Query Processings by Relational Algebra extended with Unification

4M-9

Tadashi OHMORI and Hidehiko TANAKA
The University Of Tokyo, Div. of Eng.

1 Introduction

Conventional deductive databases concentrate on fast retrieval of many fact-clauses through a few rule-clauses. Practical applications, however, will need a mass of rule-clauses and fast retrieval mechanisms for them. For this purpose, we propose *Relational Algebra extended with Unification* (RAU); a variant of relational algebra for handling unification as in [1].

This paper shows a large database of rule clauses, RAU-operators and their commutative laws.

2 Large rule-database

Deductive databases in this paper allow functor symbols. A rule-clause is abbreviated as a *rule* and a fact-clause as a *fact*. In general, deductive databases have two databases; a database of facts (factDB) and the other one of rules (ruleDB). We use two meta-predicates `ruledb` and `demo` in [2] for managing a ruleDB. `ruledb(KB,Head,Body)` says "a knowledgebase `KB` knows a rule `Head:-Body` ". `demo(T,Goal)` says " a theory `T` proves `Goal` ". In this paper, we say that a head-predicate `p` of a given rule `p(..):-...` is the rule's *kind*, and the rule *belongs to* or *expresses* its kind.

In pracical applications, two cases enlarge a ruleDB; either there are many *kinds* of rules, or many rules belong to one *kind.*

The latter case expresses "many different implementations for a common interface". In an object oriented paradigm, it is the case that a superclass C requires a common interface p and allows each subclass of C to implement p independently. Then if there are 10^4 subclasses, 10^4 rules belong to a kind p.

Figure 1 is an example of the latter case. `store(Media, User, Data)` is a common interface of a class `Media`. It says " a `User` stores a `Data` in a `Media`". In Figure 1, subclasses and values of attributes in `Media`, `User`... are expressed by compound terms. e.g. `User` has a structure of *nation(group, city(idnumber)).* Rules expressing `store` are different from each other, depending on a type (i.e. subclass) of `Media` (e.g. visual terminal,...), properties of `User` (nation, group,..) , and a type of `Data` (image, text, format,..). If there are 10^2 types of `Media` and 10^2 properties of `User`, 10^4 rules belong to the kind `store`. In the same way, `key( Data, Keyword, User)` is a common interface of a class `Data`. It says "a `Data` is registered as a `Keyword` by a `User`". Implementations of rules expressing `key` depend on a type of `Data` and properties of a `User`.

```
% store(Media,User,Data):- Cond.
       ( store(type1(T), usa(P,cal(X)),
  10^4 {      image(noaa,A) ):- q1(T,P,X,A).
 rules { store(type2(sub2(T)), usa(ic,C),
       (  :   text(f1,A):- q2(T,C,A).
          :
% key(Data,Keyword,User):- Cond.
       ( key( text(S,A), K, usa(ic,X))
  10^4 {                       :- p1(S,A,K,X).
 rules { key( image(S,A), story(K),
       (  :   japan(pie,tokyo(X)) )
          :                    :- p2(S,A,K,X).

% query
q(X,Y):-
     ruledb(kb1,
         key(D, story(aaa), japan(P,X) ),
         Cond1),
     ruledb(kb2,
         store(M, usa(P,Y), D), Cond2 ),
     demo( to, (Cond1,Cond2)).
```

Figure 1: an example of rule database

Most of queries are issued via only those common interfaces regardless of different implementations. The query in Figure 1 is written in metapredicates. It retrieves applicable combinations of rules expressing "`store` and `key`" at first, and executes them.

Query Processings by Relational Algebra extended with Unification
Tadashi OHMORI and Hidehiko TANAKA
The University Of Tokyo, Div. of Eng.

R, T : meta-relations. A, B, C : attributeID.

