	M	40175
000070	Pulaski	D21	7831	56	16	F	36170
000100	Spenser	E21	0972	54	14	M	26150
000140	Nicholls	C01	1793	56	18	F	28420
000150	Adamson	D11	4510	55	16	M	25280
000310	Setright	E11	3332	46	12	F	15900
000220	Lutz	D11	0672	55	18	F	29840
000060	Stern	D11	6423	55	16	M	32250
000300	Smith	E11	2095	48	14	M	17750
000020	Thompson	B01	3476	61	18	M	41250
000030	Kwan	C01	4738	60	20	F	38250
000010	Haas	A00	3978	66	18	F	52750

Figure 2. The Employee table

A table data structure is very much like a file structure if one adds the requirement that all the records in such a file should be structured in exactly the same way - that the records should all have the same fields in the same sequence.

Therefore tables could be regarded as more disciplined files. In addition to the property that all rows in a table have the same format, table structures must also adhere to the following basic rules:

- All data is represented by values

  Data is not represented by pointers, links or other connections. All data in the tables is represented by values which appear in a fixed column position in a row.

- An entry in a column position in a row is single valued

  This means that from the systems point of view there is only one single value for a certain column position in a row. This single value could be used to compare with other values in other columns in the same or in different tables.

- No duplicate rows occur in a table

  Every row in a table is unique.

- No duplicate column names exist within a table

  A column name appears only once in a table.

- The order of the rows is not significant

  Rows do not exist in a table in any specific order, they just belong to the table. However when rows are retrieved from tables they may be presented in an order convenient to the user.

- The order of the columns is not significant

  The order in which the columns occur is not important because columns are identified by their names not by an implied column-order.

- All relationships exist through values

  Associations between tables exist only because values can be matched in the different tables. There is no other implied link mechanism such as a predefined data structure.

These are the basic rules for table structures from relational data base theory. It should be noted that not all these aspects are of equal importance. An implementation of a relational data base system may provide variations or exceptions to these basic rules. For instance, it may be convenient to allow duplicate rows to occur as a default rule and, in addition, provide other means by which duplicates could be prevented where necessary.

Domains

A domain is the totality of all existing and possible values for one or more columns. So in addition to currently assigned values a domain also includes all possible values that may ever occur in these columns but do not currently occur. Many of those possible values might never be assigned as actual values in the columns of the table. But they are still part of the domain.

It should be noted that a domain is an abstract concept - a domain is not physically stored in a data base.

An example of a possible domain would be that of the employee numbers (EMPNO) in the EMPLOYEE table example of Figure 2 on page 10. The same domain will also be used for manager numbers (MGRNO) in the Department table below - a manager number is just an employee number for a manager.

For the DEPARTMENT table there would also be a single domain for all the possible department numbers (DEPTNO) as well as all administrative department numbers (ADMRDEPT).

DEPARTMENT

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
C01	Info Center	000030	A00
E11	Operations	000090	E01
D01	Dev Center	<>	A00
E01	Support Services	000050	A00
D21	Admin	000070	D01
A00	Spiffy Computer Co	000010	<>
B01	Planning	000020	A00
D31	Order Processing	<>	D01
E21	Software Support	000100	E01
D11	Manufacturing	000060	D01

<> = Null

Figure 3. The DEPARTMENT table

The concept of domains is important when operations on tables are considered. Table operations are often performed through matching of field values from different columns in different tables. Most operations are only meaningful if the columns involved all have the same domain.

Keys

A key is a column - or a combination of multiple columns - that distinguishes a specific row from all other rows. A key provides uniqueness for that row. In some tables it may be required to combine all columns in order to provide the key for that table. More often a single column will be enough to serve as the key for the table.

There may be more than one column that can serve as a possible key. Such columns - or combination of columns - are then called **candidate keys**. One of all possible candidate keys is chosen as the **primary key**. The remaining candidate keys - if any - are called **alternate keys**.

When a primary key from a table is used in another table, the key is referred to as a **foreign key** in the second table.

In the Department table Figure 3 **DEPTNO** will most likely be the primary key for that table. Whenever **DEPTNO** is used in another table it is a foreign key (e.g. **WORKDEPT** in the Employee table). **DEPTNAME** may or may not be a candidate key depending on whether or not the department name always uniquely identifies a row in the department table or not. **MGRNO** is another possible candidate key for the department table, but in this case the column would no longer be allowed to be null.

Null values

A null value means that there is no value assigned. In our examples this is shown as <>.

A possible reason for having a null value in a column for some row may be that the value is just not known at the time the row is created. Another reason can be that in some cases a column value simply does not apply, so no value could be assigned for a specific row. In the DEPARTMENT table Figure 3 on page 12 some departments do not have a manager assigned to them. In these cases the MGRNO has a null value.

Sometimes one might not want to allow null values for a certain column. Primary keys must not be allowed to take on null values. But also for fields other than key fields it may be desirable to disallow null values in order to enforce an application rule. An example would be that of an application requiring that all employees be assigned to a department. If so, **WORKDEPT** in the Employee table Figure 2 on page 10 should not be allowed to have null values.

Duplicate rows

A basic principle for the relational data base model is that duplicate rows should not exist in tables. All rows in a table should be unique.

Implementations of relational data base systems may however provide facilities to handle duplicates. There would then be language facilities to explicitly request if duplicates should be prevented or not.

RELATIONAL OPERATIONS

With the data structured in the form of tables, operators are needed to access and manipulate these tables. The table operators are as much part of the relational data base model as are the data structures themselves.

A general and very important property of all relational data base operations that will be described below is that the result of each operation is itself a table. This means that the result of one operation could be the input to another operation. It also means that any complete language statement could be seen as a table specification in itself. Some implementations, for instance DB2, make significant use of this basic property (in allowing virtual tables, views, to be defined).

Note: Where applicable, the following examples will be coded using DB2 SQL.

Retrieve a subset of one or more rows from a table

The basic operation is to obtain a subset of a given table.

- <u>**A subset of the rows**</u>

  This operation is called a **selection** (or **restriction**) on the table. By specifying a search criteria only some of the rows in a table are selected for the result table.

```
SELECT DISTINCT
 EMPNO,LASTNAME,WORKDEPT,PHONENO,JOBCODE,EDUCLVL,SEX,SALARY
FROM EMPLOYEE
WHERE WORKDEPT = 'D11'
```

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SEX	SALARY
000150	Adamson	D11	4510	55	16	M	25280
000060	Stern	D11	6423	55	16	M	32250
000220	Lutz	D11	0672	55	18	F	29840

Figure 4. Selection

- **A subset of the columns**

  This operation is called a **projection** on the table. The result is a table with fewer columns than the original table. All rows still remain except that duplicates in the result table are eliminated. Duplicates may occur in the result if the remaining columns do not contain a candidate key (so that they would be guaranteed to be unique).

```
SELECT DISTINCT EMPNO,LASTNAME,WORKDEPT
FROM EMPLOYEE
```

EMPNO	LASTNAME	WORKDEPT
000030	Kwan	C01
000310	Setright	E11
000050	Geyer	E01
000060	Stern	D11
000070	Pulaski	D21
000100	Spenser	E21
000300	Smith	E11
000140	Nicholls	C01
000010	Haas	A00
000220	Lutz	D11
000260	Johnson	D21
000020	Thompson	B01
000270	Perez	D21
000150	Adamson	D11

Figure 5. Projection

- **Subsets of both rows and columns**

  So far the individual operation of selection and the individual operation of projection have been described. The most common and also most important case is however when both these two operations are combined into the same single data base request. That is, a row subset and a column subset are requested at the same time.

```
SELECT DISTINCT EMPNO,LASTNAME,WORKDEPT
FROM EMPLOYEE
WHERE WORKDEPT = 'D11'
```

EMPNO	LASTNAME	WORKDEPT
000060	Stern	D11
000220	Lutz	D11
000150	Adamson	D11

Figure 6. Selection and Projection in the same request

Retrieval from multiple tables - Join

A join takes as input two tables which are matched based on one column in each table. For all matching combinations of rows, a result row is created by placing together (concatenating) all the columns from the two matching rows.

```
SELECT DISTINCT
 EMPNO,LASTNAME,DEPTNO,DEPTNAME,MGRNO,ADMRDEPT
FROM EMPLOYEE,DEPARTMENT
WHERE WORKDEPT = DEPTNO
```

EMPNO	LASTNAME	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
000220	Lutz	D11	Manufacturing	000060	D01
000030	Kwan	C01	Info Center	000030	A00
000070	Pulaski	D21	Admin	000070	D01
000270	Perez	D21	Admin	000070	D01
000100	Spenser	E21	Software Support	000100	E01
000140	Nicholls	C01	Info Center	000030	A00
000150	Adamson	D11	Manufacturing	000060	D01
000010	Haas	A00	Spiffy Computer Co	000010	<>
000020	Thompson	B01	Planning	000020	A00
000260	Johnson	D21	Admin	000070	D01
000300	Smith	E11	Operations	000090	E01
000050	Geyer	E01	Support Services	000050	A00
000060	Stern	D11	Manufacturing	000060	D01
000310	Setright	E11	Operations	000090	E01

Figure 7. Join on equal condition

This is a join on an equal condition, often referred to as a **natural join**. In this case, for each row in the **Department table**, (Figure 3 on page 12) all rows in the **Employee table** (Figure 2 on page 10) are checked for a row with a matching department number value. When found, the two rows are put together into one row. The result contains the columns from both tables except that only one of the two department number columns is kept (the values being identical because they matched).

