RSD-TR-22-87

EXTENDING THE RELATIONAL MODEL TO SUPPORT HORN-CLAUSE AS USER INTERFACE

Y.C. Lee

C.H. Cho

December 1987

CENTER FOR RESEARCH ON INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109-1109

ABSTRACT

This paper presents an approach which takes advantages of both relational and deductive databases. Although techniques of relational databases are well-developed, the relational model still need a more natural user interface. Deductive databases (DDBs) have then been suggested as alternative solutions; however, they suffer from high-cost theorem provers. The proposed approach uses Horn-clauses as user interface and the relational model as its internal representation. This approach not only offers a more natural and flexible interface to users but also takes advantage of the developed relational database techniques. Such an approach calls for an algorithm which converts Horn-clauses into relational algebraic expressions and some extensions to the relational model so as to capture the extra expressive power in Horn-clauses.

Contents

1 Introduction

Every database is a model of some part of the real world. The design of a database application thus consists of choosing appropriate data structures that would reflect the structure of the world being modeled. A desired database model for general applications must satisfy requirements such as efficient manipulation of large quantities of data, natural user interface, flexibility of modification, sharing and distribution of data, and so on.

The relational model proposed by Codd has been well-developed. Relational database techniques which include view definitions, query optimization, distributed databases, and database machines have achieved satisfactory results. View definitions allow users to specify dynamic pictures of some parts of the database. Query optimization reduces both the computation time and space of data manipulation. Distributed databases distribute data among several different sites in order to obtain a high degree of data sharing with low cost. Database machines are machines of special architectures (usually with parallel processing) such that they can execute the database operations efficiently [3,4,5,12,23,24,25].

The manipulation system used by the relational database is basically *relational algebra.* It is an algebraic system with a set of operations called *relational operations.* Usually a user uses DBMS primitives, i.e., relational operators, to specify queries or to construct virtual relations, i.e., *views.* These primitives force the user to translate problems into detailed specifications. It burdens the user with the responsibility of finding out the path and making the plan to retrieve data from database.

Deductive databases (DDBs), proposed by Chang [1,2], Kellogg, Klahr and Travis [11], Minker [16], and Reiter [20], have been suggested as alternative solutions to the problem stated above. Concentrating mainly on evaluations of queries, these approaches enroll automatic deduction, *theorem prover*, to derive answers for nonprocedural queries [6]. All the derivation rules and facts are logic axioms which provide a more natural and expressive representation.

Unfortunately, DDBs that are based on theorem prover can hardly achieve a good performance in cases of large amount of data. On the other hand, relational databases, albeit not extremely efficient by all means, have shown satisfactory performance in managing vast amount of data primarily due to the growing query optimization techniques as well as efficient access methods. In addition, relational views, if defined by *relational calculus* [25], can be as nonprocedural and natural as their corresponding derivation rules. DDBs may look more intelligent but their derivation rules must also be defined, just like the predefined views in relational databases. So, why DDBs? The answer is clear that DDBs are capable of dealing with intensional as well as indefinite facts through the use of derivation rules [20].

This paper presents an approach with advantages of both deductive and relational databases. This approach uses Horn-clauses as user interface like most of the DDBs do and uses the relational model as its internal representation. Therefore, the user interface is more natural and the well-developed relational database techniques are still applicable to this model. Such an approach requires an algorithm which converts Horn-clauses into relational algebraic expressions. It also needs some extensions to the relational model since the logic domain of Horn-clauses can not be fully covered [13].

There are two main objectives in this paper. One is to identify possible logical extensions of relational databases and of relational algebra so that not only the extended logic domain covers the same domain of Horn-clause but also query optimization techniques remain applicable to the extended domain. The other is to propose an algorithm that converts Horn-clauses into relational algebraic expressions. Section 2 gives a brief introduction of the relational model, deductive databases, and mathematical logic. It also provides an analysis of all types of clauses so that possible logic extensions can be identified. Section 3 introduces two special values, namely, the *existential* and the *universal values*, and a partial order defined on them. Together, they will allow intensional facts to be manipulated by relational operations and thus benefit from query optimization techniques. Section 4 presents an algorithm which converts Horn-clauses, used as derivation rules, into relational algebraic expressions. Thus, the extended relational database can be enhanced to perform at least the same functions as Definite DDBs (DDDBs) can. Section 5 summarizes our results and discusses some future work.

2 Background

A brief overview of relational databases, deductive databases, and mathematical logic is presented in this section. The mathematical logic constructs a conceptual framework for database models. The database concepts, which can be analyzed in terms of formal logic, forms a characterization of the hypothetical world on which database systems work.

It is known that, logically, the relational algebra can only represents a subset of the clauses in *first-order logic*; some of the clauses have no corresponding relational algebraic expressions. These clauses will be identified through the classification in Section 2.3. The method for relational databases to capture these additional expressive power will be presented in Section 3.

2.1 Relational model

The relational model proposed by Codd has a tabular structure to describe a database. In this model, the database is viewed as a collection of time-varying relations of assorted degrees. A relation can be viewed as a table of a finite number of columns and rows. The table name is the *relation name*. Columns of table are called *attributes*. Rows of the table correspond to *tuples*(records). Order of rows and columns of the table are immaterial. Such a table is called a *normalized relation* if every entry of this table is *atomic* [2,3,4,5,12,25].

¿From logic point of view, a row in the relational database is actually a *ground atomic formula* in first-order logic, and the database state can be viewed as a collection of formulas belonged to two classes [10]: *database intension* (IDB) and *database extension* (EDB).

