OMEGA: A DATABASE MANAGEMENT SYSTEM FOR ACADEMIC USE

Colleen Deegan
IBM
Gaithersburg, MD

John Atkins and Mike Henry
Department of Statistics
and Computer Science
West Virginia University
Morgantown, WV 26506

Abstract

This paper describes an implementation of the relational database management system, OMEGA. OMEGA is designed to be "query language independent" in order to promote the investigation of experimental query languages by graduate students. OMEGA also supports the query language SQL and as such, is used in undergraduate database courses to demonstrate the features of SQL.

Introduction

One of the fastest growing application areas of computer science is databases. Industry and business more and more rely on databases to remain competitive. As a consequence of the demand for database management systems, software companies are heavily engaged in the design of such systems. These database management systems must meet the diverse needs of the business community and at the same time be accessible to the entire user community, i.e., be "user friendly." It is therefore incumbent upon colleges and universities to provide well-trained and knowledgeable computer scientists to both create new database management systems and to become database administrators. To meet this challenge, college and university computer science curricula must include several courses in database design and application.

Undergraduate database courses should have available sophisticated database management systems which can be used to demonstrate to the student typical applications of databases. Graduate database courses should have systems which permit the graduate student to experiment with new query languages and novel user interfaces.

This paper describes a relational database management system, OMEGA, which meets both of these needs and is available to the academic community at no cost. OMEGA supports the query language SQL with all of its powerful facilities, and at the same time is essentially "query language independent" in that it actually executes very low level commands thereby allowing experimentation with new query languages which can easily interface with it. SQL was chosen as the supported query language because of its wide acceptance by the user community. SQL is not only the query language supported by SYSTEM R, but is also the query language supported by IBM's DB2, ORACLE, and other commercial systems. Indeed, Date [7] states that "There can be little doubt that the importance of [SQL] will increase significantly over the next few years." In view of the increasing importance of SQL, students in undergraduate database courses must have an exposure to this query language.

Overview of OMEGA

OMEGA is a relational database management system that supports very powerful retrieval facilities and incorporates several optimization features. It was designed to be "query language independent" in order to provide a mechanism for advanced students to design experimental query languages without the need for the intricate file access and manipulation code that constitutes the file manager component of a database management system.

OMEGA executes relatively simple "atoms" or quadruples which are often interrelated. The format of the atoms has the general form:

(code; old_file; new_file; condition)

The atoms are placed in an array by the query processor and the array is then passed to OMEGA when OMEGA is invoked. The array of atoms constitutes the only communication between the query processor and OMEGA. Thus, the processor is essentially a parser which parses the queries and translates a query into a sequence of atoms.

OMEGA Operations

The heart of any database management system is the ability to select tuples from a relation and test such tuples against a Boolean expression. OMEGA's selection facility uses a combination of five atoms. Since the selection process often mimics a loop, two atoms that serve the looping process are the label atom (code 13) and the branch atom (code 12). A tuple is selected for consideration with the atom whose code is 07. This atom has the format

(07; relation [(variable_name)];; tuple_name)

The variable name is optional and corresponds to the tuple variable in SQL or the RANGE OF statement in QUEL. The tuple name is the name by which the tuple is referenced in the text atom.

The tuple is tested via an atom with code 11 having the format:

(11; tuple_name; new_relation; Boolean expression)

The selected tuple, which is identified by the tuple name, is tested against the Boolean expression and if the Boolean result is true, is placed in the new_relation.

The fifth atom used in the selection process is a branch on end_of_file (code 08) and follows a select (code 07) atom.

We shall illustrate the interplay of the atoms using the well-known suppliers/parts database which appears in [5]. The schema for this database is:

S(S#, SNAME, STATUS, CITY)

P(P#, PNAME, COLOR, WEIGHT, CITY)

SP(S#, P#, QTY)

A typical query, in SQL, from the suppliers/parts database might be:

SELECT SNAME

FROM S

WHERE STATUS < 20 OR CITY = 'LONDON'

The atoms that would be executed as a result of this command would be:

```
(13;01;;)       /* label 01*/
(07;S;;*A01)  /* select a tuple from S; name it *A01 */
(08;02;;)       /* branch on EOF to label o2 */
(11;*A01;*T01; STATUS, 20,     , CITY, 'LONDON' , = , OR)
          /* test tuple *A01 and place it in relation *T01 */
(12;01;;)  /* branch to label 01 */
(13;02;;)   /* branch to label 02 */
(.....)  /* project and print atoms */
```

Note that the comments are provided for clarity and are not actually generated.

The selection atoms are usually followed by a project atom (code 17) and a print atom (code 16).

In the previous example, the remaining sequence of atoms would be:

```
(17;*T01;*T02;SNAME) /* project SNAME from *T01 and
                        place the result in *T02 */
(16;*T02;;)          /* print *T02 */
```

A more complex example will illustrate OMEGA's versatility. A relational database management system must support a form of Cartesian product and should provide for the notion of a "tuple variable." OMEGA does both as demonstrated in the following example:

SELECT SNAME, P#

FROM S, SP SPX

WHERE P# IS NOT IN

(SELECT P# FROM SP WHERE S# < > SPX.S#)

