I.J. Engineering and Manufacturing, **2012,3, 53-62**
Published Online June 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2012.03.08

Available online at http://www.mecs-press.net/ijem

Translating SQL Into Relational Algebra Tree

Using Object-Oriented Thinking to Obtain Expression Of Relational Algebra

XU Silao, HONG Mei

Computer Science and Technology School of Computer (Software), Sichuan University Chengdu, China

Abstract

When we are translating SQL into relational algebra, we need a simple but flexible form to represent the data structure involved. As an interim result of the calculation, relational algebra tree combined with object-oriented model can gives us simple, intuitive notation allowing the query to be efficiently expressed and implemented at amazing ease.

Index Terms: SQL; automatic testing of DBMS; relational algebra tree; object-oriented

1. Introduction

With the development of database system, both academic and industry has been increasingly paid more attention to the quality of database system. As an interface for manipulating and managing the database, the validity of SQL plays a significant role in software automatic testing of database system. Translating the SQL into relational algebra expression can help us to analysis the SQL efficiently. However, the translation results are some abstract mathematic tokens and thus it is imperative to find out an effective way to transform and make use of them.

Aiming at translating SQL into relational algebra, Stefano Ceri and Georg Gottlob [1] has provided us the solution with the relational model. RQP (reverse query process), proposed by Carsten Binnig [2], receives a query sentence and an expected database schema, and then generates a database instance suitable for both of them. In RQP, SQL is translated into reverse relational algebra tree directly.

We have noticed that if we traverse the reverse relational algebra tree from its root to leaf nodes, we can obtain corresponding understanding of an expression of relational algebra. Without paying enough attention to the process of this translation and how the tree is built, it is obscure for us to utilize the advantages brought by

* Corresponding author.
E-mail address: writecoffee@gmail.com, hongmei@scu.edu.cn

relational algebra. Therefore, we hope that using the object-oriented thinking and UML technique can help solve this problem.

2. Solution

A. Parsing the SQL

Based on Stefano Ceri and Georg Gottlob's work [1], we assume that the following SQL grammar has been through the naming transformation and pre-processing stages. We have come up with 23 grammar items in Table I.

The query falls into two general categories: group-by query and non-group-by query. Derived from non-group-by query, we get the unary-query and binary-query and then the exists-query, the complex-query and the simple-query can be derived from the unary query. As a start of the object-oriented thinking and the use of UML technique, the class diagram representing the relationships among them is shown in Fig. 1.

Table I Summary Of SQL Restricted Grammar

1	query	→	gb_query | ngb_query
2	ngb_query	→	unary_query | binary_query | LPARAN unary_query RPARAN
3	unary_query	→	simple_query | exists_query | complex_query
4	simple_query	→	SELECT selector FROM relation_list [WHERE simple_predicate]
5	gb_query	→	unary_query GROUP BY gb_attr [HAVING hav_condition]
6	exists_query	→	SELECT selector FROM relation_list WHERE exists_predicate
7	complex_query	→	SELECT selector FROM relation_list WHERE left_term comp_op ngb_query
8	binary_query	→	ngb_query set_op ngb_query
9	relation_list	→	ID relation_list | COMMA relation_list | ε
10	gb_attr	→	attribute_spec_list
11	hav_condition	→	function_spec comp_op constant | function_spec comp_op ngb_query
12	selector	→	attribute_spec_list
13	attribute_spec_list	→	attribute_spec_list COMMA attribute_spec | attribute_spec
14	function_spec_list	→	function_spec_list COMMA function_spec | function_spec
15	simple_predicate	→	LPARAN simple_predicate boolean simple_predicate RPARAN | attribute_spec comp_op attribute_spec | attribute_spec comp_op constant
16	exists_predicate	→	EXISTS ngb_query
17	left_term	→	attribute_spec | constant
18	function_spec	→	ID LPARAN attribute_spec_list RPARAN
19	attribute_spec	→	ID DOT ID
20	Boolean	→	AND | OR
21	set_op	→	UNION | MINUS | INTERSECT
22	comp_op	→	EQ | NOTEQ | LT | LTEQ | GT | GTEQ
23	constant	→	NUM

upper-case items denote the tokens recognized by a SQL scanner

B. Constructing Relational Algebra Tree

During the period of syntax parsing, we have used YACC [6], a compiler compile, to parse the input SQL into syntax tree with restricted SQL grammar. However, it is hard to translate the tree into relational algebra and yet make further use of it. For one thing, the SQL comprises recursive queries and for another, the expression of the relational algebra is not visualized.

