Human Factors Comparison of a Procedural and a Nonprocedural Query Language.

Charles Welty
David W. Stemple

COINS Tech. Report 79-16

August 1979

Computer and Information Science Department
University of Massachusetts, Amherst 01003

This research was funded, in part, by NSF grant MCS78-07616.

ABSTRACT*

Experiments testing the ability of subjects to write queries in two different query languages have been run. The two languages, SQL and TABLET differ only in their procedurality -- both languages use the relational data model and their Halstead levels are similar. Constructs in the languages that do not affect their procedurality are identical. Subjects were taught using manuals for the two languages that contained identical examples and problems in identical order. The results of the experiments show that subjects write difficult queries significantly better using the procedural language than subjects using the nonprocedural language. The results of the experiments are also used to compare corresponding constructs in the two languages and to recommend improvements for these constructs.

Key Words and Phrases: human factors, database systems, query languages, procedural and nonprocedural languages.

CR Categories: 4.33, 4.6, 3.72.

---

* This research was funded, in part, by NSF grant MCS78-07616. The work was done in the Computer and Information Science Dept., Univ. of Massachusetts, Amherst.

1. INTRODUCTION

More and more attention is being focused on the prime element in computer systems, namely the human element. Although in terms of cost, speed, reliability and most other "efficiency" measures, the machine is far superior to the human, humans remain a crucial part of the system. Any efficiency in the use of system resources is wasted if a system is not designed to match the needs and abilities of its users. This fact has led to the exploration of new research areas involving the human-oriented aspects of computer systems.

In the field of computer languages, human factors testing can be used to:

1. Test if a language is learnable. Failure of this test may dictate a language's demise.
2. Eliminate minor difficulties in a language. This is used in the planned evolution of the language.

Much human factors testing has been done on aspects of general purpose programming languages. The study of languages and language constructs [Gannon and Horning, 1975; Sime et al, 1973; Shneiderman, 1976a], errors in programming [Gould, 1975; Gould and Boies, 1974; Gould and Drongowski, 1974; Youngs, 1974] and supposed aids to programming [Shneiderman et al, 1975; Love, 1977; Shneiderman, 1977; Weissman, 1974] have resulted in useful insights into the programming process. In addition, directions and methods for human factors testing of general purpose programming languages are presented in [Shneiderman, 1976; Weissman, 1974; Weinberg and Schulman, 1974].

Some experimentation relating to general purpose programming languages has been done using non-programmers [Miller, 1974; Miller and Becker, 1974]. These experiments bring out the techniques used by non-programmers to solve computer oriented problems. It is hoped that understanding these techniques will aid in understanding the problems inherent in programming. Human factors studies have led to a rudimentary understanding of programmer behavior [Shneiderman and Mayer, 1975].

Special purpose languages have also been studied [Miller and Thomas, 1977; Seymour, 1978].

One type of special purpose language is the database query language -- used to retrieve information from a database. Query languages are designed for use by non-programmers. The querying of a database is only an occasional part of the intended user's job. Since the user lacks computer experience and uses the language only intermittently, a successful query language should be easy to learn, use and remember. Human factors testing of a query language can determine whether it meets these criteria.

There have been several human factors studies of query languages. Gould and Ascher [Gould and Ascher, 1975] experimented with IQF [IBM, 1972]. Their experiment showed the language to be difficult to learn. Zloof's Query by Example [Zloof, 1975] was studied by Thomas and Gould [Thomas and Gould, 1975] and found easy to learn and use. Reisner, et al [Reisner, Boyce and Chamberlin, 1975; Reisner, 1976] made a study of SEQUEL [Chamberlin and Boyce, 1974] and SQUARE [Boyce, Chamberlin, King and Hammer, 1975] with the result that SEQUEL

was found relatively easy to learn. This experiment helped in the evolution of SEQUEL 2 [Chamberlin, et al, 1970]. Lochovsky [Lochovsky, 1978] studied various data models and their associated data manipulation languages.

Most of the query language testing was done without actual systems. The languages were presented in classroom situations and testing was through classroom exams. The reason for this was twofold:

1. The purpose of the testing was to study the language before implementation.
2. Use of an actual system would have introduced the additional factors to terminal availability, system response time, subjects' typing expertise, etc.

Human factors research is readily applicable to query languages for several practical reasons:

1. There are no entrenched query languages which would be either hard or impossible to replace with a language of proven superiority.
2. Non-programming test subjects are in general supply.
3. Query languages are oriented toward individual use which is easily tested.
4. A realistic query problem is quite simple, allowing a large number of them to be done on an exam of reasonable length.
5. Query languages have a small set of data and statement types. There are usually no control sturctures. Thus, they are easy to learn.

6. Even with their simplicity and restricted problem domain, query languages have the potential to reach a very large user community.

These reasons assume added potency when compared to human factors testing of general purpose languages. General purpose languages have none of these attributes but their study has yielded useful experimental results.

2. HYPOTHESIS

One of the major issues in database query languages concerns the procedurality of query languages. Two experiments have been run. Each experiment tested the learnability of a nonprocedural query language and a procedural query language using human subjects. The experiments tested the basic hypothesis: People more often write difficult queries correctly using a procedural query language than they do using a nonprocedural query language. The languages chosen for this experiment were similar in all aspects except in the independent variable -- procedurality -- since the experiment takes the reductionist approach [Shneiderman, 1978].

SQL (formerly SEQUEL 2) [Denny, 1977] and TABLET [Stemple et al, 1978] exhibit the required properties. The similarities between SQL and TABLET are:

1. They both use the same data model, Codd's relational model [Codd, 1970].
2. They are relationally complete [Codd, 1971b].
3. They have similar language levels [Halstead, 1977].
4. Their syntactic differences are a function of their procedurality. Constructs that are independent of procedurality are identical.
5. They both use the same terminal equipment.

These similarities are detailed in [Welty, 1979].

The difference, again, is in the procedurality of the languages. Generally, a language is procedural if it specifies a step-by-step method for achieving a result. Nonprocedural

languages describe the desired result without specifying how it is to be achieved. (The idea is comparable to the difference between constructive and nonconstructive existence proofs in mathematics.) SQL is similar to Codd's relational calculus [Codd, 1971a] and TABLET (The Algebra Based Language for Enquiring of Tables) is based on Codd's relational algebra [Codd, 1971b].

Codd's relational algebra consists of a set of operations defined on relations. An operation on a relation or relations always yields another relation. A relational algebraic query specifies the ordered steps used in generating the result and, thus, is procedural.

A relational calculus query describes the elements of the desired relation. The query is purely descriptive, containing no method for achieving the desired relation. This type of query is nonprocedural.

Further discussion of the procedurality of the algebra and calculus is given by Codd [Codd, 1971a].

The experiment was set up to test the authors belief in the superiority of nonprocedural over procedural query languages. This belief in procedural methods can be explained through the use of a cognitive model of programmer behavior.

Various human factors studies including those referenced in section 1 of this paper have resulted in a cognitive model of programmer behavior [Shneiderman and Mayer, 1975]. Figure 1. illustrates the components of memory presented in this model.

