A COMPARISON OF A PROCEDURAL AND A NONPROCEDURAL QUERY LANGUAGE: SYNTACTIC METRICS AND HUMAN FACTORS

CHARLES WELTY

COINS Technical Report 79-9

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

NSF Grant MCS78-07616

ACKNOWLEDGEMENTS

Professor David Stemple provided many of the ideas and much of the motivation for this research. His eclectic background and interests -- spanning computer science, psychology and philosophy (to name a few) -- added both breadth and depth to this work. He was patient when I was discouraged and always available when I needed help.

Professor Charles Clifton of the Psychology Department at the University of Massachusetts, Amherst, was of tremendous help in the planning of the experiment and the statistical analysis of the results. His interest, insightful recommendations and tactful, constructive criticism made it a pleasure to work with him.

Dr. Phyllis Reisner of IBM Research, San Jose, California, advised and encouraged me throughout my research. I spent a week working with her in San Jose at the very beginning of my research. The ideas and encouragement she imparted to me during that week have been invaluable. She has always been willing to discuss specific experimental and statistical problems by telephone through the course of my research. I would also like to thank the management of IBM Research for making a person of such competence and worth as Dr. Reisner available to the academic community, in general, and me, in particular.

The statistical consultants, especially Ms. Trina Hosmer and Ms. Harriet Peterson, of the University Computing Center were a great help.

Professor Joseph Sardinas of the Accounting Department at the University of Massachusetts, Amherst, sponsored Accounting 397A, the

course that was used to present the query languages. His support was most helpful.

The students taking Accounting 397A, hereafter referred to as "subjects", were as conscientious and intelligent as could be wished.

Ms. Patricia Driscoll prepared all the experimental materials. She entered the manuals, quizzes and tests into a text editor. Her good humor and willingness to work long hours are much appreciated.

Many thanks also to Ms. Donna Jerkovich for typing this dissertation.

ABSTRACT

A Comparison of a Procedural and a Nonprocedural Query Language:
Syntactic Metrics and Human Factors

May 1979

Charles Welty, B.S., M.S., University of California, Berkeley,

M.S., Ph.D., University of Massachusetts, Amherst

Directed by: Dr. David Stemple

Much controversy revolves around the question of what elements are necessary for effective interfaces between casual users and computer databases. One such controversy concerns the use of nonprocedural or procedural query languages. In order to shed light on this area, an experiment testing the ability of subjects to write difficult queries correctly in a procedural and a nonprocedural language has been run. Great pains were taken to isolate the independent variable -- procedurality -- in the experiment. The subjects were taught using manuals for the two languages that contained identical examples and problems in identical order. Constructs in the languages that do not affect their procedurality are identical. Both languages use the relational model of data and have similar Halstead levels.

The results of the experiment show that subjects write difficult queries significantly better using the procedural language than subjects using the nonprocedural language do. The results of the experiment are also used to compare corresponding constructs in the two languages and to recommend improvements for these constructs.

TABLE OF CONTENTS

Appendices

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Introduction

A database management system supplies services to different types of users. One such service, thought to have the potential for explosive growth in this century, is a query language facility [Codd and Date, 1974].

A query language is an interactive language used by non-programmers to access information stored in a database. Much of query language research is oriented towards helping the "casual user" -- the person who may want to access data occasionally, but only as an ancillary part of his/her job or life.

An effective query language interface should insulate the casual user from the complexity of the underlying storage and access mechanisms of the database management system. Still, the user must interact in some way with the database system. Thus, the nature of the query language interface must be studied to discover the kind of facility that will allow "casual" use of the database and, yet, give the database management facility the parameters it needs to access the desired data. An effective query language must be learnable and easily remembered since casual use implies intermittent use.

It is not enough to simply state that a piece of software is human-oriented. The software must be tested using subjects from the proposed user population. This thesis presents an experiment testing

query languages which exemplify two different approaches to query writing -- the nonprocedural and procedural approaches.

The next section presents a brief history of the problems that have given rise to the use of human subjects experimentation in the development of software systems.

1.2 Human Factors

In the late 1940's and most of the 1950's computers were arcane pieces of machinery with a small community of users. The number of users was kept small by the scarcity of machines, the complexity of machine languages, and the requirement that the user enter his programs manually into the machine. As the general utility of computers became apparent, everything changed: computers became commonplace; high level languages evolved; operating systems were developed and the user community burgeoned.

The early computers did not compromise with their users. Users solved problems using the terminology of their own fields, translated the solution to a machine language, and operated the machine themselves. The evolution of programming languages and other software systems has resulted in many systems that conform more to the user than to the machine. General purpose programming languages conform to the needs of programmers. APL, SNOBOL, COBOL, PASCAL, SIMULA, LISP and FORTRAN span a wide range of programmer needs. Special purpose languages, on the other hand, serve non-programming users in specific problem domains -- statistics, engineering, natural science, data retrieval, etc. Though users are being served better by these languages, problems remain.

Programmers have encountered problems in the writing and maintaining of reliable software. Many of these problems seem to be inherent in the languages used by programmers. Other problems derive from the methods used by the individual programmers and groups of programmers. A variety

of programming languages, programming practices and organizations of programming groups have been proposed to ameliorate this situation. Structured programming is an umbrella term for a number of these proposals.

Structured programming concepts have led to the realization that the prime element in computer systems is 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 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; Boies and Gould, 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, 1976b, 1977; 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. The next section presents motivation for human factors research into query languages as well as a comprehensive presentation of previous work.

1.3 Human Factors Research into Query Languages

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 quickly solved, 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 structures. 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 possess none of the above attributes but their study has yielded useful experimental results.

These reasons for human factors testing of query languages are over and above the basic one: a query language that is difficult to use will not be used. A query language should be an aid to a person

1976]. Thus, Reisner's experiment both found the language to be learnable and contributed to its evolution.

Both the Reisner and the Gould and Ascher studies use languages that employ an English-like syntax. Zloof's Query By Example [Zloof, 1974] uses a graphic format that eliminates much of the syntax (and, thus, the time to learn the syntax as well as syntax-related errors) from the user's view. Query By Example was tested by Thomas and Gould [Thomas and Gould, 1975] using 39 test subjects. The subjects received less than 3 hours of instruction, wrote queries in an average of 1.6 minutes and wrote 67% of their queries correctly.

This language was shown to have excellent characteristics. It does require the use of a more complex terminal than the languages presented earlier. No experimental comparison of hard copy terminals and graphic terminals has been done.

The preceding papers have all studied restricted syntax query languages. Another major field of enquiry concerns the use of natural English for casual users.

Miller and Becker [Miller and Becker, 1974] used 14 subjects to solve problems concerning hard copy files maintained by a hypothetical company. The subjects were to type directives to a computer with no restriction on the form of the directives.

The subjects were found to:

1. Start typing the directives almost immediately.
2. Work with an entire file, rather than working record-at-a-time and, thus, did not use transfer

of control directives.

3. Generate intermediate files to be used later.
4. Use context to specify operands.
5. Use simple algorithms for the most part.
   Subjects using simple algorithms made more mistakes than the few subjects that used more complex algorithms.

This work was done with programming languages in mind but the problem is certainly a query language problem and the results are directly applicable to query language research.

Thomas looks at universal quantification in [Thomas, 1976]. His main conclusion is that people have difficulty with universal quantification. People have difficulty understanding the meaning of a statement containing a universal quantifier. They also have difficulty in expressing the relationships shown by Venn diagrams representing relationships requiring universal quantification. Also, given the chance to specify questions to solve a problem, few (7 of 185 questions) involved the explicit use of universal quantification. It seems that people do not understand and do not use universal quantification in natural English.

The query languages associated with various data models (hierarchical, network and relational) were studied by Lochovsky [Lochovsky and Tsichritzis, 1977; Lochovsky, 1978]. The relational language was found to be significantly better as measured by user performance. This may be due to the complexity of the hierarchical and network languages as well as other factors such as subject background. Many

suggestions are made for improving the languages used for each of the models.

A controversial subject in programming languages in general and query languages in particular concerns whether procedural or nonprocedural (descriptive) languages are easier for people to use. Gould [Gould, Lewis and Becker, 1976] found that subjects tended to use procedural protocols. Procedural protocols were less often ambiguous than descriptive protocols.

The same experiment investigated the use of restricted-syntax English and natural English. The restricted-syntax English protocols were less ambiguous than the natural English protocols. Also, natural English protocols written by subjects after they had written restricted-syntax protocols were less ambiguous than natural English protocols written before the subjects were exposed to the restricted-syntax.

Another study [Small and Weldon, 1978] of natural English and restricted-syntax English used natural English and a subset of SEQUEL. Twenty subjects composed queries in both notations with counter balanced orderings (one group used SEQUEL first followed by natural English, the other vice versa). Subjects using SEQUEL in the second half of the experiment outperformed all other groups. Also, subjects wrote queries faster when using SEQUEL than they did in English with no difference in query accuracy.

A paper by Shneiderman [Shneiderman, 1978] ends with an experiment that illustrates the research methods and research issues presented in the rest of the paper. The experiment will be treated here and the remainder of the paper treated in the next section.

The experiment studied natural language and a subset of SEQUEL. Each of the 22 subjects did both the natural language and SEQUEL part of the experiment with the orderings counterbalanced. The main part of the experiment was a situation problem for which the subjects had to formulate queries of their own. The number of invalid queries differed significantly between the two treatments with the SEQUEL treatment having fewer. Also, the group that used SEQUEL first had significantly fewer invalid queries than the group using natural English first.

1.3.2 Articles containing no experimental data. Shneiderman [Shneiderman, 1978] provides classifications and exposition of the primary issues in the human factors aspect of database interaction. The paper discusses the functions users wish to perform with databases and the tasks the user must accomplish to perform these operations. In addition, various interaction modes (host language embedding, self contained language, natural language, etc.) and retrieval response type are categorized. Query features are classified from the most basic -- simple mapping -- to the most complex -- universal quantification. Users are classified into three groupings:

1. Non-trained intermittent users.
2. Skilled, frequent user.
3. Professional database users.

Finally, topics directly concerned with human factors research are presented. First, research methods are categorized into introspection, field studies and controlled experimentation, with controlled experimen-

tation being the method of choice. Measurement of query composition, comprehension, debugging and modification are discussed. The last section deals with major research issues that are ready for experimental study, these include:

1. Natural language vs. artificial language.
2. Specification (nonprocedural) vs. procedural languages.
3. Linear keyword vs. two dimension positional languages.
4. Data model selection.
5. Hardware factors.

Thomas [Thomas, 1977] is concerned with both hardware and language features involved in database interaction and reports results of many studies. Included in the references are many articles from the behavioral sciences.

The papers presented in this and the previous section deal with human factors experiments and conjecture based on such experiments. Often, decisions are made that are not based on experimentation but on the opinions of database experts. Codd [Codd, 1977] points out the supericrity of descriptive (nonprocedural) over constructive (procedural) query languages for most users. Date [Date, 1977] agrees with the superiority of the descriptive approach. Codd and Date [Codd and Date, 1974; Date and Codd, 1974] also analyze the differences between the relational and network models of data and find the relational model superior. All the arguments are strong and well-founded, but need testing.

1.4 Outline of Thesis

The previous sections have presented some of the major topics for human factors research. This thesis will explore the subject of non-procedural (descriptive) and procedural (constructive) query languages. This subject has given rise to a good deal of comment both in query languages and in general purpose programming languages.

Chapter 2 of the thesis will describe the basic hypothesis and the experiment used to test this hypothesis using two query languages -- one nonprocedural and one procedural language.

Chapter 3 describes the details of the languages -- their individual constructs, their similarities and their differences. Measures of their properties are also presented.

In Chapter 4 the results of the experiment are presented, statistically analyzed and interpreted. Also, details of the types of errors made in each language are discussed. All the results of the experiment are then used to recommend changes to the languages.

Chapter 5 presents the conclusions drawn from the experiment. These conclusions include the contribution the experiment makes to language evolution, recommendations for the use of the languages, a discussion of the acceptance or rejection of the procedural/nonprocedural hypothesis and directions for future research.

C H A P T E R I I

THE EXPERIMENT

2.1 Hypothesis

One of the major issues in database query languages concerns the procedurality of query languages. As discussed in the last chapter, nonprocedural languages are thought in many ways superior to procedural languages by some database experts [Codd, 1977; Date, 1977]. There seem to be no published reports directly supporting procedural query languages. This thesis presents an experiment to test a nonprocedural query language and a procedural query language using human subjects.

The experiment 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 must be similar in all respects except in the independent variable -- procedurality -- since the experiment takes the reductionist approach [Shneiderman, 1978] -- all variables except procedurality are held constant.

SQL [Appendix A; Denny, 1977] and TABLET [Appendix B; 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 Chapter 3.

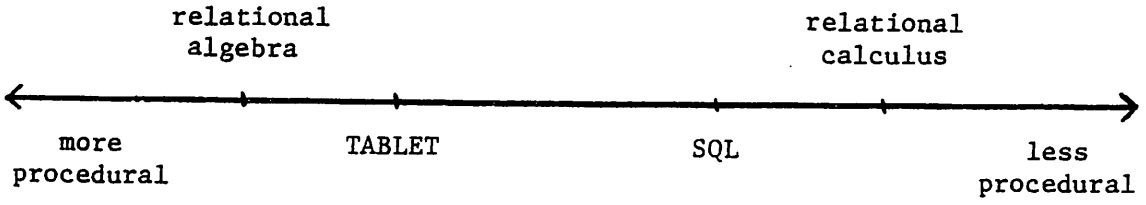
The difference, again, is in the procedurality of the languages. A thorough treatment of procedurality and the procedurality of SQL and TABLET is given in Chapter 3. Generally, a language is procedural if it specifies a step-by-step method for achieving a result. Nonprocedural languages describe the 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 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].

Perhaps a drawing of the procedurality continuum will show the relationship between the languages:



An example will make the SQL and TABLET differences more explicit. Use the following database:

EMP(NAME, DNO, SAL)

DEPT(DNO, LOC)

EMP is a relation contianing employee information: column NAME contains the employee name, column DNO contains his department number and SAL contains his salary. DEPT is a relation containing information about departments: column DNO contains the department number and LOC contains the location of the department.

```
SQL:     SELECT NAME
         FROM EMP
         WHERE DNO =
           SELECT DNO
           FROM DEPT
           WHERE LOC = 'BOSTON'

TABLET:  FORM BOSTEMPS FROM NAME, DNO OF EMP
         ADD COLUMN LOC OF DEPT BY DNO
         KEEP ROWS WHERE LOC = 'BOSTON'
         PRINT NAME
```

The SQL query does not use algebraic operations on tables. It describes the data it wants. Every line of the TABLET query performs an operation on a table and yields a transformed table. The TABLET query specifies

the ordered steps yielding the desired information. Chapter 3 contains more about the two languages.

The experiment was set up to test the overall hypothesis presented above and to test sub-hypotheses about the individual constructs used in each language. This testing will aid in the evolution of the languages by finding the strong and weak aspects of each. The weak points will be analyzed and recommendations will be made for improving the languages.

2.2 Experimental Method

The basic idea of this experiment was to test human subjects using a nonprocedural query language (SQL) and a procedural query language (TABLET). The treatments given the subjects differ only as the procedurality of the languages differ.

The two languages were taught to the subjects using uniform manuals (Appendices A and B). The manuals contained identical examples and problems presented in the same order. Each manual consisted of 12 lessons each, presenting analogous material. Everything independent of language differences was identical in the manuals. The SQL manual was prepared first, using the same order of presentation as used by Reisner [Reisner, 1976]. Since the author was involved with the design and implementation of TABLET, it was decided to let SQL determine the order of presentation of concepts in the manuals. After the SQL manual was prepared, it was cannibalized to generate the TABLET manual. Everything in the SQL manual that could remain unchanged was left alone. Only language details were changed.

The languages were presented in a 1 credit special topics course offered by the Accounting Department of the University of Massachusetts at Amherst. The course was advertised through mailings to all freshmen and sophomores in the School of Business Administration, an advertisement in the student paper, a paragraph in a pamphlet sent to all undergraduates and the course listing in the course schedule. The basic point in all the advertisement was that students could learn a simple computer language that would allow them to access information without becoming

computer experts. Most of the subjects learned of the course through the mailing. Some subjects learned of the course through the pamphlet and one student responded to the advertisement in the student newspaper.

Most of the 72 subjects that participated in the experiment were business undergraduates. Information about subject background can be found in Appendix D. The subjects were attracted to the course by the credit as well as by the opportunity to learn something about computers.

The subjects were divided into two groups. One group learned SQL and the other, TABLET. The subjects were also classified as inexperienced -- having no previous exposure to computers -- and experienced -- having taken a course in either BASIC or FORTRAN. The number of subjects in each classification was:

SQL inexperienced	17 subjects,
SQL experienced	18 subjects,
TABLET inexperienced	20 subjects,
TABLET experienced	17 subjects.

The experienced and inexperienced students attended the same classes. Only one experienced student asked questions pertaining to his previous language and his questions were privately discouraged, so the treatments were the same for the experienced and inexperienced subjects.

There were four classes, two in which SQL was taught and two in which TABLET was taught. They met as follows:

early Monday and Wednesday afternoon,

later Monday and Wednesday afternoon,

early Tuesday and Thursday afternoon,

later Tuesday and Thursday afternoon.

A toss of a fair coin determined that SQL would meet early Monday and Wednesday. The rest of the classes were scheduled to counterbalance the time difference. SQL met early Monday and Wednesday, but late on Tuesday and Thursday. TABLET met late Monday and Wednesday but early Tuesday and Thursday.

Subjects were randomly assigned to the classes. Each subject filled out a questionnaire giving his class preference and computer experience. The questionnaires were divided into two groups -- one containing only experienced and the other containing only inexperienced subjects. Subjects were randomly assigned to classes from the two groups. After this assignment was done we checked each class for people who could not attend at that time. These few people were assigned to classes that they could attend. Finally, we checked the number of people in each class that picked the class as their first choice and the number that did not choose it as their first choice. We randomly chose people from the classes to even out the number of first choice subjects in SQL classes and the number of first choice subjects in TABLET classes. This final redistribution effected only a few people.

Each section, SQL and TABLET, met twice a week for a total of 14 meetings. The first class meeting was used to hand out the manuals and Lesson 1 was read to the subjects. Subjects read their manuals at home and were quizzed at each of the next 12 meetings on the material they read. The last class meeting was a review before the final.

Subjects were required to take many tests [Appendix C] during this experiment. The 12 quizzes were used primarily as a teaching aid to

spot difficulties in learning the languages. The quizzes were identical in each class. Each quiz consisted of two to five English statements describing the data desired from a database (e.g., List the names of people making more than $10,000.). Subjects wrote the queries in their query language. All subjects were required to take all the quizzes. If a subject missed a quiz he was required to make it up before taking the next quiz. Quiz results were not statistically analyzed because subjects were allowed to compare their solutions to the correct ones before handing the solutions in. Thus, subjects had the opportunity to change their solutions. Before the review meeting, a comprehension test was given to test the subjects understanding of queries written in their language. The subjects were given queries written in the query language and asked to show which data from the samples in the database would be printed. The subjects also had to write an English statement of the purpose of the query. This was a take home quiz and was not statistically analyzed due to the possibility of cheating. Student grades were based totally on a final exam. The final was an open book exam containing 30 English statements from which the subjects wrote queries in their query language. The final was identical for the two classes. Five randomizations of problem ordering were made for the final. Last, but most important from the experimentor's point of view, was the retention test. The retention test was given 3 weeks after the final. This test followed the same format as the final but was a closed book test. The test did not affect the subjects' grades (thus removing any impetus to study) but was mandatory -- subjects would get a grade of "incomplete" if the retention

test was not taken. The purpose of the retention test was to test how well students retained their language after a period of non-use. Subjects were told their course grades after finishing the retention test.

The questions on the final and retention tests were carefully made up and matched. There was a one-to-one correspondence between the forms of the solutions on the exams. The questions were different and used different relations in the database but the forms of the solutions were identical. The test questions were grouped into two categories -- easy questions (from lessons 1-5) and hard questions (from lessons 6-12). The questions were also sub-grouped into queries using particular constructs in the languages. The results of the analyses of subject performance on these groups of problems is given in Chapter 4.

