Technical Correspondence

On "Human Factors Comparison of a Procedural and a Nonprocedural Query Language"

Chamberlin's letter

□ This letter is in response to the paper by C. Welty and D. W. Stemple, "Human Factors Comparison of a Procedural and a Nonprocedural Query Language," *ACM Transactions on Database Systems* 6, 4 (Dec. 1981), 626-649. The authors suggest that the human factors of the SQL language might be improved by introducing a JOINED BY clause which specifies how two tables are to be joined in a query, rather than by including the join criterion in the WHERE-clause of the query.

The SQL syntax for joins was influenced by the following observations:

(1) Some queries require a join of several tables.
(2) Join conditions often involve more than one column, for example:

```
TABLE1.COL1 = TABLE2.COL4
OR TABLE1.COL3 = TABLE2.COL5
AND TABLE1.COL2 > TABLE2.COL7
```

(3) Join conditions sometimes occur mixed with other types of predicates in Boolean expressions.

The first two observations can be accommodated fairly easily in a JOINED BY clause (although Welty and Stemple do not make a specific suggestion for handling these cases). But the third observation poses some basic problems for the approach suggested by Welty and Stemple, as illustrated by the following example.

A CANDIDATE table records a primary and secondary skill for each job candidate, and a rating for each:

CANDIDATE

NAME	SKILL1	RATING1	SKILL2	RATING2
Smith	Teacher	4	Writer	6

A VACANCY table records, for each vacant position in a company, its job number and required skill:

VACANCY

JOBNO	REQSKILL
501	Writer

A user wishes to state the following query: "For each vacancy, print the job number matched with the candidates who have the required skill with a rating of at least 5." In SQL, this query would be stated as follows:

```
SELECT  JOBNO, NAME
FROM    VACANCY, CANDIDATE
WHERE   REQSKILL = SKILL1 AND
          RATING1 > 5
OR      REQSKILL = SKILL2 AND
          RATING2 > 5;
```

This query cannot be stated with a JOINED BY clause unless *all* the predicates are placed in the JOINED BY clause, in which case the distinction between the JOINED BY clause and the WHERE clause becomes unclear.

In summary, I do not believe that a clear distinction exists between the join condition of a query and its other predicates, or that the SQL language would be improved by attempts to draw such a distinction.

Donald D. Chamberlin
IBM Corporation
5600 Cottle Road
San Jose, CA 95193

Welty and Stemple's reply

□ Our suggestion of using JOIN in SQL was motivated by the superior performance of subjects using the TABLET language on join problems. TABLET does use an explicit JOIN. If the difference in subject performance was only due to the existence of JOIN in TABLET and its absence in SQL, this change would help. We tend to agree with Dr. Chamberlin that adding JOIN to SQL would not help, believing instead that the difference is actually due to the underlying procedural nature of TABLET, and minor syntactic changes will probably not result in improved user performance in SQL. The possibility of a simple syntactic fix to SQL was an obvious result of the experiment and, thus, was one of the recommendations. This recommendation would have to be tested by further experimentation to see if user performance is improved. We do have doubts that it will improve performance because both SQL and TABLET used GROUP BY (identical syntax), but TABLET outperformed SQL on these problems, too.

We disagree with Dr. Chamberlin's contention that a clear distinction cannot be drawn between a join condition and other predicates. To demonstrate this using his example, we must consider the relational calculus query underlying the SQL version. The WHERE clause of a SQL query corresponds almost directly to the predicate of a relational calculus expression. SQL itself has some aspects akin to the relational calculus, others more akin to the relational algebra. JOIN is an operator in the relational algebra. In adding the JOIN clause to SQL we are transforming it to a more algebraic form. This process is analogous to that given by Codd [1]. In this algorithm the join terms of the predicate become join expressions using the join operator. In addition, $\vee$ (or) becomes $\cup$ (union) and $\wedge$ (and) becomes $\cap$ (intersection). This is a simplification of the process, but suffices for Dr. Chamberlin's example.

Applying this transformation to the SQL query in Dr. Chamberlin's letter we obtain:

```
    SELECT JOBNO, NAME
    FROM VACANCY, CANDIDATE
    JOINED BY REQSKILL=SKILL1
    WHERE RATING1 > 5
UNION
    SELECT JOBNO, NAME
    FROM VACANCY, CANDIDATE
    JOINED BY REQSKILL=SKILL2
    WHERE RATING2 > 5
```

Use of AND between the join terms would result in an INTERSECT operator being used. More complex queries naturally receive more complex transformations. The above transformation was simplified by the fact that JOBNO and NAME are the key attributes of the joined relations, as well as being the attributes being SELECTed.

Again, we do not believe that such a query will necessarily be easier to write than the current SQL form. But we do believe that a syntactic distinction which mimics the distinction between selecting more information from one table and combining information from more than one table should be present in a well-designed query language. Both

Lochovsky's experiment [2] and ours provide a basis for this.

CHARLES WELTY
Math and Computer Science Department
University of Southern Maine
96 Falmouth Street
Portland, ME 04103

DAVID W. STEMPLE
Computer and Information
Science Department
University of Massachusetts
Amherst, MA 01003

REFERENCES

1. CODD, E.F. Relational completeness of data base sublanguages. *Courant Computer Science Symposia*, vol. 6: *Data Base Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1971.
2. LOCHOVSKY, F.H. Data base management system user performance. Tech. Rep. CSRG-90, Computer Systems Research Group, Univ. Toronto, Toronto, Ont., Canada, April 1978.