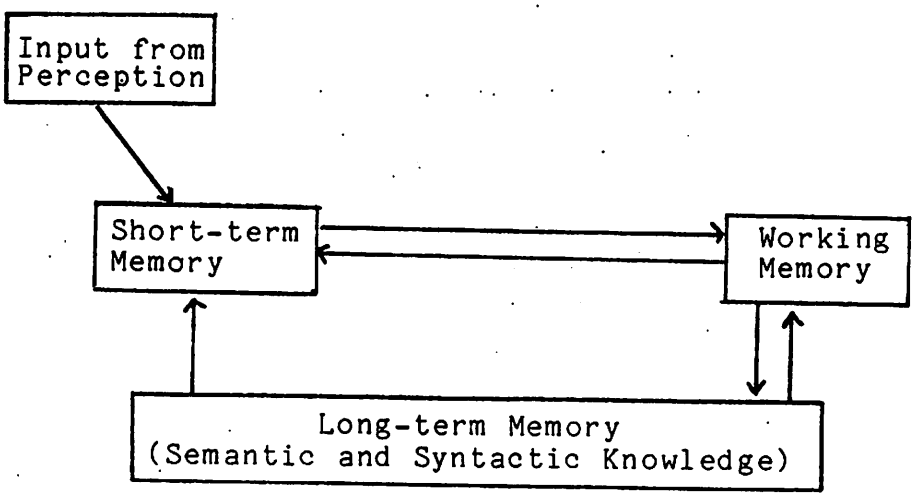
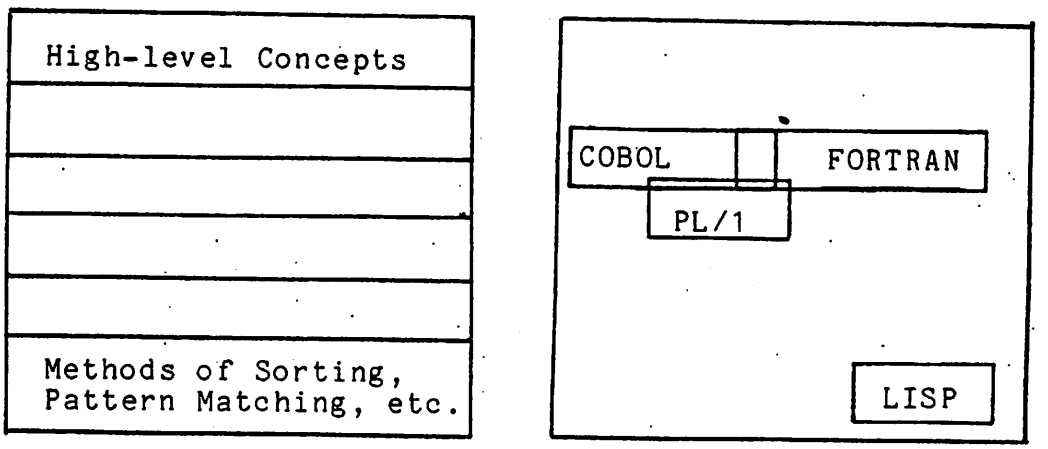


Figure 1

Memory Modules Required for Programming



Semantic Knowledge

Syntactic Knowledge

Figure 2

Long-term Memory

Short-term memory has a limited size and is a buffer between the outside world and working memory. The programmers permanent knowledge is stored in long-term memory. Working memory performs a synthesis of the short-term memory information with the relevant concepts from long-term memory.

Long-term memory is made up of two basic components (Figure 2.):

1. Semantic Knowledge concerns the general knowledge required for programming that is independent of any specific programming language.
2. Syntactic Knowledge is more precise, detailed and arbitrary (hence more easily forgotten) than semantic knowledge.

The information in the component structure of the cognitive model for an individual subject depends on that subjects background. The experimental subjects were divided into two categories -- subjects experienced with programming languages (FORTRAN, BASIC, etc) and subjects with no programming experience. The cognitive model for experienced subjects contains the syntax of their programming language as well as semantic structures necessary in programming. These semantic structures are procedural in nature and aid in the learning of other procedural language. The structures will not necessarily aid in the learning of a nonprocedural language.

Inexperienced subjects, our examples of the casual user, know English and know how to produce reports (query output) from the simple tables of the relational view. The query language needs to bridge the gap between knowing to do and knowing to express. Since the knowing to do is procedural in nature it seems worthwhile to try a procedural language. However the procedures of the language should be as close to the procedures of the person as is feasible, i.e. the semantic component of the user's cognitive world should be used where possible as the semantic component of the language. The syntax should not only be English, in order to use a part of the existing syntactic component, but should also mesh with the appropriate semantic model. (The results of the GROUP BY constructs in the two languages verify this latter point.)

3. THE LANGUAGES

The SQL and TABLET query languages are both based on the relational model of data. SQL uses English keywords in a template-like manner for the expression of queries against a database. TABLET specifies the operations that are performed on a relation. Some sample queries follow. These queries all use the COLLEGE database of Figure 3.

Q1. List the names of students from Ohio.

```
SQL:    SELECT NAME
        FROM STUDENT
        WHERE HOMESTATE = 'OHIO'

TABLET: FORM OHIOANS FROM NAME, HOMESTATE OF STUDENT
            KEEP ROWS WHERE HOMESTATE = 'OHIO'
            PRINT NAME
```

This SQL query is called a "simple mapping" and returns a value from the name column of a tuple in the STUDENT relation for which the value in the HOMESTATE column is 'OHIO'. The TABLET query first forms a working table named OHIOANS consisting only of the NAME and HOMESTATE columns of the STUDENT table. The KEEP ROWS command specifies that the rows (tuples) of OHIOANS for which the HOMESTATE column contains 'OHIO' are retained. The other rows are eliminated from the table. The values in the NAME column are then printed. Both SQL and TABLET eliminate duplicates from the tuples printed.

The calculus equivalent of the SQL query is

```
{(STUDENT.NAME):  STUDENT.HOMESTATE = 'OHIO'}
```

data base - COLLEGE

STUDENT

ID	NAME	SEX	HOMESTATE	MAJOR	REPID
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	2
2	JANE DOE	F	OHIO	ECONOMICS	9

TAKING

ID	COURSE	SECTION
1	HIST101	1
1	HIST102	2
1	POLSCI115	1
2	ECON105	3
2	ECON202	1
2	MATH101	1

FACULTY

ID	NAME	SEX	DEPT	COMHEAD	SALARY
312	BILL GRANT	M	ECONOMICS	216	20000
152	JOHN MILTON	M	HISTORY	312	14000
172	ANNE HALL	F	POLSCI	192	19000

DEPARTMENT

DEPT	BUILDING	HEAD
POLSCI	BILLINGS	172
HISTORY	BILLINGS	295
ECONOMICS	KEYNES	312
ENGINEERING	ENGINEERING	207

TEACHING

ID	COURSE	DEPT	SECTION	LIMIT	SIZE
312	ECON105	ECONOMICS	1	35	31
312	MATH101	MATH	2	40	40
152	HIST101	HISTORY	1	28	28
152	HIST102	HISTORY	2	32	19
172	POLSCI115	POLSCI	1	32	30

COURSES

COURSE	DEPT	TITLE	CREDITS
ECON105	ECONOMICS	INTRODUCTION TO ECONOMICS	3
MATH101	MATHEMATICS	COLLEGE ALGEBRA	3
HIST101	HISTORY	AMERICAN HISTORY	3
HIST102	HISTORY	EUROPEAN HISTORY	4

Figure 3

The COLLEGE Database

The SQL SELECT clause is identical to the relational calculus target list (i.e. STUDENT.NAME), except that the attribute of the target list is fully specified using the relation name. The FROM clause of the SQL query specifies the relation. The predicate of the calculus query is identical to the SQL WHERE clause except for the relation specification.

The TABLET query specifies PROJECT and RESTRICT operations [Codd, 1971b]. The FORM command says to project on the NAME and HOMESTATE attributes of the STUDENT relation. The result is bound to the name OHIOANS. The KEEP ROWS restricts OHIOANS to those tuples for which the HOMESTATE value is 'OHIO'. The PRINT command may be thought of as a project on the output file. Written in its algebraic form the query is:

```
OHIOANS ← STUDENT[NAME, HOMESTATE]
OHIOANS ← OHIOANS[HOMESTATE = 'OHIO']
OUTPUT ← OHIOANS [NAME].
```

Q2. List the average salary of economics faculty members.

```
SQL:    SELECT AVG(SALARY)
        FROM FACULTY
        WHERE DEPT = 'ECONOMICS'

TABLET: FORM ECONSAL FROM SALARY, DEPT OF FACULTY
            KEEP ROWS WHERE DEPT = 'ECONOMICS'
            PRINT AVG(SALARY)
```

Both languages allow functions in a query. The functions are MAX, MIN, AVG, SUM and COUNT. These functions apply to the given column. Duplicates are not eliminated from the column on which the function operates. There are no parallels in the calculus and algebra because neither includes functions in its definition.