The final and the retention test were given in the evening. This allowed the SQL and TABLET classes to take the tests together.

Three relational databases [Appendix C] were used in the experiment. The COLLEGE database contained information about students, teachers and courses in a college and was used in the manual for all exposition, examples and problems. The quizzes used the MANUFACTURER database. The comprehension test, final and retention test all used the MAILORDER database. There were several reasons for using more than one database. First, a single database would have to be quite large to support both the teaching and the testing of the subjects. Second, in a real learning situation the user would have to transfer his understanding from the databases used in the course to his actual database. Changing databases gives the user practice. And third, each of the small databases

would seem to be about the size of a view used by a casual user. Each database contains 6 or 7 relations which is a realistic number.

Most class meetings consisted of question answering and quizzes. Before each quiz was given, questions were answered and major errors made in the previous quiz were explained. If a question was answered in, say, the SQL class -- the analogous point would be made in the TABLET class. Great pains were taken to be sure that the material presented at each class was uniform. It took an average of 13 minutes per class to answer questions. The subjects then took the quiz. As subjects finished their quizzes, they were given a copy of the solution to compare with their answers. The subjects kept the solutions and handed their papers in. The quizzes were not returned. Quizzes did not affect the subject's grade. The purpose of this was to discourage subjects from changing their solutions while looking at the actual solutions.

Randomly selected subjects from each class were interviewed after Lesson 6, after Lesson 11 and after the final. One group was chosen after Lesson 6, another after Lesson 11 and the final interviewees were chosen from these two groups. The interviews were recorded on audio cassettes. Each interview consisted of 6 quiz-like questions. The students wrote and talked through their solutions of each problem. The interviewer asked questions about the subject's method and opinions. Each interview lasted 30 to 45 minutes. These interviews were used to aid in interpreting the results of the experiment.

After Lesson 6 and Lesson 12 each subject filled out a questionnaire giving the amount of time spent studying each lesson and rating the difficulty of each lesson on a scale of 1 to 10 (1 being easy, 10 being hard).

The entire experiment was pre-tested on 8 students at a local college. Four were taught SQL and four were taught TABLET. No statistical data came from this test. The pre-test was used to find errors in the quizzes, manuals and exams. Also, at that time the final and retention each contained 43 questions. This was judged too long and the exams were reduced to 30 problems each. Due to this pre-test, the actual experiment ran quite smoothly.

2.3 Grading the Quizzes, the Final and the Retention Test

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 misspelled 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. For example, if the FROM clause is left out of a SQL query, the columns in the SELECT can be used to determine which relation is meant. This determination may be unambiguous in the database used but may be ambiguous in another database. The compiler could correct the error in this case but not in the general case.

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 query has been made,
incomplete (IN)	incomplete query,
unattempted (UN)	no solution was attempted.

The first four categories -- correct, ML, MO, MS -- were called essentially correct responses and the other five 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.

2.4 Experimental Design

The main hypothesis (section 2.1) concerned the ability of subjects to write queries in a procedural and a nonprocedural query language. We were also interested in the effect of experience level on performance. The essentially correct responses of subjects on hard problems on the retention test was analyzed. The retention test was used because it best reflected the performance of casual users in a steady-state environment. The statistical test used was a two-way analysis of variance using experience level (inexperienced, experienced) and query language (SQL, TABLET) as the independent variables and the number of essentially correct responses by each subject for hard problems on the retention test as the dependent variable.

In addition, the same two-way analysis of variance was run using essentially correct responses on the sub-categories corresponding to different language constructs of the dependent variable. These sub-categories are not independent so the analysis of variance for all the categories cannot be run simultaneously.

A one-way analysis of variance was run to provide the group contrasts:

experienced SQL subjects vs. experienced TABLET subjects,

inexperienced SQL subjects vs. inexperienced TABLET subjects,

inexperienced SQL subjects vs. experienced SQL subjects,

inexperienced TABLET subjects vs. experienced TABLET subjects.

The results of these statistical tests are presented in Chapter 4.

CHAPTER III

THE LANGUAGES

3.1 The Relational Model of Data

Relational query languages are based on Codd's relational data model. A relation, R, on sets D1, D2, . . ., Dn (not necessarily unique) is defined as a set of ordered n-tuples <d1, d2, . . ., dn> with d1 $\in$ D1, d2 $\in$ D2, . . ., dn $\in$ Dn. The degree of R is n. D1, D2, . . ., Dn are the domains of R. The cardinality of R is the number of n-tuples in R. A sample database is given in Table 3.1.

Table 3.1 is the COLLEGE database used in the SQL and TABLET manuals. Table 3.1a shows the 6 relations comprising the database and Table 3.1b explains the relations. Subjects in the human factors experiment were given exactly the information found in Table 3.1.

The other major data models are the network model, exemplified by DBTG [CODASYL, 1971], and the hierarchical model [Tsichritzis and Lochovsky, 1976]. Many non-experimental studies have compared the various models [Michaels, Mittman and Carlson, 1976; Codd and Date, 1974; Date and Codd, 1974; Date, 1977]. Limited experimental studies of data models have also been run [Lochovsky, 1978]. In addition, McGee [McGee, 1976] gives criteria for the selection of a data model. Unfortunately, due to the wide spectrum of user types and applications, none of these papers can make an unequivocal statement about the superiority of any data model.

The relational model was chosen for this experiment because it has qualities that make it ideal for the casual user. It has one, simple data structure -- the relation. In addition, relational query languages do not require transfer of control statements, a problematic aspect of many languages. Finally, most of the research listed above points to the use of the relational model by casual users.

Database - 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

Table 3.1a

The COLLEGE Database

Most of the table and column meanings are probably obvious to you. Here is a bit more explanation:

STUDENT - Contains information about students. There is one row in this table for each student in the school. The REPID column contains the ID of each student's representative on the student senate. The representative is also a student.

TAKING - Contains the courses each student is taking this semester. There is one row for each course each student is taking.

DEPARTMENT - Contains information about the academic departments in the college. The HEAD column contains the ID of the department head.

FACULTY - Contains information about faculty members. There is one row for each faculty member. Each faculty member must be on a college committee. The COMHEAD column contains the ID of the head of a faculty member's committee. The committee head is also a faculty member.

TEACHING - Contains information about the courses being taught this semester. There is one row in this table for each section of each course taught this semester. LIMIT contains the limit on enrollment for a section, SIZE contains the actual size of the class.

COURSES - Contains information about all courses in the college, whether or not they are being taught this semester. There is one row for each course.

Table 3.1b

Further Explanation of the Relations in the COLLEGE Database

3.2 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 to be performed on a relation. Some sample queries follow. These queries all use the COLLEGE database of Table 3.1.

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 (TABLET works on tables (i.e., collections of tuples with duplicates allowed), not relations for reasons to be given in Section 3.4) 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 explanation of the TABLET commands in the previous example was incomplete in one detail. The complete TABLET query for Q1 is:

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

OHIOANS is a user table as opposed to STUDENT which is a database table. When OHIOANS is formed it is called the default user table. If no table is specified in subsequent commands, OHIOANS will be used until a new FORM statement is encountered, when the newly formed table will be the default user table. Each TABLET command transforms its user table, default or explicit. A user table is ready for printing when its various transformations result in the desired information. The PRINT command then prints the desired information.

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.

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 COURSE 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. In TABLET, the ADD COLUMN statement joins [Codd, 1971b] 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.

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.

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.

Q6. 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.

Q7. 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 illustrated by both the 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.

Q8. List the names of faculty members and the names of their committee heads, for faculty members who make more than their committee heads.

```
SQL:     SELECT A.NAME, B.NAME
         FROM FACULTY A, FACULTY B
         WHERE A.COMHEAD = B.ID
              AND A.SALARY < B.SALARY

TABLET:  FORM HIGHSAL FROM NAME, SALARY, COMHEAD OF FACULTY
         ADD COLUMN NAME (AS HEADNAME), SALARY (AS HEADSAL)
              OF FACULTY BY COMHEAD = ID
         KEEP ROWS WHERE SALARY > HEADSAL
         PRINT NAME, HEADNAME
```

In the SQL query, A and B are tuple variables that range over the FACULTY table. In effect, the FACULTY table is being joined with itself. The TABLET query first forms a new table, HIGHSAL, from FACULTY and then adds columns from the FACULTY table. The new columns must be renamed tc avoid name conflicts in the HIGHSAL table.

Appendices A and B contain more examples of SQL and TABLET.

3.3 Language Similarities

SQL and TABLET have many points of similarity, as first discussed in Chapter 2. These are:

1. They both use the relational model of data.
2. They are relationally complete.
3. They have similar language levels as measured by Halstead.
4. Their syntax is a function of their procedurality. Constructs that are independent of procedurality are identical.
5. They use the same terminal equipment.

3.3.1 The relational model. This subject has already been discussed in Section 3.1.

3.3.2 Relational completeness. A relational query language is defined to be relationally complete if it has at least the selection capability of the relational calculus [Codd, 1971b]. The relational completeness of SQL is shown in [Chamberlin and Boyce, 1974]. To show the relational completeness of TABLET it is only necessary to show that TABLET has the equivalent of all the operators found in the relational algebra [Codd, 1971b]: union, intersection, set difference, projection, generalized (theta) join, division and restriction. See Table 3.3.2.1 for the algebraic operators. The rest of this section shows the correspondence between TABLET commands and the operators of the relational algebra. The tables and column tuples defined in Table 3.3.2.1 are used throughout this section.

Operator Name	Example
UNION	R ∪ S
INTERSECTION	R ∩ S
MINUS (set difference)	R - S
PROJECTION	T[A]
JOIN	U[C θ D]V
DIVISION	U[C ÷ D]V
RESTRICTION	W[E θ F]

Where:

R, S, T, U, V and W are relations.

B, C, D, E, F, G and H are each a tuple of column names (e.g., <ID, NAME, SEX, HOMESTATE, MAJOR, REPID>).

Relations R and S are union compatible, i.e., they are of the same degree and corresponding columns are defined on the same domain.

Relations R and S consist of columns G and H, respectively. The columns of R and S occur in the order defined by G and H.

A is a renaming tuple. A is an n-tuple of the form, <a1 AS b1, . . . , an AS bn> where ai is the name of a column in relation T. The bi elements are unique (i.e., if bi = bj, then i = j).

Relation U consists of at least one more column than found in C (i.e., if C is an n-tuple, then U consists of at least n+1 columns).

Relation V consists of at least columns D (i.e., if D is an n-tuple, then V consists of at least n columns).

Relations U and V have no column names in common.

Columns C and D are defined on corresponding domains.

Relation W contains columns E and F.

Columns E and F are defined on corresponding domains.

$\theta \in \{ =, <, >, \neq, \geq, \leq \}$

C and D are of the same degree. E and F are also of the same degree.

Table 3.3.2.1

Relational Algebraic Operators and Symbols

Union is implemented using the ADD ROWS command. One table, R, is generated, then a second, S. Finally the two tables are combined with the ADD ROWS,

ADD UNIQUE ROWS OF R TO S

is the equivalent of

$S \leftarrow R \cup S$.

Intersect and minus are implemented using the KEEP ROWS command;

KEEP ROWS OF S WHERE H IN G OF R

is the equivalent of

$S \leftarrow S \cap R$.

The command

KEEP ROWS OF S WHERE H NOT IN G OF R

is the equivalent of

$S \leftarrow S - R$.

Projection is implemented by the FORM command. The command

FORM UNIQUE T1 FROM A OF T

is the equivalent of

$T1 \leftarrow T[A]$.

The result of both the TABLET and the algebraic projection is that the column name tuple for T1 is <b1, . . ., bn>, i.e., the columns are renamed.

Join is implemented using the ADD COLUMNS command. The command

ADD TO U ALL COLUMNS OF V BY C=D

implements

$U \leftarrow U[C=D]V$.

The operators $\neq$, $>$, $<$, $\geq$, or $\leq$ could be used instead of = in the algebraic join to make a generalized, or θ, join. Only the equijoin, join using =, is currently implemented in TABLET. Equijoin is the most commonly used of the possible joins.

Relational division is implemented using the GROUP commands to partition a relation. In the following column set, J is the tuple of all column names in U that are not in C. The commands

GROUP U BY J

KEEP GROUPS OF U WHERE C CONTAINS D OF V

UNGROUP U

FORM UNIQUE U1 FROM J OF U

are the equivalent of

$$U1 \leftarrow U[C \div D]V.$$

The KEEP ROWS command implements restriction. The command

KEEP ROWS OF W WHERE E θ F

is the equivalent of

$$W \leftarrow W[E \theta F].$$

Table 3.3.2.2 outlines the correspondence between the relational algebra and TABLET.

3.3.3 Halstead levels. Halstead [Halstead, 1977] has proposed measures of basic software properties. Central to these measures is the measurement of program volume, V. This is a metric for the size of an implementation of an algorithm and is defined as

$$V = N \log_2 n.$$

Algebra	TABLET
$S \leftarrow R \cup S$	ADD UNIQUE ROWS OF K TO S
$S \leftarrow R \cap S$	KEEP ROWS OF S WHERE H IN G OF R
$S \leftarrow R - S$	KEEP ROWS OF S WHERE H NOT IN G OF R
$T1 \leftarrow T[A]$	FORM UNIQUE T1 FROM A OF T
$U \leftarrow U[C\ \theta\ D]V$	ADD TO U ALL COLUMNS OF V BY C θ D
$U1 \leftarrow U[C \div D]V$	GROUP U BY J KEEP GROUPS WHERE C CONTAINS D OF V UNGROUP U FORM UNIQUE U1 FROM J OF U
$W \leftarrow W[E\ \theta\ F]$	KEEP ROWS OF W WHERE E θ F

Table 3.3.2.2

Correspondence Between Relational Algebraic Operations and TABLET Commands

The definitions and formulas for the Halstead measures are in Table 3.3.3.1. The $\log_2 n$ factor is the minimum number of bits needed to represent all the unique elements of the implementation. When multiplied by N, the result is the total number bits needed to represent the implementation.

Any particular volume measure is for an implementation of an algorithm in a particular language. If the same algorithm is implemented in a more succinct language, the volume will be reduced.

The potential volume, V*, is the volume of the implementation in a language in which the algorithm exists as a subroutine or procedure. The definition of V* is

$$V^* = (2+N2^*) \log_2 (2+N2^*)$$

where N2* is the number of different input/output parameters.

The program level, L, is a measure of the level of implementation of an algorithm. An algorithm written in a "higher-level language" [Stemple, 1977] will have a larger L value than that for the same algorithm written in a language of lower level. L is defined as

$$L = V^*/V.$$

Since V* is the smallest volume on implementation, L has values between 0 (exclusive) and 1 (inclusive). An approximation to L is

$$\hat{L} = (2/n1)(n2/N2).$$

Most of Halstead's work uses $\hat{L}$, not L.

The other Halstead metrics are given in Table 3.3.3.1. They will be explained in more detail as they are needed.

SYMBOLS

D	Difficulty (1/L)	V*	Potential volume
E	Effort	V**	Boundary volume
f	Frequency	b	Block size (Greek beta)
I	Intelligence content	n	Vocabulary size (Greek eta)
k	Redundancy factor	n1	Unique operator count
L	Program level	n2	Unique operand count
M	Number of modules	n*	Potential vocabulary
N	Program length	n1*	Potential operator count
N1	Total operators	n2*	Potential operand count
N2	Total operands	λ	Language level (Greek lambda)
T	Implementation time	v	Branch count (Greek nu)
V	Program volume		

EQUATIONS

$$n = n1 + n2$$

$$N = N1 + N2$$

$$n = kn'$$

$$n^* = n1^* + n2^* = 2 + n2^*$$

$$N = n1 \log_2 n1 + n2 \log_2 n2$$

$$V = N \log_2 n$$

$$V^* = n^* \log_2 n^*$$

$$L = V^* / V$$

$$\hat{L} = \frac{n1}{n1}\frac{n2}{N2} = \frac{2}{n1}\frac{n2}{N2}$$

$$I = LV = V^*$$

$$\lambda = LV^* = L^2 V$$

$$E = V / L = V^2 / V^*$$

$$T = E / S$$

Table 3.3.3.1

Symbols and Equations Used in Halstead Measures

The program level estimate, L, and program volume, V, were used by Halstead [Halstead, 1974] to compare the implementation of an algorithm in DSL ALPHA [Codd, 1971c], a language based on the relational calculus, with the COBOL DBTG implementation of the same algorithm. This algorithm can also be implemented in SQL and TABLET. For comparison we will also implement it in a relational algebraic language [Date, 1977].

The query used the following database:

PERSON-SKILL(P#, SKILL#)

MACH-SKILL(MACH#, SKILL#)

SCHED(JOB#, P#, MACH#, SCHED-START-DATE, SCHED-STOP-DATE)

The query is:

Given machine X (MACH#), a job number Y (JOB#). The desired start-date A (SCHED-START-DATE) and the desired stop-date B (SCHED-STOP-DATE), find the identification number (P#) of a person who has a skill (PERSON-SKILL. SKILL#) appropriate for the operation of machine X, and who is not scheduled at all between date A and date B; schedule this person if one is located.

DSL ALPHA:

```
GET (into workspace) W (at most) (1)
  PERSON-SKILL.P#: (such that)
    EXIST MACH-SKILL (with)
      (MACH-SKILL.MACH#=X)
      & (MACH-SKILL.SKILL#=PERSON-SKILL.SKILL.SKILL#)
      & NOT EXIST SCHED (with)
        (SCHED.P#=PERSON-SKILL.P#)
```

```
            & (SCHED.SCHED-START-DATE LESS-THAN B)
            & (SCHED.SCHED-STOP-DATE GREATER-THAN A)
      MOVE W INTO SCHED-RECORD
      PUT SCHED-RECORD SCHED
```

The COBOL DBTG program may be found in [Halstead, 1974; Codd and Date, 1974].

SQL:

```
      INSERT INTO SCHED(1): <Y,P#,X,A,B> WHERE P# =
        (SELECT P#
         FROM PERSON-SKILL
         WHERE SKILL# =
           SELECT SKILL#
           FROM MACH-SKILL
           WHERE MACH#=X)
    MINUS
      SELECT P#
      FROM SCHED
      WHERE SCHED-START-DATE < B
        AND SCHED-STOP-DATE > A
```

The INSERT is in a tentative format since no syntax for selecting just one tuple is shown in [Chamberlin et al, 1976].

TABLET:

```
      FORM SKILLED FROM P#, SKILL# OF PERSON-SKILL
      ADD COLUMN MACH# OF MACH-SKILL BY SKILL#=SKILL#
      KEEP ROWS WHERE MACH#=X
```

```
FORM NOTFREE FROM P#, SCHED-START-DATE,
  SCHED-STOP-DATE OF SCHED
KEEP ROWS WHERE SCHED-START-DATE < B
  AND SCHED-STOP-DATE > A
KEEP (1) ROW OF SKILLED WHERE P# NOT IN
  P# OF NOTFREE
INSERT Y,P#,X,A,B INTO SCHED
```

A database sublanguage based directly on the relational algebra is proposed by Date [Date, 1977]. In this language the scheduling query is:

```
PROJECT PERSON-SKILL OVER P#, SKILL# GIVING TEMP1
JOIN TEMP1 AND MACH-SKILL OVER SKILL# GIVING TEMP2
SELECT TEMP2 WHERE MACH#=X GIVING TEMP3
PROJECT TEMP3 OVER P# GIVING SKILLED
PROJECT SCHED OVER P#, SCHED-START-DATE,
SCHED-STOP-DATE GIVING TEMP4
SELECT TEMP4 WHERE SCHED-START-DATE < B
  AND SCHED-STOP-DATE > A GIVING TEMP5
PROJECT TEMP5 OVER P# GIVING NOTFREE
SKILLED MINUS NOTFREE GIVING FREE
SELECT FREE WHERE POSITION=1 GIVING ONEFREE
SCHED UNION (Y,ONEFREE.P#,X,A,B) GIVING SCHED
```

Again, a few liberties have been taken with the language as proposed. The POSITION=1 selection in the last SELECT is not in the language as given in Date. It is used to select the first tuple of FREE.