R[A	B]
f(a,X)	p(X)

r(f(a,X)):- p(X)

T[A	C]
X	q(X)

t(X):- q(X)

- $R \overset{u}{\bowtie} T = \dfrac{[A \quad B \quad C]}{f(a,X) \quad p(X) \quad q(f(a,X))}$
- $\sigma_{A=g(X)}\, T = \dfrac{[A \quad C]}{g(X) \quad q(g(X))}$
- $I_B\, R = \dfrac{[A \quad B]}{f(a,b) \quad p(b)}$ where p(b) is true.
- $\rho_{[\,f(A,\,B),\,C\,]}\, R = \dfrac{[A \;\; B \;\; C]}{a \;\; X \;\; p(X)}$

Figure 2: examples of RAU-operators

3 RAU-operators

A DBMS for both a large ruleDB and a large factDB must execute the query in Figure 1 as fast as possible. The query clarifies two requirements. One is a fast retrieval mechanism for a large ruleDB in case that many rules belong to one kind. The other is to avoid random accesses to a ruleDB in a disk when executing rules; because a large ruleDB may be stored in a disk.

A limited solution of the latter is a partial compilation; transformation of each rule to simpler ones which operate a factDB directly [3]. By this method, much more rules belong to one kind. e.g. a rule `p:-q,r.` is compiled into 100 rules if `q` and `r` are compiled into 10 rules. Therefore a fast retrieval mechanism is fundamental for a large ruleDB.

Our approach is simple; At first, we compile in advance each rule into programs of a variant of relational algebra such as ERA in [4]. The variant must be able to deal with functor symbols. Then, in Figure 2, let R[A,B] (or T[A,C]) be a set of rules expressing a common kind `r(A):- B.` (or `t(A):- C.`). Set-operators are defined on these sets. We call this set of rules a *meta-relation* and these operators *Relational Algebra extended with Unification* (RAU). Their formal definitions are presented in [5,7]. In Figure 2, I_BR[A,B] is a set of facts satisfying each rule in R. R[A,B] $\overset{u}{\bowtie}$ T[A,C] is a set of rules expressing "`r` and `t`". RAU is also used for compiled exressions of rules. Queries are described as tree-forms of these operators, optimized, and executed by fast set-operation algorithms.

Because I-operator executes a set of modified relational algebra programs, it needs a global query optimization for common subexpression sharings such as in [6].

4 Commutative laws

Commutative laws of RAU-operators are forms of $exp1 =_w exp2$. These laws are used for optimizing query trees.

Definition Assume that meta-relations R, T and tuples t, s are given. Then,

- $R \subset_w T \overset{\text{def}}{=} \forall t \in R, \exists s \in T, \exists\theta$: substitution, $t = s\theta$.
- $R =_w T \overset{\text{def}}{=} R \subset_w T$ and $T \subset_w R$. □

Commutative laws hold as follows [7]; (M,N,R are meta-relations. $1, 2, \ldots$ are attributeID).

1. $\sigma_{p1\wedge p2}(M \times N) =_w \sigma_{p1\wedge p2}(\sigma_{p1} M \times \sigma_{p2} N)$, where $p1$ (or $p2$) is a selection-predicate including only attributes in M (or N).
2. $\sigma I M =_w I \sigma M$.
3. $\pi_1 I_3\, M[1,2,3] =_w \pi_1 I_3\, \pi_{1,3}\, M[1,2,3]$.
4. $I_{a\wedge b}(M \overset{u}{\bowtie} N) =_w I_b((I_a M) \overset{u}{\bowtie} N)$ $=_w I_a M \overset{u}{\bowtie} I_b N$, where a (or b) is an attribute of M (or N).
5. $\rho_{tl}\, \pi_1\, I_2\, M[1,2] =_w \pi_{vl}\, I_2\, \rho_{[tl,2]}\, M[1,2]$, where tl is a list of compound term. vl is a list of distinct variable symbols in tl.

[References] [1]. Morita, VLDB86, pp.52-59. [2]. Bowen, *amalgamating languages and meta language in logic programming*, Syracuse Univ, TR June,'81. [3]. Miyazaki, ICOT-TR183.'87. [4]. Zaniolo, VLDB85. pp.458-469. [5]. Ohmori, to be appeared in IWDM87. [6]. Sellis, SIGMOD86, pp.191-205. [7]. Ohmori, master thesis, Univ. of Tokyo. '87.