Q3. List the names of students taking ECON105.

```
SQL:  SELECT NAME
      FROM STUDENT
      WHERE ID =
          SELECT ID
          FROM TAKING
          WHERE COURSE = 'ECON105'
TABLET:  FORM ECONSTUDENTS FROM NAME, ID OF STUDENT
         ADD COLUMN SOURCE OF TAKING BY ID = ID
         KEEP ROWS WHERE COURSE = 'ECON105'
         PRINT NAME
```

In the SQL query the lower mapping returns a set of ID's to the upper mapping, this is called chaining. In TABLET, the ADD COLUMN statement joins the COURSE column of the TAKING table to the ECONSTUDENTS table using equal ID values from the two tables. This operation results in ECONSTUDENTS containing 3 columns: NAME, ID and COURSE.

The calculus equivalent of the SQL query is

$\{(STUDENT.NAME) : \exists\ x \in TAKING\ (STUDENT.ID=x.ID \wedge x.COURSE='ECON105')\}$

The similarity is not syntactically explicit, but a simple transformation is all that is necessary. Both queries can be read as: list the names of all students whose ID's are the same as the ID in a tuple of the TAKING table for which the COURSE value in that tuple is ECON105.

Only the ADD COLUMN command is new to us in the TABLET query. It is the equivalent of the algebraic join operator [Codd, 1971b].

Q4. List the ID's of department heads who are also committee heads.

```
SQL:   SELECT HEAD
         FROM DEPARTMENT
       INTERSECT
         SELECT COMHEAD
         FROM FACULTY
TABLET: FORM HEADID FROM HEAD OF DEPARTMENT
        FORM COMHEADID FROM COMHEAD OF FACULTY
        KEEP ROWS OF COMHEADID
          WHERE COMHEAD IN HEAD OF HEADID
        PRINT COMHEAD
```

SQL uses the usual set operators -- UNION, INTERSECT and MINUS. TABLET forms two tables and then performs a restriction (KEEP ROWS) of the tuples in one table using the contents of the other table. TABLET uses NOT IN corresponding to SQL's MINUS. The ADD ROWS command is the TABLET analog of UNION.

The SQL set operators -- UNION, INTERSECT and MINUS -- correspond to $\vee$, $\wedge$ and - [Codd, 1971b]. Obviously UNION, INTERSECT and MINUS are algebraic in nature; they perform operations on relations and yield new relations. SQL is being shown as similar to the calculus but it does have algebraic (procedural) components. In fact, the above correspondences are shown in the reduction of a calculus query into the algebra [Codd, 1971b].

As mentioned above, TABLET does not use the set operators UNION, INTERSECT and MINUS that are part of the definition of the relational algebra. The TABLET operators perform transformations on relation so they are algebraic in nature.

Q5. List the average salary of faculty members in each department.

```
SQL:    SELECT DEPT, AVG(SALARY)
        FROM FACULTY
        GROUP BY DEPT

TABLET: FORM DEPTAVG FROM DEPT, SALARY OF FACULTY
            GROUP BY DEPT
            PRINT DEPT, AVG(SALARY)
```

Both SQL and TABLET use GROUP BY to denote the partitioning of the relation (table). In SQL, GROUP BY is a clause in the SELECT statement. GROUP BY in TABLET is a command in itself. Due to the function in the queries there are no calculus or algebraic versions.

Q6. List the name of each student and the names of the courses he is taking (eg. JOHN JONES HIST101).

```
SQL:    SELECT NAME, COURSE
        FROM STUDENT, TAKING
        WHERE STUDENT.ID = TAKING.ID

TABLET:  FORM NAMECOURSE FROM NAME, ID OF STUDENT
             ADD COLUMN COURSE OF TAKING BY ID = ID
             PRINT NAME, COURSE
```

The join operation is required by both SQL and TABLET queries. NAME and COURSE come from the STUDENT and TAKING relations,

respectively. The ID columns in the SQL WHERE clause are qualified to avoid ambiguity. The TABLET query uses the same format as in Q3.

The calculus query is

{(STUDENT.NAME, TAKING.COURSE) : STUDENT.ID=TAKING.ID}

Again, the similarity is obvious. Here SQL fully specifies the attributes that would otherwise be ambiguous.

Q7. List the ID's of students taking all the art classes offered.

```
SQL:    SELECT ID
        FROM TAKING
        GROUP BY ID
        HAVING SET(COURSE) CONTAINS
            SELECT COURSE
            FROM TEACHING
            WHERE DEPT = 'ART'
TABLET: FORM ARTCOURSES FROM COURSE, DEPT OF TEACHING
        KEEP ROWS WHERE DEPT = 'ART'
        FORM ARTSTUDENTS FROM ID, COURSE OF TAKING
        GROUP BY ID
        KEEP GROUPS
          WHERE COURSE CONTAINS COURSE OF ARTCOURSES
        PRINT ID
```

In SQL, the HAVING clause is only used to restrict a partitioned relation, so HAVING is only used with GROUP BY. The SET function has the value of all the COURSE names in a partition. The TABLET query uses two tables. One (ARTCOURSES) contains the names of

all the art courses offered. The second table (ARTSTUDENTS) contains the ID's and courses taken by all students, partitioned on identical ID values. The KEEP GROUPS command (analogous to the HAVING clause in SQL) retains those partitions in which the COURSE column contains all the art courses offered.

The calculus equivalent uses universal quantification

ARTCOURSES $\equiv$ {(TEACHING.COURSE) : TEACHING.DEPT = 'ART'}

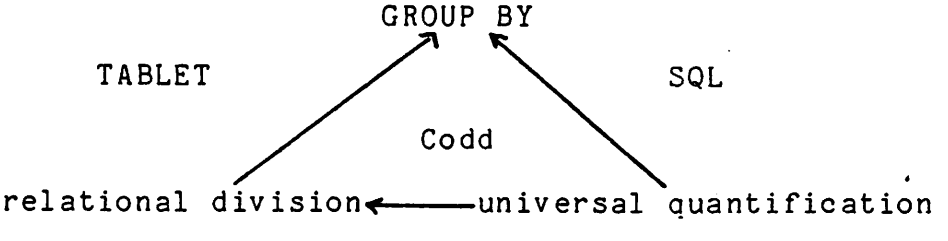
{(TAKING.ID) : $\forall$ t $\in$ ARTCOURSES $\exists$ s $\in$ TAKING

(TAKING.ID=s.ID $\wedge$ s.COURSE=t.COURSE)}.

Universal quantification is considered overly complex for most nonmathematicians, especially the casual user [Thomas, 1976]. There is very little similarity between the SQL query and the calculus query. GROUP BY specifies an operation (i.e., partitioning [Furtado and Kerschberg, 1977]) on a relation. Thus, GROUP BY is a algebraic or procedural element embedded in a nonprocedural language. Experimental results suggest that this incongruity has a bad effect on the learnability of this construct.

The equivalent algebraic query uses relational division, an operator that is fairly complex.

Codd [Codd, 1971b] shows that universal quantification becomes relational division when a calculus query is transformed into an algebraic query. So, SQL and TABLET use the identical construct for corresponding features of their underlying languages. This is illustrated by the diagram



3.1 Procedurality metric, P.

The relational calculus is agreed to be a nonprocedural language and the relational algebra a procedural language [Codd, 1971b; Date, 1977; Stonebraker and Rowe, 1977]. The calculus is nonprocedural by definition -- the desired result is described; no method for its achievement is produced. The algebra gives the method; the result of the method is the desired result.

TABLET is obviously procedural.

The nonprocedurality of SQL is open to question. The designers of SQL describe it as nonprocedural (descriptive) [Chamberlin and Boyce, 1974; Astrahan and Chamberlin, 1975]. Others [Stonebraker and Rowe, 1977] classify SQL as procedural. The problem is that procedurality is a continuum. FORTRAN programs may be nonprocedural when compared to the equivalent program in an assembly language but procedural when compared to an APL or SNOBOL implementation. SQL does have some procedural properties but is, in the main, nonprocedural. The nonprocedurality of SQL is emphasized when compared to the procedural nature of TABLET.

