AN EXTENSION OF SQL TO CAPTURE MORE INFORMATION FROM OBJECTS WITH MANY-MANY RELATIONSHIP

Jorng-Tzong Horng*, Gwo-Dong Chen**, Baw-Jhiune Liu**[1]
*Department of Computer Science & Information Engineering
National Taiwan University, Taiwan , China
**Department of Electrical Engineering, National Central University
Taiwan, R.O.C., bjliu@ncuee.ncu.edu.tw

ABSTRACT

Many decision support applications such as task assignment, truck deliveries, airline screw scheduling problems usually need to get information from objects with many-many relationship. However, current relational operators including the complete set of relational algebra and other relational operators are difficult to get required information from objects with many-many relationship.

In this paper, we extend SQL so that users can capture more information from objects with many-many relationship by using the query language directly. The relational operators were extended with six operators, namely, match, maxmatch, cover, mincover, partition, and minpartition.

INTRODUCTION

Relational models and query languages, due to their flexibility, expressive power and simplicity, are playing an increasingly important role in the development of information systems. However, existing relational query languages are not adequate for some important application domains, like decision support systems. Part of the reasons is because many decision support applications usually need to get information from objects with many-many relationship.

Current relational operators are difficult to get required information from objects with many-many relationship. As in the complete set of relational algebra [3, 6], these operations are defined to manipulate the objects rather than manipulate the relationships between objects in a relation. These operations can not be used to access certain information from objects with many-many relationship. As to other operations, the DIVISION operation can be used to get information from objects with many-many relationship. However, it emphasizes the access of objects having relationship with all objects in another object type. For example, if we want to get suppliers who supply all parts required, we use the DIVISION operator.

[1]This work was partially supported by the National Science Council under grant NSC820408E008007.

However, many decision support applications such as task assignment, truck deliveries, airline screw scheduling problems usually need to get information from objects with many-many relationship. Two more typical examples are illustrated in the following paragraphs.

(1) Let us explore a relation that records baseball players, positions played and fielding averages for the positions played. In this example, if we assume each player can play some subset of the positions and each position can be played by some subset of the players. In addition, each player has fielding averages for the positions played. A team coach might want to know how to assign a team of nine players with maximum total fielding average for a game. In this team, each player plays only one position and each position is played by only one player.

(2) A manager might want to find a minimal cost project team that represents all the skills necessary for a certain job from a collection of employees each with a different set of skills and labor cost.

Currently, to obtain these rather natural results, we need to use query statements of existing relational database systems embedded in a programming language. Thus, the application programmers require extra effort to get required information.

SQL [3, 4, 5, 6, 22, 23] is the best known of relational query languages. It has been implemented by many manufacturers such as QINT/SQL, ORACLE, UNIFY, INGRES/SQL, SYBASE, etc. [22]. SQL has become the ANSI standard for relational DBMSs [5, 23]. It allows us to ask ad hoc queries, relieving decision makers or their assistants from the need to write programs in procedural languages. Application programmers or users need only specify what he wants without regarding how the results are retrieved. Thus, application development time is reduced greatly.

The expressive power of SQL has at least as powerful as that of relational algebra. But still like the relational algebra and other existing relational query languages, SQL is not able to deal with certain kind of queries which involve the manipulation of relationships between objects with many-

many relationship in a relation. Therefore, in order to enable users to capture more information from objects with many-many relationship by using a query language directly, we propose an extension to SQL.

The approach presented here deals with a special class of queries which get required information from objects with many-many relationship. Rather than defining a new language we propose an extension of the database language SQL for the processing of this type of queries. We propose six operators, namely, match, maxmatch, cover, mincover, partition, and minpartition. We incorporate these new operators into the "WHERE" clause of SQL statements. They are used as qualifications. Therefore, users with knowledge of SQL can capture more information from objects with many-many relationship by using the extended SQL directly.

To get ideas of how to get information from objects with many-many relationship, we model the structure of objects with many-many relationship as (1) *bipartite graph* and (2) *zero-one integer programs*. These problems are both theoretically and practically important in decision support systems.

