Extension of the relational database semantic processing model

by T. Hirao

A data model consists of three parts: (1) a data definition that represents the information in an understandable manner; (2) a definition of the constraints that must hold for the information to be valid; and (3) a definition of operations that can be performed on the information. Current database management systems do not allow explicit specification of all three parts of the data model. This paper gives an approach that extends current database management systems through a technique called pre-precompilation.

A deductive database is proposed as a solution for recursive or semantic processing, utilizing the advantages of both relational and knowledge-based systems. The following introduction focuses the reader's understanding on the parts of a relational data model and states the need for extensions.

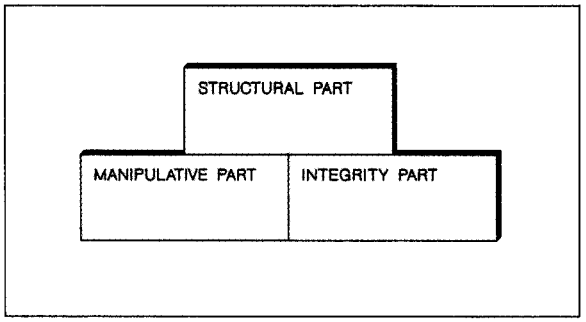
The relational data model discussed in Reference 1 consists of the following three parts: (1) a *structural part* that represents information in the form of a table; (2) an *integrity part* that applies the constraints on the table;[2,3] and (3) a *manipulative part* that operates on the table. These three parts are shown in Figure 1. The following are the characteristics of each component.[4]

The *structure* of the information translates into the format of a table, in which the elements may represent such entities as concepts, events, or objects. Although the table is simple, it is difficult to represent as a tree structure. A *tree structure* is a convenient way to represent a generalization, a specialization, or an aggregation.[3,5]

Integrity constraints assure that information is correct as regards its creation and the operations that use it. Integrity constraints are also important with respect to the semantics and maintenance of data. Therefore, we have to define precisely the integrity constraints at the time of the creation of a structure. Typical examples of integrity constraints are the ISA relations, which are functional dependencies, and the domain constraint, which relates to the properties of a value.[6] The term *ISA* is a self-referential term meaning "is a," and is used in the sense of "is a relation."

The *manipulative* part defines the four types of operations for tables: selection, insertion, deletion, and updating. The Structured Query Language (SQL) has been the standard manipulative language for the relational database model since 1986.[7]

Figure 1 Three components in the relational database model



Relational database requirements become increasingly complex each year as information processing technologies increase in function. Some of these requirements cannot be implemented using relational database technologies. For example, manipulations of the engineering data in a CAD/CAM environment have to process tree structures or recursion. However, it is difficult to do these operations, because values of attributes in a relational database are constrained to be atomic values.

Therefore, we need manipulation based not only on the data alone but also on relationships among the data, that is, on the semantics of the data. To pursue the operations that are the subject of this paper, researchers propose new data models such as the Non-First-Normal-Form model[8] (NF2) and a *deductive database model.*[9,10] We define the deductive database model later in this paper. The aims of the NF2 model and the deductive database are to establish capabilities of recursive or semantic processing. That type of data processing cannot be achieved using current database management systems.

This paper discusses characteristics and limitations of the following three current database management systems: (1) those based on the hierarchical model; (2) those based on the network model; and (3) those based on the relational model. We then propose new techniques to overcome the limitations in using the concept of the deductive database. However, we still use current technologies, such as the relational database, conventional programming languages, and so on. Next we discuss the semantics in database processing and propose techniques of resolving limitations of integrity processing, recursive processing, and the handling of ambiguous (or fuzzy) data.

Conventional databases

In order to provide basic knowledge about database management systems and knowledge-based systems,[11–15] we first review three conventional database management systems and their data models—the hierarchical, network, and relational models.

The *hierarchical model* represents information in the form of a tree, in which it is easy to understand the relationships among higher and lower information. Each box shown in Figure 2A is called a *segment* and segments are linked by pointers. These pointers facilitate the capabilities of the referential integrity constraints. The hierarchical model produces duplicated segments, as, for example, box E in Figure 2A.

