DESIGN AND IMPLEMENTATION OF A RELATIONAL QUERY LANGUAGE WITH TEAMS[1]

Jorng-Tzong Horng*, Gwo-Dong Chen**, Baw-Jhiune Liu**

*Department of Computer Science & Information Engineering
National Taiwan University, Taipei, Taiwan, China
**Department of Electrical Engineering, National Central
University, Chungli, Taiwan, bjliu@ncuee.ncu.edu.tw

ABSTRACT

In this paper, we introduce the concept "team" and incorporate the team query into data language SQL. Existing relational query languages use tuple variables of tables to specify required data. Queries that are expressed by these languages are used to get all the data which satisfy the specified conditions. While a team variable represents a subset of a table, a team query is to retrieve all the teams that meet certain constraints.

We introduce team qualifications and briefly describe team query processing. The search space of a team query may be very large. Therefore, instead of using conventional breadth-first search method, we adopt a depth-first search method for query processing. Thus, users can always get results, if one exists.

INTRODUCTION

Relational database systems are being used in data processing community. Relational database systems (DBMSs) provide nonprocedural query languages such as SQL[7, 8, 20] and set-at-a-time operators. Thus, the application programmers or users need only specify what he wants without regarding how the results are retrieved. This greatly reduce application development time.

However, relational query languages can not be used to express *teams* that meet certain team constraints. Let us take a simple team example, a baseball team coach is looking for a team of nine players from the database of a baseball team to play a baseball game. The constraints of this team are (1) there are nine players in each team, (2) there should be exactly one player for each defense position. The result of this query may have many teams. This query is different from the queries like "get all the players who can play the pitcher position." Each player in the result of the query meets the constraint "the player who can play the pitcher position." An application programmer or user needs to write procedural programs to solve such team problems.

Obviously, a team query is different from a query of relational database query languages such as SQL. SQL finds a set of individuals and all individuals in the set satisfy the condition which is specified in the query. An element of a set meets all the constraints, while an element of a team may only meet some constraint(s) of the team constraints.

A SQL query and a team query have two different aspects. The first difference is the number of results. The result of a SQL query is a set of tuples, if one exists. However, the result of a team query is a set of teams. The second difference is in the team constraints. A SQL query does not specify the set of team constraints while a team query must specify the set of team constraints. Conditions that are used in SQL are not enough to express team criteria.

Teams are common and well understood in modern society. For example, in a company, a manager needs to organize teams to perform certain jobs; in an exercise, a coach has to assign a team of players to play games; a foreign needs to assign a team to negotiate with other countries; and so on. Teams exist everywhere in the real-world. However, so far we did not find any relational database languages that address the team query in the research literature. We extend SQL to be able to express and process teams that meet certain constraints. Hence, users with knowledge of SQL can easily take advantage of the team retrieval capabilities without learning a new language.

Team queries have two categories: (1) to select an optimal team, (2) to select all possible teams. The selection of a project team with minimal cost is of the first category. Queries of this category wish to find a team which has a maximum or minimum value of a given expression such as the sum of the salaries in the whole prospective teams. The selection of all possible teams of nine players from a baseball team is of the second category.

In some cases, finding teams with extreme values like minimum or maximum is NP-complete or NP-hard. It may take a very long time to find an optimum solution. The processing of team queries is quite different from that of the conventional database queries. Conventional database query processing uses a tuple-by-tuple evaluation strategy, while team query processing uses a set-by-set evaluation strategy. The search space of a team query is all subsets of the set of tuples of a relation. The search space of a team query may be large. Therefore, instead of using conventional breadth-first search method, we adopt a depth-first search method for query processing. Thus, users can always get results, if one exists.

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

1. Related Research

Sets and set manipulation occur naturally in many applications. Database query languages that attempt to incorporate sets and set operators have been defined in recent years by Abiteboul and Gramback [2], Kuper [17], Berri et al. [3, 4], Chen [6], and Gavish and Segev [10].