Procedurality, then, is relative. Simply saying that a language is procedural or nonprocedural is usually not enough; a reference must be given. A procedurality metric would aid by defining the term and yielding values that could be compared.

In order to clarify our intuition about the much maligned concept of procedurality we present three representations of the same computation and disccuss how they vary in procedurality.

The expression to be computed is

(A*B)+(C*D)

In the simplified FORTRAN dialect GOTRAN [Andree, 1967] which allows only one operator expression on the right side of an assignment statement, we could write

```
E1 = A*B
E2 = C*D
E  = E1+E2
```

This is clearly proccedural. Note that the three bindings (to E1, E2, and E) and three operations are fully ordered.

In a language with a left-to-right evaluation rule we could write

```
E1 = A*B
E  = C*D+E1
```

whether this is more or less procedural than the preceding example is not clear but it does have one less variable binding.

IN FORTRAN we would write

```
E = A*B+C*D
```

This seems to us less procedural than the GOTRAN example. There are fewer steps, (those involving the two intermediate variable bindings are missing), and less order of evaluation is specified, since FORTRAN semantics allows either multiplication to be performed first.

A metric which matches our intuition about procedurality of programs is proposed here. The metric is based on the following properties of programs:

1. The number of variable bindings.
2. The number of operations.
3. the degree to which the bindings and operations are ordered by the language semantics.

The procedurality metric, P, is defined as

$$P = \frac{\text{number of variable bindings}}{\text{number of permissible variable binding orderings}} + \frac{\text{number of operations}}{\text{number of permissible operation orderings}}$$

In the GOTRAN example above there are three variable bindings, with only one semantically permissible order specified, and three operations, also strictly ordered by the language semantics. Thus

$$P = 3/1 + 3/1 = 6$$

In the left-to-right precedence example there are two fully ordered variable bindings and three operations with also only one permissible order of application. Thus

$$P = 2/1 + 3/1 = 5$$

The FORTRAN example has only one variable binding. The three operations can however be applied in two permissible orders. Thus

$$P = 1/1 + 3/2 = 2.5$$

We believe this metric produces a ranking of program procedurality which matches most people's intuition. It has the advantage of being concrete and based on the primitive concepts of variable binding, operation application, and their permissible orderings.

In SQL any variable binding is done in the FROM clause. The normal SQL operation is an entire SELECT specification (SELECT, FROM, WHERE and, optionally, GROUP and HAVING). Other operations are UNION, INTERSECT, MINUS, INSERT, and ←.

Each TABLET command performs an operation and binds the result of that operation to a table name.

For example, the SQL query Q3 in section 3 has no variable bindings but has 2 SELECT commands with an implied ordering so

$$P = 0 + 2/1 = 2.$$

The corresponding TABLET query has 3 variable bindings (one for every command except the PRINT command) and 4 operations. There is only one permissible order for the bindings and only one for the operations, therefore

$$P = 3/1 + 4/1 = 7$$

Using this measure on 60 queries of varying difficulty from the human factors experiments, the average P for SQL was 1.8 and 7.3 for TABLET. The means were significantly different at the .0001 level as measured by a paired t-test. This measure confirms the relative classifications of SQL as nonprocedural and TABLET as procedural.

4. THE FIRST EXPERIMENT

This experiment was run in the Spring Semseter, 1978. The subjects were 72 undergraduate students taking a 1 credit accounting course. Most subjects were business majors. The subjects were divided into two groups -- 35 subjects learned SQL and 37 subjects learned TABLET.

Subjects were also classified as "inexperienced" and "experienced". Inexperienced subjects had no previous experience with computers. Experienced subjects had a course in either BASIC or FORTRAN. A questionnaire showed that the subjects were familiar with the operators >, <, =, etc as well as the set operators union and intersection.

Subjects were motivated to take the course by the credit and a desire to learn about computers.

The languages were taught using manuals read outside class. These manuals, one presenting SQL and the other presenting TABLET, contained identical examples and problems presented in the same sequence. Each manual contained 12 lessons. Concepts that were identical in the two languages were presented identically.

There were fourteen class meetings. These meetings were devoted to answering questions on the lessons and to quizzes covering the material presented in the lessons. No lecturing was done in the class meetings. Since no material was presented in class, subjects learned the languages entirely from the manuals.

A final exam was given immediately after the course. the final consisted of 30 English questions (e.g. How many suppliers supply item 19?) and the subjects were required to write the corresponding query in their query language. The final was an open book exam and the students grades were based solely on their final exams.

A retention test was given 3 weeks after the final. This test was of identical format to the final. In fact, although the questions were different on the final and the retention test, the correct response to each question on the final was identical in structure to the corresponding query on the retention test. The retention test was a closed book test. In addition, the subjects were discouraged from studying for the retention test. The retention test did not affect student grades, but it was a required part of the course. Students had to take the test but had no motivation for studying. This delay and lack of studying was used to model the intermittent use of query languages.

The grading method for queries in all the tests was similar to Reisner's [Reisner, 1976]. Each solution was classified as one of the following:

correct - the solution was completely correct.

minor language error (ML) - the solution was basically correct but had a small error that would be found by a reasonably good translator.

minor operand error (MO) - the solution has a minor error in its data specification, perhaps a mis-spelled column name.

minor substance error (MS) - the solution yields a result that is not quite correct but its incorrectness is due to the statement of the problem.

correctable (CO) - the solution is wrong but correctable by a good compiler.

major substance error (XS) - the query is syntactically correct but answers a different question than the one specified.

major language error (XF) - a major error in the syntax (form) of the language has been made.

incomplete (IN) - incomplete query.

unattempted (UN) - no solution was attempted.

Thee first four categories -- correct, ML, MO, MS -- were called essentially correct responses. The other five categories were classified as incorrect. The correctable solutions used knowledge specific to the database being used and would not be correctable in the general case. If several errors were found in a query, the error is categorized as the lowest one found in the above list.

4.1 Results of the First Experiment

The mean number of essentially correct solutions as well as mean times to take the final and retention tests and mean study time for the lessons are found in Table 1. The test results are subdivided into various categories. These are:

easy 10 problems from Lessons 1-5 of the manuals covering (in SQL terms) simple mapping, simple mapping with arithmetic oerations,

	No. of problems	SQL mean (all 35 subjects)	TABLET mean (all 37 subjects)	SQL mean (17 inexperienced subjects)	TABLET mean (20 inexperienced subjects)	SQL mean (18 experienced subjects)	TABLET mean (17 experienced subjects)
final score	30	18.371	18.541	16.824	17.050	19.833	20.294
final time (minutes)		116.71	120.35	124.53	124.10	110.28	115.94
retention score	30	13.600	14.784	12.412	12.850	14.722	17.118
retention time (minutes)		66.514	76.351	69.412	78.050	63.778	74.353
easy final	10	8.4286	8.1351	7.9412	7.5500	8.8889	8.8235
easy retention	10	7.8286	7.4054	7.2353	6.7500	8.3889	8.1765
hard final	20	9.9429	10.4054	8.8824	9.5000	10.944	11.471
hard retention	20	5.7714	7.4054	5.1765	6.1000	6.3333	8.9412
group final	6	2.8000	2.7027	2.5882	2.2000	3.0000	3.2941
group retention	6	1.0571	1.9459	0.8824	1.3500	1.2222	2.6471
join final	3	1.3714	1.8108	1.1765	1.7500	1.5556	1.8824
join retention	3	0.5429	1.4865	0.5294	1.4000	0.5556	1.5882
chaining final	3	2.000	2.0811	2.0588	1.9500	2.3333	2.2353
chaining retention	3	2.0571	1.7027	1.7059	1.5500	2.3889	1.8824
set final	3	2.6000	1.7027	2.5294	1.9000	2.6667	1.4706
set retention	3	2.1143	0.9430	1.9412	1.0500	2.27778	0.8824
combination final	4	2.0000	2.6757	1.8235	2.4500	2.1667	2.9412
combination retention	4	1.3143	1.7838	1.3529	1.4000	1.2778	2.2353
average study time (minutes, Lesson 1-12)		29.324	36.214	33.378	39.138	25.496	32.775