Figure 2 Current database models

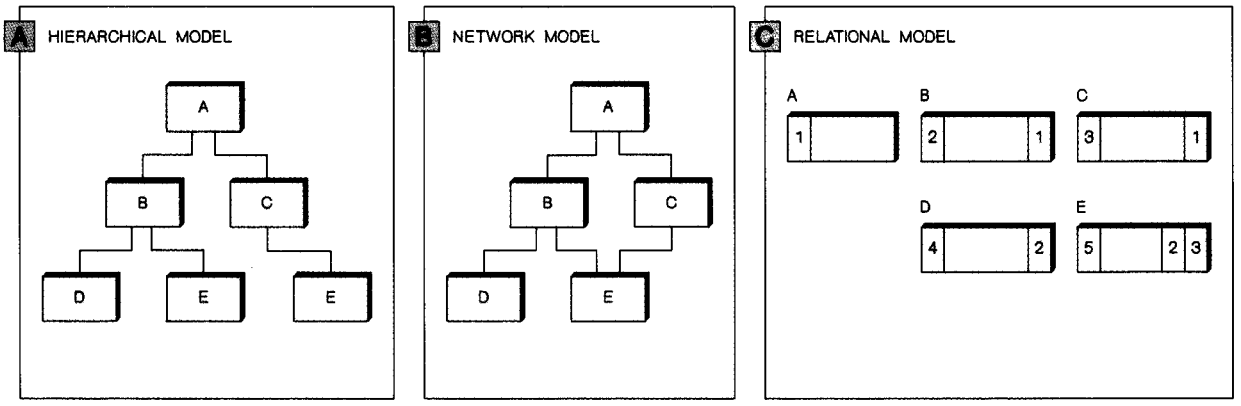


Table 1 Characteristics of three major models

Item	Hierarchical Model	Network Model	Relational Model
Object representation	Segment	Record	Tuple (row)
Constraints			
Domain	Few	Few	Some
Relation	Some	Some	Some
Referential integrity	Deletion parent to child	Deletion parent to child	Definition by DDL
Relation			
Representation	1:N	1:N	N:M
Duplication	Some	None	Some
Manipulation/access unit	CALL/segment	READ/WRITE/record	SQL/set
Exception handling	Status code	Completion code	Return code
Recursive function	Logical relation	Available	None
Deductive function	None	None	None

The *network model* represents information in the form of a network. Each box in Figure 2B is called a *record* and is linked with every other box with links that realize some capabilities of referential integrity. The network model does not produce duplicate records.

The *relational model* represents information as a table. This model may be implemented easily on a workstation as well as on large mainframe systems. Each box shown in Figure 2C is called a *table.* Relationships among tables are established by columns having values in common between pairs of tables. Referential integrity in a relational model, such as DATABASE 2™ (DB2™) Version 2, is realized with the definition of the table using a FOREIGN KEY phrase in the CREATE statement.[16] The relational data model produces duplicated columns because of their foreign key, as in the example of the 2 and 3 in Figure 2C, table E.

Conventional database management systems and their data models have characteristics as summarized in Table 1, from which we can extract problems of each model as follows.

Data redundancies. The relational model and the hierarchical model have redundancies of data, which is obvious from Figure 2. (See, for example, E in the hierarchical model and 2 in table E in the relational model.)

Lack of constraint representation. There are three types of constraints: (1) domain constraint, (2) relation constraint, and (3) referential constraint. The *domain constraint* is that an attribute value should meet certain conditions. For example, the character length for an employee number is 5. These constraints are not specified explicitly in the three database models, except for predefined data types, such as date, integer, etc. *Relation constraints* control the attributes in a tuple. For example, the maximum salary of a person whose age is under 30 must be less than a specified amount. These constraints are not supported by the hierarchical model and the network model. In the relational model, one can specify such a condition by the WITH CHECK OPTON in the CREATE VIEW statement. The *referential constraint* is that the value in the foreign key must be the same as the primary key in the referenced table. The relational model can specify the referential constraint in the CREATE TABLE statement, which has already been mentioned.

Limitations of operational capability. The hierarchical and network models have been used for a long time. In those models, we can manipulate only one segment or record at a time by the host programming language via an access path that is predetermined in the database system. On the other hand, SQL in the relational model environment is a user-oriented language that is executable interactively or through the host programming language. The relational model is also able to manipulate many tuples at a time (called a SET operation) by means of automatic navigation. However, SQL cannot process recursively, and the three models cannot do inferencing the way it is done in artificial intelligence (AI) processing.

Semantics in a database and its processing

Semantics in a database is discussed mostly in the area of a database design. One example is that of conceptual modeling—classification, aggregation, and generalization. These aspects are incorporated in such semantic data models as SHM, RM/T, SDM,

Figure 3 Relationship between the relational database and the deductive database

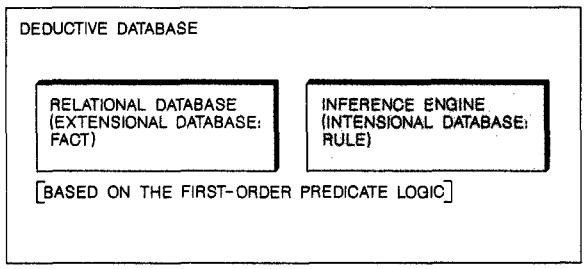


Figure 4 Advanced database

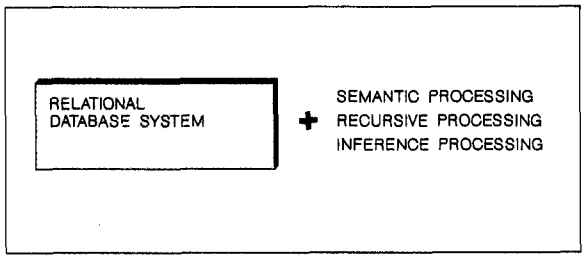


Table 2 Characteristics of relational database and knowledge-based systems

	Advantages	Disadvantages
RDB	Table representation Applicable for large databases Multiprocessing Full recovery/ restart Referential integrity	Lack of processing • recursion • outer join • list process • semantic process Cannot represent the set in the relation
KBS	Flexible processing with logic programming • inference • recursion • list & set • semantic process • ambiguous data	Limited size of the database in main storage No support of multiprocessing and recovery/restart

RDB: Relational database
KBS: Knowledge-based system

TAXIS, and IFO.[15,17] One of the aims of semantic data models is that of *integrity maintenance*, whereas only the referential integrity is supported in current database management systems. Other aims of semantic data models are those of extending the capability of data manipulation (such as recursive processing) and the processing of incomplete information.