Thus, it is quite meaningful to translate the parse tree into a new data structure which has the property of being efficiently understandable. Basing on the fact that after imposing an operation on a relation or two relations, a new expression is generated, we can now represent the relational algebra with a binary tree. Of course, our translating method is based on the algorithm proposed by Stefano Ceri [1] and we merely summarize it with an object-oriented model. The detailed process will be discussed in the following sections.

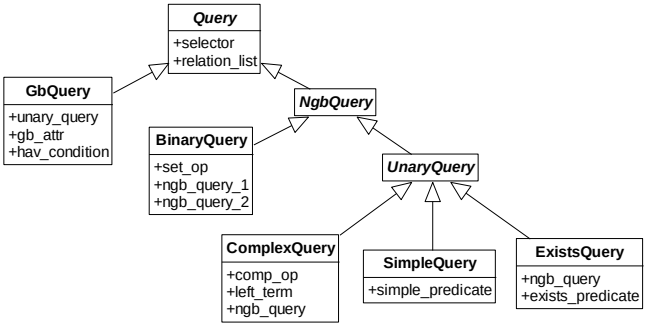

Fig 1 Hierarchy of SQL

3. Experiment

In order to examine the efficacy of the object-oriented thinking in solving the translating problem, we have to apply our solution to the implementation of each class of query.

At first, each kind of query will be modeled by UML technique with class diagram and then the relational algebra tree generated by the translator will be displayed.

C. Simple Query

Fig. 2 denotes the classes and their relationships in simple query. From the association between class FunctionSpec and class Function, we can see that the function field of FunctionSpec is nullable. When it is null, the instance denotes a group of attributes without any function applied, i.e., the attributes in projection item "PJ[S.A, S.B]". On the contrary, the attributes are aggregated by a specific function, i.e., "SUM(S.A, S.B)" in projection item "PJ[SUM(S.A, S.B), S.C]".

Especially, the fields in class SimplePredicate are of union type. According to the grammar, we recognize that the combination of its fields can be {simple_predicate, boolean, simple_predicate}, {attribute_spec, comp_op, attribute_spec} and {attribute_spec, comp_op, constant}. Therefore, there are two possible kinds of attribute for field left in class SimplePredicate, three possible kinds of attribute for field right and two possibilities for infix.

There are two scenarios [1] in the translation. The first one is that the simple_predicate item is empty and there is no "external" relation. Another one is that simple_predicate occurs, which involves further calculation of "external" relations in order to incorporate them in the Cartesian product.

In the first case, we input:

SELECT F(R.A), S.B, T.C FROM R, S, T

The SQL was translated into a relational algebra tree whose structure is shown in Fig. 3.