The join operation could also be described as follows:

- First build a table with all possible combinations of rows , e.g. each row from the first table is combined with every row from the second table. This is also called the **product** of the two tables.

- Then select from the result those rows where WORKDEPT is equal to DEPTNO.

- Finally project the result on all columns that were requested.

In relational data base systems, relationships among data are not represented in the data structures - as is the case in hierarchical data structures. Instead all relationships are carried by the values themselves as they appear in individual columns and rows. In relational data bases the mechanism to make use of relationships among data is the **Join** operator. The Join operator allows rows from different tables to be combined - the Join operator effectively replaces relationships carried as links in hierarchical structures. Therefore the join operator is one of the most important functions in a relational data base system.

A very important consequence is that relationships no longer have to be defined in advance in the data structure - the Join operator provides an access path when it is executed. It is from this basic fact that a substantial part of the advantages of relational data base systems are derived. We do not have to 'hard-wire' into the data structures all the relationships that we could possibly need - they are present as optionally available access paths through the actual values that exist in the tables.

This whole discussion applies to the logical level of data access. At the physical level the most often used access paths are often 'hard-wired' to provide optimum performance - in relational data base systems as well as in hierarchical data base systems.

Variations of relational operations

- <u>**Join on 'non-equal' conditions**</u>

  The natural join operation described above is the most common type of join.

  A variation of the natural join is to join on other conditions than that values should be equal in two columns. One could specify conditions that values should be greater than, less than, not equal to, etc. For a **greater-than-join**, each row from one table is concatenated with every row from another table which has a greater value in the joined column.

- <u>**Join a table with itself**</u>

  The join operation can also be applied to one single table - a table can be joined with itself.

  An example of this would be the case where the Department table is joined with itself on the **DEPTNO** and **ADMRDEPT** columns in order to produce a list of numbers and names for all the departments that are also administrating departments.

Another example would be the repeated joining of the Department table with itself to produce a department chart in tabular form. This repetition of the join is possible since, as was noted earlier, the result of a relational operation is itself a table.

- **Operations with null values and missing data**

  An additional class of relational operations is where the basic operations are extended to include special treatment of null values and missing rows. Such operations are called **outer** relational operations.

  The special case of an outer natural join is where a result row is created also on the additional condition that no matching row is found. This is in addition to all the result rows that were created because matching rows were found.

  In the natural join, when a row from one table does not have any matching row in the other table, then simply no result row is created. In the outer natural join, an extra result row **is** created when no matching row at all is found. This result row would contain null values for the columns from the table with the missing row.

  An example of an outer join would be to produce a list of all departments with their employees. The list should also include those departments for which there are no employee.

  Note: You will have noticed that up to this point, the rows presented in the examples have not been in any particular order. This is in keeping with a basic assumption of the relational data base model that row order is not significant. This assumption will be discussed further. However, some of the examples that follow will be presented in a specific order merely to make it easier to illustrate the point being made.

DEPTNO	DEPTNAME	EMPNO	LASTNAME
A00	Spiffy Computer Co	000010	Haas
B01	Planning	000020	Thompson
C01	Info Center	000030	Kwan
C01	Info Center	000140	Nicholls
D01	Dev Center	<>	<>
D11	Manufacturing	000060	Stern
D11	Manufacturing	000150	Adamson
D11	Manufacturing	000220	Lutz
D21	Admin	000070	Pulaski
D21	Admin	000260	Johnson
D21	Admin	000270	Perez
D31	Order processing	<>	<>
E01	Support Services	000050	Geyer
E11	Operations	000300	Smith
E11	Operations	000310	Setright
E21	Software Support	000100	Spencer

Figure 8. Outer join: The Employee and the Department tables are joined on the WORKDEPT and the DEPTNO columns and the result ordered by DEPTNO. Unique for the outer join is the additional condition that when no matching Employee row is found for a specific Department row, then a result row should be created with null values in the EMPNO and LASTNAME columns, as for D01 and D31.

If a natural join had been specified, there would not have been any Department rows for D01 or D31 in the result because there are no rows in the Employee table for employees with this work department number.

Table operations from set theory

When relational data base systems first appeared it was as a result of applying mathematical set operations to firmly structured data processing files - tables.

The next three operations - intersection, union and difference - are well defined set theory operations. They apply to some extent to operations on tables as well. However, these operations are not as useful in data processing environments as are the basic table operations described above - the selection, projection and join operations. Therefore these operations are not always implemented in relational data base systems.

A common property of these operations is that they apply only when the tables involved all have the same number of columns, and all corresponding columns have the same domain.

For illustrative purposes let us select two subsets of the Employee table. One subset being all female employees, the other being all employees with jobcode ≥ 56. Let us then project these two tables on the EMPNO, LASTNAME and WORKDEPT columns.

These two tables now have the same number of columns and all corresponding columns have the same domains.

Female Employees

EMPNO	LASTNAME	WORKDEPT
000010	Haas	A00
000030	Kwan	C01
000070	Pulaski	D21
000140	Nicholls	C01
000220	Lutz	D11
000260	Johnson	D21
000270	Perez	D21
000310	Setright	E11

Employees where JOBCODE ≥ 56

EMPNO	LASTNAME	WORKDEPT
000010	Haas	A00
000020	Thompson	B01
000030	Kwan	C01
000050	Geyer	E01
000060	Stern	D11
000140	Nicholls	C01

Figure 9. Two subsets of the EMPLOYEE table

- **Intersection**

  An intersection of two tables is a table containing the rows that are present in both tables.

EMPNO	LASTNAME	WORKDEPT
000010	Haas	A00
000030	Kwan	C01
000140	Nicholls	C01

Figure 10. Intersection: of Female Employees with Employees where jobcode ≥ 56

- **Union**

  A union of two tables is the set of all rows belonging to either the first table or the second table or both.

  Two tables, or the results of operations on two tables are concatenated. Duplicates are then eliminated.

EMPNO	LASTNAME	WORKDEPT
000010	Haas	A00
000030	Kwan	C01
000070	Pulaski	D21
000140	Nicholls	C01
000220	Lutz	D11
000260	Johnson	D21
000270	Perez	D21
000310	Setright	E11
000020	Thompson	B01
000050	Geyer	E01
000060	Stern	D11

Figure 11. Union: of Female Employees with Employees where jobcode ≥ 56

- **Difference**

  The difference between two tables are the rows from the first table that do not occur in the second table. The order in which the two tables are specified is therefore significant.

EMPNO	LASTNAME	WORKDEPT
000070	Pulaski	D21
000220	Lutz	D11
000260	Johnson	D21
000270	Perez	D21
000310	Setright	E11

Figure 12. Difference: of Female Employees with Employees where jobcode ≥ 56

Insertion

So far various retrieval operations on tables have been described. A language to operate on tables must also contain operators to insert new rows into a table, delete rows from tables and modify column values in specific rows (update of rows). Each of these operations will be considered separately starting with the insert operation.

```
INSERT INTO EMPLOYEE
    (EMPNO,LASTNAME,JOBCODE,SEX)
    ('000390','Barclay',52,'M')
```

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SEX	SALARY
000390	Barclay	<>	<>	52	<>	M	<>
000010	Haas	A00	3978	66	18	F	52750
000020	Thompson	B01	3476	61	18	M	41250
000030	Kwan	C01	4738	60	20	F	38250
000050	Geyer	E01	6789	58	16	M	40175
000060	Stern	D11	6423	55	16	M	32250
000070	Pulaski	D21	7831	56	16	F	36170
000100	Spenser	E21	0972	54	14	M	26150
000140	Nicholls	C01	1793	56	18	F	28420
000150	Adamson	D11	4510	55	16	M	25280
000220	Lutz	D11	0672	55	18	F	29840
000260	Johnson	D21	8953	52	16	F	17250
000270	Perez	D21	9001	55	15	F	27380
000300	Smith	E11	2095	48	14	M	17750
000310	Setright	E11	3332	43	12	F	15900

Figure 13. The Employee table with an additional row

The example shows the inserted new row as the first row of the table. However, the position of a specific row does not have any significance to a user. A row just belongs to the table.

New rows that are inserted may or may not contain null values for some of its columns. However, they should always contain at least the primary key.

Often the insert operation specifies insertion of a single new row. In other situations the insert may specify insertion of all the rows that have been retrieved by some other table operation. This is called a set insert.

```
INSERT INTO TEMPORARY
        (EMPNO,LASTNAME,WORKDEPT,PHONENO,
         JOBCODE,EDUCLVL,SEX,SALARY)
  SELECT EMPNO,LASTNAME,WORKDEPT,PHONENO,
         JOBCODE,EDUCLVL,SEX,SALARY
  FROM EMPLOYEE
  WHERE WORKDEPT = 'D11'
```

TEMPORARY

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SEX	SALARY
000060	Stern	D11	6423	55	16	M	32250
000150	Adamson	D11	4510	55	16	M	25280
000220	Lutz	D11	0672	55	18	F	29840

Figure 14. Insert an entire set of rows

Deletion

A delete operation follows the same form as a retrieval or insert operation. The number of rows that are deleted is dependent on the search criteria. Deletion of a single row is a special case of the general set delete operation.

```
DELETE
FROM DEPARTMENT
WHERE ADMRDEPT = 'D01'
```

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	Spiffy Computer Co	000010	<>
B01	Planning	000020	A00
C01	Info Center	000030	A00
D01	Dev Center	<>	A00
E01	Support Services	000050	A00
E11	Operations	000090	E01
E21	Software Support	000100	E01

Figure 15. The Department table after a delete

Update

Individual column values in specific rows may be modified. This is called update.