- *Database intension* corresponds to time invariant properties of database and usually consists of integrity constraints and view definitions.
- *Database extension* reflects the current state of knowledge about the part of the modeled world and is subject to frequent updates.

The relational model also provides a set of relational operations; *renaming, selection, projection, join, union, difference, and complement*. These operations are implemented in the relational *database management system* (DBMS) so that database users can use them to manipulate EDB.

2.2 Deductive databases

Logic was chosen as the principle of deductive database design since it is a effective way of representing knowledge, and it constructs a mathematical basis both for reasoning with data and for sustaining the integrity of a database.

Several researchers apply the automatic deduction techniques of theorem proving on the process of query manipulating [1,2,7,8,11,17,16,18,20,14]. Minker applies the theorem prover to both IDB and EDB while Reiter and Chang apply it to IDB only. When applying theorem prover on both IDB and EDB, every single tuple is considered as a predicate and is used to support the theorem proving as a fact. Since it is a pure theorem prover rather than a DBMS, it is not efficient when EDB $\gg$ IDB. The later strategy, applying theorem prover only on IDB, assumes that every relation contains all possible tuples and thinks of each relation as a predicate, in which all the terms are universal variables. After the success of theorem proving, the relations represented by those predicates which are used to support this proof are then processed by a relational DBMS to answer the query.

In Minker's approach [16], queries to the system consist of *well-formed formulas* in first-order logic. Knowledge is stored in a semantic network within the system. The semantic network consists of explicit facts cumulated in EDB; general clauses, which permit new facts to be derived, are stored in IDB. EDB and IDB are treated as one file; and, consequently, there is no distinction between them. The query specified by a user is resolved together with this file by a theorem prover. An answer is immediately obtained from the theorem proving procedure.

Chang's approach [1,2] allows users to state queries, derivation rules and integrity constraints. A user can make queries against virtual and base relations. If a query is free of virtual relations, it will be evaluated directly by a relational DBMS. If it contains virtual relation, then it will be processed by a deductive procedure before it is evaluated by the DBMS. This query is thus evaluated in two steps: First, apply clauses to transform it into a query which contains only base relations. Second, check if the transformed query can be answered from front-end intensional information, such as integrity constraints, within a prespecified time limit. If it works, output the answer. Otherwise, evaluate the transformed query by using a database system such as System R.

The approach taken by Reiter [20] is to equip a deductive component with the techniques for query evaluation on relational databases in such a way that achieves definite and natural interfaces. This approach is relatively more feasible for a large database, that is, a database with large EDB and comparatively small IDB. In this approach, the task of question-answering is decomposed into a theorem prover solving on IDB and an extensional processor computing on EDB. The theorem prover "sweeps through" IDB, extracting all information relevant to a given query. Exclusively, this theorem-prover never look at EDB. The end result of this sweep is a set of queries, which will be evaluated individually and extensionally. The union of answers returned from those individual queries forms a set of answers corresponding to the original query. This task decomposition concludes with two important consequences:

- The extensional processor can be realized by a relational database management system.
- Since the theorem prover never accesses EDB, IDB can be compiled by using the theorem prover as a once-only compiler.

Minker's and Chang's approaches are referred to as *definite deductive databases* (DDDBs), while Reiter's approach is called *indefinite deductive database* (IDDB). The difference between DDDB and IDDB is that the former allows only Horn-clauses but the later allows all clauses. That is, only IDDB permits indefinite rules, such as *"If A is son of B, then B is either father or mother of A."* and *"If C leads a project, then C could be a manager, a senior engineer or a research fellow"*.

One of the general advantages of DDBs is that they allows *general statements*, a set of facts, to be specified in IDB. General statements provide the capability of storing general information, such as *"Smith is the pilot for all the flights between Detroit and Cleveland."* and *"John participates all the projects in the Robotics Laboratory."* Since general statements are stored in IDB, facts in the system are now distributed to both IDB and EDB instead of being confined in EDB.

Considering the formulation of databases in terms of logic, it is worthwhile mentioning some major assumptions that govern the query evaluation (and integrity constraint) of databases. There are three assumptions, made by most approaches described, which express a certain implicit representation of negative facts and make precise the universe to which queries refer. [7]

- The *close world assumption* (CWA), which states that facts not known to be true are assumed to be false. (Since Reiter's approach involves indefinite fact, it requires *generalized closed world assumption* [20,22].)
- The *unique name assumption*, which indicates that individuals with different names are different.

- The *domain closure assumption*, which assumes that there are no other individuals existed except those in the database [21].

2.3 Mathematical logic

Both the models above lay their basis on mathematical logic either implicitly or explicitly. The relational model treats a database as a model for the given set of *sentences*. Deductive databases use logic as their knowledge representation and manipulate the model as what mathematical logic does.

In order to analyze clauses, the definitions of *sentences* and *clauses* in first-order logic should be given. The definition of sentences require that *terms*, *atomic formulas*, and *formulas* be introduced first.

[Definition 1] *Terms* are defined recursively as the following:

(1) a constant is a term;

(2) a variable is a term;

(3) if f is an n-ary function and $t_1, ... t_n$ are terms, then $f(t_1, ..., t_n)$ is a term.