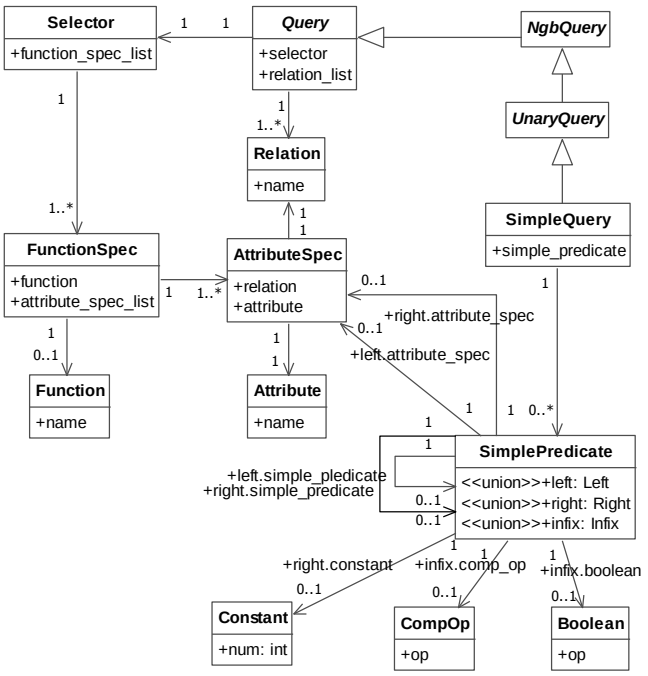

Fig 2 Class Diagram for Simple Query

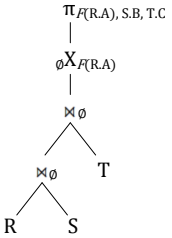

Fig 3 Relational Algebra Tree for Simple Query, Case 1.

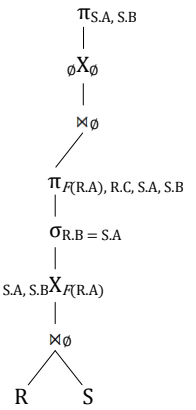

Fig 4 Relational Algebra Tree for Simple Query, Case 2.

Just like the Cartesian product, the θ-join is a binary operator. It connects two relations with specific predicate. As a matter of fact, until being optimized the θ-join node would never contain any predicate because it originally represents the Cartesian product between two expressions. After obtaining the Cartesian product of these three relations, the aggregation node and then the projection node are constructed upon this binary tree. The top-down sequence of these nodes is consistent with that of the SQL translation algorithm [1].

In the second case, we need to assume that some relations in this query have appeared in upper level. So, we can embed this simple query into another kind of query:

SELECT S.A, S.B FROM S WHERE EXISTS

SELECT R.C, F(R.A) FROM R WHERE R.B = S.A.

Fig.4 is its corresponding relational algebra tree. Attribute S.A and attribute S.B are the "external" attributes extracted from the upper level exists-query. They group the tuples of Cartesian product of S with Q by different values of the tuples of S and the results are manipulated by the aggregate function F.

D. Group-by Query

Fig. 5 is the class diagram of the group-by query. We use the class HavCondition to represent the predicate of group-by query. There are two kinds of combinations of its fields. They are {function_spec, comp_op, constant} and {function_spec, comp_op, ngb_query}. Because the first two fields of them are the same, we just need a union type to represent the third field of class HavCondition.

If the third field is a non-group-by query, it means that we have to deal with an unknown nesting query. Because the class NgbQuery is an abstract class, we can utilize the polymorphism of object-oriented language for solving the nesting query problem.

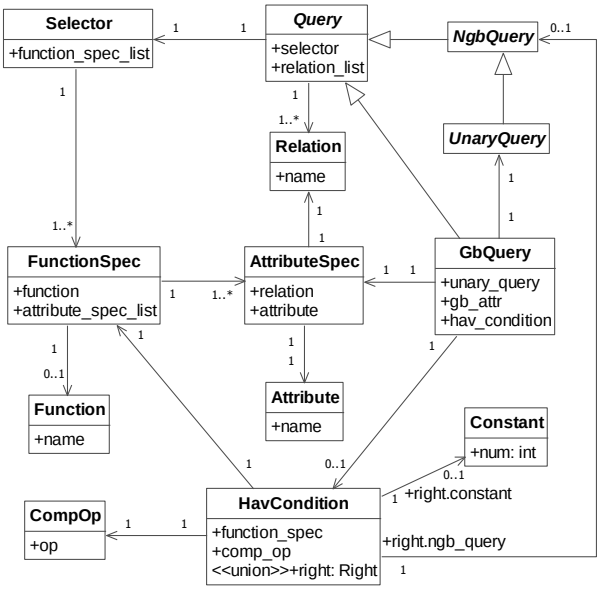

Fig 5 Class Diagram for Group-by Query

We should notice that four cases of group-by query should be distinguished. The first one is that the GROUP-BY clause has no effect. The second one is that there is no HAVING clause but the aggregate function should be evaluated. The third one and the fourth one are distinguished by the condition whether the HAVING clause has a nesting query or not. Except the first case, the unary query in group-by query should be changed into a form that its projection should incorporate all the attributes of its relations list order to correctly evaluate the functions.