Update operations apply for a set of rows just as the other table operators. The search criteria specifies which rows should be updated. The update operation itself applies uniformly to all rows selected. Individual rows are not handled differently. Updating a single row is just a special case of update where the selection criteria will select only a single row.

```
UPDATE EMPLOYEE
SET JOBCODE = 56
WHERE JOBCODE = 55
```

EMPNO	LASTNAME	WORKDEPT	PHONENO	JOBCODE	EDUCLVL	SEX	SALARY
000010	Haas	A00	3978	66	18	F	52750
000020	Thompson	B01	3476	61	18	M	41250
000030	Kwan	C01	4738	60	20	F	38250
000050	Geyer	E01	6789	58	16	M	40175
000060	Stern	D11	6423	56	16	M	32250
000070	Pulaski	D21	7831	56	16	F	36170
000100	Spenser	E21	0972	54	14	M	26150
000140	Nicholls	C01	1793	56	18	F	28420
000150	Adamson	D11	4510	56	16	M	25280
000220	Lutz	D11	0672	56	18	F	29840
000260	Johnson	D21	8953	52	16	F	17250
000270	Perez	D21	9001	56	15	F	27380
000300	Smith	E11	2095	48	14	M	17750
000310	Setright	E11	3332	43	12	F	15900

Figure 16. Update

A special form of update is to give null values to certain columns. This in effect deletes a single field value without deleting the entire row.

Certain considerations apply to how keys could be updated. Implementations of relational data base systems may differ with respect to whether key updates are allowed. In some cases a key column is not allowed to be updated. Instead that row must first be deleted and then re-inserted with a new key value.

Ordering of Table Rows

The basic assumption in the relational data base model is that table rows do not have a significant order. This however does not mean that rows could not be presented in order. On the contrary, languages to operate on tables do normally provide operators which specify that the output should be ordered.

The only real requirement is that the language operators do not depend on a specific row storage order. This allows all access to be accomplished by value association only. No access is dependent on a special position in the data base or on a 'navigation' through its contents.

Incidentally, the placement of this section on ordering under the heading of relational operations rather than under the heading of relational data structures is indicative of where it applies. Ordering is not of significance when rows are stored in the data base but instead when data is presented to its users.

In order to achieve high performance in data access it is as desirable for a relational data base system as for any other data base system to physically order data (cluster the data) according to the most used access pattern. Clustering of data in a desirable physical stored order for performance reasons is however a separate (but very important) internal consideration. It is not part of the external relational data base model.

DATA CONSISTENCY IN RELATIONAL DATA BASES

In addition to the data structures - tables with rows and columns - and the language to operate on such tables, a relational data base system also includes the facilities to control the consistency of data in tables. Such capabilities are sometimes also called integrity capabilities, although integrity also applies to how the data base management system maintains physical integrity of data. Essentially the data consistency areas of relational data base systems deal with how references between tables are kept consistent and how modification of relational data base tables are allowed to take place.

Consistency facilities in general can get very complex to implement and implementations usually provide limited facilities in this area. Consistency facilities can take many different forms. The following are some important areas for consistency.

- **Key consistency**

  Keys should always be required. Keys or any part of them should not be allowed to take on null values.

- **Consistency in data references**

  All **references** to a primary key value should be consistent. This means that all foreign key values should normally only reference existing primary key values. The data base may not be in a valid (legal), consistent state if this condition is not satisfied.

As an example, if DEPTNO is the primary key for a department table, then references to that key (e.g. WORKDEPT in employee table) might require that the corresponding DEPTNO should exist in the department table.

For **insert operations** this may mean that before a new row that contains a foreign key value is inserted, it may be required that a row in another table with that primary key value already exists. Otherwise the insert operation should be rejected. Or a row with primary key should also be inserted. An example would be that insertion of a new Employee row (that contains a Department-number value as a foreign key) would not be allowed if the corresponding Department row does not already exist.

For **delete** operations there are also alternative actions which are possible in order to preserve data consistency.

When a row in one table is deleted then all appearances of its key that may exist in other tables - foreign key occurrences for this primary key - might also have to be deleted. Either the foreign key values should be updated to null or entire rows with this key value in other tables should be deleted (cascading delete). An example of cascading delete would be that if a Project row is deleted from a Project table then also all sub-project rows should be deleted as well. Alternatively the delete operation could be rejected if other rows with this key value do exist. A Department row may not be allowed to be deleted as long as there are Employees still assigned to this Department.

On **update operations** there are also alternative actions possible. Depending on if the new value already exists as a key in another table the update could be rejected or accepted. If a primary key value is updated it may be necessary either to set to null all existing foreign key occurrences that have the old value, or to update all such foreign keys to the new value as well, or to reject the update.

Data consistency may also extend to other fields than primary and foreign keys in the data base. It might be required that certain conditions should not be allowed to occur in the data base. As an example the value in a salary column might not be allowed to be negative, or that condition should cause some procedure or program to be initiated.

In general consistency functions can get very complex to implement. Implementations of relational data base systems do not always provide facilities for all functions identified here.

THE RELATIONAL PROCESSING CAPABILITY

Up to now this chapter has presented the basic components of that type of data model called relational. As seen, the basic components of any data model are:

- A structural part

  The data structures as externally seen by the user.

- A manipulative part

  The operators that could be used on the data structures.

- A data consistency part

  The integrity or consistency rules for the data base.

An implementation of a relational data base management system will include capabilities within these areas to varying degrees. A data base management system will also include many other capabilities that do not have a direct counterpart in the underlying theoretical model.

The following general characteristics will help to distinguish a relational data base management system from other types of data base management systems.

1. A relational data base system uses tables for its data structures

   All relationships exist through values in tables. These tables do not have any user-visible navigation links.

2. A relational data base system uses set level operators to access tables

   A relational data base system provides set level operators including at least the basic capabilities of **selection, projection and join** without requiring either pre-definition of access paths and/or iterations within programs.

A data base management system that provides table data structures but not set level selection, projection and join may best be characterized as a **tabular** system.

A system that has both the above capabilities (1 and 2) is said to have the **relational processing capability** and can therefore be called a relational data base management system.

Still within this broad categorization there is a actually a wide spectrum of more or less independent functions that a relational data base management system may provide.

CHAPTER 4. HOW DB2 IMPLEMENTS RELATIONAL CONCEPTS

This chapter will describe how DB2 implements the relational data base concepts that have been described in the previous chapter.

DB2 is a relational data base management system - it has the relational processing capability. It uses tables as its data structures externally and a data manipulation language - the Structured Query Language or SQL - to access and update tables with set level operators including selection, projection and join. The SQL language is available both in an interactive environment from a TSO/SPF terminal and as an extension to COBOL, PL/I, FORTRAN or Assembler Language.

The purpose of this chapter is primarily to determine how concepts in the relational data base model map to the DB2 implementation.

TABLE DATA STRUCTURES IN DB2

Data in DB2 is externally structured as tables. Internally DB2 will use many different storage techniques which however do not appear externally. The external appearance of a table whose data is actually stored in the data base is called a **base table**. A view is also a representation of one or more tables, but is merely a description of a potential table that will be realized when that description is executed.

Base tables

A base table has a name and one or more columns. A column - which is also sometimes called a field type - is defined by a column name, a data type and a specification of whether null values are allowed or not. The data type could be one of character, decimal, floating point or integer.

Null values in DB2 have a special internal representation different from all externally allowed values. In the examples throughout this document the symbol <> is used to symbolize a null value.

Domains are not explicitly defined for columns in DB2. This means that the DB2 system itself has no knowledge of which columns are meaningful to be matched with each other.

As a default rule duplicate rows are allowed in tables, though not recommended. Duplicate rows can however be prevented by an explicit specification of a UNIQUE INDEX for one or more columns.

A relational data base request may also produce duplicate rows in the result when only some columns are selected. If duplicates are not desirable in this situation then the query could specify the keyword DISTINCT which will eliminate all duplicates from the result. An example would be a request for Jobcode, Education level and Sex from the Employee table as follows

```
SELECT DISTINCT JOBCODE,EDUCLVL,SEX
FROM EMPLOYEE
ORDER BY JOBCODE
```

JOBCODE	EDUCLVL	SEX
66	18	F
61	18	M
60	20	F
58	16	M
56	16	F
56	18	F
55	16	M
55	18	F
55	15	F
54	14	M
52	16	F
48	14	M
46	12	F

Figure 17. Duplicates eliminated

Had DISTINCT not been specified two rows for Jobcode 55, Education level 16 and Sex M would have been included in the result.