The organization of this paper is as follows. Some related researches are reviewed in Section 2. In Section 3 we give the definition of the terminology including graph matching, set covering and partitioning problems that are key concepts of this paper. In Section 4 the semantics and syntax of the extended operators are described. In Section 5, we draw some conclusions.

RELATED RESEARCH

In this section we review previous research on SQL extensions.

To extend the benefits of the database approach to other areas, in recent years there has been an emphasis on extending the power and features of database query languages. Many researchers have designed extensions to existing database languages which are better suited for the new application areas. Examples of existing extended languages include QUEL* [21] and POSTQUEL [20] which are extensions of QUEL, STBE [15] which is an extension of QBE, SQL/NF [18] and SQL for NF^2 relations [11, 16, 17] which are extensions of SQL, the work [13] which extended query languages to support graph traversal problems, the work [8] which extended SQL to support general transitive closure, OSQL [9] which extended SQL to support object-oriented queries, the work [7] which implemented the SQL database language for support of decision support capabilities, and SQLMP [2] which extended SQL to represent and formulate linear mathematical models.

BACKGROUND

In this section, we review the definition of graph matching, set covering and partitioning problems. For a more detailed description see the work [1, 10, 12, 14, 19].

Definition 1. A graph G = (V, E) is called bipartite if there exist sets V_1 and V_2 such that

$$V_1 \cup V_2 = V,\ V_1 \cap V_2 = \varnothing$$

and every edge of G is incident to one vertex of V_1 and one vertex of V_2.

Definition 2. A matching M on a bipartite graph $G(V_1, V_2, E)$ is a set of edges of E(G) no two of which are adjacent. M is a maximum matching if G has no matching M' with |M'| > |M|.

Definition 3. Let a cost $c_j > 0$ be associated with every $j \in J$. The total cost of the cover J^* is $\sum_{j \in J^*} c_j$. Let A be an m x n binary matrix, w an n-dimensional nonnegative vector, and x an n-dimensional binary vector, with wx the inner product of the two, and $\underline{1}$ an m-dimensional column vector of ones. The *set covering problem* is to find a cover of minimum cost and can be written as the zero-one integer programming

$$\text{minimize (over x) } cx \qquad (1)$$

$$\text{subject to } Ax \geq \underline{1} \qquad (2)$$

$$x_j = 0, 1, \quad j = 1, \ldots, n \qquad (3)$$

where

$$x_j = \begin{cases} 1 & \text{if } j \text{ is in the cover} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if } i \in P_j \\ 0 & \text{otherwise} \end{cases}$$

Similarly, the *'set partitioning problem* is obtained by replacing (2) with

$$\text{subject to } Ax = \underline{1} \qquad (2a)$$

Any x satisfying (2) and (3) (or (2a) and (3)) is called a *cover (partition)* solution.

OPERATORS FOR GRAPH MATCHING, SET COVERING AND PARTITIONING PROBLEMS

In this section, we described formulation of graph matching, set covering and partitioning problems in SQL. We define special operators for the "WHERE" clause of SQL expressions to give a shorthand notation of such problems. Six constructs, namely, match, maxmatch, cover, mincover, partition, and minpartition were defined. The semantics and syntax of the extended operators are described.

1. The MATCH Operation

If we want to restrict objects with many-many relationship in a pair of attributes to one-one relationship, we can not formulate this kind of query by using SQL statements directly. For example, in a certain company, n workers $W_1, W_2, \ldots, W_n$ are available for n jobs $J_1, J_2, \ldots, J_n$, each worker being qualified for one or more of these jobs. Can all the men be assigned, one man per job, to jobs for which they are qualified? This is the *assignment problem*. For matter of convenience we introduce a special construct for such selection. The construct can be used in the "WHERE" clause of SQL statements. The syntax of the construct is defined as follows:

SELECT attribute-list
FROM relation-name
WHERE MATCH((X, Y)).

Where *X* and *Y* are two attributes of the given relation *relation-name*.

The match operation of relation R on attributes X and Y—

R WHERE MATCH((X, Y))

—is a relation with the same heading as R and with a body consisting of the set of all tuples t of R such that the maximum matching "t.X and t.Y" is satisfied. The operator effectively yields a horizontal subset of a given relation, that is, that subset of the tuples of the given relation for which a maximum matching is satisfied.