Tables 3.3.3.2 through 3.3.3.5 show the Halstead analysis of the DSL ALPHA, SQL, TABLET, and algebraic implementations. The final comparison is given in Table 3.3.3.6a. Because of the length of the COBOL DBTG implementation only the result is given for it. The analysis may be found in [Halstead, 1974].

Table 3.3.3.6a shows that the COBOL DBTG measures differ greatly from the others. There is certainly variation among the measures of the non-COBOL implementations but the variation is small in comparison to the COBOL measure. So, while the SQL and TABLET measures are different, they are similar to each other and to the other relational query languages. DSL ALPHA is of lower level than SQL because of the required record and file operations MOVE and PUT that are not elements of the calculus.

Table 3.3.3.6b compares the mean language levels and volumes of the queries on the final and the retention test. Again, the measures are different but comparable.

3.3.4 Identical constructs. Both SQL and TABLET use the same form for constants. The data aggregation functions are identical -- AVG, MAX, MIN, SUM, COUNT. Both languages use GROUP BY to specify the partitioning of a table. The comparison operators -- $>$, $<$, $=$, $\geq$, $\leq$, $\neq$, -- are identical. CONTAINS, IN and SAME are used in the two languages for partitioned table operations.

In addition, neither language requires commands that transfer control of program flow. These commands are often the focus of problems in query languages as well as general purpose programming languages.

Operator	Occurrence Count	Operand	Occurrence Count
-	12	SCHED	9
.	8	SKILL	8
()	6	MACH	4
#	6	PERSON	3
&	4	P	3
=	3	W	2
EXIST	2	DATE	2
LESS THAN	2	RECORD	2
:	1	1	1
NOT	1	X	1
MOVE	1	START	1
		STOP	1
		A	1
		B	1
n1 = 11	N1 = 46	n2 = 14	N2 = 39

$$L = 2/n1\ n2/N2$$

$$= 2/11\ 14/39$$

$$= .0653$$

$$V = (N1 + N2)\ \log_2\ (n1 + n2)$$

$$= 85\ \log_2\ 25$$

$$= 409$$

Table 3.3.3.2

Calculating L and V for the DSL ALPHA Implementation of the Scheduling Problem

Operator	Occurrence Count	Operand	Occurrence Count
-	6	PERSON	1
()	1	SKILL	2
INSERT INTO	1	MACH	1
:	1	1	1
<>	1	SCHED	2
WHERE	4	Y	1
=	3	P#	4
SELECT	3	X	2
FROM	3	A	2
MINUS	1	B	2
<	1	START	1
>	1	SKILL#	2
FIRST	1	STOP	1
AND	1	MACH#	1
		DATE	2
n1 = 14	N1 = 28	n2 = 13	N2 = 25

$$L = 2/n1\ n2/N2$$

$$= 2/14\ 15/25$$

$$= .0857$$

$$V = (N1 + N2)\ \log_2\ (n1 + n2)$$

$$= 53\ \log_2\ 29$$

$$= 257$$

Table 3.3.3.3

Calculating L and V for the SQL
Implementation of the Scheduling Problem

Operator	Occurrence Count	Operand	Occurrence Count
-	10	PERSON	1
FORM FROM	2	SKILL	2
OF	5	MACH	1
ADD COLUMN	1	START	2
BY	1	SKILLED	2
KEEP ROW WHERE	2	P#	5
=	2	SKILL#	3
<	1	STOP	2
>	1	MACH#	2
NOT IN	1	DATE	4
INSERT INTO	1	X	2
KEEP () ROW WHERE	1	NOTFREE	2
INTO	1	A	2
AND	1	B	2
		1	1
		Y	1
		SCHED	6
n1 = 14	N1 = 30	n2 = 17	N2 = 40

$L = 2/n1\ n2/N2$

$= 2/14\ 17/40$

$= .0607$

$V = (N1 + N2)\ \log_2\ (n1 + n2)$

$= 70\ \log_2\ 31$

$= 347$

Table 3.3.3.4

Calculating L and V for the TABLET
Implementation of the Scheduling Problem

Operator	Occurrence Count	Operand	Occurrence Count
-	10	P#	5
PROJECT	4	SKILL#	2
OVER	5	TEMP1	2
GIVING	10	DATE	4
JOIN	1	TEMP2	2
SELECT	3	MACH#	1
WHERE	3	X	2
MINUS	1	TEMP3	2
UNION	1	SKILLED	2
{}	1	SCHED	5
()	1	START	2
.	1	STOP	2
AND	2	TEMP4	2
=	2	A	2
<	1	B	2
>	1	TEMP5	2
		NOTFREE	2
		FREE	2
		POSITION	1
		1	1
		ONEFREE	2
		Y	1
		PERSON	1
		MACH	1
		SKILL	2
n1 = 16	N1 = 47	n2 = 25	N2 = 52

$L = 2/n1\ n2/N2$

$= 2/16\ 25/52$

$= .0601$

$V = (N1 + N2)\ \log_2\ (n1 + n2)$

$= 99\ \log_2\ 41$

$= 530$

Table 3.3.3.5

Calculating L and V for the Date Algebraic Implementation of the Scheduling Problem

	DBTG COBOL	DSL ALPHA	SQL	TABLET	Date's Algebra
Level	.0112	.0653	.0857	.0607	.0601
Volume	3199	409	257	347	530

Table 3.3.3.6a

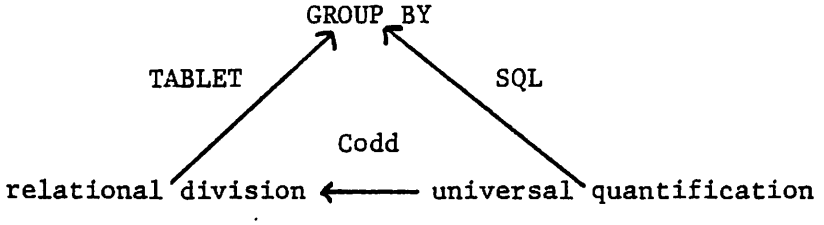
Halstead Language Levels and Volumes for the Scheduling Problem

	SQL	TABLET
Mean Level	.2734	.1670
Mean Volume	61	82

Table 3.3.3.6b

Mean Halstead Language Levels and Volumes for SQL and TABLET Queries from the Final and the Retention Test

Both SQL and TABLET use the GROUP BY construct to partition tables. We have just seen that GROUP BY is necessary in TABLET to implement relational division. Codd [Codd, 1971b] shows that universal quantification in the calculus reduces to relational division in the algebra. SQL uses GROUP BY to implement universal quantification. The following diagram illustrates the relationships.



We see that SQL and TABLET use the identical construct GROUP BY to implement the very different seeming division and universal quantification constructs of the relational algebra and calculus.

3.3.5 Terminal equipment. Both SQL and TABLET can use any type of terminal equipment -- from teletypes to graphics terminals. This similarity helps to isolate the independent variable -- procedurality -- by using languages that are as identical as possible. For example, Query By Example [Zloof, 1975] requires an interactive graphics terminal. The form of the language relieves the user of supplying much of the structure of a query. A comparison of this type of language with one having no such facility runs the risk of comparing the terminals instead of the languages.

3.4 Language Differences

SQL and TABLET differ in their procedurality. This difference is exhibited in several ways.

1. SQL is similar to the relational calculus and TABLET is based on the relational algebra.
2. The Halstead measures, while similar in comparison to non-query languages, are significantly different.
3. The procedurality metrics for the languages differ. (See section 3.4.3.)

3.4.1 Relational calculus and relational algebra. First we will look at the similarity between SQL and the relational calculus. The relational calculus is a tuple-oriented language, the predicate describes the tuples that qualify for inclusion in the target list. SQL uses tuple variables (as in Q8 in Section 3.2). The SELECT statement selects tuples that qualify given the WHERE clause and, perhaps, the HAVING clause.

The SELECT clause is similar to the relational calculus target list. The SELECT clause is the first element of a SQL query as the target list is the first part of the relational calculus query. The relational calculus requires complete specification of the domain (relation name. domain name) explicitly but the SQL query may use the FROM clause unless ambiguity arises. If ambiguity is possible, SQL uses the same form as the relational calculus uses.

A simple example of the similarity is exhibited by query Q1 in section 3.2. The SQL query is

```
SELECT NAME
FROM STUDENT
WHERE HOMESTATE='OHIO'.
```

The calculus equivalent is

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

The SQL query could use STUDENT.NAME and STUDENT.HOMESTATE but the scope of the FROM clause is over the entire SELECT statement so it is not necessary to specify the relation name when using the column names.

SQL and the relational calculus use the identical form for joining (a relational algebraic term) two relations. Taking Q7 from section 3.2 the SQL query is

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

The calculus query is

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

The similarity is obvious.

The chaining method used in SQL replaces the existential quantifier of the calculus. Looking at Q3 from Section 3.2 the SQL query is

```
SELECT NAME
FROM STUDENT
WHERE ID =
  SELECT ID
  FROM TAKING
  WHERE COURSE='ECON105'.
```

The calculus equivalent is

{(STUDENT.NAME) : ∃ x ∈ TAKING (STUDENT.ID=x.ID ∧ 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 name of a student whose ID is the same as the ID in a tuple of the TAKING table and the COURSE value in that tuple is ECON105.

The SQL set operators -- UNION, INTERSECT and MINUS -- correspond to ∨, ∧ 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].

Another feature of the relational calculus is universal quantification. The calculus solution to query Q6 from section 3.2 (i.e., List the ID's of students taking all the art courses offered.) is

ARTCOURSES←{(TEACHING.COURSE) : TEACHING.DEPT='ART'}

{(TAKING.ID) : ∀t ∈ ARTCOURSES ∃ s ∈ TAKING

(TAKING.ID=s.ID ∧ s.COURSE=t.COURSE)}.

Universal quantification is considered overly complex for most non-mathematicians, especially the casual user [Thomas, 1976].

The corresponding SQL query was

SELECT ID

FROM TAKING X

WHERE

```
        (SELECT COURSE
         FROM TAKING
         WHERE ID = X.ID)
    CONTAINS
        (SELECT COURSE
         FROM TEACHING
         WHERE DEPT = 'ART').
```

While the SQL query is very different from the calculus query, tuple variables are used. As discussed earlier, this use of tuple variables is a calculus-oriented aspect of SQL. Reisner [Reisner, 1976] showed that subjects had difficulty using tuple variables.

Tuple variables such as in the above query may be eliminated by means of GROUP BY:

```
SELECT ID
FROM TAKING
GROUP BY ID
HAVING SET(COURSE) CONTAINS
    SELECT COURSE
    FROM TEACHING
    WHERE DEPT = 'ART'.
```

Since this query lacks tuple variables it is even less calculus-like than the preceding SQL query. GROUP BY specifies an operation (i.e., partitioning [Furtado and Kerschberg, 1977]) on a relation. Thus, GROUP BY is an algebraic or procedural element embedded in nonprocedural SQL.

At this point it is clear that the simpler SQL queries correspond well to the calculus. As queries get more complex, the similarities break down. By the time we get to universal quantification there is a complete break down in the correspondence between SQL and calculus. In fact, SQL becomes more algebraic as complexity increases, as we shall see later.

TABLET is based on the relational algebra, but is not identical to it as seen in section 3.3.2 and in the example in section 3.3.3. The relational algebra is defined on relations; the relational algebraic operators transform relations into relations. Section 3.3.2 shows the correspondence between TABLET and the relational algebra.

TABLET is not defined on relations, but on tables. Tables, like relations, are collections of n-tuples, but a table may contain duplicate tuples. The necessity of retaining duplicates is illustrated in Q2 of section 3.2:

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

We have seen that the FORM command corresponds to the algebraic project operator. If the above FORM were to be the equivalent of

ECONSAL←FACULTY[ŠALARY,DEPT],

any duplicate tuples corresponding to people in the same department earning the same salary, would be eliminated.

KEEP ROWS is the equivalent of

ECONSAL←ECONSAL[DEPT='ECONOMICS'].

When we print the average salary

PRINT AVG(SALARY),

the result would be the average of the unique salaries which is not correct. Actually, the TABLET FORM command results in a table of the same cardinality as the table it is created from; ECONSAL has the same number of tuples as FACULTY. Now, when the averaging is done, we get the desired result. TABLET is a tabular algebra as opposed to the relational algebra. Each TABLET command transforms a table (or tables) into another table. In section 3.3.2 UNIQUE was used to generate a relation instead of a table.

3.4.2 Halstead measures. Section 3.3.3 used the Halstead measures to show the similarity of SQL and TABLET. When compared to the measures for a COBOL DBTG implementation, the SQL and TABLET measures are certainly close. The SQL and TABLET measures of program level and volume for the problems on the final and retention tests are significantly different ($p<.001$) as shown by a paired t-test. SQL has a higher program level mean and a lower volume mean than TABLET, showing that SQL is a higher level language than TABLET.

3.4.3 Procedurality metric. The previous two sections have shown some of the differences between SQL and TABLET. One purpose of this section is to show that those differences are directly related to the differences in the procedurality of the languages. The other purpose of this section is to propose a measure of language procedurality and relate it to these languages.

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 as mentioned in section 2.1. 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.

A metric for measuring the procedurality of programs is proposed here. The metric is based on several properties of programs:

1. The number of variable bindings.
2. The number of operations.

3. The number of possible orderings of variable bindings and operations.

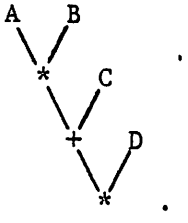
The number of variable bindings and operations increase the procedurality of a program. The number of possible orderings of variable bindings and operations tends to reduce the procedurality. For example, in the expression

E=A*B+C*D,

if operations were done left to right only, with no operator precedence, there would only be 1 way to interpret the expression,

(A*B+C)*D.

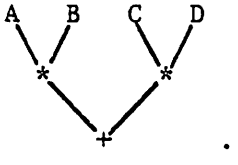
A data flow diagram for this expression is



There is only 1 possible ordering of the operations. If we used the usual FORTRAN-like operator procedure the statement is

E=(A*B)+(C*D),

and either parenthetical expression could be evaluated first. A data flow diagram for the latter expression is



This means that the values of A and B must be available for the left multiplication to occur. C and D must be available for the multiplication on the right. Both products must be available before the addition.

The procedurality metric, P, is defined as

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

For the above example (using FORTRAN precedence) there is one variable binding and three operators with 2 possible orderings so

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

Using left to right precedence, there is 1 variable binding and 3 operations but only 1 ordering, so

$$P = 1+3 = 4.$$

The P value is higher because more is specified about the operation order than in the FORTRAN-like case.

A final example would be the equivalent of the FORTRAN expression written in GOTRAN [Andree, 1967]. GOTRAN allows only 1 operator in an assignment statement so the GOTRAN version would be:

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

There are 3 variable bindings and 3 operators and only 1 possible ordering so

$$P = 3+3 = 6.$$

In GOTRAN the binding of intermediate results and the order of operations is completely specified. In the left to right parse the order of operations is specified but not the intermediate bindings. The P values of 6, 4 and 2.5 for GOTRAN, left to right parsing and FORTRAN

reflect the relaxing of the procedural constraints as we go from one language to the next.

In SQL variable binding is done in the FROM clause as in Q8 of section 3.2. A 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. If the name of the result table does not appear explicitly, the default user table is used as explained following Q1 in section 3.2.

Table 3.4.3.1 shows how the P values are calculated for the scheduling problem of section 3.3.3. Table 3.4.3.2a tabulates the results of these calculations. Table 3.4.3.2b contains the mean P values for the final and retention tests. We see that TABLET's P values are about 4 times those of SQL. So, although the language level and volumes of SQL and TABLET are quite close, as shown in Table 3.3.3.6, their procedurality differs greatly. The P values for the languages differ significantly ($p<.001$) on the 60 final and retention queries.

The P values for DSL ALPHA and SQL are fairly close. DSL ALPHA is more procedural than SQL because of the non-calculus commands -- GET, INTO, MOVE and PUT -- required for handling the workspace and updating the file.

The P values for TABLET and Date's algebra are fairly close to each other. The main reason for the difference is that TABLET combines the project and join operations in the ADD COLUMNS command. Also,

COBOL DBTG	Variable bindings are implemented using the MOVE, GET and MODIFY statements. There are 17 such statements, totally ordered. Each statement is an operation. There are 49 statements, totally ordered. P = 17/1 + 49/1 = 66
DSL ALPHA	Variable binding is implemented by the GET INTO W, MOVE and PUT statements. There are 2 such statements, totally ordered. The GET and PUT statements perform operations. There are 2 operations, totally ordered. P = 3/1 + 2/1 = 5
SQL	Variable binding is done by the INSERT command. There is 1 such command. The INSERT, SELECT and MINUS are operations. There are 5 operations. The MINUS must be done last but either top-level SELECT (SELECT P#) may be done first, so there are 2 orderings. P = 1/1 + 5/2 = 3.5
TABLET	Variable binding is done by every command. There are 7 commands, totally ordered. Each command is an operation. There are 7 commands, totally ordered. P = 7/1 + 7/1 = 14
Date's Algebra	Every command is a variable binding. There are 10 commands, totally ordered. Every command performs an operation. There are 10 commands, totally ordered. P = 10/1 + 10/1 = 20

Table 3.4.3.1

Calculating P Values for Various Implementations of the Scheduling Problem

	COBOL DBTG	Date's Algebra	TABLET	DSL ALPHA	SQL
P	66	20	14	4	3.5

Table 3.4.3.2a

P Values for the Scheduling Problem

	TABLET	SQL
P	7.3	1.8

Table 3.4.3.2b

Mean P Values for SQL and TABLET
Queries from the Final and the Retention Test

TABLET combined the selecting of 1 row with a restriction in the KEEP (1) ROW command. The algebra required two commands.

The P values for the nonprocedural languages (DSL ALPHA and SQL) differ greatly from those of the procedural languages (COBOL DBTG, Date's algebra and TABLET). Again, Table 3.4.3.2 shows that the TABLET queries have 4 times the P value of the corresponding SQL queries. DBTG's P value is far greater than any of the others. These relative P values correspond well to the author's intuition of the position of each language on the procedurality continuum (section 2.1).

Generally, more procedural languages are of lower level than less procedural languages. The Halstead levels showed that TABLET is a lower level language than SQL. This difference in Halstead measures is another reflection of the difference in the procedurality of SQL and TABLET.

The present section and the previous two sections have shown that TABLET is more procedural than SQL. For ease of discussion we will call SQL "nonprocedural" and TABLET "procedural".

C H A P T E R I V

RESULTS AND ANALYSIS

4.1 Results of the Experiment

The mean number of essentially correct solutions as well as mean times to take the final and retention tests, mean difficulty and mean study time for the lessons are found in Table 4.1.1. 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 operations, simple mapping with built-in functions, and composition (chaining). The easy problems are problems 1-10 on both the final and the retention test.
hard	20 problems from 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.

	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 inexperienced subjects)	TABLET mean (17 inexperienced subjects)
final score	30	18.371	18.541	16.824	17.050	19.833	20.294
final time (minutes)		116.71	120.35	123.53	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	5	2.0857	3.0000	1.8824	2.6500	2.2778	3.4118
combination retention	5	1.3143	2.0541	1.3529	1.6000	1.2778	2.5882
average study time (minutes, Lesson 1-12)		29.324	36.214	33.378	39.138	25.496	32.775
average difficulty (lessons 1-12; 1-easy, 10-hard)		4.1694	4.1789	4.5041	4.5885	3.8533	3.6971

Table 4.1.1

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

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 but 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	5 problems combining various language constructs. These are problems 13, 26, 27, 29 and 30 on both the final and the retention test.

Table 4.4.1 summarizes the experimental results. This table will also be referred to throughout this chapter. The problem numbers listed above correspond to the numberings in Appendix C. There were five randomizations of the exam in the actual experiment. The easy/hard dichotomy was based on the author's intuition. This intuition happens to correspond to 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.