Table 1

Mean Number of Essentially Correct Responses and Other Results
According to Subject Experience and Query Category

simple mapping with built-in functions, and composition (chaining). The easy problems are problems 1-10 on both the final and retention test.

hard — 20 problems for Lessons 6-12 of the manuals. These cover GROUP BY, set functions, join and combinations of constructs. The hard problems are problems 11-30 on both the final and the retention test.

group — 6 problems that require the GROUP BY construct. These are problems 17-22 on both the final and the retention test.

join — 3 problems requiring joining in both SQL and TABLET. These are problems 23-25 on both the final and the retention test.

chaining (or composition) — 3 problems that require printing from one table using another table for the predicate. These are problems 9-11, on both the final and the retention test.

set — 3 problems using UNION, INTERSECT and MINUS (SQL terms). These are problems 14-16 on both the final and the retention test.

combinations — 4 problems combining various language constructs. These are problems 13, 26, 27, and 30 on both the final and the retention test.

The problem numbers listed above correspond to the numberings in the Appendix. There were five randomizations of the exam in the actual experiment. The easy/hard dichotomy was based on Reisner's classification of SQL as a "layered language" [Reisner, 1976]. Reisner says that SQL queries come in two layers. The first layer is for novices, the second for more sophisticated users. The easy problems in the SQL/TABLET study correspond to layer 1. The hard problems correspond to the queries of layer 2.

	degrees of freedom	F ratio	significance
experience	1,68	4.76	(<.05)
language	1,68	1.99	(<.05)
experience x language	1,68	.05	-

a Mean study time of Lessons 1-12.

	degrees of freedom	F ratio	significance
experience	1,68	1.44	-
language	1,68	6.10	(<.01)
experience x language	1,68	.06	-

Time to take retention test.

	degrees of freedom	F ratio	significance
experience	1,68	6.51	(<.05)
language	1,68	8.71	<.005
experience x language	1,68	2.22	-

c Retention, group problems.

	degrees of freedom	F ratio	significance
experience	1,68	.33	-
language	1,68	26.53	<.001
experience x language	1,68	.19	-

d Retention, join problems.

	degrees of freedom	F ratio	significance
experience	1,68	5.76	(<.05)
language	1,68	.25	-
experience x language	1,68	.01	-

e Retention, chaining problems.

	degrees of freedom	F ratio	significance
experience	1,68	.23	-
language	1,68	42.74	<.001
experience x language	1,68	2.08	-

f Retention, set problems.

	degrees of freedom	F ratio	significance
experience	1,68	3.82	(<.05)
language	1,68	11.12	<.005
experience x language	1,68	4.12	(<.05)

g Retention, combination problems.

Table 2.

Summary of Analysis of Variance for Study Time, and Retention Scores (See Table 1 for means.)

4.2 Statistical Analysis

Subject scores on the hard problems of both the final and retention tests were analyzed using a fully crossed, two-way analysis of variance. The two independent variables were the languages (SQL and TABLET) and the experience levels of the subjects (inexperienced and experienced). The final showed no significant difference in subject performance on hard problems due to language at the .05 significance level. The experience effect was significant at the .05 level. Our main concern is the language effect so the results of the final are not investigated further. Subject performance on the hard problems of the retention test showed significance at the .05 level (actually significance was at the .01 level) due to language. The experience effect was also significant at the .05 level (in this case the actual significance was at the .005 level).

The significance of the difference in subject performance on the hard problems of the retention test leads us to investigate the problem categories presented earlier. In addition, we are interested in the average amount of time subjects spent studying the lessons and the amount of time required to take the retention test. Since we are running many statistical tests on the same data we may find significance due only to the number of tests being run. For this reason we reduce the required significance level from .05 to .007 (.05 divided by the number of variables being tested, i.e. 7). The results are given in Table 2. Effects that are of marginal significance are enclosed in parentheses.

Table 3 presents the group contrasts resulting from the one-way analysis of variance. Again, the required significance level is .007.

An interesting result is exhibited in Table 3. The difference in performance between inexperienced SQL subjects and experienced SQL subjects is minor. On the other hand there is a great deal of difference between the TABLET subjects. This shows that even minimal experience in procedural languages (i.e. FORTRAN or BASIC) aids the TABLET subjects but not the SQL subjects. This result empirically buttresses the classification of TABLET as procedural and SQL as nonprocedural.

Table 1 and Table 2 show that the performance of the SQL and TABLET subjects were significantly different in the following categories:

1. Mean study time for lessons 1-12, marginally significant (SQL subjects required less time).
2. Time required to take the retention test, marginally significant (SQL subjects required less time).
3. Hard problems on the retention test (TABLET subjects outperformed SQL subjects).
4. Join problems on the retention test (TABLET subjects outperformed TABLET subjects).
5. Set problems on the retention test (SQL subjects outperformed TABLET subjects).
6. Group problems on the retention test (TABLET subjects outperformed SQL subjects).
7. Combination problems on the final test (TABLET subjects outperformed SQL subjects).

	Significance of experienced SQL subjects vs. experienced TABLET subjects	Significance of inexperienced SQL subjects vs. inexperienced TABLET subjects	Significance of inexperienced SQL subjects vs. experienced SQL subjects	Significance of inexperienced TABLET subjects vs. experienced TABLET subjects
Mean study time (Lessons 1-12)	-	-	-	-
Time to take retention test	-	-	-	-
Retention, group	<.005	-	-	<.005
Retention, join	<.0005	<.005	-	-
Retention, chaining	-	-	-	-
Retention, set	<.0001	<.0005	-	-
Retention, combination	<.0005	-	-	<.005

Table 3.

Summary of Group Contrasts for Study Time, Lesson Difficulties and Retention Scores
(See Table 1 for means.)

TABLET subjects required more study time and more time to take the retention test than SQL subjects did. TABLET has more complex syntax and semantics than SQL has. It takes extra time to learn and write TABLET. TABLET subjects are required to learn how to manipulate tables, SQL subjects are directed mainly by SQL's syntax.

TABLET's procedurality yields rewards in the writing of hard queries. In these queries the skill acquired in table manipulations is put to use. The skill is analogous to the skill of riding a bicycle, once learned it is easily retained. SQL does not require this sort of skill of its users.

TABLET uses the join construct (ADD ROWS) where SQL uses two constructs, chaining and joining. While SQL subjects used chaining, TABLET subjects acquired experience with joining. On the problems in which both languages used joining, TABLET subjects had an advantage and performed significantly better than SQL subjects.

SQL subjects outperformed TABLET subjects on set problems. Set concepts as well as set operators, UNION and INTERSECTION, were familiar to all the subjects due to the new math. SQL uses these familiar concepts. TABLET uses concepts novel to the subjects.

TABLET subjects outperformed SQL subjects on the group problems. This is an interesting result because both languages use the same construct -

GROUP BY column name.

TABLET has many commands that perform operations on relations, but SQL queries are more tuple-oriented. The GROUP BY specifies an operation on a relation, so it is more table-oriented. Also, GROUP BY is an imperative construct, but appears as a subordinate clause within SQL's SELECT. In TABLET the GROUP BY is a separate (imperative) command. Finally, two SQL subjects said that the GROUP BY was in the wrong position in the command, suggesting it occur before the SELECT because the output specified in the SELECT is dependent on the GROUP BY clause. The SELECT clause is analogous to the PRINT in TABLET and GROUP BY does occur before the PRINT in TABLET.

TABLET subjects had more facility with novel queries than SQL subjects as exhibited by their performance on the combination problems. The step-by-step nature of TABLET allows the linear solution of problems. In SQL the user has to be aware of the total solution to a problem before starting writing. TABLET allows a piecemeal approach.