2. The MAXMATCH Operation

The match operator is an efficient way of determining a maximum matching, if one exists, between a pair of attributes. However one may, in addition, wish to take into account the effectiveness of the various relationships between a pair of attributes. As an example consider the assignment problem again. One may take into account the effectiveness of the workers in their various jobs (measured, perhaps, by the profit to the company). In this case, one is interested in an assignment that maximizes the total effectiveness of the workers. The problem can be solved by using the maxmatch operator in SQL. The syntax of the construct is defined as follows:

SELECT attribute-list
FROM relation-name
WHERE MAXMATCH((X, Y), Z).

In the given relation *relation-name*, attributes *X* and *Y* uniquely identify attribute *Z*.

The maxmatch operation of relation R on attributes X, Y, and Z—

R WHERE MAXMATCH((X, Y), Z)

is the set of all tuples t of R such that the matching "t.X and t.Y" is satisfied and the sum of the corresponding t.Z is maximum.

We will sometimes choose to minimize rather than maximize. This causes no difficulty, since we can first add negative sign to the values of Z and then find the maximum weighted matching.

3. The COVER and PARTITION Operations

We sometimes want to get specific objects of an attribute in a relation that relate to the objects of an attribute in another relation, we can formulate this query in SQL by means of a special construct. The syntax of the construct is defined as follows:

SELECT attribute-list
FROM R_1, R
WHERE COVER((R_1.X, R_1.Y_1), R_2.Y_2).

Where attributes Y_1 in R_1 and Y_2 in R_2 are defined on the same domain. Relation R_1 describes the relationship between the instances of attributes X and Y_1. The relationship between them is that each instance of X is related to a set of instances of Y_1 and each instance of Y_1 is also related to a set of instances of X. Each distinct instance of Y_1 in R_1 has a corresponding instance of Y_2 in R_2. The semantics of this construct is formally defined as finding a cover of a set.

The COVER operation of relation R_1 on attributes X and Y and relation R_2 on attribute Z—

R_1, R_2 WHERE COVER((R_1.X, R_1.Y), R_2.Z)

—is a relation with the heading (X) and a body consisting of the set of the smallest number of tuples (X:x) such that the set of tuples (X:x, Y:y) appears in R_1 for all tuples (Z:z) appearing in R_2.

The PARTITION operation is similar to the COVER operation. The PARTITION operation of relation R_1 on attributes X and Y and relation R_2 on attribute Z—

R_1, R_2 WHERE PARTITION((R_1.X, R_1.Y), R_2.Z)

—is a relation with the heading (X) and a body consisting of the set of the smallest number of tuples (X:x) such that the set of tuples (X:x, Y:y), where the set of all tuples (Y:y) are different, appears in R_1 for all tuples (Z:z) appearing in R_2.

4. The MINCOVER, SUMCOVER, MINPARTITION, and SUMPARTITION Operations

The cover operator is an efficient way of determining a minimum cover (the smallest number of sets) of a set, if one exists. However, one may, in addition, wish to take into account the cost of a cover. In this case, one is interested in a cover that minimizes the total cost of the cover. The MINCOVER operator is to find a cover of minimum cost. Finding an optimum solution of this operation is an NP-complete problem. It may take a very long time to find such solution. Hence, we provide an alternative operator for users. The alternative operator is to find feasible solutions instead of optimal solutions. The syntax of the construct is defined as follows:

SELECT attribute-list
FROM R_1, R_2, R_3
WHERE MINCOVER((R_1.X, R_1.Y), R_2.Y, R_3.Z),
or
SUMCOVER((R_1.X, R_1.Y), R_2.Y, R_3.Z) < const

Where *const* is a constant value. Every distinct instance of X in R_1 has a corresponding instance of Z in R_3 that is used as a cost of the instance. The semantics of the MINCOVER operation is formally defined as finding a cover of minimum cost, and that of the SUMCOVER operation is formally defined as finding a cover whose cost is less than the given constant-value.

CONCLUSIONS

Many decision support applications such as task assignment, truck deliveries, airline screw scheduling problems usually

need to get information from objects with many-many relationship. However, current relational operators including the complete set of relational algebra and other relational operators are difficult to get required information from objects with many-many relationship. In order to enable users to capture more information from objects with many-many relationship by using a query language directly, we propose an extension to SQL.