4.2 Statistical Analysis

The results of both the final and the retention test were analyzed using a fully crossed, two-way analysis of variance. The two independent variables were the languages (SQL and TABLET) and the experience level of the subjects (inexperienced and experienced). The results of the retention test are summarized in Table 4.2.1. The results from the final showed little significance and in no way conflicted with the results of the statistical analysis of the retention test. The results of the group contrasts (from a one-way analysis of variance) are summarized in Table 4.2.2.

4.2.1 Study time, perceived difficulty and time to take the retention test. Using the values of the means from Table 4.1.1, we see that experienced subjects required significantly less study time than inexperienced subjects. SQL subjects required significantly less study time than TABLET subjects (Table 4.2.1a). The group contrasts (Table 4.2.2) show no significant differences between groups.

Table 4.2.1b shows that inexperienced subjects rated the lessons as significantly more difficult than experienced subjects did. Table 4.2.2 shows this is also true within each language.

The time required to take the retention test was significantly longer for inexperienced subjects than it was for experienced subjects. TABLET subjects took significantly longer than SQL subjects (Table 4.2.1c). Table 4.2.2 shows that inexperienced TABLET subjects required significantly longer to take the retention test than experienced TABLET subjects.

	degrees of freedom	F ratio	significance
experience	1,68	4.76	<.05
language	1,68	3.99	<.05
experience * language	1,68	.05	-

a Mean study time of Lessons 1-12.

	degrees of freedom	F ratio	significance
experience	1,68	11.33	<.001
language	1,68	.02	-
experience * language	1,68	.27	-

b Mean difficulty of Lessons 1-12.

	degrees of freedom	F ratio	significance
experience	1,60	1.44	-
language	1,68	6.10	<.05
experience * language	1,68	.06	-

c Time to take retention test.

	degrees of freedom	F ratio	significance
experience	1,68	10.96	<.001
language	1,68	2.03	-
experience * language	1,68	.97	

d Retention score (essentially correct).

	degrees of freedom	F ratio	significance
experience	1,68	8.91	<.01
language	1,68	.65	-
experience * language	1,68	.10	-

e Retention, easy problems.

	degrees of freedom	F ratio	significance
experience	1,68	8.62	<.005
language	1,68	6.72	<.05
experience * language	1,68	1.53	-

f Retention, hard problems.

	degrees of freedom	F ratio	significance
experience	1,68	6.51	.05
language	1,68	8.71	.005
experience * language	1,60	2.22	-

g Retention, group problems.

	degrees of freedom	F ratio	significance
experience	1,68	.13	-
language	1,68	26.53	<.001
experience * language	1,68	.19	-

h Retention, join problems.

	degrees of freedom	F ratio	significance
experience	1,68	5.76	<.05
language	1,68	.25	-
experience * language	1,60	.01	

i Retention, chaining problems.

	degrees of freedom	F ratio	significance
experience	1,60	.23	-
language	1,68	42.74	<.001
experience * language	1,68	2.08	-

j Retention, set problems.

	degrees of freedom	F ratio	significance
experience	1,68	3.82	<.05
language	1,60	11.12	<.005
experience * language	1,68	4.12	<.05

k Retention, combination problems.

Table 4.2.1

Summary of Analysis of Variance for Study Time, Lesson Difficulties and Retention Scores
(See Table 4.1.1 for means.)

	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)	-	-	-	-
Mean difficulty (Lessons 1-12)	-	-	<.05	<.01
Time to take retention test	-	-	-	-
Retention score (essentially correct)	-	-	-	<.005
Retention, easy	-	-	(.0666)	<.05
Retention, hard	<.01	-	-	<.005
Retention, group	<.005	-	-	<.005
Retention, join	<.0005	<.005	-	-
Retention, chaining	-	-	-	-
Retention, set	<.0001	<.0005	-	-
Retention, combination	<.0005	-	-	<.005

Table 4.2.2

Summary of Group Contrasts for Study Time, Lesson Difficulties and Retention Scores
(See Table 4.1.1 for means.)

4.2.2 Overall retention scores. The overall retention score was significantly higher for experienced than for inexperienced subjects. The scores for SQL and TABLET subjects were not significantly different (Table 4.2.1d). Table 4.2.2 shows that experienced TABLET subjects significantly outperformed inexperienced TABLET subjects on the retention test. The same is true for the easy problems (Table 4.2.1e and Table 4.2.2).

4.2.3 Hard problems. The experienced subjects did significantly better than inexperienced subjects on the hard problems. TABLET subjects outperformed SQL subjects on the hard problems (Table 4.2.1f). This result tends to confirm our original hypothesis. Table 4.2.2 shows that the experienced TABLET subjects did significantly better on hard problems than the experienced SQL subjects did. Also, experienced TABLET subjects significantly outperformed inexperienced TABLET subjects.

4.2.4 Group problems. Experienced subjects outperformed inexperienced subjects on the group problems (Table 4.2.1g). TABLET subjects outperformed SQL subjects. Table 4.2.2 shows much the same split for group problems as it does for hard problems but at higher significance levels.

4.2.5 Join problems. Interestingly, the experience level makes no difference for join problems (Table 4.2.1h). TABLET subjects outperform SQL subjects in join problems. Table 4.2.2 shows that both the experienced and inexperienced TABLET subjects outperform the corresponding SQL subjects.

4.2.6 Chaining problems. Experienced subjects outperform inexperienced subjects on chaining problems (Table 4.2.1i). There are no differences due to languages or within languages (Tables 4.2.1i and 4.2.2).

4.2.7 Set problems. SQL subjects outperform TABLET subjects across the board on set problems as shown in Tables 4.2.1j and 4.2.2. As in join, there is no difference due to experience.

4.2.8 Combination problems. Experienced subjects outperformed inexperienced subjects with marginal significance (Table 4.2.1k). TABLET subjects significantly outperformed SQL subjects. In this case the interaction term (experience × language) is significant. Table 4.2.2 shows that experienced TABLET subjects significantly outperformed experienced SQL subjects but the difference was not significant for inexperienced subjects. So the overall language difference was primarily due to the experienced subjects.

4.2.9 General results. Generally, experienced students outperformed inexperienced students. SQL subjects took less time to learn their language, found it less difficult to learn, finished the retention test faster and did better on set problems than the TABLET subjects did. TABLET subjects outperformed SQL subjects on hard problems, group problems and join problems. Where language differences occurred, they usually occurred between the experienced SQL and TABLET groups. These results will be discussed later.

4.3 Error Analysis

The previous section was concerned with the statistical analysis of the experimental results. The statistics show that differences exist in the subjects ability to write queries in the two languages but do not show what types of errors caused these differences. This section will show the types of errors that were found.

Errors were divided into two groups, minor and major. The types of errors in these two classifications will be discussed first. In addition, there were errors due to individual constructs in each language. The last subsection will cover these errors.

In order to make the problem grading more accurate, each problem was graded for all students before going on to the next problem. So, all problem 1's were graded, then problem 2, etc. This allowed for analogous errors to be treated analogously. This method also resulted in the grader being unaware of whose problem was being graded.

All examples in this section refer to the database of Table 4.3.1. This database was used for both the final and the retention test.

4.3.1 Minor errors. Minor errors were errors due to inadvertencies in the writing of a query. These are errors that would be found by the compiler but do not reflect any misunderstanding of the language.

There were few minor language errors. For example, several SQL students put most of the dots (.) in the fully qualified column name (e.g., ITEM.ITEMNO) but would leave one dot out. They obviously knew to use the dots, but carelessly forgot one. Also, if several unrelated minor errors of different type were made, these were classified as minor

Database - 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

Table 4.3.1a

The MAILORDER Database

This database contains information about a mailorder company. This company is old and established and sells only high quality items.

DEPARTMENT	Contains information about the departments in the company. There is one row for each department. FLOOR tels which floor of the building the department is on.
ITEM	Contains information about the items sold by the company. There is one row for each item. RETAIL contains the retail price of the item. The item is reordered if ONHAND is less than or equal to REORDER. ALTERNATE gives the item number of the item to be substituted for this one if the ONHAND value is 0. Obviously, ALTERNATE contains the item number of another item in the ITEM table.
SELLS	Contains information about which departments sell which items. QUOTA gives the quantity of the item that must be sold by this department each month. There is one row for each item sold by each department.
SUPPLIER	Contains information about the companies that supply items to this mailorder company. There is one row for each supplier company.
SUPPLIES	Contains information about the items supplied by each supplier. WHOLESALE is the wholesale price per item charged by the supplier. ONORDER tells how many of the item is presently on order from the supplier. There is one row for each item supplied by each company.
CHARGEACCTS	Contains information about the charge accounts held by customers of the company. There is one row for each account. TOTALBILL contains the total amount owed by the customer. LIMIT contains the customer's credit limit. A customer is over his limit if TOTALBILL is greater than LIMIT. RATING contains the customer's credit rating (1 is poor, 10 is excellent). Each customer must be referred by another charge customer in order to get his account. REFERREDBY contains the account number of the person who referred this customer to the company.
CHARGED	Contains information about the items charged by the customer. PERITEM is the price the customer paid per item charged. It is included because the customer may have purchased a sale item or purchased an item before its RETAIL price in the ITEM table went up. (If the customer returns the item, this is the amount he gets back.)

Table 4.3.1b

Further Explanation of the Relations
in the MAILORDER Database

language errors since they showed a slight miscomprehension of the language. If several minor errors occurred that showed some basic confusion, the solution was marked as having a major language error. Both SQL and TABLET subjects showed some problem with parentheses in function references. The most common error was writing a function reference as

(COUNT) SUPPNAME

instead of

COUNT (SUPPNAME).

This was classified as a minor language error. Small misspellings of language keywords also were classified as minor language errors. Extraneous WHERE's as in

```
WHERE ONHAND > '3000'
  AND WHERE ALTERNATE = '8'
```

were also minor language errors. Subjects often spelled the AVG function call as AVE resulting in a minor error for the query. Another common error was using 2·LIMIT for 2*LIMIT. Subjects were used to the dot (·) operator for multiplication and tended to use it. Some TABLET subjects used WHERE instead of BY in the ADD command. This was judged as a minor language error because the compiler will recognize the WHERE as being improperly placed when it is expecting BY and can ask for it. Another error found in both SQL and TABLET was incorrect naming of tables and columns in the SQL assignment command (using ←) and the TABLET FORM command. Illegal symbols were sometimes used or names would start with a numeric character. A minor TABLET error was caused by some subjects thinking that the keyword FORM was required when referencing a table name, so phrases such as

KEEP GROUPS WHERE ITEMNO IN ITEMNO OF FORMS

would result when S was a table formed earlier in the query. The query is perfectly correct if the FORM is eliminated. Most of the minor language errors were common to both languages. SQL and TABLET do have some minor language errors peculiar to each one, but the most usual occurred in both languages.

Minor operand errors were minor errors in column names, table names or constants. Simple misspellings such as ACCNTNO for ACCTNO were obviously minor errors. Using the plural for a singular column (e.g., NAMES for NAME) was also classified as a minor error. Putting quotes around column names in non-ambiguous cases such as

TOTALBILL > 'LIMIT'

were allowed because the compiler would find that TOTALBILL is a numeric column and 'LIMIT' is non-numeric. Also, leaving quotes off was a minor error if there was no ambiguity as in

SUPPNAME = WARMTH,INC.

The compiler would find no column with the name WARMTH,INC and print a message. An ambiguous reference would be

SUPPNAME = 'ITEMNO'

for

SUPPNAME = ITEMNO

or the reverse. The compiler would accept either of these. Fortunately, this is an unlikely circumstance and never occurred in the testing.

A minor substance error is one that causes a slightly incorrect set of tuples to be printed. The set of tuples printed must include

some of the proper set. For example, if the query were "List the items with retail price over $5.00.", the WHERE clause should be

WHERE RETAIL > 5.00

but

WHERE RETAIL ≥ 5.00

would be classified as a minor substance error. If the restriction is indirectly specified as in the query, "List the items that need to be reordered.", the correct response is

WHERE ONHAND ≤ REORDER

but

WHERE ONHAND < REORDER

is classified as a minor substance error. The subject must remember or look up the definition of "REORDER" since it is not directly specified in the query, so the second response is only a minor error.

4.3.2 Major errors. All incorrect queries fall into one of the major error categories.

The first category is that of correctable errors. These are errors that are correctable by a good compiler. For example, the SQL query

SELECT ITEMNO, DESCRIPTION

WHERE RETAIL > 5.00

is missing the FROM clause. The only relation in the database containing the necessary columns is ITEM, so the compiler could add the FROM clause. This is classified as a major error because it is correctable due to fortuitous circumstance. If another relation contained the same columns as used in the query, it would not be correctable.

Queries containing major substance errors are queries that are syntactically correct but answer a different question than the one specified. A common example is the use of UNION instead of INTERSECT in SQL. Both SQL and TABLET have problems with chaining or composition (SQL terms) problems. For example the query, "List the descriptions of items whose alternates cost more than $100.", has the following correct solution:

```
SQL: SELECT DESCRIPTION
     FROM ITEM
     WHERE ALTERNATE =
       SELECT ITEMNO
       FROM ITEM
       WHERE RETAIL > 100.
```

It is a common error to reverse ALTERNATE and ITEMNO in the WHERE clause. Many errors are not language related but reflect the difficulty people have with writing algebraic equations. For example, one query concerns bills that are more than twice the average bill, the correct TABLET KEEP ROWS command is

```
KEEP ROWS WHERE TOTALBILL > 2*AVG(TOTALBILL).
```

A common error is to write

```
KEEP ROWS WHERE TOTALBILL*2 > AVG(TOTALBILL).
```

Another confusion arises when using tuple variables in SQL to join a relation with itself. The correct solution to the query, "List each item description and the description of its alternate for items that cost more than their alternates.", is

```
SELECT A.DESCRIPTION, B.DESCRIPTION
FROM ITEM A, ITEM B
WHERE A.ALTERNATE = B.ITEMNO
  AND A.RETAIL > B.RETAIL.
```

Often, < is used for > because of the confusion over whether A or B is the alternate tuple (B is the alternate). TABLET subjects had the analogous problem; the BY clause in the ADD COLUMNS command was often incorrect. Also, TABLET subjects were not sure of which columns to include in the FORM command. For example, the TABLET solution to the preceding problem would be

```
FORM DESCS
  FROM DESCRIPTION, RETAIL, ALTERNATE OF ITEM
ADD COLUMN DESCRIPTION (AS ALTDESC), RETAIL (AS ALTRET)
  OF ITEM
  BY ALTERNATE = ITEMNO
KEEP ROWS WHERE RETAIL > ALTRET
PRINT DESCRIPTION, ALTDESC.
```

The usual mistake was to include ITEMNO in the FORM command and to reverse the columns in the BY clause. Only 1 of 35 SQL subjects wrote a correct response for a query of analogous form to this one on the retention test; 2 of 37 TABLET subjects wrote it correctly.

The most common major substance error in both languages was writing a simpler query than the one desired. For example, people often wrote a query for "List the names of suppliers that supply items sold by the camping department.", when the desired query was, "List the names of

suppliers that supply only items sold by the camping department.". The "only" was forgotten, thus eliminating the need for the GROUP BY and IN constructs.

A query contains a major language error if the query is illegal and the error shows a basic misunderstanding of the language. A common SQL error is to answer the query, "Which account or accounts have the largest total bill? (List the account number.)", with the query

```
SELECT ACCTNO
FROM CHARGEACCTS
WHERE TOTALBILL = MAX(TOTALBILL)
```

instead of

```
SELECT ACCTNO
FROM CHARGEACCTS
WHERE TOTALBILL =
  SELECT MAX(TOTALBILL)
  FROM CHARGEACCTS.
```

This nested form for the function was a problem for many subjects. Only 7 of the 35 SQL subjects wrote this query correctly, 23 TABLET subjects wrote it correctly. TABLET does not use this nested syntax.

TABLET subjects confused 'KEEP ROWS' with 'KEEP GROUPS'. SQL subjects had the analogous problem of using 'WHERE' for 'HAVING'.

Incomplete and unattempted responses are, obviously, responses that were either incomplete of that did not exist.

4.3.3 Generic errors. These errors may be major or minor and are related to constructs used in the languages.

TABLET subjects had a great deal of trouble using the ADD ROWS, KEEP ROWS . . . IN and KEEP ROWS . . . NOT IN constructs. These constructs are analogous to the SQL UNION, INTERSECT and MINUS which were very successful. These were classified as SET problems in section 4.1. None of the 37 TABLET subjects wrote a correct query using the ADD ROWS construct on the retention test. 23 of the 35 SQL subjects wrote the query correctly using the UNION operator. On the final, 13 TABLET subjects wrote it correctly as opposed to 33 SQL subjects. 27 TABLET subjects wrote the correct response to a query that could use the KEEP ROWS . . . IN construct but they all used an alternate method using the ADD COLUMNS construct. They query was, "List the account numbers of accounts with credit rating of 10 who have charged item 19.". Two possible responses were:

```
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 ITEM19 WHERE ACCTNO IN ACCTNO OF RATING10
```

and

```
FORM RATING10ITEM19
  FROM ACCTNO, RATING OF CHARGEACCTS
ADD COLUMN
  OF CHARGED
```

```
  BY ACCTNO = ACCTNO
KEEP ROWS WHERE RATING = 10
  AND ITEMNO = 19
PRINT ACCTNO.
```

All subjects chose the latter method.

There was no alternate for the KEEP ROWS . . . NOT IN construct and only 9 TABLET subjects wrote the query correctly. 21 SQL subjects wrote it correctly using the MINUS operator. The query was, "List the names of suppliers located in Boston that do not supply item 19.". Many subjects in both the SQL and TABLET groups used ITEMNO ≠ 19 as part of their query which was incorrect, so some of the problems with this query is a misunderstanding of the concept of set difference.

SQL subjects had trouble with the GROUP BY construct. GROUP BY would be left out of or used incorrectly in queries requiring it. The correct SQL response to the query, "List the suppliers whose average wholesale price is over $500.", is

```
SELECT SUPPNAME
FROM SUPPLIES
GROUP BY SUPPNAME
HAVING AVG(WHOLESALE) > 500.
```

Only 4 SQL subjects wrote the above SQL query. 20 SQL subjects wrote

```
SELECT SUPPNAME
FROM SUPPLIES
WHERE AVG(WHOLESALE) > 500
```

which is not even a legal query as explained in the previous section. 19 TABLET subjects wrote this query correctly.

SQL subjects also had trouble with join problems on the retention test. Only 7 SQL subjects wrote 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.". The correct response is

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

The joining restriction was forgotten. 23 TABLET subjects wrote their queries correctly.

An interesting sidelight to the last problem is that 5 SQL subjects tried to use GROUP BY in their responses. This illustrates the fact that SQL subjects use GROUP BY when it is not required in addition to not using it when it is required. TABLET subjects also used the GROUP BY in this problem, but due to the nature of TABLET, the extraneous GROUP BY did not effect the correctness of the response. The correct TABLET response is

```
FORM NAMEITEM
  FROM NAME, ACCTNO, TOTALBILL, LIMIT OF CHARGEACCTS
ADD COLUMN ITEMNO
  OF CHARGED
  BY ACCTNO = ACCTNO
KEEP ROWS WHERE TOTALBILL > LIMIT
PRINT NAME, ITEMNO.
```

Usually, TABLET subjects placed the extraneous

GROUP BY ACCTNO

before the PRINT command. This has no effect on the correctness of the query. 8 TABLET subjects added GROUP BY to their queries, none of the GROUP BY's caused the queries to be in error, but 4 of the responses were incorrect due to other factors. So, both SQL and TABLET subjects used the extra GROUP BY, but TABLET is more forgiving of this mistake than SQL is.

4.4 Interpretation of the Results

Table 4.2.1 shows that the performance of the SQL and TABLET subjects showed significant differences in the following categories:

1. Mean study time for Lessons 1-12.
2. Time required to take the retention test.
3. Hard retention problems.
4. Join retention problems.
5. Set retention problems.
6. Group retention problems.
7. Combination retention problems.

Tabl3 4.4.1 also shows that experienced subjects outperformed inexperienced subjects. This section will deal with these differences in performance as well as an interesting observation on the way SQL and TABLET subjects handle chaining problems. First, a cognitive model of programmer behavior will be presented that will aid in our understanding of the results of this experiment.