Views

All operations on tables will always create tables as a result. This is a consequence of the definition of relational operators. Most any specification of a data base retrieval request could be stored in the system as a definition of a special type of table - a **view table**. Views can be thought of as an alternative representation of data in base tables with the following characteristics:

- A view can be a subset of rows from a base table
- A view can be a subset of columns from a base table
- A view can include data from multiple tables - joined tables
- A view can include derived, calculated fields
- A view can be defined in terms of other views

Data in a view is not stored as a separate set of rows but is built from the underlying base tables when the view is used in SQL.

When views are defined in terms of other views this is called **views-on-views**. Such nesting of view references can be done to any depth.

Because a view is a stored specification of a data base request, a view can contain data from multiple different tables by the use of join operators in the view specification. As an example the Join of the Employee table with the Department table in Figure 7 on page 17 could have been defined as a view and therefore referenced in other SQL statements as if it were a table.

View definitions are a very powerful and important facility of DB2. They provide an additional level of data independence and view definitions therefore have many possible applications.

The following is an example of a possible view of the Employee table. The rows where Jobcode ≤ 59 is an example of a row subset which provides a security-by-content capability. Elimination of all other columns than the Workdept column is an example of column subsets. The Average-of-salary and Count-of-rows are examples of built-in functions.

```
CREATE VIEW DEPTSALARY AS
 SELECT WORKDEPT,AVG(SALARY),COUNT(DISTINCT EMPNO)
 FROM EMPLOYEE
 WHERE JOBCODE <= 59
 GROUP BY WORKDEPT
```

DEPTSALARY

WORKDEPT	AVG(SALARY)	COUNT(EMPNO)
C01	28420	1
D11	29123	3
D21	26933	3
E01	40175	1
E11	16825	2
E21	26150	1

Figure 18. View of Employee table

Indexes

Indexes are present in a relational data base system primarily as a performance vehicle. Any column in a base table may have an index. An index may also be defined over multiple columns from the same table. The index can have a specification of UNIQUE in which case duplicate entries in the indexed column(s) are not allowed.

The use of an existing index is determined entirely by the DB2 system itself, it is transparent to application programs. There are no language statements in SQL which refer to an index other than those to define them - CREATE and DROP.

Primary keys

The user does not have to explicitly define a key to DB2. The equivalent of a key definition is established by:

- specifying that one or more columns have an INDEX which is UNIQUE and

- specifying that nulls are not allowed for those columns - NOT NULL - in the definition of the table.

By this specification the system will enforce the rule that the key should always be present for every row in the table and that it should uniquely identify that row.

TABLE OPERATIONS IN DB2

The language used in DB2 is called SQL (Structured Query Language). This language not only includes statements for retrieval operations as the acronym might suggest, but also modification operations, data definition operations and some built-in functions.

The language is available as a stand-alone language to be used interactively from a TSO/SPF terminal or embedded in a host programming language such as COBOL, PL/I, FORTRAN or Assembler Language.

Retrieval operations

The following relational data base operators are supported in the SQL language.

- Selection

- Projection

- Join

- Union

Elimination of duplicates in the result of table operations is optional. As a default rule, duplicates are allowed in tables. However the user can specify that potential duplicates should be eliminated by the DISTINCT keyword.

For join operations SQL provides support for the natural join operation and joins on <, <=, =, >=, >, ¬<, ¬= and ¬> conditions.

Many examples in the preceding chapter on the relational data base model are formulated using the SQL language syntax (to distinguish SQL examples from informal problem descriptions all SQL statements are capitalized).

Modification operations

All three basic modification operations are supported in SQL.

- Insert
- Delete
- Update

All these three operations could be performed either as set operations or as record-at-a-time operations. Previous examples of modification operations also use the SQL syntax.

Built-in functions

SQL provides integrated functions for summation, grouping, ordering and basic statistics - calculate an average of the values in a column, find the minimum or maximum value in a certain column.

SQL Environments

SQL is available both interactively from a TSO/SPF terminal and also as embedded statements in COBOL, PL/I, FORTRAN or Assembler Language programs.

The interactive TSO/SPF terminal environment is primarily intended for data base administrators and application programmers to interactively, on-line create tables, test SQL statement sequences and maintain data bases. The query interface for end-users is provided as a separate Program Product which executes in a DB2 environment and provides specialized end-user interfaces for non-DP professionals.

Before being executed, SQL queries that are embedded in programs go through the following steps:

- Precompilation by the SQL precompiler

  The SQL syntax is checked and a modified source program is produced in which SQL statements have been replaced by calls to DB2 code. Also a form of the SQL statements is saved in a "data base request module" - (DBRM) - to be used later in the BIND process.

- Compilation by the host language compiler

  The program is then compiled and linked using the normal COBOL, PL/I or FORTRAN Compiler or the Assembler.

- BIND

  Next the program is 'bound'. During this step, the SQL statements (DBRM) produced by the precompiler are analyzed in order to produce an access module. The bind process chooses the most appropriate method to execute the SQL statements and produces the code for it, based on data requirements, indexes available, etc. This separation of the accessing "technique" from the program logic is an important aspect of the DB2 implementation.

- Execute

  The program is executed and the data base calls that were produced by the precompiler are now processed by DB2 according to the access module built during the 'bind' process.

This sequence is essentially the same whether SQL is used embedded in COBOL, PL/I, FORTRAN or Assembler Language programs or used interactively from a TSO/SPF terminal. In the host program environment these preparatory steps can be executed separately, while in the interactive environment they are all executed as one combined task by the system.

A major advantage of the precompilation and BIND process is that it removes the interpretation of SQL syntax, authorization checking and the establishment of an optimized access strategy from the run time code. This entire process is performed before program execution.

Cursor operations

A conventional host language like COBOL, PL/I, FORTRAN or Assembler Language can not easily deal with a set of objects - like table rows - for which there is no known upper limit.

In order to handle the result of a retrieval request which contains multiple rows a mechanism is required by which each individual row could be obtained from the result table and stored away in a host language data structure or host variable.

This mechanism is called a **cursor**. The idea behind the use of a cursor is to first define the complete result of a table operation by normal relational operators and then fetch one row at a time from that result table. This is done by moving the cursor from row to row in the result table.

```
EXEC SQL DECLARE DEPD01 CURSOR FOR
            SELECT DEPTNO,DEPTNAME,MGRNO
            FROM DEPARTMENT
            WHERE ADMRDEPT = 'D01' ;

DCL variable1


EXEC SQL OPEN DEPD01;
         DO WHILE ......
            EXEC SQL FETCH DEPD01 INTO Variable1 ;
            Process Variable1
         END;
EXEC SQL CLOSE DEPD01;
```

Figure 19. Cursor use in host languages

TABLE DEFINITION AND UTILITIES

SQL for data definition

In addition to language operators for data base retrieval and modification, SQL also includes operators for data base definition. These operators perform the following functions

- Create a new table or view
- Alter the definition of a given table
- Create an index
- Drop(delete) an index
- Drop(delete) a table or view
- Define storage objects and defaults

Generally these statements can all be executed dynamically from an interactive TSO/SPF terminal provided that the proper authorization requirements are satisfied. Once accepted by DB2 the definitions take immediate effect. Therefore the use of SQL through the TSO/SPF interactive interface will considerably simplify the data definition task.

As an example all of the following operations could be done through the TSO/SPF interface in one single interactive session:

- Create a new table
- Create an index on the table
- Enter some test data
- Use the table for testing SQL retrieval statements
- Update and change data in the table
- Add new columns and data as necessary
- drop the table and index

Catalog support

The definitions of all DB2 objects are held in a **Catalog** within the system in the form of tables.

Because the catalog is maintained within DB2 as a set of tables, it can be interrogated using the SQL language itself. With SQL retrieval statements a user can directly query the DB2 systems catalog to find current information on existing tables, column definitions, view definitions etc. The same language that is used for data definition, data retrieval and update is also used to access the run-time data base definitions themselves. The catalog can however not be modified directly via SQL INSERT, DELETE or UPDATE statements. This is only allowed by using the appropriate data definition language statements.

DB2 authorization

DB2 provides an extensive authorization facility designed to protect all the various objects that constitute a DB2 system. These facilities not only provide functions to grant and revoke privileges against tables, views, programs etc; they also make it possible to design the authorization environment in a number of different ways, ranging from a very centrally controlled system to distributing authority to a number of disperse user groups.

Considering the emphasis of DB2 and relational data base systems in general on ease of use, it is assumed that this may often mean a wide range of user types accessing DB2 data. For this reason, it is vital that an extensive authorization mechanism be provided.

DB2 utilities

Although utility functions are not part of the relational data base model, an implementation of any data base management system requires many utility functions for the day-to-day operation, control and recovery of the data bases.

Utility functions in DB2 include:

- Reorganize one or more tables
- Load a table with data from sequential files
- Take an image copy (backup) of tables in the data base
- Merge previously created image copies
- Recover tables from image copies
- Statistics and log utilities

No utilities are needed for data base definition because this task is accomplished through SQL.

In this chapter relational data base capabilities will be compared to hierarchical. The purpose is to put relational data base capabilities into perspective for those readers who have a background with hierarchical systems, especially IMS/VS.

The comparison will partly be at a concepts level - it will have general applicability to relational and hierarchical data base systems. However, in many cases comparisons will also be made at the implementation level; in this case the IMS/VS implementation of a hierarchical model will be compared to the DB2 implementation of a relational model.