COL [2] is a logic-based language for complex objects. In addition to tuple and set constructs, the language also provides so called "data functions". LDL [3, 4] is an attempt to combine the benefits of logic programming with those of relational query languages. In LDL, sets and set operators are introduced into this language. In HiLog [6], sets can be represented naturally by parameterized predicates. The work [10] addresses the problem of optimizing queries that involve set operations (set queries) in a distributed relational database system. A set query is defined to be any query that can be represented as a sequence of relational operations followed by set operations between one set of tuples and a group of sets of tuples.

Although, the above languages support sets and set manipulation, however, they did not identify the concepts of team.

2. Overview of This Paper

This paper is organized as follows. In Section 2 the language is briefly described and a number of motivating examples are given. Section 3 defined basic and special team qualifications. In Section 4, we briefly describe query processing techniques of basic team qualifications. The last section is the conclusion.

TEAM-ORIENTED QUERIES

In this section we first briefly describe the syntax of team-oriented queries and then present a number of motivating examples. For a detailed description of the language, one is referred to the work [13].

1. Query Specification in Team-Oriented Query Language

Let $U = \{ R_1, R_2, \ldots, R_p \}$ be the universal set of relations, and $W = \{A_1, A_2, \ldots, A_n\}$ be the universal set of attributes. A team is a subset of the set of tuples over the specified attributes of a relation that satisfies a set of qualifications and a set of team qualifications. More formally, a team can be defined as a 4-tuple:

$$TEAM = (R, A', \Theta s, s\Theta s),$$

where $R \in U$, $A' \subseteq W$, Θs is a set of qualifications, and $s\Theta s$ is a set of team qualifications. The set of tuples over the set of attributes A' of relation R constructs the domain of a team.

A qualification Θs is made up of a number of clauses of the form:

Attr **rel-op** const-value

where Attr $\in$ W, rel-op is normally one of the operators $\{=, \geq, \leq, \neq, <, >\}$, and const-value is a constant value. Clauses can be arbitrarily connected by the Boolean operators AND, OR, and NOT to form a general qualification. A qualification here corresponds to a selection condition of the SELECT operation.

A team qualification $s\Theta s$ is any qualification that involves set operations between two sets. A team qualification may be of the form :

tuple-set($Attr_1$, $Attr_2$, ..., $Attr_n$) **set-op** tuple-set($Attr_1$, $Attr_2$, ..., $Attr_n$), or
tuple-set($Attr_1$, $Attr_2$, ..., $Attr_n$) **set-op** set-tuple-const-value, or
aggregate-fun(tuple-set($Attr_1$, $Attr_2$, ..., $Attr_n$)) rel-opconst-value

where $Attr_1$, $Attr_2$, ..., $Attr_n \in W$, $n \geq 1$, tuple-set($Attr_1$, $Attr_2$, ..., $Attr_n$) is a set of tuples over the specified attributes $Attr_1$, $Attr_2$, ..., $Attr_n$, set-op is normally one of the set operators $\{\equiv, ==, \subset, \subseteq, \supset, \supseteq\}$, set-tuple-const-value is a set of n-tuple constant values, and aggregate-fun is normally one of the aggregate functions {SUM, AVG, COUNT, MIN, MAX}. An n-tuple constant value is a tuple which is composed of n constant values. The semantics of the well-known set operations such as $\subset$, $\subseteq$, $\supset$, $\supseteq$, are the same in our definition. A bag is like a set except that its elements can be duplicate. In our definition, We particularly distinguish the difference between the definitions of the set equality ($\equiv$) and bag equality (==). Set A is equal to set B if they both have the same members. Bag A is equal to bag B if they both have the same members and the numbers of the same member in these two sets are equal.

2. Example Team Queries

We present two queries to illustrate various features of the language. In the following queries, underlined attributes are keys.

Query 1. The query uses a Baseball Database with two relations, which are given below.

PLAYERS(<u>player</u>, year-of-birth, year-joined, batting-avg, team-name)
PLAYER-POSITION(<u>player</u>, <u>position</u>, #-of-times)

The relation PLAYERS gives the number, year-of-birth, year-joined, batting average, team-name of each player. The relation PLAYER-POSITION gives each player that plays certain positions. The #-of-times is the number of times of a position that a player had been played. Find a team of nine players and their corresponding positions that satisfies the following conditions.