4.4.1 A cognitive model of programmer behavior. Various human factors studies including those referenced in Chapter 1 have resulted in the rudiments of a cognitive model of human behavior [Shneiderman and Mayer, 1975]. The rest of this section refers to this model.

Figure 4.4.1.1 illustrates the components of memory involved in programming. 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

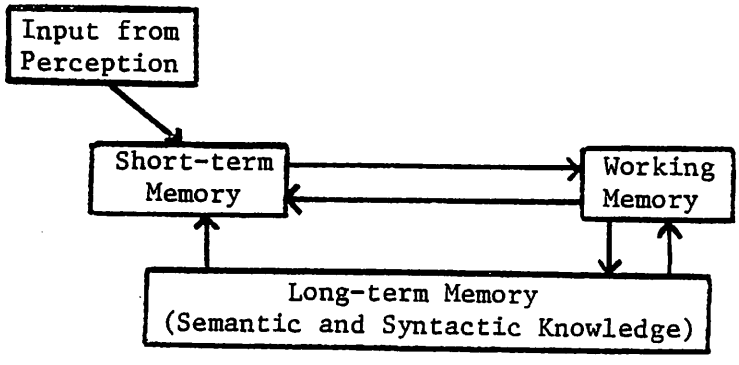


Figure 4.4.1.1

Memory Modules Required for Programming

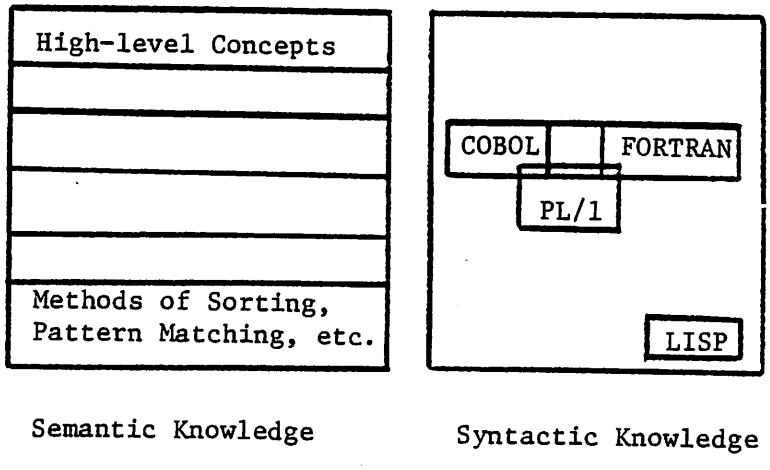


Figure 4.4.1.2

Long-term Memory

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 4.4.1.2):

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.

One of the Reisner's results [Reisner, 1976] can be used to illustrate the use of this model. Reisner found that the results for experienced SEQUEL and SQUARE subjects did not differ significantly but those for inexperienced subjects did.

The experienced subjects already had a semantic knowledge structure built in their long-term memory. Their language, SEQUEL or SQUARE, was added to their syntactic knowledge structure. SEQUEL and SQUARE differ primarily in their syntax. SEQUEL has a vertical, English keyword-oriented syntax and SQUARE, a horizontal, position and operator-oriented syntax. Experienced programmers, with their well-defined semantic structure, could see through the syntactic differences and there was no difference in their performance.

Inexperienced subjects, on the other hand, were building their semantic structure but were more tied up in the syntax of the language. The semantics must, at some point, be drawn from the syntax. SQUARE is syntactically more complex than SEQUEL, especially for non-mathema-

tically oriented subjects. SEQUEL uses English keywords whereas SQUARE uses position, parenthesization and operators to perform the same functions. SEQUEL looks like English and SQUARE resembles mathematical notation. It was inevitable that this syntactic difference would work against inexperienced users.

The models presented in Figures 4.4.1.1 and 4.4.1.2 will also be helpful in the discussion of the SQL-TABLET experiment.

4.4.2 Mean study time for Lessons 1-12. Table 4.2.1a shows that TABLET subjects studied the lessons significantly longer than did SQL subjects. SQL is referred to as a language in which a template exists that must be filled in [Reisner, 1976]. The SELECT-FROM-WHERE is the template and the columns, tables and relationships must be added. The syntax of SQL is its main component.

TABLET, on the other hand, requires knowledge of table transformations, of building and joining tables. TABLET has a richer and, thus, more complex semantic component. Naturally, TABLET requires more studying because of the richness of the semantic component that must be built.

Also, TABLET syntax is more complex because it has the capability of specifying all the operations used in arriving at a desired result. SQL does not specify the operations. The Halstead measures given in Chapter 3 show TABLET to be of a lower syntactic level than SQL. Thus, it takes more time to master the TABLET syntax than to master the SQL syntax.

4.4.3 Time required to take the retention test. TABLET subjects required significantly longer to take the retention test than SQL subjects did (Table 4.2.1c). As in the previous section, this difference relates to the syntactic level and semantic complexity of TABLET. It takes more time to write a TABLET query because the query is longer and is semantically more complex than a SQL query.

4.4.4 Hard retention problems. TABLET subjects wrote significantly more essentially correct responses to hard problems on the retention test than SQL subjects did (Table 4.2.1f). This is the point at which we benefit from the operational skills required in the use of TABLET. TABLET subjects have built a semantic structure that they can use to solve complex problems. Also, looking at Table 4.1.1 we see that although TABLET subjects outperformed SQL subjects only slightly on the hard final problems, the performance of TABLET subjects on hard retention problems was significantly better than that of SQL subjects. SQL subjects did not retain their language as implied by Shneiderman and Mayer in section 4.4.1; SQL has primarily a syntactic component, so it is more easily forgotten.

It is interesting to compare these results with the results of Reisner that were discussed earlier. Looking at Table 4.2.2 we see that the experienced TABLET subjects performed significantly better than experienced SQL subjects but the difference for inexperienced subjects is not significant. This is the reverse of the case for Reisner. The main reason for the difference being greatest for the experienced TABLET subjects is that TABLET is procedural as are the

languages that the experienced subjects know, BASIC and FORTRAN. Thus, the semantic structure of the experienced subjects matched the procedural methods required by TABLET. There is no such match between the semantic structure and SQL for the experienced SQL subjects.

The inexperienced subjects have no semantic structure derived from a previous programming language. Yet, inexperienced TABLET subjects outperformed inexperienced SQL subjects on both the final and retention (Table 4.1.1). This difference is not significant (Table 4.2.2) but leads to the following conjecture: the semantic structure learned by the TABLET subjects helps them even if it was not reinforced by a comparable, preexisting structure. The inexperienced subjects have learned a skill and like riding a bicycle, that skill is retained through a period of non-use. The semantic and syntactic structures seem to support each other; the semantic part says that it is possible to perform a specific operation on a table and that aids (through an associative memory?) in arriving at the applicable syntactic form. Another factor aiding TABLET for inexperienced users may be a natural human predilection for procedural problem solving. This predilection will be discussed further in Chapter 5.

4.4.5 Join retention problems. TABLET subjects wrote significantly more essentially correct responses to join problems on the retention test that SQL subjects did (Table 4.2.1h). TABLET subjects were aided by the fact that TABLET uses the same constructs for these problems as it does for chaining (SQL term) but SQL uses different constructs. So, TABLET subjects had previous experience. Notice that the difference

between SQL and TABLET for chaining problems is not significant. Also, notice that Table 4.2.2 shows the performance of TABLET subjects was independent of computer experience.

4.4.6 Set retention problems. SQL subjects wrote significantly more essentially correct responses to set problems on the retention test than TABLET subjects did (Table 4.2.1i). This result is probably tied to the experience today's college students have had with the set operations union and intersection in elementary school. All the subjects in this experiment stated on a questionnaire that they were familiar with union and intersection. Set difference (MINUS) was unfamiliar to the subjects, but having the idea of sets probably made it an easy concept to pick up. So, the SQL subjects had a pre-existing semantic structure for sets and, specifically, the union and intersection operators. Table 4.2.2 shows the performance of SQL subjects to be basically independent of computer experience. This lends credence to the idea that the SQL subjects had a common reason for their performance.

TABLET subjects had the semantic structure for sets as well as the union and intersection operators. But TABLET used a very different method and different operators.

It would be interesting to take an older group of subjects or subjects from a population that had no experience with sets and see how SQL and TABLET compare under these circumstances. Obviously, SQL is aided by the new math. The results of this experiement could change if traditional mathematics teaching were to return to the nation's grade schools.

4.4.7 Group retention problems. TABLET subjects wrote significantly more essentially correct responses to group problems on the retention test than SQL subjects did (Table 4.2.1g). SQL and TABLET use the identical language construct for partitioning a table --

GROUP BY column name(s).

GROUP BY is used under identical conditions in both languages; there is no difference in when it is used. Also, it is a new concept to all the subjects; there is no comparable idea in the set theory of the new math, so subjects had no pre-existing semantic structure to aid them with group problems. All of this is important because the significant differences shown in previous constructs (set problems and two table problems) could be attributed to previous training and syntactic differences in the constructs used.

Two SQL subjects (one in class, the other in the final interview) said that the GROUP BY clause was incorrectly placed in the SELECT. They though that GROUP BY should appear before the SELECT clause. SQL places restrictions on the selected information when GROUP BY is used -- only values that are constant over a group may be selected. But, the GROUP BY appears after the SELECT, so there is a restriction in the SELECT due to a construct that appears later. In TABLET the PRINT command follows the GROUP BY command.

Also, GROUP BY is an imperative as is SELECT yet it is a subordinate clause that appears along with the descriptive FROM, WHERE and HAVING clauses that make up the predicate of the SELECT.

In TABLET, GROUP BY does appear before the PRINT command. Also, it is a command in itself, not subordinate to another command.

In terms of the structure of long-term memory, GROUP BY has the identical syntactic component in both SQL and TABLET, but the semantics closely matches the rest of TABLET whereas there is a mismatch between the descriptive nature of SQL and the imperative nature of SQL and the imperative nature of GROUP BY.

Another way to approach the problem of the GROUP BY in SQL is to look at the English semantics of a query using GROUP BY as opposed to its SQL semantics. Let's do this by looking at an example. Using the database introduced earlier:

EMP(NAME, DNO, SAL)

DEPT(DNO, LOC).

List the number of employees in each department who made more than $10000.

```
SQL:      SELECT DNO, COUNT(*)
          FROM EMP
          WHERE SAL > 10000
          GROUP BY DNO

TABLET:   FORM NUMEMPS FROM DNO SAL OF EMP
          KEEP ROWS WHERE SAL > 10000
          GROUP BY DNO
          PRINT DNO, COUNT(*)
```

A SQL query, taken as two imperative English sentences, says to "select then group". This is clearly not the correct semantics of the SQL query. In fact, the do-this-then-do-that nature of the query's

English semantics is foreign to SQL semantics in general. Multiple SELECT's are not semantically sequenced in their syntactic order. A second SELECT is conceptually sequenced during the first SELECT. Thus SQL's semantics contains nesting as its basic ordering mechanism. This seems to be appropriate for SELECT's but does not fit the grouping operation, at least in its imperative form.

Examination of the TABLET query shows that no such dissonance between the English and TABLET semantics is present. This could also be the factor which leads to better retention of the capability of using this feature correctly.

The fact that experienced TABLET subjects outperformed all other groups (Table 4.2.2) implies that the TABLET GROUP BY operator corresponds closely to the subjects' procedural experience.

This result is the most interesting one of all the construct-specific results. It is the cleanest element of the experiment because the syntax and semantics of GROUP BY are identical in the two languages. Any differences must be due to the language in which the construct is embedded.

4.4.8 Combination retention problems. TABLET subjects wrote significantly more essentially correct responses to combination problems on the retention test than SQL subjects did (Table 4.2.1k). This difference cannot be tied to any one construct. The fact that TABLET allows its users to combine constructs with greater facility than SQL does is of great importance. It implies that the TABLET user will have an easier time handling novel situations than SQL users will.

4.4.9 Chaining retention problems. There was no difference due to language for chaining problems on the retention test. There is an interesting observation that can be made about the methods that TABLET subjects used in solving chaining problems.

Chapter 5 of the SQL manual introduces chaining. The first example query is, "List the courses being taught by Bill Grant.". The SQL query is

```
SELECT COURSE
FROM TEACHING
WHERE ID =
  SELECT ID
  FROM FACULTY
  WHERE NAME = 'BILL GRANT'.
```

Chapter 5 of the TABLET manual introduces analogous queries using the ADD ROWS command. The above query is shown in TABLET as

```
FORM COURSENAME
  FROM COURSE, ID OF TEACHING
ADD COLUMN NAME
  OF FACULTY
  BY ID = ID
KEEP ROWS WHERE NAME = 'BILL GRANT'
PRINT COURSE.
```

Notice that the TABLET example follows the same order of table reference as the SQL manual does -- first the TEACHING table then the FACULTY table. This order is required in SQL, but TABLET can use the reverse ordering,

as in

```
FORM NAMECOURSE
  FROM NAME, ID OF FACULTY
ADD COLUMN COURSE
  OF TEACHING
  BY ID = ID
KEEP ROWS WHERE NAME = 'BILL GRANT'
PRINT COURSE.
```

This option is presented once in the TABLET manual. All other examples, problem solutions and quiz solutions use the SQL ordering.

Quiz 4 quizzed the subjects on this type of problem. Only 3 of the 37 TABLET subjects solved Problem 1 of Quiz 4 using the non-SQL ordering. This seemed reasonable after considering the SQL bias of the lesson. Also, the quiz was open book and subjects probably used the lesson examples as models for their solutions.

As time passed, the non-SQL ordering became more prominent. On Quiz 5, a review quiz, Problem 1 required the ADD COLUMN command. This time 23 TABLET subjects used the non-SQL ordering. On the retention test Problem 9 allowed a SQL and a non-SQL ordering. Again, 23 TABLET subjects (not all the same) used the non-SQL ordering. Although 23 TABLET subjects used non-SQL orderings on both the review quiz and the retention test, only 13 subjects used the non-SQL ordering in both. This suggests approximately a 50% likelihood of a person using either ordering.

In the interviews (occurring after Lesson 6, after Lesson 11 and after the final) many subjects said they started a query with what they

knew (e.g., the person's name in the above query) and worked towards the unknown (e.g., the course being taught). TABLET allows this ordering. SQL does not.

In terms of the cognitive model, it seems that the TABLET subjects have a robust semantic component of their long-term memory. This robustness is exhibited by the ability of subjects to write queries in a way that is natural for them even if the method used is contrary to the presentation of the method.

The fact that there was no significant difference due to language in the ability of subjects to write chaining queries shows that the ideas used are simple in both SQL and TABLET. The ability of TABLET subjects to use a more natural form may lead to improved performance on difficult queries that require chaining as a subproblem.

4.5 Recommendations for Language Changes

The results of this experiment show that there are difficulties with the following problem categories:

1. Join problems in SQL.
2. Set problems in TABLET.
3. Group problems in SQL.

Recommendations for language changes are presented in the rest of this section.

4.5.1 Recommendations concerning join problems in SQL. The difference between the performance of SQL and TABLET subjects on join problems (Tables 4.2.1h and 4.2.2) is primarily due to the experience TABLET subjects received in using the ADD COLUMNS command. ADD COLUMNS is used in easier TABLET problems for which SQL used the chaining or composition technique. The obvious, but undesirable, change is to eliminate the chaining method and always use joining so that subjects have more experience. For example, the response to the query, "List the names of people who have charged item 19.", using chaining is

```
SELECT NAME
FROM CHARGEACCTS
WHERE ACCTNO =
  SELECT ACCTNO
  FROM CHARGED
  WHERE ITEMNO = '19'.
```

This would use the joining method,

```
SELECT NAME
FROM CHARGEACCTS, CHARGED
WHERE CHARGEACCTS.ACCTNO = CHARGED.ACCTNO
  AND ITEMNO = '19'.
```

The problem with this is it requires a fairly complex query for a simple problem. SQL is referred to as a layered language [Reisner, 1976] because it has a set of simple constructs, easily mastered by novices, and a set of more complex constructs that can be mastered with experience. Chaining is simple but joining is more complex, so replacing chaining with joining would eliminate a large class of queries from the SQL's simple layer.

Lochovsky [Lochovsky, 1978] recommends eliminating the join specification from the WHERE clause and adding an explicit join specification to the language. At present, the correct response to the query, "List the names of people who have exceeded their credit limit and the item number 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. The designers of TABLET adopted such a construct (the BY clause) independently.

4.5.2 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 rating 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, proven concepts.

4.5.3 Recommendations concerning group problems in SQL. It is tempting to use the student comments referred to in section 4.4.7 and put the GROUP BY clause earlier in the SELECT. The place it should really go is before the SELECT itself. Making this change seems to have 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.

C H A P T E R V

CONCLUSION

5.1 Summary

A human factors experiment testing the ability of subjects to write queries in two different query languages has been run. One purpose of the experiment was to test the hypothesis: people more often write difficult queries correctly using a procedural query language than they do using a nonprocedural query language. Another purpose of the experiment was to compare corresponding constructs in the two languages and recommend improvements for constructs that were found difficult.

The two languages chosen were SQL and TABLET. Both languages use the relational model of data and are relationally complete. They have similar Halstead levels. They also use the same built-in functions. The languages differ in their procedurality -- SQL is nonprocedural and TABLET is procedural.

The subjects of the experiment were 72 undergraduates at the University of Massachusetts taking an accounting course. The subjects were divided into two groups -- 35 learning SQL and 37 learning TABLET. Subjects in each group were classified as inexperienced (having no previous computer experience) and experienced (having had a course in BASIC or FORTRAN).

The languages were taught using language manuals that presented the same examples and problems in the same order for each language. No lectures were given. There were 14 class meetings in the course,

each of which was devoted to answering questions and quizzing the students. At the end of the course a final was given. Three weeks later a retention test was given. Both tests required students to write queries in the language they learned.

The results of the retention test are the main results of the experiment. These query languages were designed as an aid to the casual user, a user who uses the language intermittently. The delay between the end of the course and the retention test models this intermittence. It was also felt that the delay would bring out differences in the retention of the languages.

The results of the retention test supported the hypothesis: subjects using the procedural language (TABLET) wrote significantly more essentially correct responses to difficult questions than were written by subjects using the nonprocedural language (SQL). Comparing the results for corresponding constructs in the languages suggested several changes to those constructs:

1. Using the set operations UNION, INTERSECT and MINUS in TABLET.
2. Rephrasing the GROUP BY clause in SQL.
3. Changing the method of joining tables used in SQL.

5.2 General Comments on Procedurality

There has long been argument about procedural and nonprocedural general purpose programming languages. The advocates of nonprocedural languages seem to be winning this argument. The superiority of nonprocedural languages seems to be seen as axiomatic. This axiom may be just that, an axiom that, once assumed, leads to interesting but not necessarily useful results.

Since programming languages and query languages are methods of problem solving, results of problem solving studies should be of interest. Some of these studies are occurring as this is written [Clement, Lochhead and Soloway, 1979]. These studies look at two methods of solving problems. One method uses static, descriptive (nonprocedural) equations to represent a problem and lead to its solution. The other uses dynamic (procedural) methods. Initial results of these studies show the procedural method superior to the nonprocedural method.

Most people are procedurally oriented. For example, we learn to do things by doing them.

There are some people who seem to think less procedurally or who enjoy describing problems or problem solutions nonprocedurally. These people are often mathematicians. As mathematicians have more influence on programming languages, the languages become less procedural. This nonprocedurality does have a mathematical elegance but may not be the correct orientation for the average programmer and, especially, not for the naive user of database query languages.

The result of this SQL-TABLET experiment adds credence to the supporters of procedural languages. The subjects of the experiment were naive in the use of computers. Approximately half the subjects had no experience. The other half had, at most, one course in computer programming and that was, for most subjects, an introduction to BASIC. This background was chosen because it would seem to be typical of people who would get involved as users of query languages -- they might have a little training, but that training would be a single, required college course with little or no subsequent use of computers.

It may be argued that computer programmers are completely different in background and predilection than the subjects of this experiment. This may be true, in part, but anyone who has programmed for long realizes how little mathematical skill is really required of most programmers. Programmers are people, and results of the SQL-TABLET experiment show that people -- especially people with a little background with computers -- perform better using a procedural language.