[Definition 2] Let P be an n-ary predicate (relation) ($n \geq 0$) and $t_1, ..., t_n$ be terms, then $P(t_1, ..., t_n)$ is an *atomic formula*. An atomic formula or the negation of the atomic formula will be referred to as a *literal*. Notice that predicates in first-order logic is similar to relations in the relational model.

[Definition 3] *Formulas* are defined by recursion as the following:

(1) atomic formula is formula;

(2) if A is a formula, then so is $\neg A$;

(3) if A and B are formulas, then so are $A \wedge B$, $A \vee B$, and $A \rightarrow B$;

(4) if A is a formula and x is an individual variable, then $(\forall x)A$ and $(\exists x)A$ are formulas.

[Definition 4] A *sentence* is a formula without free variable. That is, every variable in this formula is bound by either $\forall$ or $\exists$.

In order to analyze the sentences in first-order logic, we have to convert the sentences into a certain class of formulas called clauses. A clause is defined as a formula consisting of a disjunction of literals. Every sentence can be converted to a set of clauses by the following procedure [19]:

(1) Eliminate implication symbols by making the substitution $\neg A \vee B$ for $A \rightarrow B$.

(2) Reduce scqpes of negation symbols such that each negation symbol, $\neg$, applies to at most one atomic formula.

(3) Standardize variables such that in each sentence there is no variable being quantified twice.

(4) Eliminate existential quantifiers by converting it into Skolem form. For example, replace $(\forall y)[(\exists x)P(x, y)$ by $(\forall y)P(f(y), y)$, where function f is called a Skolem function.

(5) Convert to prenex form by moving all the quantifiers to the front of the sentence and let the scope of each quantifier include the entire sentence. This sentence consists of a list of quantifiers called a prefix followed by a quantifier-free formula called a *matrix*.

(6) Put matrix in conjunctive normal form, that is, the conjunction of a finite set of disjunctions of literals.

(7) Eliminate universal quantifiers.

(8) Eliminate $\wedge$ symbols by replacing expression of the form $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n$ with the set of conjunctions of literals $\{\varphi_1, \varphi_2, ..., \varphi_n\}$.

Although the clause is defined as a disjunction of literals, it can be written in an equivalent form. Thus, the simplified general form of clauses

$$\neg P_1 \vee \neg P_2 \vee ... \vee \neg P_i \vee ... \neg P_k \vee R_1 \vee R_2 \vee ... \vee R_j \vee ... \vee R_q$$

is

$$P_1 \wedge P_2 \wedge ... \wedge P_i \wedge ... \wedge P_k \rightarrow R_1 \vee R_2 \vee ... \vee R_j \vee ... \vee R_q$$

where P_i and R_j are positive literals.

Now, we follow the approach in [7] to analyze the clauses in first-order logic and classify them into several types. Depending on the respective values of k and q in the clause above, there are various types of clauses, some of them being integrity constraints, intensional facts and indefinite assertion that are associated with IDB while the rest are derivation rules that manipulate the facts in DDBs:

- *Type 1: $k = 0, q = 1$.* $\rightarrow R_1(t_1, t_2, ..., t_m)$

  Since there is no "pre-condition", $R_1(t_1, t_2, ..., t_m)$ is always true and considered as a fact. If $t_1, t_2, ..., t_m$ are constants, then it is a single fact like a tuple in a relational database. Otherwise, it is called a *general statement* and is stored in IDB as an *intensional fact* of DDBs. A general statement has no counter part in relational databases.

- *Type 2: $k \geq 1, q = 1$.* $P_1 \wedge ... \wedge P_k \rightarrow R_1$

  This clause may be considered as either an *integrity constraint* or a *definition of the predicate R_1 in terms of the predicates $P_1, P_2, ..., P_k$*. If considered as a definition of the predicate R_1, then this clause is a *derivation rule* in DDB and may be considered as a *query* or *view definition* in a relational database if every argument of predicate R_1 is a variable which occurs in some of the predicates $P_1, P_2, ..., P_k$. Otherwise, it can not be interpreted in the relational model.

- *Type 3: $k = 1, q = 0$.* $P_1(t_1, t_2, ..., t_m) \rightarrow$

  It is equivalent to the clause $\neg(P_1(t_1, t_2, ..., t_m))$ and thus stands for either an *integrity constraint* or *negative fact*. Since the negative fact can be implied by the absence of positive fact under closed world assumption, we would like to consider it as an integrity constraint in this paper.

- *Type 4: $k > 1, q = 0$.* $P_1 \wedge P_2 \wedge ... \wedge P_k \rightarrow$

  Such clause is usually thought of as an *integrity constraint* by the same reason in *Type 3*.

- *Type 5: $k = 0, q > 1$.* $\rightarrow R_1 \vee R_2 \vee ... \vee R_q$

  This clause is an *indefinite assertion*, since we only know that some of the literals is true but do not know exactly which one of them is true.

- *Type 6: $k \geq 1, q > 1$.* $P_1 \wedge ... \wedge P_k \rightarrow R_1 \vee R_2 \vee ... \vee R_q$

  The clause may be interpreted as either an *integrity constraint* or the *definition of some indefinite data*.

Type 1 through *Type 4* are called *Horn-clauses*. Neither *Type 5* nor *Type 6* is allowed in DDDBs. The only deductive databases that can handle these two types of clauses is IDDB [7].

(1) Eliminate implication symbols by making the substitution $\neg A \vee B$ for $A \rightarrow B$.

(2) Reduce scqpes of negation symbols such that each negation symbol, $\neg$, applies to at most one atomic formula.