(a). They were born after 1960 and joined before 1990.
(b). The number of a position which was played by a player must be greater than 50 times.
(c). One player plays only one position and one position is played by only one player.
(d). The nine players are composed of 3 players, 2 players, 2 players, and 2 players. They belong to teams Elephant, Dragon, Lion, and Tiger respectively.
(e). The total batting average of the team is greater than 0.28.

The team query can be expressed as follows:

```
SELECT <TP.player, TP.position> IN TEAM
       <PP.player, PP.position> TEAM-PLAYERS TP
FROM   PLAYERS P, PLAYER-POSITION PP
WHERE  P.player = PP.player and P.year-of-birth >
       1960 and P.year-joined < 1990 and PP.#-of-
       times > 50
```

and (SELECT position FROM TP) == { pitcher, catcher, 1st, 2nd, 3rd, short, left, center, right}
and (SELECT TP.player, COUNT(TP.position) FROM TP GROUP BY TP.player) ≡ {(*, 1)}
and (SELECT P.team-name, COUNT(P.player) FROM P, TP WHERE P.player=TP.player GROUP BY P.team-name) == {(Elephant, 3), (Dragon, 2), (Lion, 2), (Tiger, 2)}
and AVG (SELECT P.batting-avg FROM P, TP WHERE P.player=TP.player) > 0.28

Query 2. The query uses a Orchestra Database with six relations which are given below.

INSTRUMENT(iname, first-player)
MUSICIAN(mname, sex)
SYMPHONY(sname, time)
ORCHESTRA(sname, iname, mname)
TOUR(city, sname)
PLAYS(mname, iname, #-of-times)

The relation INSTRUMENT gives the name of the first player of each instrument. MUSICIAN contains information about players. The relation SYMPHONY gives the duration in minutes of each symphony. The relation ORCHESTRA gives the set of players that execute each instrument in each symphony. TOURS contains the set of symphony names to be played in each city. The relation PLAYS gives each player that plays certain instruments. The #-of-times is the number of times of an instrument that a player had been played. Now we want to find a group of ten players to play symphonies. It satisfies the following conditions.

(a). The ten players consist of 3 players for violin, 1 player for conductor, 2 players for cello, 2 players for vola, and 2 players for bass drum.
(b). Each player in the group must have been acted the first player of some instrument.
(c). The ten players are composed of 6 male musicians and 4 female musicians.
(d). The #-of-times of an instrument which was played by a musician must be greater than 300 times.
(e). Each player must have been played the symphony "Tchaikovsky" in the city New York.

The team query can be expressed as follows:

SELECT <OT.mname, OT.iname> IN TEAM <PL.mname, PL.iname> ORCH-TEAM OT
FROM INSTRUMENT IN, MUSICIAN MU, ORCHESTRA ORCH, TOUR, PLAYS PL
WHERE PL.mname = ORCH.mname and TOUR.sname = ORCH.sname and PL.#-of-times > 300 and TOUR.city = "New York" and TOUR.sname = "Tchaikovsky" and PL.mname = IN.first-player
and (SELECT iname, COUNT(mname) FROM OT GROUP BY iname) == { (conductor, 1), (violin, 3), (cello, 2), (viola, 2), (bass drum, 2)}
and (SELECT mname, COUNT(iname) FROM OT GROUP BY mname) ≡ {(*, 1)}
and (SELECT MU.sex, COUNT(MU.mname) FROM MU, OT WHERE OT.mname = MU.mname GROUP BY MU.sex) == {(F, 4), (M, 6)}

BASIC TEAM QUALIFICATIONS

In this section, we turn our attention to a collection of team qualifications. The team qualifications are usually divided into two groups. One group includes team qualifications from mathematical theory, which are applicable because each operand is defined to be a set of tuples. These team qualifications include BTQ1, BTQ2, BTQ3, and BTQ4. The other group consists of special team operations which are specifically defined for query optimization. These operations include STO1 and STO2, STO3, STO4, STO5, and STO6. Before starting our definition, we use the following notations:

(a) $sf(\hat{t})$ = SELECT <attribute-list> FROM $\hat{t}$ {WHERE <search-condition>};
(b) $sqlf(\hat{t}, t_1, ..., t_n)$ = SELECT <attribute-list> FROM $\hat{t}, t_1, ..., t_n$ WHERE <search-condition>;
(c) $sgf(\hat{t})$ = SELECT <attribute-list> FROM $\hat{t}$ WHERE <search-condition> GROUP BY<attribute-name>{,<attribute-name>};
(d) bsqlf(t) = SELECT <attribute-list> FROM t { WHERE <search-condition> };
(e) $agf1(\hat{t}, t_1, ..., t_n)$ = Aggregate-fun($sqlf(\hat{t}, t_1, ..., t_n)$ or $\hat{t}$) IS MAXIMUM;
(f) $agf2(\hat{t}, t_1, ..., t_n)$ = Aggregate-fun($sqlf(\hat{t}, t_1, ..., t_n)$) IS MINIMUM;
(g) $agf3(\hat{t}, t_1, ..., t_n)$ = Aggregate-fun($sqlf(\hat{t}, t_1, ..., t_n)$) < constant-value
(h) constant-set = $\{c_1, ..., c_m\}$;
(i) tuple-constant-set = $\{(s_i, n_i), i=1, ..., p\}$.

Where $\hat{t}$ is a team name; t and t_i, i=1, ..., n, are general table names; s_i, i=1, ..., p are instances of some attribute of a relation; n_i, i=1, ..., p are positive integers. A <search-condition> is a boolean expression. A <attribute-list> is a list of attributes. Aggregate-fun is normally one of the aggregate functions {SUM AVG, COUNT, MIN, MAX}. A constant-value is a constant value. A constant-set is a set of constant values. A tuple-constant-set is a set of 2-tuple constant values.

Basic Team Qualifications

(BTQ1) $sf(\hat{t})$ ≡ constant-set or bsqlf(t)
(BTQ2) $sf(\hat{t})$ == constant-set or bsqlf(t)
(BTQ3) $sqlf(\hat{t}, t_1, ..., t_n)$ θ constant-set or bsqlf(t), θ ∈ {≡, ==, ⊂, ⊆, ⊃, ⊇}
(BTQ4) $sgf(\hat{t})$ θ tuple-constant-set , θ ∈ {≡, ≥}

Special Team Operations

(STO1) (constant-set or bsqlf(t)) **match** (constant-set or bsqlf(t))
(STO2) (constant-set or bsqlf(t)) **maxmatch$_A$** (constant-set or bsqlf(t))
(STO3) (bsqlf(t)) **mincover$_A$** (constant-set or bsqlf(t))
(STO4) (bsqlf(t)) **cover$_C$** (constant-set or bsqlf(t))
(STO5) (bsqlf(t)) **minpart$_A$** (constant-set or bsqlf(t))
(STO6) (bsqlf(t)) **part$_C$** (constant-set or bsqlf(t))

Aggregate Function

(AGF1) $agf1(\hat{t}, t_1, ..., t_n)$
(AGF2) $agf2(\hat{t}, t_1, ..., t_n)$
(AGF3) $agf3(\hat{t}, t_1, ..., t_n)$

For more detailed description about basic team qualifications and special team operations, one is referred to the contents of[13].

QUERY PROCESSING

In this section we describe query processing techniques of each team qualification.

Two important things must be considered in processing a team query. One is the choice of a team candidate, and the other is the evaluation of team qualifications. In this paper, we focus on the former, that is , how to reduce the search space. In fact, many team candidates can be avoided generating in query processing. For example, consider the following team query :

```
SELECT <TEAM-SP.s#, TEAM-SP.p#> IN TEAM
        <SP.s#, SP.p#> TEAM-SP
FROM SP
WHERE   (SELECT s# FROM TEAM-SP) ==  {s1, s2,
        s3, s4}
and     (SELECT p#, COUNT(s#)  FROM TEAM-SP
        GROUP BY p#) ≡ {(*, 1)}
```

If the first team qualification is evaluated to **True**, then the set of tuples over s# in TEAM-SP must contain one s1, one s2, one s3, and one s4. If the set of tuples over s# contain other values then the team qualification will be evaluated to **False**. Thus we only have to consider the team candidates that contain one s1, one s2, one s3, and one s4 and ignore other candidates when we choose team candidates. The technique of choosing team candidates can greatly reduce the search space. In this case, the first team qualification plays an important role in generating team candidates. A team query is specified by giving a set of team qualifications. We choose the first team qualification in a sequence of team qualifications as a team generator (TG), and other team qualifications as team qualifications (TQs). A team generator uses the information which is specified in the team qualification to generate team candidates. The team candidate is then evaluated by team qualifications. If the team candidate is evaluate to **True**, the team candidate becomes a team and is returned.