Although relational data base capabilities in this chapter are compared to hierarchical data base capabilities, most comparisons will apply also to comparisons with a network data base system. Implementations of hierarchical data base management systems - like IMS/VS - actually have much in common with network data base management systems even if the data structure that is presented to users is still hierarchical at the external level.

This chapter will consider in turn:

- Data structure correspondences
- Language functions
- Data consistency facilities
- Performance implications.

CORRESPONDENCE IN DATA STRUCTURE

The correspondence of Tables to Segment types

As a basis for comparison a rough analogy is that a table corresponds to an IMS/VS segment type. An IMS/VS data base corresponds to a set of multiple tables, probably with many columns in common.

The following figure shows the hierarchical equivalence to the Employee and Department table structures used in the previous examples.

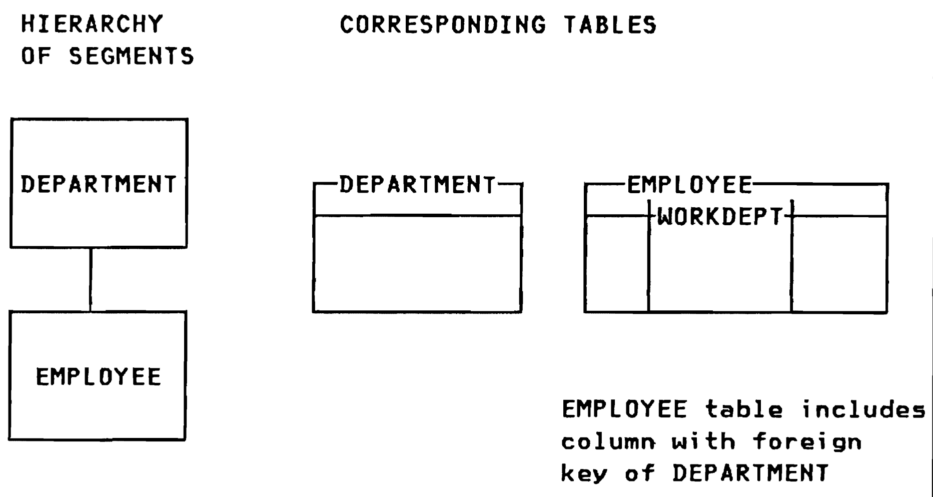

Figure 20. Hierarchy - Table correspondence

Another possible hierarchical structure that corresponds to the EMPLOYEE and DEPARTMENT table would be one in which a DEPARTMENT segment type is organized as a dependent of an EMPLOYEE segment type. Or they could even be structured as two separate IMS/VS data bases, possibly with logical relationships between the DEPARTMENT and EMPLOYEE segment types. In general, though, an IMS/VS segment type roughly corresponds to a table.

Relationships in tables and hierarchies

A parent to child relationship in a hierarchy corresponds to columns for which the values could be matched to values in some other column in a relational data base system.

As an example, values in the WORKDEPT column in the EMPLOYEE table could be matched with values in the DEPTNO column in the DEPARTMENT table. Also, MGRNO in the DEPARTMENT table could be matched with EMPNO in the EMPLOYEE table. A futher example is that ADMRDEPT could be matched with DEPTNO - both in the DEPARTMENT table.

There are a number of interesting points to notice in this analogy.

- In hierarchies there are two ways of representing relationships.

  Either the data structure is used as in the case of a parent-child link. Or the relationship is carried through actual field values that appear in the segments themselves.

- In a relational data base relationships are always represented by values in table columns.

  There is only one way to represent relationships in tables. A consequence of this is a considerable simplification of the necessary language operators that deal with relationships in the table data structure, as well as greater data independence.

- There is always a possible set of tables that corresponds to a set of hierarchical data base structures.

  Any application can be implemented either using a relational or a hierarchical system.

  The links implied in the hierarchical structure are represented by extra columns in the tables if implemented in a relational system. The extra columns contain the necessary concatenated keys of corresponding parent segment types in a hierarchy. As an example see Figure 20 on page 40.

The value of having many potential access paths

Because relationships in tables are only represented by values in the tables themselves, this gives some significant advantages as compared to hierarchies.

- At the user level all relational access is accomplished by associative addressing - by comparing values.

  Data base requests to retrieve or update data are always through the use of actual values in the tables. This simplifies operations on tables.

  This is in contrast to the use of a hierarchical data structure where access requires that one follows the predefined links in the hierarchical structure. This is often referred to as navigation because the programmer or user expresses the data base requests in terms of an implied position within the hierarchical structure and a movement along the segments in the structure.

- Many potential access paths exist to data

  If the data itself exists, then the relationships between different columns also already exist. There is no requirement to add new secondary indexes and/or change application code accordingly for the purpose of having a new way of entry into the data base. Neither is there a need to first unload the data base, change the data description and then reload the data base again with the new data base descriptors in order to have a new relationship on existing data established.

  For performance reasons however, when new access requirements evolve it may be desirable to add indexes or make other changes to the physical storage of data with no effect on programs.

- If a new relationship is required for new data, then extra columns are defined for the new data or new tables are created. By the presence of the new columns and tables the new relationships automatically exist.

- The user can always formulate queries using as a starting point the columns that are most significant for the query. This is in contrast to having to use a predefined structure that has other data as its basis for structuring (the root segment). The predefined structure may have been defined based on other data access requirements. New access requirements may not as conveniently be expressed in terms of the old structure.

  Using the example shown in Figure 20 on page 40, a query that was only concerned with information about employees, in a relational system, would only have to deal with that table. In the hierarchical approach shown, the query has to deal with data about departments as well.

- All access is symmetrical

  A consequence of access through field values only is that all access is symmetrical. The user does not code differently depending on what access path is used. Access through different indexes, as an example, is not different in the way you code as it is in IMS/VS. The use of an index is determined completely at the internal system level and is not specified by the user in DB2. This simplifies coding for the user since he does not have to use different coding techniques for similar coding situations.

CORRESPONDENCE IN LANGUAGE FUNCTIONS

Set level operations versus record at a time operations

When relational data base concepts were first introduced one of the objectives was to provide higher level operators for data base requests. This was possible because of the introduction of table structures where all relationships between different data elements are kept in data values instead of some in the data structure and others in the data. With all relationships represented in a single uniform way it was much easier to have language operators access the data on a higher (less detailed) level. Operations from mathematical disciplines on sets could then be applied to data organized in the form of tables. Characteristics of these operations are:

- Operations apply to entire collections (sets) of rows from tables - not just individual rows.

- The result of each table operation is in itself a table.

  Therefore data base requests could be expressed so that the result of one operation becomes the input to another.

- All data base requests use the same principal form because all relationships in the data base are represented in the same uniform manner - as field values.

Record-at-a-time access returns instead one individual, single record as the result of each data base request. The coding that the user has to do to access the data base at this level therefore becomes more detailed.

In principle it would be possible to define a language based on set level operators to work not only on tables but also on hierarchies. However, because there are multiple ways to represent relationships in hierarchies this is more complex. Current implementations such as IMS/VS do not provide this capability; instead data base operations are record-at-a-time operations.

Table operations are defined as set operations - they work on entire collections of rows. The table language can be used as a standalone language or it can also be embedded into a conventional host language like COBOL, PL/I, FORTRAN and Assembler Language. For the embedded version the operations of insert, delete and update present no problems as set operations. Also there is no problem when retrieval requests return a single row as a result. However, in SQL, retrieval requests that return a set of rows require that the rows are then accessed one at a time from the set using a cursor.

Retrieval

Retrieval in a relational data base system is generally performed as selections, projections and joins. A relational data base request corresponds roughly to a series of individual calls in the hierarchical environment. The following general comparisons could be made between these two environments:

- A single SELECT statement in DB2 corresponds to a series of GU, GN, GNP calls in DL/I.
- SELECT statements using join conditions are used to combine data from different tables.

  SELECT statements using join conditions therefore correspond to those retrieval calls which depend on links carried in the hierarchical structure i.e. GNP, GN.
- When relational data base set retrieval operators are issued in a conventional host language like COBOL, PL/I, FORTRAN or Assembler Language a CURSOR is used to obtain each row from the result.

  This is because there is no known upper limit to the number of individual rows in the result. Therefore to handle the retrieval of sets of rows in host languages, DB2 uses a Cursor which is a position holder in a set of rows. Cursors are first OPENed; Then FETCH statements are used to pick up one row at a time from the result and make it available to the program. Finally the Cursor is CLOSEd.
- View definition can substitute for a considerable part of the programming by eliminating irrelevant data that otherwise would have to be handled by program code. Not only could irrelevant data be eliminated, but the view could also predefine (through join operations) the specific data of interest for this application. Further if the same set of data is used in more than one application, redundant coding in multiple programs would be eliminated through the predefined view specification.

Modification

Delete, insert and update operations are normally specified as set operations in a relational data base system. IMS/VS always handles these operations one record at a time.

- Delete

  Delete operations apply to all rows that satisfy the data base request specification.

On the other hand a DL/I delete call will propagate down to dependent segment types while there is no cascading delete function in DB2.

- Update

  This operation is performed as a set level operation with DB2. All rows that satisfy the data base request specification are updated. The update applies uniformly to all rows.

  The corresponding function in IMS/VS is performed as a series of Get Hold and Replace calls, one for each segment occurrence.

- Insert

  Insert operations could be performed either as a set operation or as a single row operation with DB2. Figure 14 on page 24 is an example of a set insert operation where all rows for department D11 are selected from the DEPARTMENT table and inserted into the TEMPORARY table.