The relational operators are extended with six operators, namely, MATCH, MAXMATCH, COVER, MINCOVER (SUMCOVER), PARTITION, and MINPARTITION (SUMPARTITION). We incorporate these new operators into the "WHERE" clause of SQL statements. Therefore, users with knowledge of SQL can capture more information from objects with many-many relationship by using the extended SQL directly.

REFERENCES

[1] J. A. Bondy and U.S.R. Murty, "Graph Theory with Applications," The Macmillan Press, London, 1976.

[2] J. Choobineh, "SQLMP: A Data Sublanguage for Representation and Formulation of Linear Mathematical Models," ORSA Journal on Computing, Vol. 3, #4, 358-375, Fall 1991.

[3] E.F. Codd, The Relational Model for Database Management, Version 2. Reading, MA: Addison-Wesley, 1990.

[4] C.J. Date, A critique of the SQL database language. in Relational Database: Selected Writings by Date, C.J., Addison-Wesley, Reading, MA, 1986. This paper was originally published in ACM SIGMOD Record, 14, 3, 1984.

[5] C. J. Date, A Guide to the SQL Standard, Addison-Wesley, Reading, Mass., 1987.

[6] C. J. Date, "An Introduction to Database Systems," Volume I, Fifth Edition, Reading, MA: Addison-Wesley, 1990.

[7] R. Dattero, R. G. Ramirez, and J. Choobineh, "Derived Relations with Exceptions: Decision Support Capabilities," Journal of Management Information Systems, Vol. 6, pp. 83-101, Spring 1990.

[8] J. Eder, "Extending SQL with General Transitive Closure and Extreme Value Selections," IEEE Transactions on Knowledge and Data Engineering, 2(4), pp.381-390, December 1990.

[9] D. Fishman, et al., "Iris: An Object-Oriented Database Management System," ACM Transactions on Office Information Systems, 5(1), pp.48-69, January 1987.

[10] R. S. Garfinkel and G. L. Nemhauser, "Integer Programming," John Wiley & Sons, 1972.

[11] V. Linemann, "Non First Normal Form Relations and Recursive Queries: an SQL-Based Approach," in Proc. 3rd IEEE Int. Conf. Data Engineering, Los Angeles, pp 591-598, 1987.

[12] L. Lovasz and M.D. Plummer, "Matching Theory," Elsevier Science Publishers, 1986.

[13] M.V. Mannino and L.D. Shapiro, "Extensions to Query Languages for Graph Traversal Problems," IEEE Transactions on Knowledge and Data Engineering, 2(3), pp.353-363, December 1990.

[14] J.A. Mchugh, "Algorithmic Graph Theory," Prentice-Hall, 1990.

[15] G. Ozsoyoglu, Z. M. Ozsoyoglu and V. Matos, "Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions," ACM TODS, 12(4), 566-592, 1987.

[16] P. Pistor and F. Anderson, "Designing a Generalized NF2 Model with an SQL-Type Language Interface," In Proc. 12th Int. Conf. Very Large Databases, Kyoto, Japan, pp. 278-285, 1986.

[17] P. Pistor and Traunmueller, "A Database Language for Sets, Lists, and Tables," Inform. Systems 11(4), 323-336, 1986.

[18] M. A. Roth, H. F. Korth and D. S. Batory, "SQL/NF: A Query Language for 1NF Relational Databases," Inform. Systems, 12(1), 99-114, 1987.

[19] H.M. Salkin, and K. Mathur, Foundations of Integer Programming, Elsevier Science Publishing Co., Inc., 1989.

[20] M. Stonebraker and L. Rowe, "The Design of POSTGRES," in Proc. 1986 ACM-SIGMOD Int. Conf. Management of Data, Washington, DC, May 1986.

[21] M. Stonebraker and L. Rowe, and Hanson, E. Extending a database system with procedures. *ACM Transactions on Database Systems*, 12, 3 (1987), 350-376.

[22] R.F. van de Lans, "Introduction to SQL," Addison-Wesley, 1988.

[23] Database Language SQL, Document ANSI X3.135-1986. Also: ISO/TC97/SC21/WG3 N143.