(3) Standardize variables such that in each sentence there is no variable being quantified twice.

(4) Eliminate existential quantifiers by converting it into Skolem form. For example, replace $(\forall y)[(\exists x)P(x, y)$ by $(\forall y)P(f(y), y)$, where function f is called a Skolem function.

(5) Convert to prenex form by moving all the quantifiers to the front of the sentence and let the scope of each quantifier include the entire sentence. This sentence consists of a list of quantifiers called a prefix followed by a quantifier-free formula called a *matrix*.

(6) Put matrix in conjunctive normal form, that is, the conjunction of a finite set of disjunctions of literals.

(7) Eliminate universal quantifiers.

(8) Eliminate $\wedge$ symbols by replacing expression of the form $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n$ with the set of conjunctions of literals $\{\varphi_1, \varphi_2, ..., \varphi_n\}$.

Although the clause is defined as a disjunction of literals, it can be written in an equivalent form. Thus, ıe simplified general form of clauses

$$\neg P_1 \vee \neg P_2 \vee ... \vee \neg P_i \vee ... \neg P_k \vee R_1 \vee R_2 \vee ... \vee R_j \vee ... \vee R_q$$

is

$$P_1 \wedge P_2 \wedge ... \wedge P_i \wedge ... \wedge P_k \rightarrow R_1 \vee R_2 \vee ... \vee R_j \vee ... \vee R_q$$

where P_i and R_j are positive literals.

Now, we follow the approach in [7] to analyze the clauses in first-order logic and classify them into several types. Depending on the respective values of k and q in the clause above, there are various types of clauses, some of them being integrity constraints, intensional facts and indefinite assertion that are associated with IDB while the rest are derivation rules that manipulate the facts in DDBs:

3 Extension of Relational Model

The discussion in previous section has shown that intensional facts, which include general statements and *general clauses*, are not taken care of by the relational model. A general clause is a clause of *Type 2* with some variables in R_1 but not in any of $P_1, P_2, ..., P_k$ or with (Skolem) function in R_1. The main purpose of this section is thus to enhance the expressive power of the relational model by implementing the facilities to manipulate these intensional facts. In order to do so, two special values are introduced, a partial order is defined (to remove some redundancies), and some modifications of the relational operations are made.

3.1 Existential and universal values

Our approach to supporting the general statements is to define the *universal* value ϖ and *existential* value λ so as to store the general statements as tuples in relations of database. In other word, the general statements, which used to be in the IDB of DDBs, will be in the EDB of the extended relational databases. These two values are defined as the following:

Define an universal value, ϖ, in relational database to represent the variable in a general statement. Under the Domain closure assumption, ϖ represents the set of all values that correspond to the same attribute name.

Define an existential value, λ, in relational database to represent the Skolem function. It is an incomplete fact that denotes "there exists some value" but fails to indicate the exact value.

Accordingly, a predicate $P(c_1, c_2, ..., f_1(x_1), f_2(x_2), ..., x_1, x_2, ...)$ can be represented by the tuple $\langle c_1, c_2, ..., \lambda_1, \lambda_2, ..., \varpi_1, \varpi_2, ...\rangle$ as a single tuple of a relation.

3.2 Partially ordered sets and redundant tuples

In a multi-tuple relation with universal or existential value, a tuple might be subsumed by the fact that represented by another tuple. For example: "*John Smith takes course EECS 585*" is implied by "*John Smith takes all courses*". The tuple which is covered by another tuple is called a *redundant tuple* of the relation.

In order to identify the redundant tuples of relations with universal and existential values, we have to define a partially ordered set for the domain of each attribute. Let $S = D \bigcup \{\varpi, \lambda\}$ where D is the *active domain* [15] of attribute A. The set S together with a partial ordering relation "$\leq$" is referred to as the partially ordered set P. P is said to be the partially ordered set for attribute A. The partial ordering $\leq$ satisfies the following for every a, b and c in S:

- *Reflexive :* $a \leq a$;
- *Antisymmetric:* $a \leq b$ and $b \leq a$ implies $a = b$;
- *Transitive :* if $a \leq b$ and $b \leq c$, then $a \leq c$.

The partial ordering relation P is defined as: let $a, b \in S$, then the ordered pair $(a, b) \in P$ if and only if $a = \lambda$ or $b = \varpi$ or $a = b$

[Definition 5] Let P_i be the partially ordered set for attribute A_i. A tuple $\langle a_1, a_2, ..., a_n \rangle$ of relation $r[A_1, A_2, ..., A_n]$ is said to be *redundant* if there exists another tuple $\langle b_1, b_2, ..., b_n \rangle$ in the same relation r such that

$$(a_i, b_i) \in P_i, 1 \leq i \leq n$$

[Application] All the redundant tuples can be removed without changing the state of this database, since the fact it represents is covered by some other tuple.

3.3 Modification of relational operations

Since values ϖ and λ are different from other attribute values, some modifications of the relational operations should be made to accommodate these two values based on the partial order defined above. Two methods are proposed to modify the relational operations. One modifies every relational operation directly while the other modifies the relational algebraic expression instead.

The first method is to modify the relational operations as the following:

- *Renaming:* No modification is made, since this operation has nothing to do with attribute values.
- *Selection:* If there is a ϖ at the "selected" attribute, then expand it with respect to the selected attribute as the following:

Replace tuple $\langle a_1, ..., a_{i-1}, \varpi, a_{i+1}, ..., a_n \rangle$ by the set of tuples $\langle a_1, ..., a_{i-1}, \alpha, a_{i+1}, ..., a_n \rangle, \forall \alpha \in D_i$.