  In IMS/VS insert always applies for a single segment occurrence (or in the special case of a path call to the segments in a single hierarchical path).

Built-in functions

Built-in functions are provided in DB2 for summation (SUM), ordering (ORDER BY), and basic statistics (MIN, MAX, AVG, COUNT).

There are no corresponding functions in IMS/VS. All such functions have to be coded in application programs.

DATA CONSISTENCY CAPABILITIES

Generally speaking data consistency facilities are a characteristic of the implementation. With the present implementation of DB2 consistency of data across tables in general has to be maintained by the programs and procedures that operate on the data bases. To a certain extent this responsibility is however simplified by the language functions that are available in SQL.

IMS/VS provides some data consistency capabilities through its hierarchical data structures and some others through the logical insert/replace/delete rules.

- Key consistency

  Primary key consistency in DB2 is accomplished by specifying that a primary key column should not be allowed to take on null values, and by specifying a unique index for the column.

  IMS/VS requires keys to be present and to be unique if so specified but there are options available to depart from this.

- Consistency in data references

  Consistency in references of foreign keys will have to be maintained by program procedures in DB2. When a row in one table is deleted, references to that row from other tables may have to be removed. For instance if a row in the EMPLOYEE table is deleted and this Employee also happens to be a manager of a department then the MGRNO in the DEPARTMENT table also has to be removed. And there might be other references to this Employee that would also have to be removed.

  In IMS/VS delete requests are propagated down to dependent segment occurrences.

  The DB2 system does not require that certain conditions exist before new rows are inserted or existing rows are updated. For instance if a new Employee row is added to the data base it may be necessary to check that the Department assigned for this employee actually exists in the DEPARTMENT table. This must be done by the program.

  In IMS/VS a dependent segment can not be inserted before its parent segment exists. Also insert/replace/delete rules will reject and propagate some data base modification requests under the condition that relationships are implemented as 'logical relationships'. This includes differentiating between "physical delete" and "logical delete", and automatically deleting a parent that loses all its children, if so specified in the delete rule. Other forms of relationships may require program code and procedures in order to maintain data consistency.

  Tables can be joined in DB2 on any condition that the user determines is appropriate. DB2 will not verify if a join does indeed make sense or not. This is a user responsibility. In IMS/VS the possible access paths are all predefined by the hierarchical data structure and secondary indexes and no other alternative access paths can be used.

PERFORMANCE IMPLICATIONS

High level access to data bases through the relational data base model and the various increased ease-of-use benefits that are obtainable with this approach, as one could expect, are not always free. The precise performance differences between the relational model and other models like DL/I will vary depending on the functions being used and the skills of the person using the product.

There is no clear-cut answer as to whether one or the other system generally provides the better performance. The following are some of the implications on performance that could be expected with a relational data base system:

- Applications differ greatly in how predictable and varied their data usage patterns are. Some always use data in the same way, accessing it in the same order, while others are less consistent, and usage patterns are not predictable. DL/I provides a number of tuning options that are especially useful in an environment where the majority of access is through a well known path. As soon as multiple and variant access requirements must be satisfied and balanced against each other, the overall optimum storage organization will be more difficult to establish. This latter case is more likely to be the environment where DB2 will be most often used.

- Using high level languages like SQL will generally require greater memory and CPU resources. Among other things, this depends on the use of facilities such as sorting which apply to set level operations but do not really have an equivalent in a record-at-a-time environment.

- In DB2, the system has the major responsibility in choosing access paths. In a hierarchical system such as IMS/VS, the programmer is responsible for the "navigation" through the data base to achieve the objective. Which system performs better depends, among other things, on the skills of the IMS/VS programmer and the efficiency of the DB2 automatic path selection process.

- Because in relational data bases relationships are represented as values rather than by data structures, one would assume that this will require slightly more disk storage because actual key values will be stored in the data base as opposed to the pointer(s) in the corresponding hierarchical structure. This will however depend on the total number of key values actually present in a data base as compared to the number of pointer slots that actually occupy storage for a specific data base organization. The relative length of key values as compared to the data will also impact the overall result.

A general conclusion is that differences in this area will not be very significant.

In general performance implications as discussed here should be balanced against the increased productivity that could be achieved with the relational data base approach in data base programming, change and maintenance. **Since the potential benefits and costs associated with them are not of the same unit, a general non-subjective overall assessment is not possible.** The importance of the various cost factors discussed above will certainly vary considerably both with the application environment and with the production environment.

CHAPTER 6. ADVANTAGES OF A RELATIONAL DATA BASE SYSTEM

In this chapter the advantages of the relational data base approach, as compared to a hierarchical approach, will be summarized. Preceding chapters have identified the fundamental characteristics of the relational data base approach as well as differences and similarities between relational and hierarchical. The purpose of this chapter is to concentrate on the potential advantages with the intention to later identify which type of applications are more suited to a relational data base implementation than to a hierarchical one.

Comparisons and conclusions in this chapter will take as a basis actual IBM implementations of a hierarchical and a relational data base system. Therefore DB2 as a relational data base system will be compared to IMS/VS as a hierarchical data base system.

ADVANTAGES RELATED TO UNDERSTANDING

Simple Concepts

A main idea behind relational data base systems is that the basic principles for structuring data as well as for accessing data ought to be simplified as compared to today's implementations. There should be few broad simple principles to apply rather than a number of independent ad-hoc rules.

The way to structure data should be simple and straightforward in order to make it possible for more users to deal directly with data bases.

The languages to operate on data should be less detailed than today's practices and the operations be more generalized.

These motivations have led to the relational data base alternative which structures data externally in the form of tables with rows and columns and provides language functions which could operate on tables as a whole - as set operations - as opposed to individual rows.

The relational approach has also put more emphasis on the theoretical foundations of table operations. This has a value in that the use of the language in new coding situations becomes more predictable. It is possible to predict the outcome of new operations based on the theory and the underlying basic principles.

The relational approach is therefore the result of an attempt to provide simple concepts for data base structuring as well as for data base access.

Uniform Language for Data Definition and Data Manipulation

In line with the emphasis of simple concepts also comes the idea of providing a uniform language both for data base access and for data base definition. Having one consistent language for these two tasks simplifies communication between people who define the data bases and those who will later access them.

The uniformity of language for data definition and data access has also contributed to the fact that with SQL, some aspects of data base programming have now been moved over to data base definition. An example would be the following:

```
CREATE VIEW NEWTABLE AS
 SELECT EMPNO,LASTNAME,DEPTNO,DEPTNAME,ADMRDEPT
 FROM EMPLOYEE,DEPARTMENT
 WHERE WORKDEPT = DEPTNO
 AND ADMRDEPT = 'D01'
```

NEWTABLE

EMPNO	LASTNAME	DEPTNO	DEPTNAME	ADMRDEPT
000060	Stern	D11	Manufacturing	D01
000150	Adamson	D11	Manufacturing	D01
000070	Pulaski	D21	Admin	D01
000260	Johnson	D21	Admin	D01
000220	Lutz	D11	Manufacturing	D01
000270	Perez	D21	Admin	D01

Figure 21. View definition: Eliminates need for program code that otherwise would have been necessary in order to combine the two tables and to obtain the proper subset of rows.

The View definition here eliminates program code to match the two tables Employee and Department, to eliminate all rows not having ADMRDEPT D01, to eliminate columns not of interest in the result. Data base definition is taking over some tasks that otherwise would have required code in the programs accessing the data base; the distinction between data base definition and data base access becomes less important. This also leads to an improvement in productivity because what was previously necessary program code in many programs, may now be a single data specification.

ADVANTAGES RELATED TO DESIGN

Logical and physical design

From a theoretical standpoint the design of a data base on a logical level should not be any different for different data base management systems.

The logical data base design effort should resolve questions on which records are best suited to model the real business, which fields should go together in a record type, when it is better to split a record type into multiple record types and what relationships exist between record types.

It should be the responsibility of the physical design task to structure these record types into physical data bases according to the requirements, facilities and peculiarities of the actual data base management system.

However, an experienced designer will in practice take into account, during the logical design phase, the options that exist for physical design. The boundary is not always as clean and distinct as it should ideally be.

When this reality is applied to relational data base design a very important characteristic of the relational data base approach stands out. In this approach all relationships between data are modelled in the same way - as values in the actual tables. Therefore many complex considerations are eliminated. These are considerations like which relationships should be implemented in the data structures (hierarchies or networks) and which should still be implemented as values. This leads to a simplification in going from a logical design into physical design, especially if the application need for data is not completely known - which is often the case - or if the applications need for data does indeed change over time - which very often happens.

Another way to put this is the following. After the logical design of a set of record structures is done then this logical data base design maps directly into relational tables. This is in contrast to a network or hierarchical system where there is a further step where one has to decide which relationships should be implemented as part of the data structure.

The advantage of the relational approach with respect to this data base design activity is that table structures will have a better chance to survive changing requirements and needs.

Easy to define and change a given table

When the result of data base design is implemented and defined to the data base management system there are a number of tasks that are simplified with relational data bases.

- Logical objects are clearly separated from physical.

  The data base administrator could clearly define physical aspects while application designers could provide logical definitions.