This paper focuses on the processing of the semantic differences between data types by means of integrity maintenance, recursive processing, and the processing of ambiguous data. Processing details are discussed in later sections of this paper.

As a basis for later discussion, we briefly mention characteristics of relational database systems and knowledge-based systems. Currently, a relational database system is a database management system that maintains the static business data and provides for full recovery, restart, and so on. However, a relational database system does not provide for recursive processing. A knowledge-based system, on the other hand, is an application composed of a knowledge base of facts and rules that use the flexible processing of recursion or inference formation. However, a knowledge-based system is limited in its use as a database system. Table 2 summarizes the advantages and disadvantages of relational database systems and knowledge-based systems.

By combining the advantages of both systems, we can overcome the disadvantages of the relational and knowledge-based systems. We can consider the relational database system as a database management system that maintains static data, and we can consider a knowledge-based system as an application system that exploits the capabilities of a relational database management system.

Toward a new database system

Because relational database systems and knowledge-based systems have characteristics that complement each other, we can construct a new database management system that can expand the capabilities of a database management system and knowledge-based system. Let us consider the new database system as a deductive database system because it uses a knowledge base. This new database system has the following four characteristics:

- It is based on first-order predicate logic.
- It can manipulate incomplete or ambiguous information.
- It can make inferences using facts and rules.
- It can maintain the integrity of data.

Figure 3 represents the relationship between the relational database and the earlier deductive data-

base. There are three ways to implement the new deductive database: The *advanced database* shown in Figure 4 adds new function to the relational database to enable new capabilities of semantic processing, including recursion and inference. This approach need not change existing programs, which is a great advantage. However, the current database does not have these capabilities. Many researchers propose new data models, such as the NF2, the semantic data model, and the deductive data model. Table 3 summarizes the characteristics of these three models. Each data model shown in Table 3 has two types of data, one is data itself and the other is metadata, which describe characteristics of real data. The properties of the data and metadata are shown in Figure 5.

Metadata incorporate the following five definitions:

- *Schema* defines the table, the column and its domain name, the primary key, and the foreign key.
- *Domain* defines data types and characteristics of the domain.
- *Structure* defines the relationships among tables or columns in the same table.
- *Constraint* defines the referential integrity and the relation integrity.
- *Operation* defines the alert and trigger that are executed at the time of a special event; operation also defines procedures that are used for inferencing.

The meaning of soft and hard data is as follows. *Soft data* are ordinary data that are manipulated by users. *Hard data* are new types of data, typically historical data, that require control information that is stored in soft data. For example, CAD/CAM data are hard data that require information of the creator, the dates of any modifications, and their relationship to other CAD/CAM data in soft data. Figure 6 shows an example of metadata.[10] The important fact is that we can manipulate both metadata and soft (natural) data in a consistent way.

Figure 5 Types of data

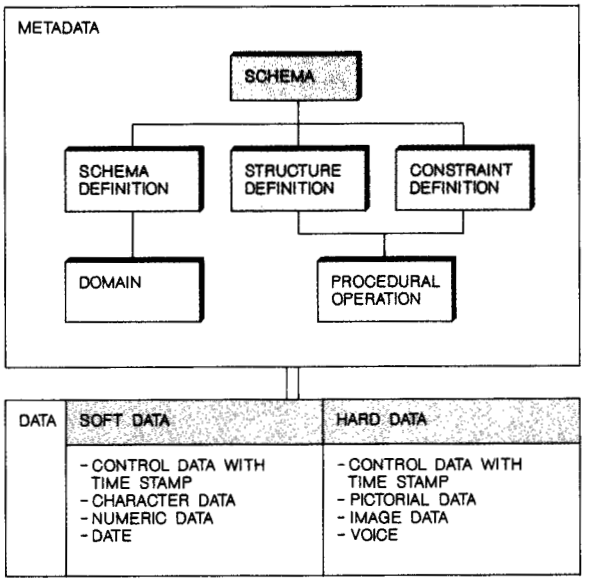


Figure 6 Example of metadata

```
DATABASE DEFINITION
 SCHEMA(PERSONNEL):
  TABLE  DEPT = (DEPT#,DNAME,MGR)          PKEY(DEPT#)
  TABLE  EMP  = (EMP#,ENAME,AGE,SEX,DNO)  PKEY(EMP#)
 TYPE-DOMAIN:
  TYPE ID = {DEPT#,EMP#,MGR,DNO}   CHAR(4)
  TYPE NAME = {DNAME,ENAME}        CHAR(20)
  TYPE AGETYP = {AGE}              SMALLINT < 70
  TYPE SEXTYP = {SEX}              CHAR(1) ['F','M']
 STRUCTURE:
  RELATION D-TO-E  [DEPT:DEPT#,EMP:DNO]
  RELATION E-TO-D  [EMP:EMP#,DEPT:MGR]
 CONSTRAINT:
  C(D-TO-E)  DELETION SET NULL
  C(E-TO-D)  DELETION SET NULL
 PROCEDURAL:
  PREDICATE AGE >= 30 WHERE MGR = EMP#
  PREDICATE EMP,COUNT(*) <= 20
            WHERE DNAME LIKE 'B%' AND DEPT# = DNO
```