The thrust of this section is that while the SQL-TABLET experiment was mainly a test of two languages using a specific set of 72 subjects, its result may point to a generic tendency of human beings to use procedural rather than nonprocedural methods to solve problems. Obviously, the right level of procedurality is required -- few people would argue that an assembly language would outperform SQL or TABLET based on the higher procedurality of the assembly language. Assuming that programmers are not completely disjoint from the rest of mankind, the results of these experiments should aid in deciding the direction of programming

languages. Obviously, much more research is needed before we can tell how global the results of our research is.

5.3 Language Evolution

The evolution of SEQUEL to SEQUEL 2 was partially the result of planned experimentation [Reisner, 1976]. The strengths and weaknesses of SEQUEL were found by experimenting using subjects from a realistic population. The resulting language has now been tested and the required improvements made.

Compare this to the evolution of FORTRAN. As different users and manufacturers saw new applications and realized the shortcomings of the language, they made their own revisions to the language. Now, the dialects of FORTRAN are such that the ANSI standard is the lowest common denominator and is almost ridiculously stringent. If FORTRAN had been tested thoroughly early in its existence, features such as mixed mode expressions, logical variables, various formatting features and other constructs might be standard. Also, the operation of the DO statement might not be rooted in history but in the needs of programmers.

Now, it is too late for FORTRAN. It could be argued that the complete testing of any general purpose programming language is too complex to be reasonable. Even if this argument is true, language designers should apply the principles derived from experimentation and not rely solely on intuition.

The next section describes the next stage of TABLET's evolution.

5.4 Further Research

One result of this experiment is the suggestion that TABLET use the set operations UNION, INTERSECT and MINUS. An experiment is currently being planned that will incorporate these constructs into an otherwise unchanged TABLET. None of the SQL constructs will be changed, so SQL will be a constant.

The other change in the experiment will be the subject population. Only inexperienced subjects will be used. Separating the effects due to experience was difficult in the present experiment. The result for inexperienced subjects is of greater interest.

A further experiment, testing the revised SQL against a constant TABLET, is being considered.

5.5 Recommendations for the Uses of the Language

Another result of this experiment is an idea of the circumstances under which each language should be used. TABLET is obviously best for intermittent users with some experience using a procedural language if the users will be solving difficult problems. If the situation requires a language that needs very little learning time and will be used mainly for easy queries, SQL is the choice.

The choice for inexperienced people is difficult to determine, especially if the users will be solving difficult problems. Future research will help resolve this question.

5.6 Final Remarks

The use of human factors experimentation in computer science is new and controversial. Some computer scientists think it does not belong in computer science but in psychology. This is a totally false perception. The results of such experiments are of much greater interest to computer scientists than to psychologists. We, computer scientists, want to know how to design languages. Psychologists was to know how human beings relate to the world. Experiments such as the one described in this paper have too many variables to supply any definite results to a psychologist. The results show a computer scientist which language or approach yields superior user performance. The exact mental processes responsible for the result, while certainly of interest, are secondary to the result itself.

Another major reason for human factors testing in computer science is to substantiate the claims made for a language. A language may be designed and implemented with the intention of making it easy to use, but testing of the language is required to see if it performs as designed. A parallel situation occurs when a more efficient sort algorithm is devised. A proof of its efficiency is required. Unfortunately, human factors testing is not as precise a tool as a mathematical proof but as the only tool available, human factors testing is constantly being improved by ongoing research.

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.

Clement, J., Lochhead, J., and Soloway, E. (1979). Translating between symbol systems: Isolating a common difficulty in solving algebra word problems, Cognitive Development Project, Department of Physics and Astronomy, University of Massachusetts, Amherst, February.

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.

Codd, E.F. (1971c). A database sublanguage founded on the relational calculus, RJ 893, July 26, IBM Research, San Jose, California.

Codd, E.F., and Date, C.J. (1974). Interactive support for non-programmers: The relational and network approaches, RJ 1400, June 6, IBM Research, San Jose, California.

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

Date, C.J. (1972). Relational database systems: A tutorial, Proc. 4th International Symposium on Computer and Information Science, Miami Beach, December, Plenum Press, New York.

Date, C.J., and Codd, E.F. (1974). The relational and network approaches: Comparison of application programming interfaces, RJ 1401, June 6, IBM Research Lab, 5600 Cottle Road, San Jose, California.

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

Durding, B.M., Becker, D.A., Gould, J.D. (1977). Data organization, Human Factors, 19(1), pp. 1-14.

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.

Gould, J.D., Lewis, D., and Becker, C.A. (1976). Writing and following procedural, descriptive and restricted syntax language instructions, RC 5943, April 9, IBM Watson Research Center, Yorktown Heights, New York.

Halstead, M.H. (1974). Software physics comparison of a sample program in DSL ALPHA and COBOL, RJ 1460, October, IBM Research Lab, San Jose, California.

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

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

Ledgard, H., and Singer, A. (1977). The case for human engineering, COINS Technical Report, 77-11, September, University of Massachusetts, Amherst, Massachusetts.

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 structure on program understanding, Proc. ACM Conference on Language Design and Reliable Software, SIGPLAN notices, Vol. 12, No. 3, March.

McGee, W.C. (1976). On user criteria for data model evolution, ACM TODS, Vol. 1, No. 4, December, pp. 370-387.

Michaels, A.S., Mittman, B., and Carlson, C.R. (1976). A comparison of relational and CODASYL approaches to data-base management, ACM Computing Surveys, Vol. 8, No. 1, March, pp. 125-151.

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 experimentation 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 database 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 flowcharts 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.

Small, D.W., and Weldon, L.J. (1977). The efficiency of retrieving information from computers using natural and structured query languages, Report SAI-78-655-WA, September, Science Applications, Inc., 1911 N. Ft. Myer Drive, Suite 1200, Arlington, Virginia.

Stemple, D.W. (1977). A data base management facility and architecture for the realization of data independence, PhD thesis, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts.

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. (1977). Psychological issues in data base management, Proc. Very Large Data Bases, IEEE, New York, New York, pp. 169-185.

Thomas, J.C. (1976). Quantifiers and question-asking, RC 5866, February, IBM Watson Research Center, Yorktown Heights, New York.

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

Tsichritzis, D.C., and Lochovsky, F.H. (1976). Hierarchical data-base management, ACM Computing Surveys, Vol. 9, No. 1, March, pp. 105-123.

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. (1978a). SQL manual, University Computing Center, University of Massachusetts, Amherst, Massachusetts, November.

Welty, C. (1978b). TABLET manual, University Computing Center, University of Massachusetts, Amherst, Massachusetts, November.

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.

A P P E N D I X A

SQL MANUAL

TABLE OF CONTENTS

Lesson 1. Introduction to tables and data bases.

You probably know that computers can store large amounts of information. This information is of little importance to people unless they can get the specific pieces of information they want. Much science fiction such as Star Trek and Star Wars present computers that respond beautifully to spoken questions:

Captain Kirk - "Where is the Klingon third fleet?"

Computer - "Alpha Centauri, first quadrant, fourth sector."

Unfortunately, at present it is not possible to communicate with a computer in this way.

We are going to learn a language for use in asking questions of a computer. This sort of language is called a query language and is used to retrieve information from a computer. The purpose of this course is twofold:

1. To aid anyone who may have need of the computer in the future.

2. To test the reaction of people to the language in order to improve the language.

Your participation in this course will result in your learning about a useful tool and will aid in the development of that tool.

The language we will study was designed for use by people who have no knowledge of computers. The language uses a stilted form of written English to communicate with the computer. This language allows us to state exactly what the computer is to do. The computer is a big, fast, expensive machine. It will do exactly what you ask it to do and this leads to a problem. It is much like the fairy tale about King Midas. King Midas asked that

everything he touched turn to gold. He forgot to exclude food, water, his daughter, etc. The computer is much the same - you get what you ask for.

The first thing we must do is find out what information is in the machine and what it looks like. The information may be of any type and oriented towards any use. Many people with diverse interests use the computer to store information. Historians, sociologists, scientists, librarians and businessmen all store information in computers. Though the particular information is different for the various users it is all stored in simple tables.

The following table contains information about students in some college:

STUDENT

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

The table has a name: STUDENT. Each column in the table also has a name - ID, NAME, SEX, HOMESTATE and MAJOR. There is a row in this table for each student in the college. The actual table in the computer would have many rows, the few rows shown above are just samples.

Often it takes more than one table to store all the information required. For example, the college may also want to store information about the courses each student is currently taking:

TAKING

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

The table named TAKING has three columns - ID, COURSE and SECTION. Each row of this table contains the information about a course being taken by a student. The sample rows shown above show that student number 1 is taking 3 courses. The course name and section number are given in addition to the student ID. From the ID given in the STUDENT table we see that John Jones is taking the three courses, HIST101, HIST102 and POLSCI115. The rows in this table are not really stored in any order, they are shown in a certain order above just as an example.

You might ask why we don't store all the information in one table such as:

STUDENT-TAKING

ID	NAME	SEX	HOMESTATE	MAJOR	COURSE	SECTION
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	HIST101	1
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	HIST102	2
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	POLSCI115	1
2	JANE DOE	F	OHIO	ECONOMICS	ECON105	3
2	JANE DOE	F	OHIO	ECONOMICS	ECON201	1
2	JANE DOE	F	OHIO	ECONOMICS	MATH100	1

statement. We are using the STUDENT table so the FROM clause contains STUDENT. The WHERE clause describes the row of the table containing the information to be printed. We want the row in which the NAME column contains 'John Jones'. We write this in SQL as: WHERE NAME='JOHN JONES'. Notice that the STUDENT table contains all the columns used in this query. The columns used are HOMESTATE, MAJOR and NAME (NAME is in the WHERE clause).

The phrase contained in the WHERE clause (in this case NAME='JOHN JONES') is called an attribute. An attribute is a quality exhibited by something. For example an attribute of MacIntosh apples is that they are red. In this query language we are talking about the attributes of rows in a table. In Example 2.1 we want to print the row with the attribute that 'JOHN JONES' is in the NAME column. Attributes of rows correspond to attributes of actual things. A row in the student table that has 'OHIO' in the HOMESTATE column corresponds to a student from Ohio.

The quotes around 'JOHN JONES' are necessary to show that this is actually information that is stored in the table. Whenever we refer to a value that is stored in a table we put quotes around it, for example - 'ECONOMICS', '2', 'JANE DOE', 'OHIO' and 'ECON101'.

Actually, numbers do not have to be in quotes, the quotes are optional. The following are all correct

```
  3    '3'
 45    '45'
3.8   '3.8'
```

The entire query says to print the home state and major (SELECT HOMESTATE, MAJOR) from the STUDENT table (FROM STUDENT) for the student whose name is John Jones (WHERE NAME = 'JOHN JONES').

Obviously, the words SELECT, FROM and WHERE must be spelled correctly. The order of the SELECT, FROM and WHERE clauses is always the same as in Example 2.1

All the queries in this lesson and the next few lessons will use only one table. You can recognize a query that uses only one table by determining which columns are to be printed (they will be used in the SELECT clause) and the columns involved in the attribute (used in the WHERE clause) If you can find a table containing all these columns, you can use the methods presented in this and the next few lessons. These queries will always be of the form:

SELECT _____

FROM _____

WHERE _____

Now you try one.

Problem 2.1

English: Print Mary Smith's ID number. (Notice that Mary Smith's information is not in the small sample data base given. This is normally the case - if you knew the answer you wouldn't ask the question.)

SQL:

The answers to the problems are found at the end of the text. Try to solve the problem before looking up the solution.

Example 2.2

English: What building is the Economics department in?

The DEPARTMENT table contains the BUILDING column (for the SELECT clause) as well as the DEPT column (for the attribute).

SQL:

```
SELECT BUILDING
FROM DEPARTMENT
WHERE DEPT = 'ECONOMICS'
```

In this case the attribute is DEPT = 'ECONOMICS'. We want to print the building name contained in the row having a DEPT value of 'ECONOMICS'.

Problem 2.2

English: What department does John Milton teach in and what is his salary?

SQL:

Problem 2.3

English: What is Bill Williams' major? (Bill is a student.)

SQL:

All the queries written so far have been assumed to return

information from one row of the table. Actually, this assumption could be wrong. Take Example 2.1:

```
SELECT HOMESTATE, MAJOR
FROM STUDENT
WHERE NAME='JOHN JONES'
```

This attribute (NAME='JOHN JONES') could be exhibited by several rows, there may be more than one John Jones in the college. If there were three John Jones's we might get the response:

```
MASSACHUSETTS     HISTORY
   NEW YORK       BIOLOGY
      MAINE       ENGLISH.
```

The fact that many people may have the same name is the reason for the ID columns in the tables. If we said:

```
SELECT HOMESTATE, MAJOR
FROM STUDENT
WHERE ID='1'
```

and each person had a unique ID, we would be assured of getting:

```
MASSACHUSETTS     HISTORY.
```

Often we do wish to obtain information from many rows of a table.

Example 2.3

English: List the names of all history majors.

SQL:

```
SELECT NAME
FROM STUDENT
WHERE MAJOR='HISTORY'
```

This query will list the names of all history majors in the

school. Many rows have the attribute that the value in the MAJOR column is 'HISTORY'. Again, we see that the NAME and MAJOR columns required in the query are both found in the STUDENT table.

Problem 2.4

English: List the courses taken by the student whose ID is 2.

SQL:

Example 2.4

English: What are the majors of students from Massachusetts?

SQL:

```
SELECT MAJOR
FROM STUDENT
WHERE HOMESTATE='MASSACHUSETTS'
```

Obviously, this query will print out lots of majors. Even if many students have the same major, each major will be printed only once. Duplicates are eliminated from the list. We might get a response like:

```
HISTORY
BIOLOGY
FRENCH
ENGLISH
  .
  .
  .
```

We would not get:

HISTORY

BIOLOGY

ENGLISH

HISTORY

FRENCH

BIOLOGY

.

.

.

The underlined majors are duplicates and would not be printed. Obviously, repeated values are really just wasted. The information needs to be printed only once. SQL deletes duplicates from all lists.

Problem 2.5

English: List the courses and their titles for courses taught in the economics department.

SQL:

All the SQL queries you have seen so far have looked like:

```
SELECT column names
FROM table name
WHERE column name = 'value'
```

The SELECT clause lists the columns containing the information to be printed. The FROM clause contains the name of the table

containing all the column names found in the query. The WHERE clause specifies the attribute of the rows containing the information to be printed.

If we want to print information from all the rows in a table, no WHERE clause is needed:

```
SELECT NAME
FROM FACULTY
```

will print the names of all faculty members.

Problem 2.6

English: List the names and majors of all the students.

SQL:

Lesson 3. More on attributes.

All of the attributes have been of the form:

column name = 'value'

It is possible to use comparisons other than =, these are:

> greater than

< less than

>= greater than or equal to

<= less than or equal to

≠ not equal.

Example 3.1

English: List the names of all faculty members who make less than $11,000.

SQL:

```
SELECT NAME
FROM FACULTY
WHERE SALARY < '11000'.
```

The attribute of these faculty members is that they make less than $11,000. This attribute is written in SQL as: SALARY < '11000'. Notice that we do not use the dollar sign ($) or comma (,) in the value '11000'. The computer looks at the value purely as a number with no interpretation of its meaning, so the $ is not used. Commas are just not used in numbers in this language.

This query will print the names of faculty members making less than $11,000, but not the names of people making exactly $11,000. If we want to include those people making exactly $11,000 the attribute would be: SALARY <= '11000'.

Problem 3.1

English: List the courses worth 4 credits or more.

SQL:

Problem 3.2

English: List the courses with more than two sections.
(Assume that if a course has a section 3 or higher it must have sections 1 and 2.)

SQL:

Now for another slightly different type of problem.

Example 3.2

English: List the faculty members' ID, course and section number for all full classes.

A class is full if the LIMIT and SIZE columns of the TEACHING table have the same value in the same row. In the sample data base MATH100 is full. We want to print the faculty ID , course and section number. All of this information is in the TEACHING table.

SQL:

```
SELECT ID, COURSE, SECTION
FROM TEACHING
WHERE SIZE = LIMIT
```

This query will print the desired information for rows having the

attribute that the SIZE value and the LIMIT value are the same. Usually, just to be safe, we would say

SIZE >= LIMIT

in case someone counted wrong and allowed 41 students into a class whose limit is 40. If = were used alone, the class would not be printed as full and it certainly should be printed.

Notice that LIMIT is not in quotes. Column names are never quoted. If the attribute were SIZE >= 'LIMIT', the computer would look for the value 'LIMIT' in the SIZE column. That would be incorrect. Also if we were to write the attribute COURSE = ECON105 (instead of COURSE = 'ECON105') the computer would try to find a column named ECON105, which does not exist. Remember, quotes are put around values but not around column names.

Problem 3.3

English: Some departments are located in a building with the same name as the department (engineering for example). List out the department names and building names for which the names are not the same.

SQL:

At this point we have seen two types of attributes:

1. column name = 'value'
2. column name = column name

In addition, we can use > , < , >=, <=, and ≠ instead of =.

Let's try some more complicated attributes.

Example 3.3

English: What are the ID's of students taking section 2 of ECON119?

Let's state this query in terms of the attributes a row in the TAKING table must have in order to be printed: Print the ID if the section number is 2 and the course is ECON119.

SQL:

```
SELECT ID
FROM TAKING
WHERE SECTION = '2'
   AND COURSE = 'ECON119'
```

The AND says that both the attributes must apply to this row. In other words the row must have both SECTION = '2' and COURSE = 'ECON119' in order to be printed. The order of the two attributes does not matter. The following WHERE clause has the same effect as the one above:

```
WHERE COURSE = 'ECON119'
  AND SECTION = '2'.
```

Problem 3.4

English: List the names of history majors from Ohio.

SQL:

Sometimes we may have more than just two attributes that must hold. When this happens we can add more AND's followed by

the other attributes. A WHERE clause with 4 attributes all of which must hold would look like:

```
WHERE attribute1
  AND attribute2
  AND attribute3
  AND attribute4
```

Problem 3.5

English: List the names of women economics faculty members making more than $15,000.

SQL:

Now what if we wanted to print a row if any one or more of a set of attributes holds?

Example 3.4

English: List the names of all students from Ohio and all students from Iowa.

Again let's state the problem with respect to a particular student: Print the name of a student if he is from Ohio or if he is from Iowa.

SQL:

```
SELECT NAME
FROM STUDENT
WHERE HOMESTATE = 'OHIO'
  OR HOMESTATE = 'IOWA'
```

The OR means that a person qualifies if he is from Ohio or Iowa.

In terms of rows, we print a row if that row has 'OHIO' or 'IOWA' in the HOMESTATE column.

Notice that the English statement of the query used 'and' but that we used OR in the SQL query. When a query has many attributes it is necessary to give some thought as to whether AND or OR is to be used. The English statement of the problem is often misleading.

In Example 3.4 it happens that both attributes concerned the same column, HOMESTATE. The attributes do not have to use the same columns.

Problem 3.6

English: List the names of all faculty members who are in the Economics department or who make less than $10,000.

SQL:

Do you see the difference between AND and OR? AND is used when every attribute must hold. OR is used if any of the attributes may hold. Many attributes may be used with OR. If we had 4 attributes we would write:

```
WHERE attribute1
  OR attribute2
  OR attribute3
  OR attribute4.
```

A row would be printed if any of the attributes held.

Lesson 4. Arithmetic and functions.

It is often useful to perform arithmetic in a query.

Example 4.1

English: The economics faculty has decided to aid the finances of the college by having every member take a $500 pay cut per year. List the ID, name and new yearly pay for members of the economics faculty.

SQL:

```
SELECT ID, NAME , SALARY-500
FROM FACULTY
WHERE DEPT = 'ECONOMICS'
```

This query is very straightforward. The possible arithmetic operations are:

*	MULTIPLICATION
/	DIVISION
+	ADDITION
-	SUBTRACTION

Problem 4.1

English: It has been decided to let 10 more people into all the full classes. List the course name, section number and new limit of enrollment for the full classes.