- Definition of data base structures could be accomplished directly from an on-line terminal, and with no disruption of current data base operations.

- Data base definitions take immediate effect.

- Additional columns can be added without the requirement to unload and reload the data base.

- View definitions may insulate one design from subsequent changes.

  Users of an existing view are in general not impacted when data bases are expanded by new columns because the view can shield them from the additional columns.

- If the data base design result is not perfect from the very start, a relational data base system makes it easier to correct. A relational data base system is more forgiving to an incomplete data base design.

ADVANTAGES RELATED TO PROGRAMMING AND DATA BASE ACCESS

Powerful set operations in data base programming

A major benefit that is achieved with relational data base systems is the use of set operations to manipulate data in the data base. This at once provides the user with a higher level data manipulation language which simplifies data access.

Set operations are applicable when a relational data base is accessed from a terminal for all the different types of access - retrieve, insert, update and delete. Most of these operations can also be used directly in a conventional programming language - COBOL, PL/I, FORTRAN or Assembler Language. So the advantages of set operations extend automatically to this environment as well.

Because of limitations in the host languages themselves, retrieval of sets of rows from tables can not be directly handled in the same way as from an interactive terminal. There are no language facilities available in COBOL, PL/I, FORTRAN or

Assembler Language. to easily deal with sets for which the upper limit of rows is not known. So retrieval of sets will require the use of Cursors in order to handle a retrieved set. This limits the value of set operators in the host language environments to some degree. In effect the individual rows of the set are now dealt with one at a time.

However many of the advantages that exist for set level operations still remains in the host language environment. Delete, update and insert operations can still be specified as set operations. And application program can be developed from the basis of very selective view definitions.

Simplified Data Base programming through view definitions

View definitions are very powerful in providing just the subset of data that a particular application should be concerned with. View definitions therefore will reduce the need for programming of commonly used selections of data. View definitions will also shield the application from data that is part of the data base but is irrelevant for this application. Not only can columns that are not of interest be shielded, individual rows that do not qualify in the view are also eliminated for this particular application. This means that there will be less logic in the program itself to deal with data that is not of interest. Also, there will probably be fewer exceptional cases for the program to cope with.

Simplified Data Base testing

When program development is in a preliminary or early stage it will be convenient to try out relational data base request sequences directly from a terminal as a stand-alone data base test without involvement of the actual program. It will also be convenient to set up and check test data outside the actual program through the interactive terminal interface.

These operations are simplified because the relational data base language is self-contained - it does not require a host language in order to be used.

Symmetry in data access

A simplification that is accomplished with relational data base systems is that all access can be done in the same uniform way. All access is based on value comparison - not value comparison in some cases and data structure navigation in other. There are no language statements which depend on whether a particular index is used or whether access is based on hashing or on one or the other of potential alternative data structure paths.

Many possible access paths available

Tables can be combined based on any two compatible columns. A user is not limited to a predefined data structure. This has a value when access requirements change so that there is a need to access data based on new information. Because all relationships are represented by values in the tables, all these values are equally available for access.

ADVANTAGES RELATED TO MAINTENANCE

Ease of Change in data bases

Data base maintenance caused by data base changes is simplified in ways similar to data definition.

- Many changes to tables do not require data bases to be unloaded and reloaded
- Columns can be added without impact on existing users because view definitions can preserve a previous view of a table
- In many cases, base tables can be split or merged without impact, because their appearance can be produced from the new base tables through the use of views.
- There will be less impact due to changes because all relationships are already carried through actual values in the tables and not through links in an explicit data structure

ADVANTAGES RELATED TO SECURITY

Security by Data Content through Views

View definitions have already been mentioned as a powerful facility to simplify programming. A view only needs to contain the data that this particular application has to deal with. This capability also has a security side; a user can be screened off from data that he is not allowed to have access to - or even see. The user does not even have to be aware of the fact that he does not see a full table.

An example would be that of restricting access to the employee table by the following view definition.

```
CREATE VIEW DEPEMPLOYEE AS
 SELECT EMPNO,LASTNAME,WORKDEPT,PHONENO
 FROM EMPLOYEE
 WHERE WORKDEPT = 'D11'
```

DEPEMPLOYEE

EMPNO	LASTNAME	WORKDEPT	PHONENO
000060	Stern	D11	6423
000150	Adamson	D11	4510
000220	Lutz	D11	0672

Figure 22. A view definition on the Employee table

With this view of the employee table a user could not see salary information at all and only employee information for employees in department D11. The user is further restricted to see only nonsensitive data in the employee table - the employee number, lastname, work department and phone number columns. The example is fairly trivial in its subsetting but there is a potential for many possible variations. Every subset of data that could be expressed as a view could also effectively be a security specification.

Note that security by content in this way is not specified in procedural program code but available through the data definition facilities.

Moreover, SQL statistical operators may be used in views to restrict some users to aggregate data, such as averages and totals, and not allow access to individual values.

ADVANTAGES RELATED TO OPERATION

Although the operational aspects of relational data base systems are really an implementation issue and not an inherent feature of all relational data base systems, nevertheless the basic objectives for relational data base systems will carry over also to the operational area for the following reasons.

- Concepts are simplified so that, in general, it is easier to have a more understandable overview of the system and its components. Simplicity will in general also improve reliability - in systems usage as well as in the system itself.

- Different objects are kept separate to a larger extent.

For instance there is a cleaner boundary between physical storage definitions and logical table definitions.

- In the spirit of providing clean, simple concepts to the users, more internal options are kept in the system itself.

  This has both advantages and drawbacks. On one hand fewer externalized options make it easier to operate the data bases just because there are fewer components to keep track of and to be knowledgeable about. Obviously this also means that meaningful options and performance alternatives may have been sacrificed for increased simplicity - increased ease-of-use.

ADVANTAGES OF A HIERARCHICAL APPROACH

Preceding sections of this chapter have concentrated on the potential advantages of a relational approach. Even if the objective of this chapter is to identify advantages specifically of relational data base systems so that suitable application areas for relational data base systems could then be identified, it would not be fair to only identify advantages. Disadvantages of the relational data base approach as compared to a hierarchical should be described as well.

By implication those aspects of a data base system that have not been included earlier in this chapter are in general advantages of a hierarchical data base system as compared to a relational. Even if this document does not go into quite the same detail, there are many facilities in hierarchical systems that compare favorably with relational systems implementations.

The following is a short summary of characteristics for such application and data base environments.

- Relationships in the data base form a natural hierarchy which is used by all major applications.

  No significant other relationships exist. Data content is rather stable with little anticipated new or changed data requirements. The application processing requirements follow this hierarchical structure strictly and do not change.

- Data has to be accessed one record at a time by the nature of the data or the application.

  Therefore there are no benefits from set operations on data to be gained here. This situation will occur when individual records require very specific and unique treatment in most operations. It also occurs where the application itself requires recursion in data access as do many procedures in 'bill-of-material' applications.

- Applications require access only to a single, individual data record.

  Many applications, especially in high volume, predefined, repetitive transaction environments, merely read a few input parameters and then directly access an individual record in a data base for further processing. In this situation there are no specific advantages of set level programming.

- Resource consumption is more important than flexibility in change and maintenance.

  For applications that follow a hierarchical access pattern the hierarchical physical storage options may in some situations provide more efficient access characteristics than a relational implementation that is based on the use of indexes.

- There is a need for detailed tuning capabilities.

  The relational data base approach will emphasize ease-of-use and simplified operation. Although such characteristics do not have to conflict with detailed tuning possibilities, when there is a trade off between ease-of-use and detailed tuning alternatives. It is in line with the relational approach to favor the ease-of-use aspect.

In general, when a comparison is made on an implementation level, a specific hierarchical implementation may provide features that are not present in a specific release of a relational implementation. Such differences are however not fundamental to the hierarchical or relational approach as such.

CHAPTER 7. RELATIONAL APPLICATIONS

There is no one particular application area that could be said to be exclusively suited for relational data base systems and not for other data base systems. Instead, some application development tasks will be easier to perform with DB2; they will apply to many different application areas, but to a varying degree.

This chapter will first describe the general characteristics of application development with DB2. After that some typical application classes that are suitable for DB2 development will be identified.

CHARACTERISTICS OF DB2 APPLICATION DEVELOPMENT

Application development with DB2 differs in a number of ways from the way application development normally is done with IMS/VS. There are some common application development tasks that will be easier to accomplish with DB2. The extent to which these common tasks apply to a certain application will then determine the type of applications that are more suited for DB2. The following are some of the more important characteristics of DB2 application development:

- It is easy to define a new table.
- It is easy to define a view.
- It is easy to add a new column to an existing table.
- It is easy to insert small volumes of test data into such tables.
- All of these activities can be done interactively from a terminal, on-line without dependencies on other users.
- It is then easy to start using such tables interactively from a terminal before going into programming with COBOL, PL/I, FORTAN or Assembler Language.
- Sequences of DB2 data base requests for a new application may be tested interactively from a terminal without a need for programming.
- Preliminary results can therefore be shown to 'end-users' of the planned application at a much earlier stage of application development.
- In some cases this approach may extend even further so that end-users themselves do the initial application development.

This initial design will most likely not be at a detailed level but, rather, show the main input and output data flow.

These characteristics mean that it will be easier to rapidly put up an initial data structure (a set of tables), make preliminary data base retrieval and update requests on those tables and then modify the initial set of data base tables according to early test experiences.