Table 3 Characteristics of new data models

Item	NF2	Semantic Model	Deductive Model
Representation	Non-normalized table	Graph or logic	Table
Constraints	Yes	Yes	Yes
Relationship	N:M	Function, abstraction	Knowledge (rule)
Language	Extended SQL	Function	Predicate logic
Unit of operation	List, set	Set of objects	Record
Function of deduction	—	Inheritance	Derive

Figure 7 Logic database

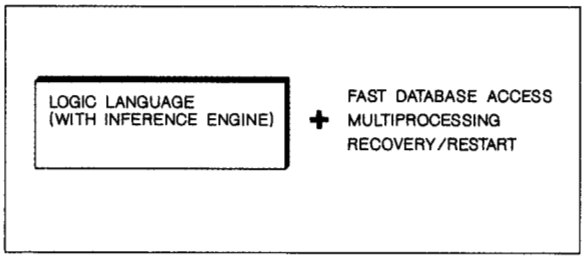


Figure 8 Compromise approach

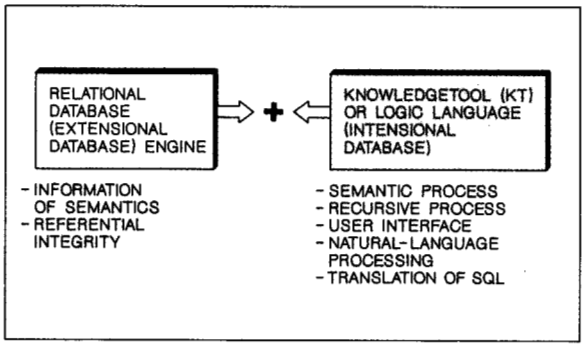


Figure 9 An example table

Employee Table

EMPNO	NAME	DEPTNO	AENO	SALARY	AGE
A0000	J.Hull	A001	A0000	700000	45
A0001	F.Date	B001	A0000	600000	35
A0002	T.Teorey	C001	A0000	500000	36
A0003	C.Hunt	D001	A0000	550000	40
A0004	A.Martin	B001	A0001	400000	36
A0005	S.Ohta	C001	A0002	420000	34
A0006	P.Huit	D001	A0003	340000	30
A0007	G.Gull	B001	A0004	280000	28
A0008	J.Buch	B001	A0004	270000	30
A0009	E.Harada	C001	A0005	250000	26
A0010	K.Kelly	D001	A0006	200000	25
A0011	P.Sowa	D001	A0006	250000	25
A0011	F.Role	D001	A0006	240000	27

(Primary Key) ← (Foreign Key)

Note: AENO is the column of the administrative person.

The *logic database* shown in Figure 7 adds database function, multiprocessing capability, and the functions of recovery and restart to the logic programming language. The logic programming language can do recursive processing. Therefore, it is easy to implement a logic database. However, there are some limitations. The database space is limited to the memory of the running address space. Existing programs must be changed in order to access a logic database. We have to create the interface routine for coordinating a conventional programming language and a logic programming language.

The *compromise approach* shown in Figure 8 benefits from the advanced database and logic database approaches in that it incorporates cooperation between relational databases and the logic programming language. Although the overhead of this approach may be less than that of the logic database approach, it is greater than that of the advanced database approach. Therefore, we should consider that this approach is a step in the migration to an advanced database approach. This paper discusses a compromise approach because of its implementability at the present time. Thus we shall discuss semantic processing, recursive processing, and the processing of ambiguous data.

Semantic processing

Integrity constraints. One of the aims of a deductive database is that of integrity maintenance—referential integrity and relation integrity. Current relational database systems cannot define the integrity rule among data explicitly, except for referential integrity, which is realized in DB2 Version 2. Therefore, it is difficult to maintain data integrity. To have the capability of integrity maintenance, SQL should have the new functions shown in Table 4. New functions of constraint checking are essential in order to develop a new deductive database. In this section, we discuss a technique that can maintain the integrity among data in a relational database. A database that uses this technique can maintain consistency and reduce redundancies in the database.

The double precompiling technique. In order to process semantics, we must specify such semantic information as integrity constraints. To do this, we use a special table named a *semantic table*, which contains the information of integrity constraints. That information will be created by a table creator, using the SQL INSERT statement or the data load utility supplied by DB2.

To illustrate, consider the example employee table shown in Figure 9. This table has some integrity

constraints. For example, the column AENO, which means the administrative employee number, is the foreign key of the column EMPNO. The AGE value must be greater than or equal to 25 and less than or equal to 50, and so on. We can now incorporate the integrity-constraints information into the semantic table shown in Figure 10. This figure shows sample data for the employee table in Figure 9. The semantic table will be used twice—at pre-precompile time and at the execution time of the user program, which is shown in Figure 11.

Semantic processing is executed in the following sequence of steps.

Program coding. First, code the program using new statements, instead of the standard SQL statements for the input of the pre-precompiler. In this paper, we use the symbol $ preceding standard SQL statements, by which they are recognized by the pre-precompiler as statements for semantic processing. Thus the $INSERT statement is used for semantic processing rather than the standard INSERT statement.

Pre-precompile. The program prepared in the first step is pre-precompiled, which translates the statements with the $ character into standard SQL statements plus some other statements that are needed for semantic processing. This process uses information in the semantic table.

Normal processing. The conventional relational database system has to do normal processing, which includes precompile, compile, and link edit. This includes an interface routine supplied by the new deductive database system.