If there is a λ at the "selected" attribute, then remove this tuple.

After the conversion of all the tuples ϖ or λ at the "selected" attribute, the selection is performed as usual.

- *Projection, union* and *Cartesian product* work as usual at first and then remove all the redundant tuples.

The method suggested above requires a modification to existing relational operations. This requirement may appear to be costly and undesirable. The second method suggested below will modify, instead, the relational algebraic expressions which are to be evaluated and involve some relations that contain ϖ. Assuming that the relation $r(R)$ has been defined with some value ϖ in attribute A, r in the relational algebraic expression will be replaced by

$$r\prime(R) \leftarrow \pi_R \delta_{A,tmp \leftarrow tmp,A} \{ r \bowtie [(\{\langle \varpi : A \rangle\} \times \delta_{tmp \leftarrow A} \sigma_{A \neq \varpi} \pi_A(r)) \cup \sigma_{tmp = A}(\sigma_{A \neq \varpi} \pi_A(r) \times \delta_{tmp \leftarrow A} \sigma_{A \neq \varpi} \pi_A(r))] \}$$

In a sense, the expanded relation $r\prime(R)$ is nothing but a view constructed from $r(R)$ and the augmented active domain of A, which is a relation exactly as the following:

$$\{ \langle \varpi \; a_1 \rangle, \langle \varpi \; a_2 \rangle, ..., \langle \varpi \; a_n \rangle, \langle a_1 \; a_1 \rangle, \langle a_2 \; a_2 \rangle, ..., \langle a_n \; a_n \rangle \}.$$

The modified relational algebraic expression, albeit more complicated, can often be optimized as a query tree.

4 Horn-Clauses as User Interface

The second goal of this paper is to support the Horn-clause as user interface. This section will first analyze Horn-clauses and then propose an algorithm which converts a query or view definition specified in the form of Horn-clause to an relational algebraic expression. It also provides an example to demonstrate the proposed algorithm and then discusses the result of this section.

4.1 Analysis of Horn-clauses

Since Horn-clauses in DDDBs cover the clauses of *Type 1* through *Type 4*, we can find the logic domain which is covered by DDDBs but is not dealt with by the relational model by examining the clauses of *Type 1* through *Type 4*. An investigation of Horn-clauses based on the classification of Section 2.3 first suggests the following:

- All the clauses of *Type 3, 4* and some clauses of *Type 2* which are interpreted as integrity constraints are enrolled in the control part of the database and are not to be manipulated or derived.
- Each clause of *Type 1* with no variable or function corresponds to nothing but a constant tuple in relational algebra.
- The clauses of *Type 1* with variables or functions are general statements stored in IDB. Such intensional facts can not be treated by conventional relational operations that are performed on EDB only; and, as a result, the intensional facts has no corresponding part in conventional relational databases. However, with the existential and universal values, it can be stored as a "special" single tuple in EDB and be manipulated by a similar manner. Therefore, general statements can be interpreted in the extended relational model.

It turns out that clauses of *Type 2*, which are used as derivation rules in DDDBs, deserve a further study in order to examine their distinctions from query or view definitions in relational algebra. The derivation rules, clauses of *Type 2*, in DDDBs are of the form

$$P_1(x_1^{(1)}, ..., x_{i_1}^{(1)}) \wedge ... \wedge P_k(x_1^{(k)}, ..., x_{i_k}^{(k)}) \rightarrow R(y_1, ..., y_j)$$

where, $k \geq 1$ and each P_m $(1 \leq m \leq k)$ can be either of the forms $x_i \theta x_j$ or $x_i \theta c$, or corresponds to a relation or view of the database.

Primarily dealing with the above derivation rules, an algorithm is proposed next to convert Horn-clauses as a whole into relational algebraic expressions. It shows that the derivation rule above can be replaced by a view definition if all the arguments at the right-hand side (RHS) of "$\rightarrow$" in the derivation rule are variables and if all of them appear at the left-hand side (LHS) as well. The case that some RHS variables do not appear at LHS is very similar to the general statement of *Type 1* and will also be taken care of with the help of existential and universal values. Therefore, the extended relational model covers the entire logic domain that can be represented by Horn-clauses [9,26,27].

4.2 Algorithm

Given a derivation rule, we can first manipulate the LHS in order to get a corresponding relational algebraic expression, $\Re$, by the first four steps of the algorithm. Then, at Step 5 of the algorithm, we can process the RHS, $R(y_1, y_2, ..., y_j)$, based on $\Re$ to produce the equivalent view definition in relational algebra.

In the algorithm below, we assume that c is constant, v is variable or function, A_i is the corresponding attribute name for the position v_i is located and "$\leftarrow$" stands for "assignment".

Algorithm:

Input : $P_1, P_2, ..., P_k, R$: predicates

$\bar{R_{l_1}}, \bar{R_{l_2}}, ..., \bar{R_{l_r}}$: relational schemes corresponding to $P_{l_1}, P_{l_2}, ..., P_{l_r} (1 \le l_1 \le ... \le l_r \le k)$

Output : $\Re$: relational algebraic expression

Step 1: Collect all the predicates of the set $\{P_1, P_2, ..., P_k\}$ that correspond to some relations or views of the database and convert them into relations or views by the following rules: (Note: In the following, the notation "$\mapsto$" means "convert to," and the notation "$\leftarrow$" means "represent the relational algebraic expression.")