AND S.S# = SP.S#

The atoms that are executed as a result of this SQL command are:

```
    (06;S,SP(SPX);*T01;)
    /*Cartesian product of S and SP; SPX references SP; result
is relation *T01 */
    (13;01;;) /* label 01 */
    (07;*T01;;*A01)
    /*select tuple from *T01 and name it *A01 */
    (08;02;;)  /* on EOF, branch to label 02 *
    /* atoms generated by inner select */
    (13;03;;) /* label 03 */
    (07;SP;;*A02) /* select tuple from SP and name it *A02 */
    (08;04;;) /* on EOF, branch to label 04 */
    (11;*A02;*V02;S#,SPX.S#,  ) /*test tuple *A02 and
                  place in relation *V02 */
    (12;03;;) /*branch to label 03 */
    (13;04;;) /*label 04 */
    (17;*V02;*V03;P#) /* project P# from *V02 and place in
                      *V03 */
              /*end of atoms from inner block; result of inner
SELECT is relation *V03 */
    (11;*A01;*V04;P#,*V03,IS_NOT_IN,S.S#,SP.S#,=,AND)
          /* test tuple *A01 and place in *V04 */
    (12;01;;)  /* branch to label 01 */
    (13;02;;) /*label 02 */
    (17,*V04;*V05;SNAME:P#) /* project SNAME and P# from
                V04 and create *V05 */
    (16;*V05;;) /* print *V05 */
```

Most sophisticated query languages have a "group by" facility whereby tuples are grouped together by common values on one or more fields and then groups are selected and/or a built-in function is applied to teach group. SQL, QUEL, and QBE all provide for this "group by" option. OMEGA supports a group by option and provides the standard six built-in functions: SUM, MAX, MIN, AVG, COUNT, and SET. The first five functions are self-explanatory, SET is applied only to groups. The SET function takes the values from one or more attributes in a group and makes these values a relation so that the set of values from that group may be compared to another relation.

As an example, consider the following SQL query:

```
SELECT P#, AVG(QTY)

FROM SP

GROUP BY P#

HAVING SET(S#) =

(SELECT S# FROM SP)
```

The atoms generated as a result of this query are:

```
(14;SP;*G01;P#)
  /* group tuples in SP by P# and place in relation *G01 */
(17;SP;*T02;S#) /*project S# from SP and place in *T02 */
(15;*G01;*G03,SET(S#),*T02,=) /* select groups from *G01
        for which the set of S#'s equals the relation *T02 */
(17;*G03;*T04;P#:AVG(QTY) /*project the P# value from *G03
        and compute the average QTY value from each group */
(16;*T04;;) /*print *T04 */
```

OMEGA provides for all of the file maintenance operations that are necessary in a database management system. These operations include the ability to create a file, introduce new tuples to the file, modify tuples in the file, and to delete tuples from the file. These operations are standard in any database management system and vary little from system to system.

Optimization

The files in OMEGA are each direct access files (hash files). Although tuples are stored in the files in a sequential fashion, the direct access file format was chosen to facilitate optimization. The design of OMEGA requires the creation of a substantial number of intermediate, temporary files (files with the name *Tnn, *Vnn, *Gnn). The overhead resulting from the creation of numerous temporary files can seriously degrade the performance of the system. For this reason, the temporary files are in fact logical files whenever possible. This means that if a new file is to be created from an existing relation and it has the same schema as the existing relation, then the new file will be a list of pointers into the existing file. This optimization technique is exploited in three operations: (1) selecting tuples from one relation and placing them in a second relation (code 11), (2) grouping tuples based on an attribute (code 14), and (3) selecting groups that satisfy some Boolean condition (code 15).

In a high-level query language, many queries involve nested select commands. The results of the nested selects are used to determine if a tuple from an outer select block meets a condition and is to be included in a new file. In some cases, the relation resulting from an inner select block does not depend on the tuple being tested in the outer block and therefore, need not be re-computed with every iteration of the outer block. OMEGA recognizes those situations where relations that result from inner blocks need only be computed once.

As an example, consider the following SQL query:

```
SELECT S#

FROM SP SPX

WHERE

   (SELECT P#

   FROM SP

   WHERE S# = SPX.S#)

     CONTAINS

   (SELECT P#

   FROM SP

   WHERE S# = 'S3')
```

In this example, the relation resulting from the first SELECT clause (in the outer WHERE clause) must be computed for each tuple being tested from the outer SELECT. However, the second SELECT clause is independent of the outer SELECT block and therefore will be computed once and used by OMEGA in each iteration from the outer SELECT.

Conclusion

OMEGA has provided an opportunity for computer science students to gain a hands-on experience with an actual database management system. Students indicate that their learning experience and their interest in the course was very much increased by their ability to access an actual database management system.

Graduate students are using OMEGA as a starting point in several graduate projects. These projects are investigating new query languages and new ideas in user interfaces. For example, graduate students are involved with writing pre-processors to make OMEGA accessible from a high level language. Others are exploiting the direct access file structure of OMEGA to further optimize record retrieval and a third group is using OMEGA to explore artificial intelligence applications to produce "user-friendly" interfaces.

References

1. M. M. Astrahan, et.al., "System R: Relational Approach to Database Management," ACM Transactions on Database Systems ,1, No. 2 (June 1976).

2. M. M. Astrahan, et.al., "System R, A Relational Database Management System," IEEE Computer Society: Computer 12, No. 5 (May 1979).

3. D. D. Chamberlain and R. F. Boyce, "SEQUEL: A Structured English Query Language," Proc. 1974 ACM SIGMOD Workshops on Data Description, Access and Control.

4. D. D. Chamberlain, et.al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," IBM J. R&D, 20, No. 6 (November 1976).

5. C. J. Date, An Introduction to Database Systems, 4th Ed., Vol. 1, Addison Wesley, 1985.

6. C. J. Date, "Some Principles of Good Language Design," SIGMOD Record, 14 (3), November 1984.

7. C. J. Date, "A Critique of the SQL Database Language," SIGMOD Record, 14 (3), November 1984.

8. Jeffrey D. Ullman, Principles of Database Systems, 2nd Ed., Comp. Sci. Press, 1982.