Here we present two examples using the KnowledgeTool™, which is an IBM-supplied artificial intelligence (AI) tool based on PL/I. KnowledgeTool (KT) is used for implementing expert systems. We use the KT in the new deductive database system because that system requires knowledge-based processing. The function to be performed by the program is to update the employee table by inserting the data for a new employee.

Given:

```
$INSERT INTO Employee
VALUES ('A0015','T. HIROTA',
        'C001','A0002',25000,25)
```

Figure 10 An example of semantic table

Semantic Table

TNAME	COLUMN	PKEY	FKEY	REFT	RI	UP	DOWN
Employee	EMPNO	Yes					
Employee	NAME						
Employee	DEPTNO		Yes	DEPT	NULL		
Employee	AENO		Yes	Employee	NULL		
Employee	SALARY						
Employee	AGE					25	50

Figure 11 Semantic processing with the pre-precompiler

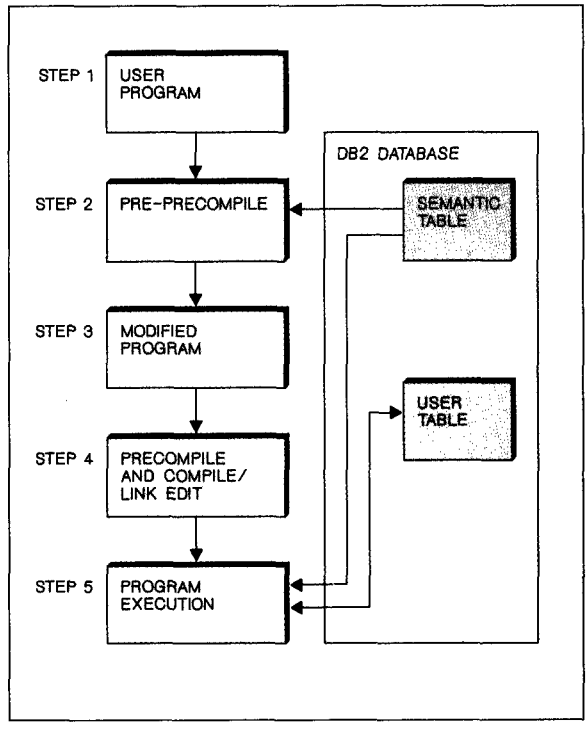


Table 4 New SQL for the deductive database

Statement	Functions
SELECT	Current + Recursion and inferencing
INSERT	Current with RI (RESTRICT) + Constraint check and value check
DELETE	Current with RI (RESTRICT, SET NULL, CASCADE)
UPDATE	Current with RI (RESTRICT) + Constraint check and value check

Figure 12 The derived relation

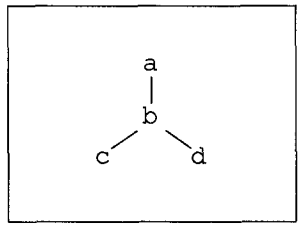
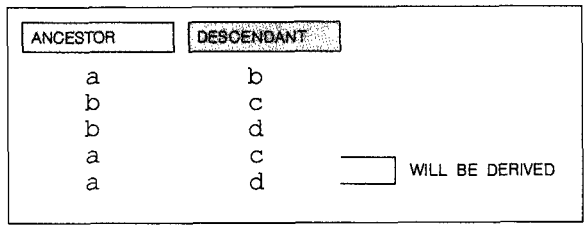


Figure 13 Derived relations in the relational database



Translation:
Allocate A for the semantic table.
Allocate B for data to be inserted.

```
WHEN (S1- > A &
  S2- > B (S2- > AGE > S1- > DOWN,
  S2- > AGE < S1- > UP ))
BEGIN;
  EXEC SQL INSERT INTO
    Employee VALUES (...);
END;
```

This semantic processing checks age restrictions.

Next, delete information of the employee number A0008 from the table in Figure 9.

Given:

```
$DELETE FROM Employee
WHERE ENO = 'A0008'
```

Translation:
Allocate A for the semantic table.
Allocate B for conditions of DELETE.

```
WHEN (S1- > A (S1- > FKEY = 'Yes') &
  S2- > B (S1- > REFT = S2- > TNAME)
BEGIN;
  EXEC SQL DELETE FROM Employee
    WHERE EMPNO = :S2- > EMPNO;
  EXEC SQL UPDATE :S1- > TNAME
    SET :S1- > FKEY = NULL;
END;
```

This processing uses referential integrity. In this case, the DELETE rule is SET NULL.

Execution. We can execute the program, which is pre-precompiled and compiled. Although we use KT, this pre-precompile approach may be adapted to a traditional programming language such as PL/I or COBOL.