SQL:

Example 4.4

English: Give 10% raises to all faculty members making less than $10,000. Print the faculty members' names and the amount of their raises.

SQL:

```
SELECT NAME , .10 * SALARY
FROM FACULTY
WHERE SALARY < '10000'
```

Arithmetic may also be used in the attribute.

Example 4.5

English: Print the course and section number for classes that are less than half full.

SQL:

```
SELECT COURSE, SECTION
FROM TEACHING
WHERE SIZE < LIMIT/2
```

Problem 4.4

English: List the names of faculty members who would make more than $30,000 if they were given $5000 raises.

SQL:

Functions may also be used in the attribute.

Example 4.6

English: List the faculty members who make more than the average salary in the school.

SQL:

```
SELECT NAME
FROM FACULTY
WHERE SALARY >
   SELECT AVG(SALARY)
   FROM FACULTY
```

The second SELECT finds the average salary which is then used in the attribute of the first SELECT. The average is calculated using all the faculty salaries because there is no WHERE clause in the second SELECT.

Problem 4.5

English: List the faculty member or members who make the largest salary in the school.

SQL:

Example 4.7

English: What are the section and course names of courses having the smallest enrollment in the college?

SQL:

```
SELECT COURSE, SECTION
FROM TEACHING
WHERE SIZE =
   SELECT MIN(SIZE)
   FROM TEACHING
```

Say that the smallest course has 4 people in it. This query would print out the course names and sections of all courses with 4 in the SIZE column.

Problem 4.6

English: Which courses have the most sections?

SQL:

Example 4.8

English: List the names of faculty members who make more than the highest salary made in the economics department.

```
SQL:  SELECT NAME
      FROM FACULTY
      WHERE SALARY >
         SELECT MAX(SALARY)
         FROM FACULTY
         WHERE DEPT = 'ECONOMICS'
```

Here we have used a WHERE clause in the second SELECT so we find the maximum salary only for the economics department.

Problem 4.7

English: List the courses and sections for courses that have fewer people in them than the smallest course taught by the math department.

SQL:

Lesson 5. Chaining.

What happens if we want to print information from columns of one table but the attribute involves columns from another?

Example 5.1

English: List the courses being taught by Bill Grant.

There is no table containing all the necessary information. Course names are in TEACHING, but faculty names are not. We see that the TEACHING table contains the ID of a faculty member and the FACULTY table contains the ID and faculty name. We want to SELECT from the TEACHING table but use the FACULTY table for the attribute.

Let's state the query in terms of a particular course. Print the course for which the ID is the same as the ID of Bill Grant (uniqueness of ID's means that the course is taught by Bill Grant).

We want to print the name of a course with a particular ID, so we write the incomplete query:

```
SELECT COURSE
FROM TEACHING
WHERE ID =
```

Then we want to get the ID from the FACULTY table where the name is Bill Grant:

```
SELECT ID
FROM FACULTY
WHERE NAME = 'BILL GRANT'
```

These two parts combine to form:

```
SELECT COURSE
FROM TEACHING
WHERE ID =
   SELECT ID
   FROM FACULTY
   WHERE NAME = 'BILL GRANT'
```

(The second SELECT is indented just to make the query more readable. The indentation is not required.)

The sample data base shows that ECON105 is one of the courses that will be printed.

This process of using one table for the printing and another table for the attribute is called chaining. There must be a column that links the tables together for the chaining to work. In Example 5.1. the ID column in the TEACHING table links to the ID column of the FACULTY table. The FROM clause contains the name of the table used in the SELECT and WHERE clauses of its SELECT statement. The top SELECT statement uses columns COURSE and ID from the TEACHING table and the bottom SELECT statement uses ID and NAME from the FACULTY table. Therefore, each FROM clause has only one table name in it.

Problem 5.1

English: Print the name of the person teaching section 2 of ECON105.

SQL:

Example 5.2

English: List the names of teachers of 4 credit courses.

Teachers names are only in the FACULTY table. The TEACHING table contains the courses taught by the faculty members. The COURSES table contains the credits for the courses. FACULTY links to TEACHING by ID. TEACHING links to COURSES by COURSE, so:

SQL:

```
SELECT NAME
FROM FACULTY
WHERE ID =
   SELECT ID
   FROM TEACHING
   WHERE COURSE =
      SELECT COURSE
      FROM COURSES
      WHERE CREDITS = '4'
```

This is called a 3 level chain because it uses 3 SELECT statements chained together.

Now you try one.

Problem 5.2

English: List the names of students taking 1 credit courses.

SQL:

Now, we have queries that look like this:

```
SELECT column names
FROM table name
WHERE column name =
   SELECT column name
   FROM table name
   WHERE ...
         .
         .
         .
```

It is not necessary that the linking columns in the two tables have the same column name. Up to this point they all have had the same names, but that was by coincidence.

Example 5.3

English: List the names of students whose major department is located in the Keynes building.

SQL:

```
SELECT NAME
FROM STUDENT
WHERE MAJOR =
   SELECT DEPT
   FROM DEPARTMENT
   WHERE BUILDING = 'KEYNES'
```

Problem 5.3

English: Print the name and salary of the head of the economics department.

SQL:

Example 5.4

English: What is the name of the head of the committee that John Milton is on?

All faculty names are in the FACULTY table, so the top SELECT (the one that says what to print) will use the NAME column of the FACULTY table. We want the person whose ID is the same as the COMHEAD value in John Milton's row. We chain from the FACULTY table to the FACULTY table.

```
SQL: SELECT NAME
     FROM FACULTY
     WHERE ID =
        SELECT COMHEAD
        FROM FACULTY
        WHERE NAME = 'JOHN MILTON'
```

Look at the sample data base. Do you see that the head of John Milton's committee is Bill Grant? We are chaining between rows of the same table.

Problem 5.4

English: Who is John Jones's representative?

SQL:

Lesson 6. Combinations of forms.

This review lesson will present new combinations of the queries that we have already seen. There are many possible combinations of forms. The purpose of this lesson is to show you how some forms are combined. In the tests you may see combinations that are not presented here.

Example 6.1

English: List the names of teachers who make more than $20,000 and teach MATH100.

SQL:

```
SELECT NAME
FROM FACULTY
WHERE SALARY > '20000'
   AND ID =
      SELECT ID
      FROM TEACHING
      WHERE COURSE = 'MATH100'
```

Let's analyse this query. We want to print the faculty NAME, which is only in the FACULTY table. One of the attributes concerns SALARY, also in the FACULTY table. The other attribute concerns COURSE. Both attributes must apply so we use an AND between the attributes. The second attribute (COURSE = 'MATH100') requires a chain from the FACULTY to the TEACHING table using ID as the link.

All queries, including combinations of forms, must be broken down into what is being printed and what the attributes are. Then tables must be found that contain the required columns and

that are correctly linked together.

In Example 6.1 one attribute uses SALARY, from the FACULTY table and the other uses COURSE from the TEACHING table. We have combined the AND with chaining.

Earlier we learned that attributes may be on either side of the AND or OR. Let's look at Example 6.1 written incorrectly as:

```
SELECT NAME
FROM FACULTY
WHERE ID =
      SELECT ID
      FROM TEACHING
      WHERE COURSE = 'MATH100'
   AND SALARY > '20000'
```

Since indentation is ignored, the computer will think that the AND is part of the second WHERE:

```
WHERE COURSE = 'MATH100'
   AND SALARY > '20000'
```

The computer will say that you have made an error because SALARY is not a column in TEACHING (the FROM clause specifies the table containing all the columns used in a SELECT). It is necessary to use parentheses to show which parts of the query go together:

```
SELECT NAME
FROM FACULTY
WHERE ( ID =
      SELECT ID
      FROM TEACHING
      WHERE COURSE = 'MATH100' )
  AND SALARY > '20000'
```

Parentheses may be put around any attribute but is only really necessary in cases like the above. Writing the query in the original order (Example 6.1) would eliminate the need of the parentheses. There are some cases in which they really are required and you should be careful when lots of forms are combined.

Now you try a problem that combines different features than the one just shown.

Problem 6.1

English: List the names of the faculty members whose classes would overflow (be over their limit) if 10 more people signed up.

SQL:

Example 6.2

English: What is the average salary of teachers of ECON105?

SQL:

```
SELECT AVG(SALARY)
FROM FACULTY
WHERE ID =
   SELECT ID
   FROM TEACHING
   WHERE COURSE = 'ECON105'
```

Here we have combined a function (AVG) with chaining.

Problem 6.2

English: How many credits is Jane Doe taking?

SQL:

Example 6.3

English: What is the total enrollment of courses worth 4 credits or more?

SQL:

```
SELECT SUM(SIZE)
FROM TEACHING
WHERE COURSE =
   SELECT COURSE
   FROM COURSES
   WHERE CREDITS >= '4'
```

Example 6.4

English: How many history majors from Ohio are taking ECON105?

SQL:

```
SELECT COUNT(ID)
FROM STUDENT
WHERE MAJOR = 'HISTORY'
   AND HOMESTATE = 'OHIO'
   AND ID =
      SELECT ID
      FROM TAKING
      WHERE COURSE = 'ECON105'
```

You could also say:

```
SELECT COUNT(NAME)
```

because duplicates are not eliminated and if there were two Jane Doe's they would both be COUNTed. COUNT(*) would also work.

Problem 6.3

English: Which 4 credit courses are taught by Bill Grant?

SQL:

Problem 6.4

English: How many students is Anne Hall teaching?

SQL:

Problem 6.5

English: List the names of people teaching ECON105 and the names of people teaching POLSCI115.

SQL:

Lesson 7. UNION, INTERSECT and MINUS.

Example 7.1

English: List the ID's of students and faculty in the economics department.

To get the student ID's we say:

```
SELECT ID
FROM STUDENT
WHERE MAJOR = 'ECONOMICS'
```

For the faculty ID's we say:

```
SELECT ID
FROM FACULTY
WHERE DEPT = 'ECONOMICS'
```

We put the two sets of ID's together with UNION.

SQL:

```
SELECT ID
FROM STUDENT
WHERE MAJOR = 'ECONOMICS'
UNION
SELECT ID
FROM FACULTY
WHERE DEPT = 'ECONOMICS'
```

UNION is used when you want to combine the results of two queries. Example 7.1 would result in a list of ID's in one column like:

```
19
47
219
116
```

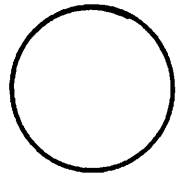
.

.

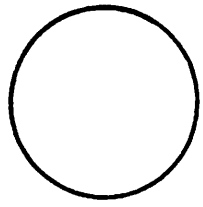
.

The student and faculty ID's will be mixed together, the student ID's will not be printed first followed by the faculty ID's.

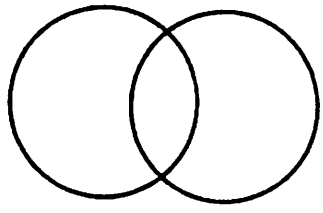
Let's use a diagram to explain Example 7.1. Let a circle represent all ID's of economics students.



Another circle represents the ID's of the economics faculty members:



The set of all the economics ID's is represented as:



The circles may intersect (overlap) because a student may also be a faculty member. (If ID's were something like social security number each person would have one ID and would not need two if he were both a student and a faculty member.)

The UNION prints out all the ID's from both circles, duplicates are eliminated as usual.

Problem 7.1

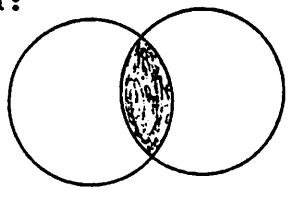
English: List the ID's of students taking ECON113 and the ID's of faculty members teaching ECON113. (Don't let the 'and' confuse you.)

SQL:

Example 7.2

English: List the ID's of economics students who are also faculty members.

Looking at the two circles drawn earlier we want to print out the shaded area:



This is the part of the circles that contain the ID numbers held both by students and faculty members. The shaded area is called the intersection of the two circles. In SQL we do an intersection using an INTERSECT.

```
SQL: SELECT ID
     FROM FACULTY
     WHERE DEPT = 'ECONOMICS'
INTERSECT
     SELECT ID
     FROM STUDENT
     WHERE MAJOR = 'ECONOMICS'
```

There may be no ID's in the intersection. In this case nothing would be printed except a message saying that nothing was found. This is the case when no students are also faculty members.

Problem 7.2

English: List the courses taken by student 19 that are taught by faculty member 247. (The student does not have to be in the section taught by the faculty member, just has to be taking a course which is taught by that faculty member)

SQL:

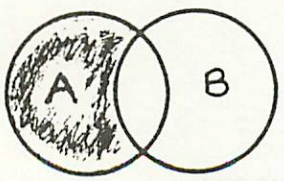
Problem 7.3

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

SQL:

UNION and INTERSECT are used when we want to print a list using the results of two queries. The two queries connected by UNION and INTERSECT may occur on either side of the UNION and INTERSECT, the resulting query is identical either way.

There is one more way to use the results of two queries together. If we have the results of two queries shown as circles A and B below:



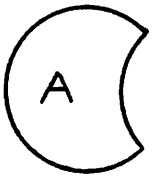
Then

A

MINUS

B

prints the shaded portion. It looks like



a circle with a bite out of it. The computer prints everything returned by query A that is not in query B. If A and B do not intersect, the result would be all of A.

Example 7.3

English: List the courses taken by John Jones that are not taught by John Milton.

Here circle A will represent the courses taken by John Jones and circle B will be the courses taught by John Milton. A MINUS B will yield the desired result. Here queries A and B are a little more complex than the ones we have seen before in this lesson. This is just to illustrate that the queries connected by UNION, INTERSECT and MINUS do not need to be simple.

```
SQL: SELECT COURSE
     FROM TAKING
     WHERE ID =
        SELECT ID
        FROM STUDENT
        WHERE NAME = 'JOHN JONES'
MINUS
     SELECT COURSE
     FROM TEACHING
     WHERE ID =
        SELECT ID
        FROM FACULTY
        WHERE NAME = 'JOHN MILTON'
```

The two queries may not be reversed. If we write

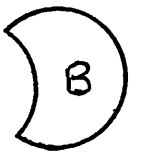
```
     SELECT COURSE
     FROM TEACHING
     WHERE ID =
        SELECT ID
        FROM FACULTY
        WHERE NAME = 'JOHN MILTON'
MINUS
     SELECT COURSE
     FROM TAKING
     WHERE ID =
        SELECT ID
        FROM STUDENT
        WHERE NAME =  'JOHN JONES'
```

We would get the answer to the question: What courses are taught by John Milton that are not taken by John Jones?

Using diagrams:

A MINUS B gives

B MINUS A gives



Problem 7.4

English: List the names of departments located in Keynes hall that have no students from Ohio as majors. (With this simple data base we have to use some pretty stupid queries to illustrate all the points.)

SQL:

Problem 7.5

English: Which department heads are not also committee heads?

SQL:

Problem 7.6

English: List the departments having no faculty members.

(Remember, only departments with faculty members will be in the FACULTY table.)

SQL:

The preceding queries only print one column. It is possible to print many columns. The only requirement is that the columns appear in the same order in the top SELECT clauses of the two queries. If we are printing several columns we want all the values in any particular column to be of the same type.

Example 7.4

English: List the ID's and majors of students from Ohio as well as the ID's and departments of faculty members making more than $10,000.

```
SQL: SELECT ID, DEPARTMENT
     FROM FACULTY
     WHERE SALARY > '10000'
UNION
     SELECT ID, MAJOR
     FROM STUDENT
     WHERE HOMESTATE = 'OHIO'
```

The order of the columns in the two SELECTs is the same - first the ID followed by the department (major). We would see:

247 ECONOMICS

116 HISTORY

9 GEOGRAPHY

•

•

•

Lesson 8. Groups.

Example 8.1

English: List the ID's of students taking more than 5 courses.

We want to count the courses taken by each student and print the ID's of those students taking more than five. Each student has an individual ID. We want the rows grouped so that within any group all the rows have the same ID number:

TAKING

ID	COURSE	SECTION
1	HIST101	1
1	HIST102	2
1	POLSCI115	1
2	ECON105	3
2	ECON219	2
2	ECON201	1
2	MATH101	1
2	PE117	8
2	PE116	4
3	LOGIC12	1

The sets of rows between the lines are called groups. When we use the functions we studied earlier (COUNT, AVG, MAX, MIN and SUM) with groups they work only within a group.

Here is what the query looks like:

```
SQL: SELECT ID
     FROM TAKING
     GROUP BY ID
     HAVING COUNT(COURSE) > '5'
```

The

```
GROUP BY ID
```

says that the rows of the TAKING table are grouped to have identical ID values within each group, as shown in the table above. The HAVING clause specifies group attributes. The function in the HAVING clause works only within groups, so

```
HAVING COUNT(COURSE) > '5'
```

counts the number of courses within a group and specifies that we only want those groups with more than 5 courses in them. We print the ID's (SELECT ID), duplicates are eliminated.

Usually the column named in the SELECT clause is also the one in the GROUP BY. We will go into this in more detail later.

Problem 8.1

English: List the departments having fewer than 10 faculty members.

SQL:

Example 8.2

English: List the departments in which the average faculty member makes more than $20,000.

```
SQL: SELECT DEPT
     FROM FACULTY
     GROUP BY DEPT
     HAVING AVG(SALARY)  > '20000'
```

Problem 8.2

English: List the courses with total enrollment (for all the sections) less than 10.

SQL:

Functions may also be used in the SELECT clause.

Example 8.3

English: List the total payroll for each department.

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

In this example the department was not explicitly asked for, but if the DEPT was left out of the SELECT we would just get a list of sums without knowing which department each sum was for.

When a GROUP BY is used, only certain things may be in the SELECT clause. Only values that are constant for each group may be in the SELECT clause. Obviously the column in the GROUP BY is constant over a group because that is the meaning of GROUP BY. A function is also constant. Each group has only one value for the AVG, COUNT, MAX, MIN and SUM of any column.

Another way to explain the rule for the SELECT clause when a GROUP BY is used is: Each group will result in only one line of output. You must SELECT items that have only 1 value for the group.

Problem 8.3

English: List the number of sections of each course being taught.

SQL:

The two forms can be combined so we have functions in both the SELECT clause and the HAVING clause.

Example 8.4

English: List the department and average salary for the departments in which the top salary is more than $20,000.

```
SQL: SELECT DEPT, AVG(SALARY)
     FROM FACULTY
     GROUP BY DEPT
     HAVING MAX(SALARY) > '20000'
```

Problem 8.4

English: Print how many sections there are for those courses in which the total enrollment (for all sections) is greater than 100.

SQL:

A WHERE clause may also be used in a query using a GROUP BY. The WHERE clause gives the attributes of the rows in the table and the HAVING clause specifies group attributes.

Example 8.5

English: List the departments in which the average salary for women is more than $20,000.

```
SQL: SELECT DEPT
     FROM FACULTY
     WHERE SEX = 'F'
     GROUP BY DEPT
     HAVING AVG(SALARY) > '20000'
```

The order of the clauses is always as shown in Example 8.5. The WHERE clause says that the result consists only of those rows with the attribute SEX = 'F'. So the result is only for women faculty members. The GROUP BY says that the women are grouped in departments. The HAVING clause says that only the groups with the given attribute - AVG(SALARY) > '20000' - are printed.

Problem 8.5

English: List the majors having more than 10 people from Ohio in them.

SQL:

GROUP BY is only used with functions. The problem is to recognize whan we need and when we don't need GROUP BY.

Example 8.6

English: How many people are on the faculty of the geography department? (Same as Example 4.3.)

Here we want the COUNT(ID) but only for the geography department. We don't have to use GROUP BY because we are only considering the geography department and will only get one value printed.

```
SQL: SELECT COUNT(ID)
     FROM FACULTY
     WHERE DEPT = 'GEOGRAPHY'
```

Example 8.7

English: How many people are on the faculty of each department?

Now we want the COUNT(ID) for each department. We must use GROUP BY to achieve this.

```
SQL: SELECT DEPT, COUNT(ID)
     FROM FACULTY
     GROUP BY DEPT
```

We SELECT the DEPT column so we can see which department goes with each faculty count. There are several rows printed as a result of this query.