Tables 2 and 3 show more significant results for the retention test than for the final, especially for experienced subjects. At the final both SQL and TABLET subjects had attained about the same level of performance. As time went on, subjects experienced with BASIC or FORTRAN (both procedural languages) retained TABLET because TABLET's procedural nature matched their procedural experience. TABLET retention was reinforced by this experience. SQL did not match their experience and SQL performance deteriorated.

5. THE SECOND EXPERIMENT

A second experiment was run in the Spring semester, 1979. The results of the first experiment are not very conclusive with respect to inexperienced subjects. The only significant difference in this group are in the use of the set and join constructs.

It was decided to run the experiment again with an increasegd sample of inexperienced subjects. The second experiment was primarily the same as the first but did have several differences.

TABLET was changed to use the UNION, INTERSECT and MINUS operators. Thus, Q4 in section 3 would be

```
FORM HEADID FROM HEAD OF DEPARTMENT
FORM COMHEADID FROM COMHEAD OF FACULTY
FORM BOTH FROM HEAD OF HEADID
    INTERSECT HEAD OF COMHEADID
PRINT HEAD.
```

In order to insure a sufficiently large supply of subjects we offered $50 to each student in addition to the 1 credit. In the first experiment the subjects were unpaid.

The other difference was that the classes had a different instructor than in the first experiment.

Other aspects of the experiments were identical.

T-tests were used for the statistical analysis of the results since the subjects were in two homogeneous groups -- inexperienced SQL subjects and inexperienced TABLET subjects.

5.1 Results of the Second Experiment

On the second experiment we use a t-test to check the main hypothesis. TABLET subjects outperformed SQL subjects on hard final problems at the required significance level, .05 (the actual significance level was .01, means were 8.1 and 10.5 essentially correct responses out of 20 for SQL and TABLET respectively). The results of the retention test were not significant. For this reason the final is examined further but not the retention. As in the first example, the acceptance level for the additional variables is .007. The results of the t-tests for the second experiment are presented in Table 4.

The results show the same basic pattern as the results of the first experiment. TABLET subjects did better on the hard problems as a whole, specifically on the join and combination problems. An interesting result of this experiment is that the differences were more significant on the final than in the retention test. The reverse was true in the first experiment.

SQL subjects again required less time to take the final test and less study time.

The use of UNION INTERSECT and MINUS in TABLET helped. The difference in performance between SQL and TABLET subjects on the final test was not significant. The TABLET changes seemed to help.

To explain the performance on the final we again consider the cognitive model presented earlier. Experienced subjects in the first experiment could do well on the final independently of the language for the most part. As time passed, their

	SQL mean	TABLET mean	Value	Significance
Time studying for final	135.0	107.7	1.09	(<.05)
Time to take final	112.3	119.3	-.95	(<.05)
final group	2.3	2.2	.33	-
final join	1.3	2.1	-4.06	<.001
final chaining	2.1	2.2	-.65	-
final set	1.8	1.7	.65	-
final combinations	1.7	2.3	2.62	(<.01)

Table 4

Results of the Second Experiment.

Means for the SQL and TABLET subjects and significance levels of the t-tests.

performance on the nonprocedural langauge (SQL) deteriorated more than for the procedural language because the procedural language conformed to their pre-existing semantic component of memory.

The inexperienced subjects of the second experiment did not have any procedural training with computers. It would seem that either TABLET was more "natural", i.e. TABLET fit the subjects pre-existing non-computer experience better than SQL, or that the procedural experience gained by TABLET subjects in learning the language resulted in performance superior to that of the SQL subjects.

In general, the second experiment confirmed the results of the first but with a group of inexperienced subjects. The second experiment also shows this experiment to be replicable -- a prime element of a good experiment.

6. RECOMMENDATIONS FOR LANGUAGE CHANGES

The results of this experiment show that there are problems with the following language elements:

1. Join in SQL.
2. Sets in TABLET.
3. Grouping in SQL.

The queries used in this section refer to the MAILORDER database used on the final and retention tests, see the Appendix.

Recommendations concerning join problems in SQL

The difference between SQL and TABLET for join problems is primarily due to the experience TABLET subjects received in using the ADD COLUMNS command while SQL used chaining. TABLET continues using ADD ROWS for joining but SQL uses a construct different than chaining.

Lochovsky [Lochovsky, 1978] studied a language directly based on the relational calculus and recommended eliminating the join from the predicate and adding an explicit join specification. At present, the correct response to the query, "List the names of people who have exceeded their credit limit and the item numbers of items they have charged," is

```
SELECT  NAME, ITEMNO
FROM CHARGEACCTS, CHARGED
WHERE CHARGEACCTS.ACCTNO = CHARGED.ACCTNO
   AND TOTALBILL > LIMIT.
```

The joining is done by the

```
CHARGEACCTS.ACCTNO = CHARGE.ACCTNO
```

in the where clause. A possible substitute query would be

```
SELECT NAME, ITEMNO
FROM CHARGEACCTS, CHARGED
JOINED BY CHARGEACCTS.ACCTNO = CHARGED.ACCTNO
WHERE TOTALBILL > LIMIT.
```

The JOINED BY could be simplified to

```
JOINED BY ACCTNO
```

since this is unambiguous. This syntax makes SQL even less calculus-oriented and more algebraic.

Recommendations concerning set problems in TABLET

TABLET should make use of the set theory background that is so common today. Explicit use of the UNION, INTERSECT and MINUS operators is recommended. For example, the correct response to the query, "List the account numbers of accounts with credit ratings of 10 who have charged item 19," was

```
FORM RATING10
   FROM ACCTNO, RATING OF CHARGEACCTS
KEEP ROWS WHERE RATING = 10
FORM ITEM19
   FROM ACCTNO, ITEMNO OF CHARGED
KEEP ROWS WHERE ITEMNO = 19
KEEP ROWS OF RATING10 WHERE ACCTNO
   IN ACCTNO OF ITEM19
PRINT ACCTNO
```

The last KEEP ROWS command, above, would be replaced by

```
FORM BOTH
   FROM ACCTNO OF RATING10
   INTERSECT ACCTNO OF ITEM19.
```

The query is still not as succinct as the SQL equivalent but is based on the same concepts.

The second experiment used UNION, INTERSECT and MINUS in TABLET. While inexperienced SQL subjects significantly outperformed inexperienced TABLET subjects on the first experiment (at the .0005 level), the difference on the second experiment was not significant. In fact, the means were nearly identical. This suggests an improvement due to the set constructs in TABLET.

Recommendations concerning group problems in SQL

It is tempting to use the student comments referred to in section 4.2 and put the GROUP BY clause earlier in the SELECT. The place it should really go is before the SELECT itself. Making this change has ramifications that result in a very procedural, TABLET-like language.

The simplest change is to make the GROUP BY seem more like a subordinate clause and not an imperative. Using the participle, GROUPED or GROUPING would have this effect. For example, the correct response to the query, "For each supplier list the supplier name and the average wholesale price of the items he supplies.", is

```
SELECT SUPPNAME, AVG(WHOLESALE)
FROM SUPPLIES
GROUP BY SUPPNAME.
```

The SQL query would become

```
SELECT SUPPNAME, AVG(WHOLESALE)
FROM SUPPLIES
GROUPED BY SUPPNAME
```

or

```
SELECT SUPPNAME, AVG(WHOLESALE)
FROM SUPPLIES
GROUPING BY SUPPNAME.
```

Since the group restriction clause uses the keyword HAVING, it would seem to be a matter of taste as to which GROUP participle is used. The participle reduces the imperative nature shown in GROUP BY. This also has a nice correspondence to English sentence structure and reads well. The GROUPED BY seems to read better than GROUPING BY, especially if a HAVING clause were to follow.