The first case is simple and when we input the following query we get the relational algebra tree shown in Fig. 6.

SELECT R.A FROM R WHERE R.B > 7

AND R.C = 'Tom James' GROUP BY R.C

In the second case, projection items of the unary_query field in the group-by query should be rewritten by incorporating all the attributes of the relations_list and we have used a table to record the relation-attribute pairs occurred in the query while constructing the syntax tree. For instance:

SELECT F(R.A) FROM R
WHERE R.C = 7 GROUP BY R.B

is translated into a relational algebra tree shown in Fig. 7.

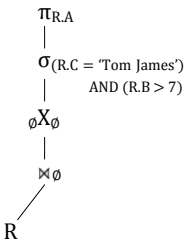

Fig 6 Relational Algebra Tree for Group-by Query, Case 1

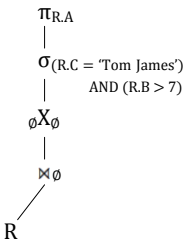

Fig 7 Relational Algebra Tree for Group-by Query, Case 2.

In the third case, we need to evaluate the aggregate function in the HAVING clause and incorporate them with that of the term unary_query. For instance,

SELECT F1(R.A) FROM R WHERE R.C = 7
GROUP BY R.B HAVING F2(R.C) > 2

is translated into a relational algebra tree shown in Fig. 8. Function F1 and F2 apply to the tuples grouped by attribute R.B.

In the fourth case, we need to evaluate the nesting query in the HAVING clause. We embedded a simple query into the group-by query as the following example:

SELECT F1(R.A) FROM R WHERE R.C = 7 GROUP BY
R.B HAVING F2(R.C) > SELECT S.C FROM S.

Two sub-queries are linked by a semi-join with a predicate, F2(R.C) > S.C, extracted from the HAVING clause. In addition, this semi-join can be transformed into a θ-join following with a projection on its left term.

E. Exists Query

The class diagram that describes the exists-query is shown in Fig. 10. The key task is to interpret the term ngb_query.

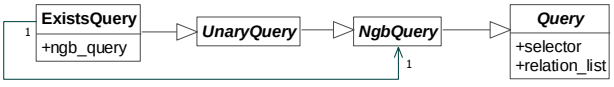

Fig 8 Class Diagram for Exists-Query

Exists-query should be discussed in two cases. The first case is that there is no connection between the field ngb_query and the field relation_list in the class ExistsQuery. Whether there is common relation or not is calculated by method connect [1] and the "external" relations are obtained by method other [1]. For instance,

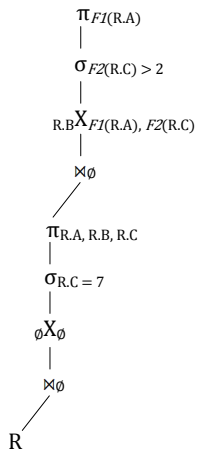

Fig 9Relational Algebra Tree for Group-by Query, Case 3

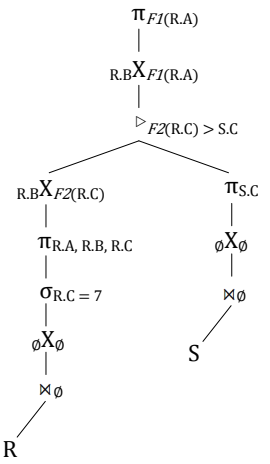

Fig 10 Relational Algebra Tree for Group-by Query, Case 4

SELECT R.A FROM R WHERE
EXISTS SELECT S.A FROM S WHERE S.B > 7

is translated into a relational algebra tree shown in Fig. 11. In order to keep the integrity of the relational algebra tree we retain the aggregation node which has no effect and this will be eliminated in the post-processing.

The second case is that these two fields are related. From the example below, the relation set calculated by method connect is {R} and the attribute set obtained from method other is empty.