Let $\bar{R_m}(A_1, ..., A_{i_m})$ be the relation corresponds to $P_m(x_1, ..., x_{i_m})$,

$$P_m(v_1, v_2, ..., v_{i_m}) \quad \mapsto \quad \Re_m \leftarrow \bar{R_m}(A_1, A_2, ..., A_{i_m})$$

$$P_m(c_1, ..., c_l, v_{l+1}, ..., v_{i_m}) \quad \mapsto \quad \Re_m \leftarrow \pi_{A_{l+1}, ..., A_{i_m}} \sigma_{A_1 = c_1, ..., A_l = c_l} \bar{R_m}(A_1, A_2, ..., A_{i_m})$$

If v_i is exactly the same variable or function as some v_j in P_m, then

$$\Re_m \leftarrow \pi_{A_1, ..., A_{i-1}, A_{i+1}, ..., A_{i_m}} \sigma_{A_i = A_j} \Re_m$$

If there are more than one pair of v_i and v_j, then repeat this process until there is no duplicate variables or functions left.

For all the distinct variables or functions $v_{m_1}, ..., v_{m_p}$ of P_m and their corresponding attribute names $A_{m_1}, ..., A_{m_p}$ in $\bar{R_m}$, $1 \leq m_1 \leq ... \leq m_p \leq i_m$,

$$\Re_m \leftarrow \delta_{V_{m_1}, ..., V_{m_p} \leftarrow A_{m_1}, ..., A_{m_p}} \Re_m$$

(The V_i in this renaming is exactly the same "symbol" as v_i. We use v_i as a term in the predicate and V_i as an attribute name).

Step 2: Join (nature join) all the results from step 1 to form $\Re$.

Step 3: Collect the predicates left in step 1 except R, which is the predicate at RHS, to form the set S. This set would then consist of only predicates that are associated with "θ".

Step 4: For each $s_l = x_i \theta x_j$ or $s_l = x_i \theta c$ in S, $l = 1, ..., | S |$ do the following:

$$\Re \longleftarrow \sigma_{X_i \theta X_j} \Re$$

or

$$\Re \longleftarrow \sigma_{X_i \theta c} \Re$$

Step 5: Let $R(f_1, ..., f_q, u_1, ..., u_r, c_1, ..., c_s, v_1, ..., v_t)$ be the predicate at RHS and the corresponding relation scheme of R is $R_V[F_1, ..., F_q, U_1, ..., U_r, C_1, ..., C_s, A_1, ..., A_t]$, where c stands for constant, f stands for Skolem function, u stands for variable which is absent from LHS and v is variable that appears at LHS.

Depending on the respective values of q, r, s and t in R, the result from Step 4, "$\Re$", will be manipulated by one of the following cases:

Case 1: $q = r = 0, s > 0, t = 0$

All of the terms are constants.

$$\Re \leftarrow \pi_{C_1, ..., C_s}(\{\langle c_1, ..., c_s \rangle\} \times \Re)$$

This is a special view definition where the view, a single fact, is verified by checking condition $(P_1 \wedge ... \wedge P_k)$ of the LHS. The result of $\Re$ can be either $\{\langle c_1, ..., c_s \rangle\}$ or ϕ depending on whether the condition is true or not.

Case 2: $q = r = 0, s = 0, t > 0$

All of the terms are variables that appear at LHS.

$$\Re \leftarrow \delta_{V_1,...,V_t \leftarrow A_1,...,A_t}(\pi_{V_1,...,V_t}\Re)$$

The relational algebraic expression $\Re$ is a view definition on base relations and views that correspond to predicate $P_1, ..., P_k$.

Case 3: $q = r = 0, s > 0, t > 0$

Some of the terms are constants and the others are variables that appear at LHS.

$$\Re \leftarrow \{\langle c_1, ..., c_s \rangle\} \times (\delta_{V_1,...,V_t \leftarrow A_1,...,A_t}\pi_{V_1,...,V_t}\Re)$$

It is a view definition similar to the one in *Case 2*, except that some attributes of the view being defined are constants.

Case 4: $q > 0$ or $r > 0$

Some of the terms are variables that are absent from LHS or functions.

Predicate R can be thought of as being formed by two independent components: $R_p(v_1, ..., v_t)$ and $R_g(f_1, ..., f_q, u_1, ..., u_r, c_1, ..., c_s)$. While the corresponding relational algebraic expression for R_p can be easily constructed by a process similar to *Case 2*, R_g is analogous to the general statement of *Type 1* clause. Since R_g has nothing to do with the LHS, it can be represented by a tuple $\langle \lambda_1, ..., \lambda_q, \varpi_1, ..., \varpi_r, c_1, .., c_s \rangle$, where λ is an existential value and ϖ is an universal value. Similar to *Case 2*, the relational algebraic expression for R_p is:

$$\delta_{V_1,...,V_t \leftarrow A_1,...,A_t}\pi_{V_1,...,V_t}\Re$$

The Cartesian product of the relational algebraic expression corresponding to R_p and the tuple representing the general statement R_g is thus the relational algebraic expression desired. That is,

$$\Re \leftarrow \{\langle \lambda_1, ..., \lambda_q, \varpi_1, ..., \varpi_r, c_1, ..., c_s \rangle\} \times (\delta_{V_1,...,V_t \leftarrow A_1,...,A_t}\pi_{V_1,...,V_t}\Re)$$

In this algorithm, we have shown that most of the derivation rules have corresponding view definitions in conventional relational algebra, while some derivation rules that involve general information must be supported by existential and universal values.