7. DISCUSSION AND CONCLUSIONS

Thse experiments can be seen as testing languages at three levels of abstraction:

1. SQL vs TABLET. The testing of the actual languages involved, an engineering experiment.
2. Calculus-like vs algebraic relational query languages. Testing the two types of relational query languages.
3. Nonprocedural vs procedural query languages. Extending the results of the experiment to make statements about query languages in general.

At level 1 we found TABLET to be superior to SQL in difficult queries in general, in several of its language constructs and in the combination of constructs. SQL was superior in one construct (set operations) and in learning and writing time. There was little difference in the easy problems. Thus, TABLET would be recommended when difficult queries are the norm. SQL has the advantage in shorter learning and writing times and should be used if these are basic to the application, especially if most queries will be easy.

At this level of abstraction we also make suggestions for language changes. The experiment has aided in the evolution of these languages.

At the next level, level 2 our results show a superiority of the algebraic over the calculus-like language. Obviously, the data structures used in the algebraic language and the allowable algebraic operators must be on the right level and be understandable to the user population.

The results of this experiment contradict the intuition of some database experts [Codd, 1971a; Date, 1977]. This illustrates the need for testing intuition.

Level 3 is the level of our original hypothesis about procedural and nonprocedural query languages. While we have not tested all possible procedural and nonprocedural query languages, we have tested languages that exemplify the two approaches. At this level the original hypothesis has been supported: Subjects more often wrote difficult queries correctly using the procedural query language (TABLET) than they did using nonprocedural query language (SQL). We are not arguing that the more procedural a langauge is the easier it is to learn. A study of SQL against any Von Neumann assembly langauge would be a rout. An assembly language is definitely more procedural than SQL but is not at the proper syntactic level. For an analysis of the comparative syntactic levels of SQL and TABLET see [Welty, 1979].

Referring again to the cognitive model presented earlier, the results of the experiments suggest that even people with no computer experience are procedurally oriented. The semantic component may not contain elements applicable to programming languages, but contains methods and models used in daily life. The procedural nature of TABLET corresponds well to these procedural methods and models.

Why, then, do many database researchers support nonprocedural languages? While the experiments do not directly test this question, a look at the cognitive model may help. Most researchers tend to know many languages. They have syntactic and

semantic models of many languages, database models, programming method, etc. This strong base allows them to abstract problems away from the procedural solutions (necessary to novices) into nonprocedural solutions. Nonprocedural solutions have an appealing elegance. The problem then, is that the researchers design languages based on their powers of abstraction, forgetting the required procedural foundations.

8. SUMMARY

Two experiments have been described testing the performance of subjects in writing queries of a relational database. In both experiments, one group of subjects learned SQL -- a nonprocedural, relational calculus-like query language -- and the other group learned TABLET -- a procedural langauge based on the relational algebra. The experiments tested the hypothesis that people more often write difficult queries correctly using a procedural query language than they do using a nonprocedural query language. In addition, the experiments were used to compare specific constructs in the languages in order to improve these constructs.

The first experiment used 72 subjects. Approximately half the subjects had no computer experience and the other half had one course in FORTRAN or BASIC. The subjects were divided into two groups -- one learning SQL, the other learning TABLET. The subjects learned the languages from manuals augmented with question-answering sessions and short quizzes. Subjects were then tested with a final test and, three weeks later, a retention test.

The results of the tests supported the hypothesis. In addition, problems were found with several language constructs. These results were found for the two groups as a whole and for the experienced subjects. The results for the inexperienced subjects were inconclusive.

A second experiment was run testing a large number (78) of inexperienced subjects. The subjects were divided in two groups and the experiment was basically identical to the first experiment. The results confirmed those of the first experiment.

Recommendations for language changes based on these experiments have been made.

APPENDIX

data base - MAILORDER

DEPARTMENT

DEPT	FLOOR
CLOTHES	3
CAMPING	2
SHOES	3

ITEM

ITEMNO	DESCRIPTION	RETAIL	ONHAND	REORDER	ALTERNATE
1	SHEEPSKIN SLIPPER	23.00	933	1000	2
2	ACRYLIC SLIPPER	19.00	2079	2000	8
3	WOOL BLANKET	27.50	1957	2000	4
4	THERMAL BLANKET	19.50	3056	2000	19

SELLS

DEPT	ITEMNO	QUOTA
CLOTHES	1	500
CLOTHES	2	500
CAMPING	3	400
CAMPING	4	600

SUPPLIER

SUPPNAME	LOCATION
JIM'S SPORTS	BOSTON
WARMTH, INC	ANCHORAGE

SUPPLIES

SUPPNAME	ITEMNO	WHOLESALE	ONORDER
JIM'S SPORTS	1	15.00	500
JIM'S SPORTS	2	10.00	0
JIM'S SPORTS	3	16.50	500
WARMTH, INC	3	17.00	250
WARMTH, INC	4	10.50	0

CHARGEACCTS

ACCTNO	NAME	SEX	TOTALBILL	LIMIT	RATING	REFERREDBY
101	JAMES LEE	M	1543.97	1500	9	108
102	SUE JONES	F	296.95	1000	7	101

CHARGED

ACCTNO	ITEMNO	QUANTITY	PERITEM
101	1	4	20.00
101	3	2	27.50
102	1	1	23.00
102	2	1	15.00

1. List the names of suppliers who supply item 19.

2. Which accounts have exceeded their credit limit? List the account numbers.

3. Which customers with a credit rating higher than 8 were referred by account 108? (List the customer names.)

4. For the Christmas season each charge account will have its credit limit doubled. Print the account numbers and new limits.

6. List the item numbers of items for which the quantity on hand is more than twice the reorder quantity.

7. How many suppliers supply item 19?

8. Which item or items have the highest wholesale price? (List the item number.)

9. List the locations of suppliers that supply item 19.

10. List the names of people who have charged hiking boots.

11. List the descriptions of items whose alternates cost more than $100.

12. List the names of people whose bills are more than twice the average bill.

13. What is the average credit limit of people who have bought item 19?

14. List the names of suppliers located in Boston as well as the names of suppliers who supply item 19.

15. List the item numbers of items supplied by Warmth, Inc that were charged by account 107.

16. List the account numbers of accounts with credit rating of 10 that have not charged item 19.

17. List the departments in which the average quota is more than 750 items.

18. For each department list the department name and the average quota for that department.

19. List the suppliers and the total number of items on order for suppliers whose largest order is over 100 items.

20. List the names of suppliers that have orders for over 1000 items each of which costs $10.00 or more.

21. List the suppliers that supply all the items sold by the camping department.

22. List the departments that sell only items supplied by Warmth, Inc.

23. List the names of people and the item numbers of items they have charged.

24. List the descriptions of items that need to be reordered and the names of suppliers that supply them.

25. List each item description and the description of its alternate for items that cost more than their alternates.

26. List the departments located on the third floor that sell more than 100 <u>different</u> items.

27. List the names of suppliers and the average <u>retail</u> price of items they supply.

28. Due to inflation the retail price of all items that need to be reordered will be doubled. Create a table containing the item description and new retail price.

29. List the account numbers with the highest total bill in each credit rating category. (List the account numbers and the ratings.)

30. Find the item numbers of items with a retail price over $1000 that are sold by the camping department and charged by account number 109.

RETENTION

1. List the names of charge account customers with a credit rating of 10.

2. Which items need to be reordered? List the item numbers.

3. Which items with more than 3000 on hand have item 8 as their alternate? (List the item numbers.)

4. List the account numbers with credit rating less than 3 that have exceeded their credit limit and were referred by account 108.