In Section 3, we introduce four basic team qualifications BTQ1, BTQ2, BTQ3, and BTQ4. Each can be used as a team generator. Different team generators use different algorithms to generate team candidates.

CONCLUSIONS

Team query processing uses a set-by-set evaluation strategy. The search space is all subsets of the set of tuples of a relation. The search space of a team query may be very large. Therefore, instead of using conventional breadth-first search method, we adopt a depth-first search method for query processing. Thus, users can always get results, if one exists, in a predicate restricted amount of time. In the selection of teams with extreme values, a team query perhaps can not find such teams in a small amount of time, however a team query can generate teams with near-optimal values by incorporating genetic algorithms or simulated annealing algorithms into query optimization algorithms.

Further research includes improving the implementation of the evaluation algorithm and the incorporation of further optimization techniques.

REFERENCES

[1] Balas, E. and Martin, C.H., "Pivot and Complement - A Heuristic for 0-1 Programming," Management Science 26(1), pp. 86-96, January 1980.

[2] Abiteboul, S. and Grumbach, S., "A Rule-Based Language with Functions and Sets," ACM Transaction on Database Systems, March 1991.

[3] Beeri, C., Naqvi, S., Ramakrishnan, R., Shmueli, O., and Tsur, S., "Sets and Negation in a Logic Programs," Proceedings of 6th Principle of Database Systems, 1987.

[4] Beeri, C., Naqvi, S., Ramakrishnan, R., Shmueli, O., and Tsur, S., "Set Constructors in a Logic Database Languages," MCC Technique Report.

[5] Bennett, K., Ferris, M.C., and Ioannidis, Y.E., "A Genetic Algorithm for Database Query Optimization," Proceedings of the Fifth International Conference on Genetic Algorithms, 1991.

[6] Chen, W.D., Kifer, M., and Warren, S., "Hllog: A foundation for Higher-Order Logic Programming," SUNY at Stony Brook Technical Report also in Proceedings of Logic Programming 1989.

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

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

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

[10] Gavish B. and Segev, A., "Set Query Optimization in Distributed Database Systems," ACM Transaction on Database Systems, Sept. 1986.

[11] Goldberg, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[12] Holland, J.H., Adaption in Natural and Artificial Systems, Ann Arbor: The University of Michigan Press, 1975.

[13] Horng, J.T., Chen, G.D., and Liu, B.J., "A Team-Oriented Query Language," Department of Computer Science & Information Engineering Technical Report, National Taiwan University, Taiwan, 1992.

[14] Ioannidis Y.E. and Wong, E., "Query Optimization by Simulated Annealing," in Proc. of the 1987 ACM-SIGMOD Conference on the Management of Data, San Francisco, CA, May 1987, pp. 9-22.

[15] Ioannidis, Y.E. and Kang, Y.C., "Randomized Algorithms for Optimizing large Join Queries," in Proc. of the 1987 ACM-SIGMOD Conference on the Management of Data, Allantic City, NJ, May 1990, pp. 312-321.

[16] Kirkpatrick, S., Gelatt, Jr. C.D., and Vecchi, M.P., "Optimization by Simulated Annealing," Science 220, 4598 (May 1983), pp. 671-680.

[17] Kuper, G., "Logic Programming with Sets," Proceedings of 6th Principle of Database Systems, 1987.

[18] Liepins, G.E., Hilliard, M.R., Richardson, J., and Palmer, M., "Genetic Algorithms Applications to Set Covering and Traveling Salesman Problems," in Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies, edited by D.E. Brown, C. White, III, Kluwer Academic Publishers, 1990.

[19] Lin, F.T., Kao, C.Y., and Hsu, C.C., "Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems," To Appear in IEEE Trans. on Systems, Man, Cybernetics, 23(3), May 1993.

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