SELECT R.A FROM R WHERE EXISTS
SELECT S.A FROM S WHERE S.B = R.A

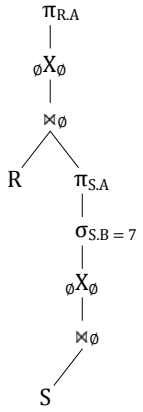

Fig 11 Relational Algebra Tree for Exists-Query, Case 1.

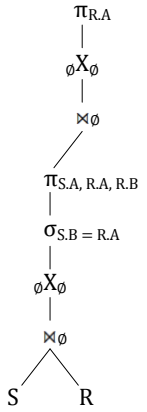

Fig 12 Relational Algebra Tree for Exists-Query, Case 2.

So, the term ngb_query has already dealt with all the relations involved in this query and there is no “external” relation. We can perceive this effect from Fig. 12.

F. Complex Query

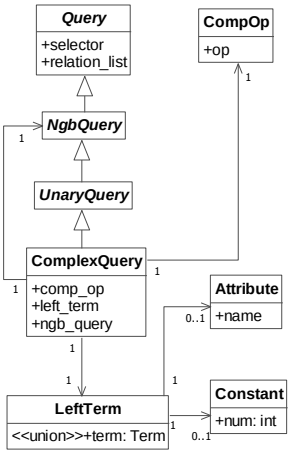

Fig 13Class Diagram for Complex-Query Relational

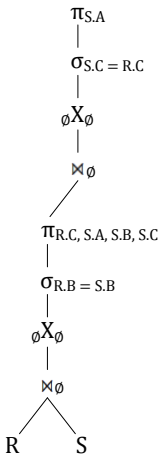

Fig 14 Algebra Tree for Complex Query

The class diagram is shown in Fig. 13. The complex-query contains a comparison between a left_term and a nesting non-group-by query. Being somewhat alike the exists-query, complex-query uses the connect [1] method to calculate the common relations and the other [1] method to obtain the "external" attributes list and then translate the comparison into a selection operation. We use the following example to reflect this effect:

SELECT S.A FROM S WHERE S.C =

SELECT R.C FROM R WHERE R.B = S.B.

Fig. 14 is the translation result and from this we can see that relation S is the connecting relation. The sub-query has involved all the relations in this query and the upper query just need to apply the selection "S.C = R.C" on that expression.

G. Binary Query

A binary-query should be translated into two sub-queries linked by a binary operator (INTERSECT, UNION, and DIFFERENCE) and its description class diagram is shown in Fig. 15. The binary-query translation requires the sub-query to be associated with "external" attributes calculated by method other [1] respectively in order to become useful for higher-level queries. For example,

SELECT R.A FROM R WHERE EXISTS

(SELECT S.B FROM S INTERSECT

SELECT T.B FROM T WHERE T.C = R.C)

is translated into a relational algebra tree shown in Fig. 16. The "external" attribute set of sub-query "SELECT S.B FROM S" is empty and the "external" attribute set of sub-query "SELECT T.B FROM T WHERE T.C = R.C" is {R.A, R.C}. From the "external" attributes sets, we notice that the first sub-query lacks of relation R which is required in order to perform the intersection with the second sub-query. Hence an additional Cartesian product of the first sub-query with R is required. Fig. 15 shows us the integrated construction.

We can also use minus operator to express an intersection since operation $A \cap B$ is equal to $A - (A - B)$. Difference operation is required to substitute the intersection operation in RQP algorithm [2].

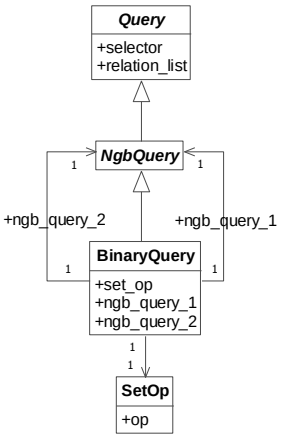

Fig 15Class Diagram for Binary-Query

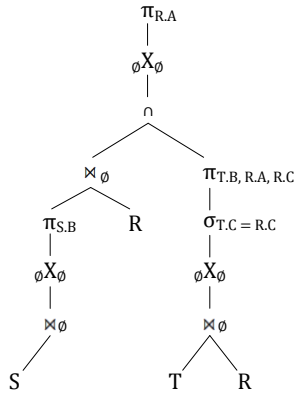

Fig 16 Relational Algebra Tree for Binary-Query.