5. Due to inflation the retail price of each item will be doubled. List the item number and the new price.

6. List the account numbers of people who would still be over their credit limit if their limit were doubled.

7. How many items have a retail price less than $5.00?

8. Which account or accounts have the largest total bill? (List the account number.)

9. List the names of people who have charged item 19.

10. On which floor is the department that sells hiking boots located?

11. List the names of people who were referred by accounts with credit ratings of 8 or higher.

12. List the descriptions of items whose retail price is more than twice the average retail price.

13. What is the average retail price of items whose retail price is more than twice the average retail price.

13. What is the average retail price of items sold by the camping department?

14. List the item numbers of items charged by account 107 as well as the item numbers of items supplied by Warmth, Inc.

15. List the account numbers of accounts with credit rating of 10 who have charged item 19.

16. List the names of suppliers located in Boston that do not supply item 19.

17. List the suppliers whose average wholesale price is over $500.

18. For each supplier list the supplier name and the average wholesale price of the items he supplied.

19. List the departments and their total quotas for departments whose highest quota is over 1000 items.

20. List the ratings for which the average bill for women having that rating is over $3000.

21. List the departments that sell all the items supplied by Warmth, Inc.

22. List the accounts that have charged only those items that the camping department sells.

23. List the supplier names and the descriptions of the items they supply.

24. List the names of people who have exceeded their credit limit and the item numbers of items they have charged.

25. List the name of each account holder and the name of the person who referred him for account holders with higher credit ratings than the account they were referred by.

26. List the customers with credit rating of 7 who have charged more than 100 <u>different</u> items. (List the names.)

27. List the names of customers and the average price per item of items they charged.

28. For Christmas the credit limit of people who have exceeded their limit will be doubled. Create a table containing their names and new credit limits.

29. List the item numbers of items with the highest wholesale price supplied by each supplier. (List the item numbers and the supplier names.)

30. List the departments located on the third floor that sell items that need to be reordered.

REFERENCES

Andree, R.V. (1967). Computer Programming and Related Mathematics, John Wiley and Sons, Inc., New York.

Astrahan, M.M., and Chamberlin, D.D. (1975). Implementation of a structured English query language, CACM Vol. 18, No. 10, October, pp. 580-588.

Boyce, R.F., Chamberlin, D.D., King, W.F., and Hammer, M.M. (1975). Specifying queries as relational expressions: The SQUARE data sublanguage, CACM, November, pp. 621-628.

Chamberlin, D.D., Astrahan, M.M., Eswaran, K.P., Griffiths, P.P., Lorie, R.A., Mehl, J.W., Reisner, P., and Wade, B.W. (1976). SEQUEL 2: A unified approach to data definition, manipulation, and control, IBM Journal of Research and Development, Vol. 20, No. 6, November, pp. 560-575.

Chamberlin, D.D., and Boyce, R.F. (1974). SEQUEL: A structured English query language, Proc. 1974 ACM SIGFIDET Workshop, Ann Arbor, Michigan, April, pp. 249-264.

Codd, E.F. (1970). A relational model of data for large, shared data banks, CACM, Vol. 13, June, pp. 377-397.

Codd, E.F. (1971a). A database sublanguage founded on the relational calculus, Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, pp. 35-68.

Codd, E.F. (1971b). Relational completeness of database sublanguages, Courant Computer Science Symposia, Vol. 6: Data Base Systems, Prentice-Hall, New York, pp. 65-98.

Date, C.J. (1977). An Introduction to Database Systems, Second edition, Addison-Wesley.

Denny, G.H. (1977). An introduction to SQL, a structured query language, RA 93, May, IBM Research, San Jose, California.

Furtado, A.L., and Kerschberg, L. (1977). An algebra of quotient relations, IFSM T.R. No. 14, February, Department of Information Systems Management, University of Maryland, College Park, Maryland.

Gannon, J.D., and Horning, J.J. (1975). The impact of language design on the production of reliable software, IEEE Trans. on Reliable Software, Vol. 1, No. 2, pp. 10-22.

Gould, J.D. (1975). Some psychological evidence on how people debug computer programs, International Journal of Man-Machine Studies, Vol. 7., pp. 151-182.

Gould, J.D., and Ascher, R. (1975). Use of a IQF-like query language by non-programmers, RC 5279, IBM Watson Research Center, Yorktown Heights, New York.

Gould, J.D., and Boies, S.J. (1974). Syntactic errors in computer programming, Human Factors, 16(3), pp. 253-257.

Gould, J.D., and Drongowski, P. (1974). An exploratory study of computer program debugging, Human Factors, 16(30), pp. 258-277.

Halstead, M.H. (1977). Elements of Software Science, Elsevier North-Holland, Inc., New York.

IBM (1972). Interactive query facility user's guide, GH-1223.

Lochovsky, F.H. (1978). Data base management system user performance, Technical Report CSRG-90, April, Computer Systems Research Group, University of Toronto.

Love, T. (1977). An experimental investigation of the effect of program sturcture on program understanding, Proc. ACM Conference on Language Design and Reliable Software, SIGPLAN notices, Vol, 12, No. 3, March.

Miller, L.A. (1974). Programming by non-programmers, International Journal of Man-Machine Studies, Vol. 6, pp. 237-260.

Miller, L.A., and Becker, C.A. (1974). Programming in natural English, RC 5137, November, Watson Research Center, Yorktown Heights, New York.

Miller, L.A., and Thomas, J.C. (1977). Behavioral issues in the use of interactive systems, International Journal of Man-Machine Studies, Vol. 9, pp. 509-536.

Reisner, P. (1976). Use of psychological experimentaiton as an aid to development of a query language, IEEE, Trans. on Software Engineering SE-3, 3(1977), pp. 218-229.

Reisner, P., Boyce, R.F., and Chamberlin, D.D. (1975). Human factors evaluation of two data base query languages -- SQUARE and SEQUEL, Proc. AFIPS 1975 NCC, Vol. 44, AFIPS Press, Montvale, New Jersey, pp. 447-452.

Seymour, W. (1978). Diary of a human factors experiment, TR 77-14, January, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts.

Shneiderman, B. (1976a). Exploratory experiments in programmer behavior, International Journal of Computer and Information Science, Vol. 5, pp. 123-143.

Shneiderman, B. (1976b). Human factors experiments in programming: Motivation, methodology and research directions, ISM TR No. 9, September, Department of Information Systems Management, University of Maryland, College Park, Maryland.

Shneiderman, B. (1977). Measuring computer program quality and comprehension, International Journal of Man-Machine Studies, Vol. 9, pp. 1-13.

Shneiderman, B. (1978). Improving the human factors aspect of data-base interactions, ACM TODS, Vol. 3, No. 4, December, pp. 417-439.

Shneiderman, B., and Mayer, R. (1975). Towards a cognitive model of programmer behavior, TR No, 37, August, Computer Science Department, Indiana University, Bloomington, Indiana.

Shneiderman, B., Mayer, R., Mckay, D., and Heller, P. (1975). Experimental investigations of the utility of detailed flow charts in programming, CACM Vol. 20, No. 6, June, 1977, pp. 373-381.

Sime, M.E., Green, R.R.G., and Guest, D.J. (1973). Psychological evaluation of two conditional constructions used in computer languages, International Journal of Man-Machine Studies, Vol. 5, pp. 105-113.

Stemple, D.W., Becker, M., Welty, C., and Mayfield, W. (1978). TABLET: The algebra based language for enquiring of tables, TR 78-19, November, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts.

Stonebraker, M., and Rowe, L.A. (1977). Observations on data manipulations languages and their embedding in general purpose programming languages, Memo No. UCB/ERL M77/53, July, Electronic Research Lab, University of California, Berkeley, California.

Thomas, J.C., and Gould, J.D. (1975). A psychological study of query by example, AFIPS, Vol. 44, National Computer Conference, pp. 439-445.

Weinberg, G.M., and Schulman, E.L. (1974). Goals and performance in computer programming, Human Factors, 16(1), pp. 70-77.

Weissman, L. (1974). Psychological complexity of computer programs: An experimental methodology, SIGPLAN notices, June, pp. 25-35.

Welty, C. (1979). A comparison of a procedural and a nonprocedural query language: Syntactic metrics and human factors. COINS TR 79-9, University of Massachusetts, Amherst, Massachusetts.

Youngs, E.A. (1974). Human errors in programming, International Journal of Man-Machine Studies, Vol. 6, pp. 361-376.

Zloof, M.M. (1975). Query by example, Proc. 1975 NCC AFIPS, Vol. 44, pp. 431-438.