4.3 Example

An example is provided in order to illustrate the proposed algorithm. Presumably, a set of examples, one for each case in Step 5 of the algorithm, would be needed. However, Case 4, which is the most complicated one, is general enough to cover others.

[Example 1]

Let $P_1(u, v, f(u)) \wedge P_2(v, w, \text{“}aaa\text{”}) \wedge P_3(w, x, y, x) \wedge (v > y) \wedge (x = \dot{0}) \rightarrow R(f(u), z, w, \text{“}bbb\text{”})$ be the Horn-clause to be converted to relational algebraic expression. Let $\bar{R}_1(A_1, A_2, A_3)$, $\bar{R}_2(B_1, B_2, B_3)$ and $\bar{R}_3(C_1, C_2, C_3, C_4)$ be the relation schemes corresponding to P_1, P_2 and P_3 respectively. The results of each step using this example are shown as the following:

Step 1: P_1 is converted to $\delta_{u,v,f(u) \leftarrow A_1,A_2,A_3} \bar{R}_1$;

P_2 is converted to $\delta_{v,w \leftarrow B_1,B_2} \pi_{B_1,B_2} \sigma_{B_3=\text{“}aaa\text{”}} \bar{R}_2$; and

P_3 is converted to $\delta_{w,x,y \leftarrow C_1,C_2,C_3} \pi_{C_1,C_2,C_3} \sigma_{C_1=C_4} \bar{R}_3$.

Step 2: $\Re$ is the join of results in Step 1. That is, $(\delta_{u,v,f(u) \leftarrow A_1,A_2,A_3} \bar{R}_1) \bowtie (\delta_{v,w \leftarrow B_1,B_2} \pi_{B_1,B_2} \sigma_{B_3=\text{“}aaa\text{”}} \bar{R}_2) \bowtie (\delta_{w,x,y \leftarrow C_1,C_2,C_3} \pi_{C_1,C_2,C_3} \sigma_{C_1=C_4} \bar{R}_3)$.

Step 3: $S = \{v > y, x = \dot{0}\}$.

Step 4: $\Re$ is $\sigma_{v>y,x=\dot{0}}[(\delta_{u,v,f(u) \leftarrow A_1,A_2,A_3} \bar{R}_1) \bowtie (\delta_{v,w \leftarrow B_1,B_2} \pi_{B_1,B_2} \sigma_{B_3=\text{“}aaa\text{”}} \bar{R}_2) \bowtie (\delta_{w,x,y \leftarrow C_1,C_2,C_3} \pi_{C_1,C_2,C_3} \sigma_{C_1=C_4} \bar{R}_3)]$.

Step 5: The result $\Re$ is $\{\langle \lambda, \varpi \rangle\} \bowtie \pi_w\{\sigma_{v>y,x=\dot{0}}[(\delta_{u,v,f(u) \leftarrow A_1,A_2,A_3} \bar{R}_1) \bowtie (\delta_{v,w \leftarrow B_1,B_2} \pi_{B_1,B_2} \sigma_{B_3=\text{“}aaa\text{”}} \bar{R}_2) \bowtie (\delta_{w,x,y \leftarrow C_1,C_2,C_3} \pi_{C_1,C_2,C_3} \sigma_{C_1=C_4} \bar{R}_3)]\} \bowtie \{\langle \text{“}bbb\text{”} \rangle\}$.

4.4 Discussion

As mentioned earlier, cases in Step 5 of the proposed algorithm can be merged into one. It is for the purpose of illustration to divide this step into four cases. Case 4, without the condition $q > 0$ or $r > 0$, is actually general enough to cover all the cases.

Output of this algorithm can be stored as a view definition. Although the resulting relational algebraic expression may be very complicated, it can often be simplified by query optimization techniques.

Clauses which are used as queries might lead to a new problem. Up to now, the clause that input to the algorithm is thought of as a view definition, but it can also to be a query that inquires for information from the database. Because of these special tuples, which represent either general statements in base relations or clauses that fall into Case 4 of Step 5 in the algorithm, the result of this query may consist of existential value λ and universal value ϖ. Since the universal value is used for internal representation only, it should not be a part of the query output. Therefore, values of active domains should be collected to answer the query.

Corresponding to each attribute, there is an active domain. The procedure to compute the active domain can be defined as a view definition as in Section 3.3. The view definition will often be simplified by query optimization techniques together with the relational algebraic expression. An active domain is most likely associated with the same attribute of the same relation. If, for some special reason, the values of an active domain are distributed in several different relations, they must be unioned to form the active domain.

5 Conclusions

Aiming at supporting Horn-clauses as user interface, our approach extends the conventional relational model into a model that can deal with intensional facts by storing them in the EDB, via existential and universal values. In addition to the existential and universal values that are introduced to the relational model as attribute values for all attribute domains, a partial order is defined to identify and to remove redundancies. Since these two special values are different from other attribute values, some modifications of the relational operations are made to accommodate them based on the partial order defined. Although these intensional facts are stored in EDB, they are logically equivalent to those in the IDB of DDBs.

The relational model with the proposed extension is therefore able to deal with the following types of axioms, which are covered exactly by DDDBs.

- The single fact of *Type 1* can be represented by a constant tuple.