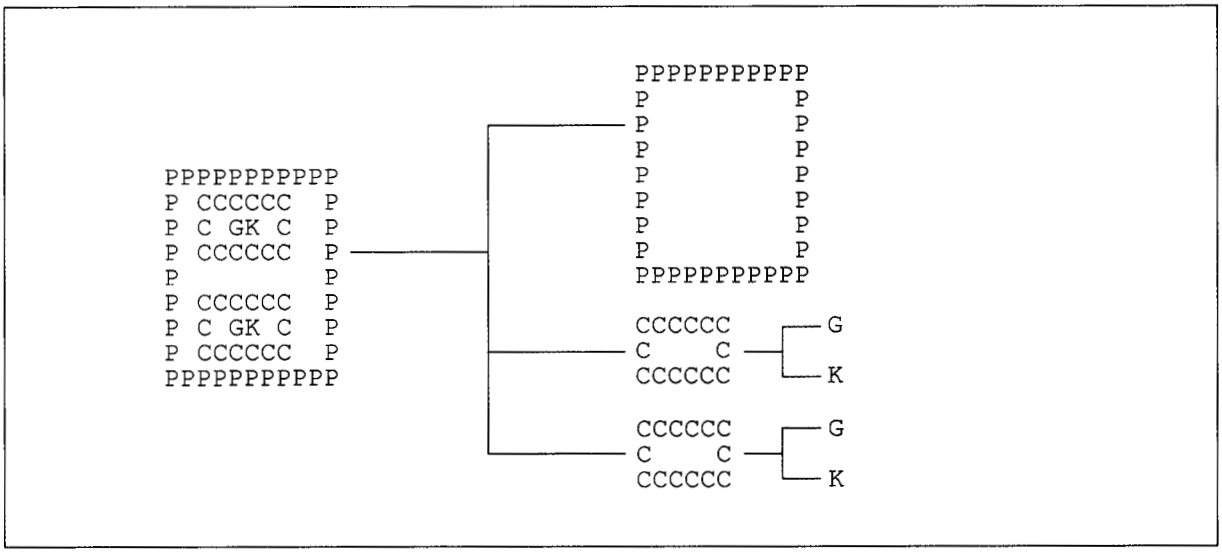
Recursive processing

Defining recursive processing. When a relation *R(a,b)* exists between the items *a* and *b* and a relation *R(b,c)* exists between the items *b* and *c*, a new relation *R(a,c)* is also established. The relation *R(a,c)* is called the *derived relation.* This derived relation is represented by the decision tree shown in Figure 12. The process of tracing the decision tree is known as *recursive processing* and is included in such AI architectures as PROLOG. Using PROLOG, we can derive *c* from *a* as indicated in Figure 12. However, current relational database systems have to record the relation *R(a,c)* as shown in Figure 13, because relational database systems cannot do recursive processing. If we can perform recursive processing, we can derive the relation without storing redundant data in the database. We now discuss the necessity of recursive processing and its implementation in a relational database environment.

The necessity of recursive processing. Consider the parts-relation shown in Figure 14, which is a typical relation requiring recursive processing as in the case of a bill-of-materials overview for all parts of a certain product. This type of relation is suitable for representing a hierarchical database, because the parts relation is represented hierarchically.

To avoid the complication of specifying the way to enter a certain record into a hierarchical database, we should migrate from the hierarchical database to the relational database gradually. However, it is difficult to store parts records in a relational database efficiently. Currently, we use an identifier that enables us to recognize the hierarchical relationship of parts to one another, and we add logic that manipulates the identifier in the program. This is the way we develop applications that can trace the hierarchy upward or downward, using the identifier and adding the capability of recursive processing to a relational database.

Figure 14 The parts-relation



The implementation of recursive processing. In order to do recursive processing, we have to declare the information of recursiveness in some table *R*, and we have to code in a special form that specifies recursion.

Preparing the declarative information of recursiveness. Because of the characteristics of a foreign key, recursiveness can be considered as the relationship between a primary key and the foreign key in tables. Consider the table shown in Figure 9. Given that there is a relationship between EMPNO (employee number) and AENO (administrative employee number), these are the primary key and the foreign key with respect to one another. If we want the name of an employee's manager, we trace columns in the following sequence:

EMPNO → AENO = EMPNO → AENO...

That is, we trace the sequence of the primary key and the foreign key. Thus, for example, managers of the employee number A0006 are A0003 and A0000.

We can record the information of the primary key and the foreign key in a DB2 catalog by executing CREATE statements as follows:

```
CREATE TABLE Employee ( .....
   PRIMARY KEY (EMPNO)
   REFERENCES Employee
   ON DELETE SET NULL)

CREATE UNIQUE INDEX Xemp
   ON Employee (EMPNO)
```

Of course, we can prepare the special table that retains the information of the primary key and the foreign key, as in the semantic table in Figure 10.

Coding statements for recursive processing. Many requests may be satisfied using recursiveness. For example, find the names of all managers in my management chain; or find the name of the manager whose second line is the president. If we want to specify these requests in one statement, we must obey predefined rules of syntax and procedures. This paper proposes a new syntax of SQL statements as follows:

- A statement is for recursive processing.
- The number of times and in which direction the recursion is to be done are given.

Let us review the new SQL statements in detail. Expansion to the SQL statement is

```
$SELECT column list FROM table list
    RECURSIVE
    USING starting-column-of-recursion
      ,{ALL| n}
  WHERE condition
```