The net effect is that a number of early tasks in the ordinary development cycle will be simplified.

APPLICATIONS IN PROTOTYPING AND IN EARLY DESIGN PHASES.

As a consequence of the basic characteristics described above, development of the following types of applications will be simplified:

- Prototyping new applications.

  A new application is developed in a way that shows only the main input and output data flow. Details of transaction processing and handling of exceptional cases are not implemented at this level. Such details are left for later implementation steps when the main data flow is firmly defined.

  The important idea with prototyping is to provide a fairly complete but very high level implementation within a short time. The potential users can soon get an idea of how the final application will look without having all the details implemented. This stimulates new feedback ideas from the users and increases early participation in the design process. New suggestions, changes and design alternatives can be explored much more easily with a prototype than in the final implementation.

- Developing small applications which have a limited life-time.

  The entire application may have such a limited life-time that it is not reasonable to go through a complete and thorough application development and coding cycle using conventional development methods. Neither the resources available nor the calendar time may allow such applications to be developed using standard application development methods. The quick design and implementation capabilities of a relational data base system, including standard output display facilities, may open these new application areas.

- Implementing applications which have frequently changing requirements.

If one could foresee frequently changing requirements for a specific application, then the facilities that are provided by the data base system for quick and easy change will become more important. A relational data base system is normally more suited for that type of environment because of its more flexible methods to change existing data structures. The relational data base tables themselves can be accessed based on many different, alternative relationships in the data.

- Applications where user specifications are very incomplete.

  These types of applications are very similar to those where one could expect frequent changes.

  Often it is not only that user specifications are very incomplete - the 'real' user requirements may not be known. Other times an application area may seem so trivial that thorough design practices do not have to be used or the qualified resources necessary for a thorough design may not be present.

  Often this is an indication of the fact that the 'real' user requirements have not emerged yet. Application design in smaller, iterative steps is especially advantageous for those applications, because there is a greater need to frequently check an intermediate design result with the users.

APPLICATIONS FOR DATA ANALYSIS ON EXTRACTED DATA

An area specially suited for relational data base systems is one in which the data to be operated upon is first extracted from the operational environment where it may constantly be updated. Such an extract now represents a snapshot of a production oriented data base. Many types of data analysis will require all their data to be consistent over the analysis period and can not allow the data to be continuously changing as it normally does in an operational environment.

Extracted data for such environments is also often a subset of the entire population. The size of the data base is generally smaller in order to make analysis operations faster and more convenient.

If there are requirements to restructure such a snapshot of data in many different ways, to look at summaries, groupings , subsets of the data in many ad-hoc unplanned ways, then the table structure will be much easier to work with. Therefore modelling, planning and forecasting applications will be easier to develop with a relational data base system because these types of applications are more dependent upon the use of many different relationships in data.

If arithmetic calculation requirements or requirements for drawing charts, drawing diagrams etc. are extensive, then a more specialized package may be more appropriate.

The role of a relational data base system for such environments is to provide these specialized packages with properly adopted subsets of files. A relational data base is then used more as a 'copy manager' to provide suitable subsets to the specialized packages. The strength of a relational data base lies here in its ability to easily combine various different tables on quite different criteria/conditions. This is in contrast to the specialized file analysis packages which may have superior analyzing and graphics facilities. However, specialized analysis packages often have quite limited facilities to combine different files freely with each other.

MODELLING, PLANNING AND FORECASTING APPLICATIONS

In the previous section on applications based on extracted data, modelling, planning and forecasting applications were identified as possible typical candidates for implementation with a relational data base system. These types of applications are often implemented as standalone applications independent of the way the basic data is originally obtained - by extracting or by some other procedures.

The value of a relational data base system for these applications lies primarily in the ease by which various alternative and changing relationships in the data can be analyzed. The nature of these applications is to restructure the data, rearranging it according to new hypothesis and explore the consequences of alternative relationships. Therefore the flexibility provided by relational data base systems will be of special importance.

DATA BASES WITH LARGE OR UNKNOWN NUMBER OF RELATIONSHIPS

The more possible relationships that exist between various parts of the data, the more complex also is the task of choosing the best hierarchical or network predefined data structure. A predefined data structure is more vulnerable to changing requirements, new access patterns, changed volumes of data and overlooked design alternatives.

A relational data base has a great advantage in that all relationships are represented in the same way in the data itself. New or changed access requirements do not have to be reflected in a changed predefined data structure. There is not as great a need to correctly anticipate the most likely access paths from the beginning. The advantage of having this freedom will be more and more obvious the more relationships exist in the data itself.

IMPLEMENTATION OF APPLICATION GENERATORS

The relational data base structure itself will be very well suited for implementations of application decision tables in relational table format. Such decision tables can then drive more generalized applications than are normally developed as standard application programs. In this sense table driven applications are a step towards more generalized Application Generators .

The key for such applications is the relative simplicity by which generalized function decision tables may be set up using tables themselves as the driving mechanism. The DB2 Sample Application is a good example where this approach has proved to be beneficial.

QUERY APPLICATIONS

The class of applications normally characterized as **Query** will, in a specific way, be the most important source of applications for relational data base systems.

Query applications have in common the fact that the exact procedure for access to data bases can not be defined in advance. Information requests are ad-hoc in nature. Also the typical requester of information is often not a data processing professional, but rather a person with other main responsibilities than Query formulation.

To satisfy this category of users interfaces have to be provided that are better adapted to the needs of end-users who are non-DP professionals. The interactive DB2 TSO interface alone is not sufficient for this category of users.

A query package like QMF, based on a relational data base management system will therefore be an ideal tool for an Information Center environment. Here a DP professional could assist the ad-hoc user in specifying suitable extracts of operational data and then formulate query or update requests using the query terminal interface to the data base.

A relational data base lends itself as the basis for query packages better than does a hierarchical or network data base system. This is because in a relational data base there are no relationships that are already built into the data structure. Therefore there is no bias for or against a specific way of accessing the data base.

In a hierarchical or network data base system a user has to code differently when using relationships that are built into the predefined data structure as compared to using relationships that are represented as values in the data base. Experience has also shown that query products based on a hierarchical or network data structure tend to be more complex whenever the predefined relationships in the data structure can not be directly used. The query formulation is not symmetric with respect to the way relationships are used for access, and this asymmetry shows up as increased complexity for the users. A relational data base has therefore much more potential for query products that could be built on top of the relational data base management system itself.

PRIVATE DATA BASES VERSUS SHARED DATA BASES

The more isolated the data that a single person or a small group of people is using, the more suited it will be for implementation in a relational data base system. This is because the methods for changing existing data structures are simplified in relational data base systems. The advantages of simplified data structure changes will pay off much quicker the less impact there is on other users shared data. Relational data base systems are therefore especially suited for data bases which are departmental - which affect only limited group of users.

When the data base becomes larger and is shared by many different users, the problems of control and maintenance will become more important, and also more complex. These problems are limited not only to the data base system but will also require well established routines and responsibilities for change and tight control procedures. The problems of change control and maintenance tend to increase more with the size and shared usage of the data base rather than with the type of data base management system that is used. So the advantage of having flexible tools for quick and easy change may not be that dominant in large shared data base environments.

Advantages of a relational data base system are therefore most obvious where there are less requirements to share data with other applications and user groups.

EVOLUTION OF RELATIONAL AND NON-RELATIONAL APPLICATIONS

It should be clear from the preceding sections that the dividing line between relational and non-relational data base applications is not very sharp. Many applications may in fact be developed using different types of data base management systems.

A considerable number of applications will continue to be best served by a non-relational data base management system. Characteristic for such applications is that they are using predefined and stable data base structures in a repetitive manner. Performance considerations and detailed tuning facilities may be more important for such applications than flexibility in change and access of the data.

Relational data base systems tend to be more suited for applications where there is a clear need to use many different and varying relationships in the data and where ease of change and high level access to data is the more important aspect.

As experience with relational data base systems increases and the implementations of relational data base management systems mature, it is likely that more and more applications will be found to be best served by a relational data base system. More and more environments will find that flexibility in data base access and flexibility of change is a key factor in successful data base expansion and evolution.

GG24-1581-00 **READER'S COMMENT FORM**
IBM DATABASE 2 Relational Concepts

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

What is your occupation?

____

Number of latest TNL applied:

____

Thank you for your cooperation.

If you wish a reply, please give your name and mailing address:

____

____

____

____

Reader's Comment Form

Cut or Fold Along Line

Fold

IBM INTERNATIONAL SYSTEMS CENTER
Department 471
Building F27
555 Bailey Avenue
P. O. Box 50020
San Jose, California 95150
U.S.A.

Fold

IBM

GG24-1581-00 **READER'S COMMENT FORM**

IBM DATABASE 2 Relational Concepts

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, please give your name and mailing address:

____

____

____

____

What is your occupation?

____

Number of latest TNL applied:

____

Thank you for your cooperation.

Reader's Comment Form

Cut or Fold Along Line

Fold

IBM INTERNATIONAL SYSTEMS CENTER
Department 471
Building F27
555 Bailey Avenue
P. O. Box 50020
San Jose, California 95150
U.S.A.

Fold

IBM

GG24-1581

IBM DATABASE 2 Relational Concepts GG24-1581 Printed in the U.S.A.