H. Postprocessing

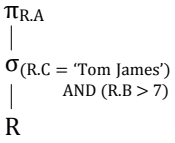

Fig 17Redundancy Eliminated Relational Algebra Tree of Fig. 6

```
PJ[F(R.A); S.B, T.C]
  FN[F(R.A); ∅]
    JN[∅]
      JN[∅]
        EXP[R]
        EXP[S]
      EXP[T]
```

Fig 18Output Of Relational Algebra Tree Shown In Fig. 3

Except for the post-processing in [1], here we need to eliminate the tree nodes which have no effect on the expression, such as aggregation node missing the aggregate attribute, θ-join node linking only one expression without predicate or selection node missing predicate. For example, the relational algebra tree in Fig. 6 can be optimized to the one shown in Fig. 17.

I. Output The Expression Of Relational algebra

Sometimes we hope that the translating result can be reused in different development platform and thus a convenient way is required. For this, we design an EBNF grammar for the output of the expression. The grammar is shown in Table II.

Table II Output Grammar Of Relational Algebra

tree	→	TAB tree | node ENTER NEWLINE tree | ENDFILE
node	→	RSDWORD LBRACKET content BRACKET
content	→	function_list SEMI attribute_list | predicate | ID
attribute_spec_list	→	attribute {COMMA attribute_spec_list} | NULLSET
function_spec_list	→	function {COMMA function_spec_list} | NULLSET
function	→	ID LPARAN attribute_spec_list RPARAN
attribute	→	ID DOT ID
predicate	→	LPARAN predicate bool_op predicate RPARAN | function cmp_op attribute | attribute cmp_op attribute | attribute cmp_op CONSTANT
bool_op	→	AND | OR
cmp_op	→	GT | GTEQ | LT | LTEQ | EQ | NOTEQ
boolean	→	AND | OR
set_op	→	UNION | MINUS | INTERSECT
comp_op	→	EQ | NOTEQ | LT | LTEQ | GT | GTEQ
constant	→	NUM | STRING

upper-case items denote the tokens recognized by a SQL scanner

Given the grammar, we can construct a top-down syntax parser easily and reconstruct the relational algebra tree in another platform. Fig. 18 is an example output for the relational algebra tree shown in Fig. 3.

4. Conclusion

Based on the analyses of each types of query, we can figure out that the object-oriented technique has made the representation of relational algebra intuitive. Class diagram becomes a vivid notation for elements of nodes of relational algebra tree and the grammar of SQL becomes more comprehensive for us. However, when extreme performance is required, object-oriented technique may not be feasible and traditional implementation would be preferred.

References

[1] Stefano Ceri, Georg Gottlob, “Translating SQL Into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries”, Software Engineering, IEEE Transactions, vol. SE-11, issue 4, pp. 324 – 345, April 1985

[2] Carsten Binnig, Donald Kossmann, Eric Lo, "Reverse Query Processing," icde, pp.506-515, 2007 IEEE 23rd International Conference on Data Engineering, 2007

[3] Agrawal, R., “Alpha: an extension of relational algebra to express a class of recursive queries”, Software Engineering, IEEE Transactions, vol. 14, issue 7, pp. 879 – 885, July 1988

[4] John R. Levine,Tony Mason,Doug Brown, “Lex & Yacc”, O’Reilly & Associates, 1992

[5] Thomas Connolly and Carolyn Begg, “Database Systemes: A Practical Approach to Design, Implementation, and Management”, 4th ed., Pearson Education, 2005

[6] S. C. Johnson, “YACC: Yet another compiler compiler”, Bell Lab., Murray Hill, NJ, Comput. Sci. Tech. Rep. 32, 1975

[7] Kenneth C. Louden, “Compiler Construction: Principles and Practice”, PWS Publishing Company, 1997