where

Figure 15 Results of the pre-precompiler

```
(a)
$SELECT NAME FROM Employee
  RECURSIVE USING EMPNO ,ALL
  WHERE EMPNO = 'A0006'

is translated to the following statements,

  DCL  KEYAREA    CHAR(5);
  DCL  BAENO      CHAR(5);
  DCL  AENO(100) CHAR(5);
  DCL  BNAME      CHAR(20);
  DCL  NAME(100,100)  CHAR(20) INIT ('FFFFF');
  DCL HIAR_CNT FIXED BIN(31) INIT(0),
      REC_CNT  FIXED BIN(31) INIT(0);
  EXEC SQL DECLARE CURSOR C1 FOR
       SELECT NAME,AENO FROM Employee WHERE
         EMPNO = :KEYAREA;
  HIAR_MAX = 999999 ;
  AENO(1) = 'A0006' ;
  DO WHILE(SQLCODE = 0 & HIAR_MAX-HIAR_CNT>0) DO;
     REC_CNT = 0;
     KEYAREA = AENO(HIAR_CNT) ;
     EXEC SQL OPEN C1;
     HIAR_CNT = HIAR_CNT + 1 ;
     DO WHILE (SQLCODE = 0) DO;
        REC_CNT = REC_CNT + 1 ;
        EXEC SQL FETCH C1 INTO :BNAME , :BAENO ;
        NAME(HIAR_CNT,REC_CNT)= BNAME  ;
        AENO(HIAR_CNT) = BAENO ;
     END;
     EXEC SQL CLOSE C1 ;
   END;
```

- RECURSIVE indicates this statement contains recursive processing
- USING indicates the direction of the recursion; a column name is the primary key or the foreign key
- ALL|n indicates the number of recursive processes; the default is ALL, which means to get all data through the whole recursive process

The sample requests previously described are now coded, using the new SQL syntax as follows:

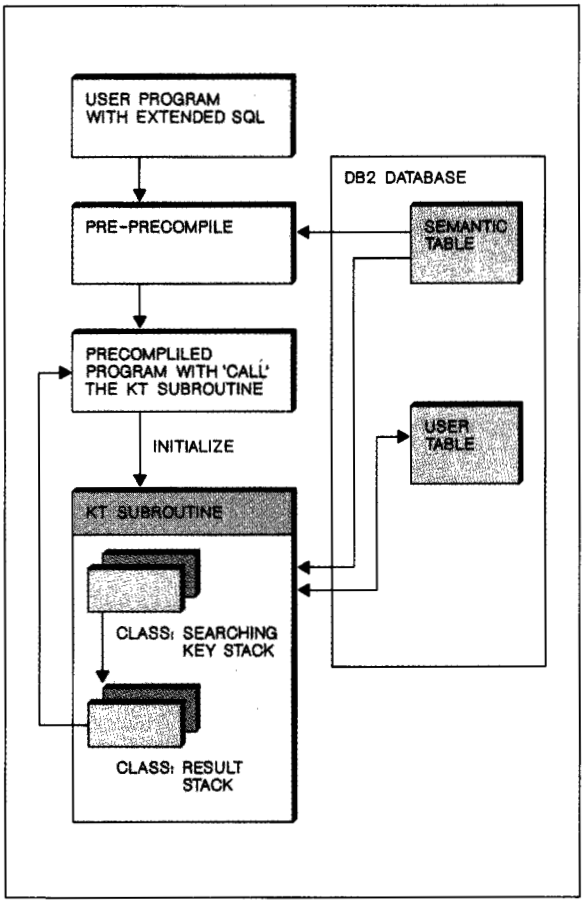
(a) $SELECT NAME FROM Employee
RECURSIVE USING EMPNO ,ALL
WHERE EMPNO = 'A0006'

(b) $SELECT NAME FROM Employee
RECURSIVE USING AENO ,2
WHERE AENO = 'A0000'

We can now process these requests. First, the statements have to be translated into conventional SQL form using the pre-precompiler. One approach is shown in Figure 15, which is the result of the pre-precompiler from the request (a) just given. Because it is easy to expand the current application using this programming language, this type of implementation is valuable for those who use a traditional programming language.

Implementation using the KnowledgeTool. Another implementation is required for users who use such AI tools as KT, which can allocate storage dynamically in responding to the new data. The KT also provides class-type variables that can select members immediately after conditions have been met. These capabilities are convenient to pass several answers to the program at the time of the execution of the extended SQL SELECT statement. We can also make inferences using rules stored in the knowledge base, after extracting the facts from the database. We can implement the pre-precompiler and routines for recursive processing using the skeleton of the process shown in Figure 16. The pre-precompiler interprets the

Figure 16 The recursive process using KT



extended SQL statement and checks the primary key and the foreign key in the semantic table. Then it creates two types of stacks—one stack for searching and the other for storing results. The KT subroutine is called by the user program at execution time and uses the KT functions. Using the skeleton in Figure 16, implementation of recursive processing is easy.

Processing ambiguous data

Definition of ambiguous data. *Ambiguous data* are defined as incomplete information that is stored as null values in a relational database. There are two meanings of null value: (1) don't-care value, which is not permanently stored; and (2) don't-know value, which is not yet stored. Examples of ambiguous data are shown in Figure 17.

Manipulating ambiguous data. Current relational database systems treat these two types of ambiguous data as null values and make no distinction between them. However, we want to treat null values in another way.

Consider the queries against ambiguous data:

- Name the persons who are proficient in English.
- Find the persons whose scores on the TOEIC are about 800 (TOEIC is Test of English for International Communication).

We want to retrieve the right information using such queries. In order to manipulate ambiguous data in DB2, we propose the pre-precompiler method, for which two tables must be created prior to pre-precompilation. An ordinary table is created, which contains real data as well as additional columns representing ambiguous data, an example of which is given in Figure 18. A keyword table is prepared, which contains special keywords for pre-precompiling, as shown in Figure 19.