- The general statements of *Type 1* can be treated in the EDB via two special values, *existential* and *universal* values.
- The clause of *Type 2* can be handled with regard to its interpretation.
  (1) If it is used as an integrity constraint, then it is enrolled in the control part of a relational DBMS.
  (2) If it is purely a derivation rule, then it corresponds to a view definition in the relational model.
  (3) If it is a derivation rule that involves also general facts, then it is thought of as an extended view definition via existential and universal values.
- The clauses of *Type 3* as well as *Type 4* can be represented as integrity constraints in the relational DBMS.

In summary, this paper suggests that the relational model with the extension presented in this paper have the same expressive power and underlying mathematic logic as DDDBs. The algorithm presented can be implemented as a translator from Horn-clauses, nonprocedural derivation rules, to view definitions in the relational database. The extension of the relational model requires only minor modifications to its operations. The proposed approach not only offers a more natural and flexible interface to users but also takes advantage of the developed relational database techniques. This paper however does not suggest how to deal with indefinite facts, which are only handled by the IDDBs as proposed by Reiter [20]. The capability of incorporating indefinite facts is of high potential in expert systems and the like. It is anticipated that the logical extension of relational databases to indefinite facts will be much more complicated and will be studied in the future.

References

[1] C. L. Chang, "DEDUCE— a Deductive Query Language for Relational Data Bases," *Pattern Recognition and Artificial Intelligence*, Academic Press, Inc., New York, 1976, pp. 1C8-134.

[2] C. L. Chang, "DEDUCE 2: Further Investigations of Deduction in Relational Data Bases," *Logic and Data Bases*, Gallaire and Minker, Eds. Plenum Press, New York, 1978, pp. 201-236.

[3] E. F. Codd, "Extending the Database Relational Model to Capture More Meaning," *ACM Trans. Database Systems*, vol. 4, no. 4, December 1979, pp. 397-434.

[4] *An Introduction to Database System*, vol. 2, Addison-Wesley, Reading, Ma., 1983.

[5] C. J. Date, *Relational Database*, Addison-Wesley, 1986.

[6] H. Gallaire, J. Minker, and J. M. Nicolas, "An Overview and Introduction to Logic and Data Bases," *Logic and Data Bases*, Gallaire and Minker, Eds. Plenum Press, New York, 1978, pp. 3-30.

[7] H. Gallaire, J. Minker, and J. M. Nicolas, "Logic and Databases: a Deductive Approach," *ACM Computing Surveys*, vol. 16, no. 2, June 1984.

[8] J. Getta and H. Rybinski, "HOLMES: a Deduction Augmented Database Management System," *Inform. Systems*, vol. 9, no. 2, 1984, pp. 167-179.

[9] P. A. V. Hall, "Relational Algebras, Logic, and Functional Programming," Proc. ACM-SIGMOD Conference on Management of Data, Boston, Ma., June 1984, pp. 326-333.

[10] T. Imielinski, "Query Processing in Deductive Databases with Incomplete Information," Proc. ACM-SIGMOD Conference on Management of Data, Washington, D. C., May 1986, pp. 268-280.

[11] C. Kellogg, P. Klahr, and L. Travis, "Deductive Planning and Pathfinding for Relational Data Bases," *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 179-200.

[12] H. F. Korth and A. Silberschatz, *Database System Concepts*, McGraw-Hill, 1986.

[13] Gabriel M. Kuper and Moshe Y. Vardi, "On the Expressive Power of the Logical Data Model" (Preliminary Report), Proc. ACM-SIGMOD Conference on Management of Data, Austin, Tx., May 1985, pp. 180-187.

[14] E. L. Lozinski, "Evaluation Queries in Deductive Databases by Generating Subqueries," Proc. IJCAI, Los Angeles, Ca., August 1985, pp. 173-177.

[15] D. Maier, *Theory of Relational Databases*, Computer Science Press, Rockville, Md., 1983.

[16] J. Minker, "An Experimental Relational Database System Based on Logic," *Logic and Databases*, Plenum Press, New York, 1978, pp. 107-147.

[17] J. R. McSkimin and J. Minker, "The Use of a Semantic Network in a Deductive Question-answering System," Proc. IJCAI, Cambridge, Ma., 1977, pp. 50-58.

[18] J. M. Nicolas, K. Yazdanian, "Integrity Checking in Deductive Data Bases," *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 325-344.

[19] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA. 1980.

[20] R. Reiter, "Deductive Question-Answering on Relational Data Bases," *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 149-177.

[21] R. Reiter, "Equality and Domain Closure in First-Order Databases," *J. ACM*, vol. 27, no. 2, April 1980, pp. 235-249.

[22] J. C. Shepherdson, "Negation as Failure: A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption," *J. Logic Programming*, 1984:1, pp. 51-79.

[23] J. M. Smith and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. Database Systems*, vol. 2, no.2, June 1977, pp. 105-133.

[24] T. J. Teorey and J. P. Fry, *Design of Database Structures*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[25] J. D. Ullman, *Principles of Database Systems*, 2nd Ed., Computer Science Press, Rockville, Md., 1983.

[26] J. D. Ullman, "Implementation of Logical Query Languages for Databases," *ACM Trans. Database Systems*, Sep 1985, pp. 289-321.

[27] G. Vossen and V. Brosda, "A High-Level User Interface for Update and Retrieval in Relational Databases – Language Aspects," Proc. ACM-SIGMOD Conference on Management of Data, Austin, Tx., May 1985, pp. 343-353.