The process of pre-precompilation is as follows:

1. Code the program with a special keyword given in the keyword table.

   Example 1. $SELECT EMPNO FROM VT1
   WHERE TOEIC = $high

   Example 2. $SELECT * FROM VT1
   WHERE TOEIC = $about800

Figure 17 Ambiguous data

EMPNO	TOEIC	
E1	812	
E2	-	← about 800
E3	-	← not yet assigned
E4	-	← not assigned
E5	730	
E6	590	

Figure 18 Table with ambiguous data

T1

EMPNO	TOEIC	TOEICF1	TOEICF2	
E1	812	-	-	
E2	-	780	820	← about 800
E3	-	700	1000	← high
E4	-	-	-	← unknown
E5	730	-	-	
E6	590	-	-	

← View (VT1) →

2. The pre-precompiler translates the extended SQL statement to the standard SQL statement using the keyword table.

   Example 1.

   ```
   $SELECT EMPNO FROM VT1
   WHERE TOEIC = $high
   ```

   produces

   ```
   SELECT EMPNO FROM T1
   WHERE TOEIC
   BETWEEN 700 AND 1000
   UNION
   SELECT EMPNO FROM T1
   WHERE TOEICF1 > = 700
   AND TOEICF2 < = 1000
   AND TOEIC IS NULL
   ```

   Example 2.

   ```
   $SELECT * FROM VT1
   WHERE TOEIC = $about800
   ```

Figure 19 Keyword table for the pre-precompiler

T2

TABLE	COLUMN	KEYWORD	VALUE1	VALUE2
VT1	TOEIC	$high	700	1000
VT1	TOEIC	$about800	780	820
VT1	TOEIC	@unknown		

produces

```
SELECT * FROM T1
WHERE TOEIC BETWEEN 780 AND 820
UNION
SELECT * FROM T1
WHERE TOEICF1 > = 780
AND TOEICF2 < = 820
AND TOEIC IS NULL
```

3. The program, after translation, is the program that contains only standard SQL statements. Therefore, we can continue the next ordinary step of precompiling. It is important to decide the special keyword like $high, and to standardize the meanings of the keywords for users in order to use the keyword correctly.

Concluding remarks

The implementation of a new experimental deductive database is discussed. This database uses the same first-order predicate logic as relational databases. Therefore, prototypes of the new deductive database are easily implemented using the relational database.

Another component is that of the object-oriented database, which incorporates the data and procedures. The use of object-oriented databases and deductive databases is also a topic of research in the relational model. In the future, we hope to combine databases and knowledge engineering.

Acknowledgments

The author is grateful to M. Egawa, H. Tsuchino, and I. Hayashi for their advice and comments during the preparation of this paper. The author also thanks George Stierhoff and the referees for their helpful comments.

Cited references

1. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13,** No. 6, 377–387 (1970).
2. E. F. Codd, "Relational Database: A Practical Foundation for Productivity," *Communications of the ACM* **25,** No. 2, 109–117 (1982).
3. C. J. Date, *Relational Database: Selected Writings*, Addison-Wesley Publishing Co., Reading, MA (1986).
4. D. C. Tsichritzis and F. H. Lochovsky, *Data Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1982).
5. J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems* **2,** No. 2, 105–133 (1977).
6. C. J. Date, *An Introduction to Database Systems, Volume II*, Addison-Wesley Publishing Co., Reading, MA (1983).
7. C. J. Date, *A Guide to the SQL Standard*, Addison-Wesley Publishing Co., Reading, MA (1987).
8. R. Hull, "A Survey of Theoretical Research on Typed Complex Objects," *Database*, J. Paredaens, Editor, Academic Press, London (1987), pp. 193–256.
9. H. Gallaire, J. Minker, and J.-E. Nicolas, "Logic and Databases: A Deductive Approach," *ACM Computing Surveys* **16,** No. 2, 153–185 (1984).
10. B. E. Jacobs, *Applied Database Logic I: Fundamental Database Issues*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1985).
11. C. J. Date, *An Introduction to Database Systems, Volume I*, Addison-Wesley Publishing Co., Reading, MA (1986).
12. T. Hirao, *Relational Database System*, Kindai-kagaku-sha (1986, in Japanese).
13. Y. Kambayashi, "Semantics of Data Structures," *Journal of the Information Processing Society of Japan* **27,** No. 2, 129–139 (1986, in Japanese).
14. J. D. Ullman, *Principles of Database Systems*, 2nd edition, Computer Science Press, MD (1982).
15. J. Peckham and F. Maryanski, "Semantic Data Models," *ACM Computing Surveys* **20,** No. 3, 153–189 (1988).
16. *IBM DATABASE 2 General Information Manual*, GC26-4073-3, IBM Corporation (1987); available through IBM branch offices.
17. M. L. Brodie, "On the Development of Data Models," *On Conceptual Modeling*, M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, Editors, Springer-Verlag, Inc., NY (1984), pp. 19–47.

Takayuki Hirao *IBM Japan Ltd., 1, Kanda Izumi-cho, Chiyoda-ku, Tokyo 101, Japan.* Mr. Hirao is an advisory instructor in the IBM Japan Education Center, where he is working on education activities for customers. He joined IBM Japan in 1974, working as a systems engineer (SE) until 1980. He received his B.S. in applied mathematics from Tokyo Education University (now renamed Tsukuba University). Mr. Hirao is the author of *Relational Database Systems* (in Japanese) and a member of the Information Processing Society of Japan and Japanese Society for Artificial Intelligence.

Reprint Order No. G321-5417.