Notice that the use of the function was dictated by the form of the data base. If the DEPARTMENT table had a SIZE column containing the number of faculty members in each department, the COUNT function would not have been needed for these queries. The solution to Example 8.6 would have been:

```
SELECT SIZE
FROM DEPARTMENT
WHERE DEPT = 'GEOGRAPHY'
```

The solution to Example 8.7 would have been:

```
SELECT DEPT, SIZE
FROM DEPARTMENT.
```

Obviously, the way information is stored in the data base affects your queries.

Lesson 9. Sets.

Example 9.1

English: List the ID's of students who are taking all the courses taught by the art department.

The query:

```
SELECT COURSE
FROM TEACHING
WHERE DEPT = 'ART'
```

produces the names of all courses being taught by the art department. (The COURSES table is not used because it may contain courses not being taught this semester.) Let's think of these course names as a set of course names.

If the TAKING table is grouped by ID, each group contains all the courses being taken by a particular student:

```
SELECT ID
FROM TAKING
GROUP BY ID
```

We want to print out the ID's of students whose set of courses (the courses in his group) contains those courses taught by the art department.

SQL:

```
SELECT ID
FROM TAKING
GROUP BY ID
HAVING SET(COURSE) CONTAINS
   SELECT COURSE
   FROM TEACHING
   WHERE DEPT = 'ART'
```

The top query SELECTs and GROUPs using the ID column. The meaning of SET(COURSE) is the set of courses in a particular group. In other words the set of courses taken by a particular student. SET is a function like MAX, MIN, AVG, COUNT and SUM except that SET produces the set of items in a particular column of a group instead of the single numeric value produced by MAX, MIN, etc.

Let's say that a particular student is taking:

ART101

PHIL202

ART201

ART303

CRAFTS113

The above courses will be his SET(COURSE). If the art department is teaching:

ART101

ART201

ART303

this semester, then this student qualifies to have his ID printed because his set of courses CONTAINS the set of courses taught by the art department.

A student taking

ART201

PHIL202

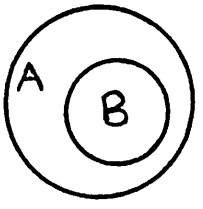
PHIL303

ART303

would not qualify for printing because he is not taking ART101 so

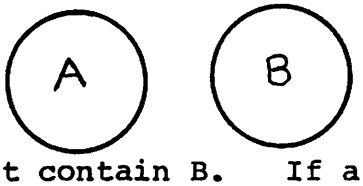
his set does not contain all the courses offered by the art department.

If we say that A is the set of courses taken by a student and circle B is the set of courses taught by the art department then we are looking for the cases when the following diagram represents the situation:

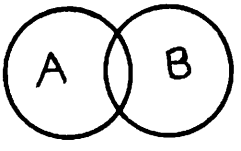


The circle A CONTAINS the circle B. We print the ID's for people whose course diagrams look like the one above.

If a person were taking no art courses the diagram would be:



Obviously A does not contain B. If a person were taking a few but not all of the art courses the diagram would be:



Still, A does not contain B.

Problem 9.1

English: List the ID's of faculty members who teach all the courses taught by the art department.

SQL:

Another possible query is:

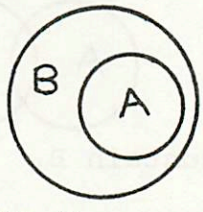
Example 9.2

English: List the ID's of students taking only art classes. (This is different than Example 9.1.)

We want to print the ID's of students whose set of courses is contained in the set of courses taught by the art department.

```
SQL: SELECT ID
     FROM TAKING
     GROUP BY ID
     HAVING SET(COURSE) IN
        SELECT COURSE
        FROM TEACHING
        WHERE DEPT = 'ART'
```

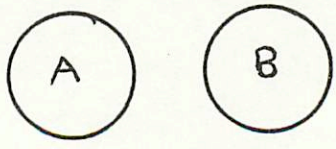
If A is all the courses taken by a student and B is all the courses taught by the art department, then ID's will be printed for students whose diagram is:



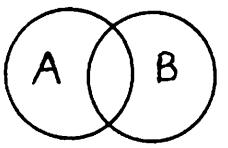
The student may not be taking all the courses taught by the art department but he is taking only courses in art.

IN in the SQL query means that the courses taken by the student must be completely contained in the set of courses offered by the art department. By the diagram, A is IN B. Notice that A IN B is the opposite of A CONTAINS B.

A student taking no art courses would have this diagram



Obviously A is not in B. A student taking art classes as well as some other classes would have the diagram



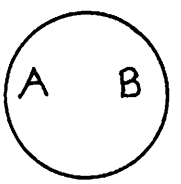
Neither of the two above cases would cause the student's ID to be printed.

Problem 9.2

English: List the departments that have only women faculty members.

SQL:

Another type of query takes care of the situation that A and B must be identical



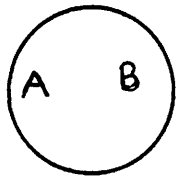
Everything in A is in B and everything in B is in A. We use the = symbol for this case. The sets A and B are identical.

Example 9.3

English: Does any department have both only women faculty members and also have all the women faculty members in the college?

This query might be asked to see if there is blatant sex discrimination in the college. Perhaps home economics is the only department with any women in it at all and it is entirely

women. If A is the women in a department and circle B is all the women teachers in the college, we will print the department name if:




```
SQL: SELECT DEPT
     FROM FACULTY
     GROUP BY DEPT
     HAVING SET(ID) =
        SELECT ID
        FROM FACULTY
        WHERE SEX = 'F'
```

Problem 9.3

English: List the ID's of students taking only art courses but who are taking all the art courses offered.

SQL:

Lesson 10. Joining.

Now we want to write queries that will print or compare columns that come from different tables. We want to bring columns together from two different tables and put them in the SELECT or WHERE clause. This process is called 'joining'.

Example 10.1

English: List the names of the department heads and the buildings their departments are located in.

We want to print NAME from the FACULTY table and BUILDING from the DEPARTMENT table. If we simply wanted to print the names of the department heads we could say:

```
SELECT NAME
FROM FACULTY
WHERE ID =
   SELECT HEAD
   FROM DEPARTMENT
```

We can't print the BUILDING because the topmost SELECT can only contain columns from the FACULTY table. We must put the DEPARTMENT table into the FROM in order to print the BUILDING.

SQL:
```
SELECT NAME, BUILDING
FROM FACULTY, DEPARTMENT
WHERE ID = HEAD
```

Now we have two table names in the FROM clause. We are still printing when the ID equals the HEAD value, but we can put both the ID and HEAD columns in the WHERE clause because of the two table names in the FROM.

Problem 10.2

English: List the course name, title and section for all full courses.

SQL:

Problem 10.3

English: List the names of all women students from Ohio and the courses they are taking.

SQL:

Let's try another variation of joining.

Example 10.3

English: List each student name along with the name of his representative.

Remember, representatives are also students so their names are in the STUDENT table. The student names are also in the STUDENT table. We know that we will print

JOHN JONES JANE DOE

because the REPID value in John Jones's row is 2 which is Jane Doe's ID number. Jane Doe is John Jones's representative. We want to SELECT the NAME column of the STUDENT table for each name, but want to print the NAMEs from different rows. What we do is tell the machine that we have two tables, called A and B, and join those two tables as we have before.

```
SQL: SELECT A.NAME, B.NAME
     FROM STUDENT A, STUDENT B
     WHERE A.REPID = B.ID
```

The FROM clause assigns the names required for the SELECT and WHERE clauses. More attributes may be added with AND's. Do you see which table each name is coming from? Does the representative name come from table A or table B? (Answer: Table B)

Problem 10.4

English: List the names of faculty members whose salary is larger than their committee head's salary. List the names of the committee heads, too.

SQL:

Lesson 11. Combinations of forms.

Example 11.1

English: List the names of the faculty members and the average size of the courses they teach.

The faculty member names are in the FACULTY table but the course sizes are in the TEACHING table. This means we are printing from two tables and must join the FACULTY and TEACHING tables. We also want to print the average class size which means we must have a grouped table.

```
SQL: SELECT NAME, AVG(SIZE)
     FROM FACULTY, TEACHING
     WHERE FACULTY.ID = TEACHING.ID
     GROUP BY TEACHING.ID
```

We could use

```
     GROUP BY FACULTY.ID
```

with the same effect because the WHERE clause says that the ID's from the two tables will be the same. In Lesson 8 we usually SELECTed the column found in the GROUP BY. In Example 11.1 we grouped by ID but printed NAME. The actual rule (also given in Lesson 8) is that we only SELECT values that are constant within a group. Obviously if we group on ID, the NAME will be the same within each group. Only one name is associated with each ID.

Problem 11.1

English: List the course names and the average salary of people teaching those courses.

SQL:

Example 11.2

English: List the names of women faculty members and the average size of the courses they teach.

```
SQL: SELECT NAME, AVG(SIZE)
     FROM FACULTY, TEACHING
     WHERE FACULTY.ID = TEACHING.ID
        AND SEX = 'F'
     GROUP BY TEACHING.ID
```

Problem 11.2

English: List the course names and average salaries of men teaching those courses.

SQL:

Here's another combination.

Problem 11.3

English: List the names of faculty members teaching all the courses taught in the art department.

SQL:

Example 11.3

English: List the names of department heads who are also committee heads.

Do you see that the following solution won't work?

```
SELECT NAME
FROM FACULTY
WHERE ID =
   SELECT HEAD
   FROM DEPARTMENT
INTERSECT
SELECT NAME
FROM FACULTY
WHERE ID =
   SELECT COMHEAD
   FROM FACULTY
```

We could have two different faculty members with the same name. If one of them is a committee head and the other is a department head we would print out that name. This is because we are intersecting names and not ID's.

SQL:

```
SELECT NAME
FROM FACULTY
WHERE ID =
   ( SELECT HEAD
     FROM DEPARTMENT
   INTERSECT
     SELECT COMHEAD
     FROM FACULTY)
```

The INTERSECT yields the ID's for people who are both committee heads and heads of departments. We then use these ID's to print the correct names. The parentheses are necessary to show that we do the inside query (to obtain the ID's) first. If the parentheses were missing we would attempt to intersect the names of department heads with the ID's of committee heads. That would be a meaningless query. Nothing would be returned because probably no department head has a name that is only a number (Prof. 213?).

Problem 11.4

English: List the names of heads of committees with more than 30 people on them.

SQL:

Problem 11.5

English: List the names of committee heads and the names of department heads having more than 30 people in their committee or in their department. (This is a tricky one.)

SQL:

Example 11.4

English: List the departments having no women on their faculties.

At first glance it might seem that the following would work:

```
SELECT DEPT
FROM FACULTY
WHERE SEX = 'F'
GROUP BY DEPT
HAVING COUNT(ID) = '0'
```

The WHERE says that the table has only women in it, therefor the groups will consist only of departments having women in them. The departments with no women in them have no groups in the table.

SQL:
```
SELECT DEPT
FROM DEPARTMENT
MINUS
SELECT DEPT
FROM FACULTY
WHERE SEX = 'F'
```

In this case the top query selects all departments in the school. The bottom one selects those departments having women in them. The MINUS results in the departments having no women being printed.

Problem 11.6

English: List the departments having no students from Ohio as majors.

Lesson 12. Naming queries and using them.

It is sometimes useful to give a name to the result of a query and use the named query as a table in other queries.

Example 12.1

English: Assign a name to the result of the query selecting the name and home state of students in the college.

SQL: NAMESTATE ← SELECT NAME, HOMESTATE
FROM STUDENT

NAMESTATE is the name assigned to the result of the query. The symbol ← is supposed to be an arrow pointing to the left that says to perform the query and then assign the specified name to the result. A table name must start with a letter and contain only letters and numbers. No blank spaces may be in the name. The table could really be called anything: XZ17U would be acceptable as the name. Simple names like A and B are often used, too. Usually, the names should have some meaning just to be easier to use. Now we have the table:

NAMESTATE

NAME	HOMESTATE
JOHN JONES	MASSACHUSETTS
JANE DOE	OHIO

Problem 12.1

ENGLISH: Form a table containing the names of faculty members in the art department.

SQL:

It is also possible to assign names to the columns in the new table. If no new name is assigned to a column, the name is the same as in the table it came from.

Example 12.2

English: Form a table with faculty member names, their salary and the dollar amount of a 10% raise.

SQL: FACRAISE(NAME,PRESENTSAL,RAISE) ← SELECT NAME,SALARY,.10*SALARY
FROM FACULTY

The column names in parentheses (NAME, PRESENTSAL, RAISE) are given to the columns in the FACRAISE table. The SQL query yields 3 columns and these columns are given the names in parentheses. The NAME column from FACULTY has its name unchanged. The SALARY column becomes PRESENTSAL. The column .10 * SALARY had no name since it was just created by this query. Now it has the name RAISE. It is necessary that all columns in a table have a name, so it is only necessary to give the .10 * SALARY column a name. The other two columns could have been left alone. The new table is

FACRAISE

NAME	PRESENTSAL	RAISE
BILL GRANT	20000	2000
JOHN MILTON	14000	1400
ANNE HALL	19000	1900

Problem 12.2

English: Create a table containing course names, section numbers and the number of students needed to fill the class (the number of places left in the class). Don't include full or overflowing classes.

SQL:

Example 12.3

English: Print the name and salary of the person making the most money in each department.

If we make a new table:

MAXSAL

DEPT	MAXSAL

containing the department name (DEPT) and the maximum salary (MAXSAL) we can SELECT NAME from FACULTY and column MAXSAL from table MAXSAL after joining on the DEPT column.

First we create a table containing the department names and the maximun salary in each department:

```
MAXSAL(DEPT,MAXSAL) ← SELECT DEPT, MAX(SALARY)
                      FROM FACULTY
                      GROUP BY DEPT
```

Then we join the MAXSAL table just created with the FACULTY table. Now we have the name of a faculty member, his salary and the maximum salary all in the same row and can compare his salary to the maximum.

```
SELECT NAME, MAXSAL
FROM FACULTY, MAXSAL
WHERE FACULTY.DEPT = MAXSAL.DEPT
   AND SALARY = MAXSAL
```

The entire query is:

```
SQL: MAXSAL(DEPT,MAXSAL) ← SELECT DEPT, MAX(SALARY)
                           FROM FACULTY
                           GROUP BY DEPT

     SELECT NAME, MAXSAL
     FROM FACULTY, MAXSAL
     WHERE FACULTY.DEPT = MAXSAL.DEPT
        AND SALARY = MAXSAL
```

We could have said

```
     SELECT NAME, SALARY
```

Because the attribute says SALARY = MAXSAL. The following sample tables may help in understanding the above query:

FACULTY

NAME	SALARY	DEPT
JACK LEE	14000	ECONOMICS
BILL GRANT	20000	ECONOMICS
JILL EVERS	22000	ECONOMICS
JOHN MILTON	14000	HISTORY
JOE KRAFT	20000	HISTORY
CHRIS WILLS	20000	HISTORY

MAXSAL

DEPT	MAXSAL
ECONOMICS	22000
HISTORY	20000

We will print:

```
     JILL EVERS    22000
     JOE KRAFT     20000
     CHRIS WILLS   20000.
```

This type of query is the most difficult yet and is the most difficult we will cover. You may recognize it from the English query by first seeing, 'most money in each department.' This

phrase requires the MAX function on the SALARY column as well as grouping on the DEPT column. Now, we notice that we want a particular name out of each department. We can't directly obtain this information because the table is grouped on DEPT and will have many names in each group. The names are not constant within a group as required for printing.

We formed a new table (MAXSAL) from the group information. We then joined the FACULTY and MAXSAL tables using the DEPT column so we had the maximum salary associated with the names of the people in the department. We used the added attribute that a person's salary must be equal to the maximum in order for his name and salary to be printed.

Problem 12.3

English: List the ID of the teacher, the course, section and enrollment for the class (or classes) in each department with the largest enrollment.

SQL:

Problem solutions.

2.1
```
SELECT ID
FROM STUDENT
WHERE NAME = 'MARY SMITH'
```

2.2
```
SELECT DEPT, SALARY
FROM FACULTY
WHERE NAME = 'JOHN MILTON'
```

2.3
```
SELECT MAJOR
FROM STUDENT
WHERE NAME = 'BILL WILLIAMS'
```

2.4
```
SELECT COURSE
FROM TAKING
WHERE ID = '2'
```

2.5
```
SELECT COURSE, TITLE
FROM COURSES
WHERE DEPT = 'ECONOMICS'
```

2.6
```
SELECT NAME, MAJOR
FROM STUDENT
```

3.1
```
SELECT COURSE
FROM COURSES
WHERE CREDITS >= '4'
```
(The attribute for this query could also be written as CREDITS > 3 and the query would still be correct. It is possible to have correct solutions to the examples and problems in this text that are different from the solutions presented. If your solution to a problem or example is different from the one given, check its correctness with your instructor.)

3.2
```
SELECT COURSE
FROM TEACHING
WHERE SECTION > '2'
```

3.3
```
SELECT DEPT, BUILDING
FROM DEPARTMENT
WHERE DEPT ≠ BUILDING
```

3.4
```
SELECT NAME
FROM STUDENT
WHERE MAJOR = 'HISTORY'
   AND HOMESTATE = 'OHIO'
```

3.5
```
SELECT NAME
FROM FACULTY
WHERE SEX = 'F'
   AND DEPT = 'ECONOMICS'
   AND SALARY > '15000'
```

```
3.6  SELECT NAME
     FROM FACULTY
     WHERE DEPT = 'ECONOMICS'
        OR SALARY < '10000'

4.1  SELECT COURSE, SECTION, LIMIT + 10
     FROM TEACHING
     WHERE SIZE >= LIMIT

4.2  SELECT COUNT(COURSE)
     FROM TAKING
     WHERE ID = '9'

4.3  SELECT AVG(SIZE)
     FROM TEACHING
     WHERE ID = '312'

4.4  SELECT NAME
     FROM FACULTY
     WHERE SALARY + 5000 > '30000'
```

(You could say SALARY > '25000' by doing the arithmetic yourself, but the way shown above uses the ideas from the lesson.)

```
4.5  SELECT NAME
     FROM FACULTY
     WHERE SALARY =
        SELECT MAX(SALARY)
        FROM FACULTY

4.6  SELECT COURSE
     FROM TEACHING
     WHERE SECTION =
        SELECT MAX(SECTION)
        FROM TEACHING

4.7  SELECT COURSE, SECTION
     FROM TEACHING
     WHERE SIZE <
        SELECT MIN(SIZE)
        FROM TEACHING
        WHERE DEPT = 'MATH'

5.1  SELECT NAME
     FROM FACULTY
     WHERE ID =
        SELECT ID
        FROM TEACHING
        WHERE SECTION = '2'
        AND COURSE = 'ECON105'
```

```
5.2  SELECT NAME
     FROM STUDENT
     WHERE ID =
        SELECT ID
        FROM TAKING
        WHERE COURSE =
           SELECT COURSE
           FROM COURSES
           WHERE CREDITS = '1'

5.3  SELECT NAME, SALARY
     FROM FACULTY
     WHERE ID =
        SELECT HEAD
        FROM DEPARTMENT
        WHERE DEPT = 'ECONOMICS'

5.4  SELECT  NAME
     FROM STUDENT
     WHERE ID =
        SELECT REPID
        FROM STUDENT
        WHERE NAME = 'JOHN JONES'

6.1  SELECT NAME
     FROM FACULTY
     WHERE ID =
        SELECT ID
        FROM TEACHING
        WHERE SIZE + 10 > LIMIT

6.2  SELECT SUM(CREDITS)
     FROM COURSES
     WHERE COURSE =
        SELECT COURSE
        FROM TAKING
        WHERE ID =
           SELECT ID
           FROM STUDENT
           WHERE NAME = 'JANE DOE'

6.3  SELECT COURSE
     FROM COURSES
     WHERE CREDITS = '4'
        AND COURSE =
           SELECT COURSE
           FROM TEACHING
           WHERE ID =
              SELECT ID
              FROM FACULTY
              WHERE NAME = 'BILL GRANT'
```

This problem is one that has another, very different solution.

```
      SELECT COURSE
      FROM TEACHING
      WHERE COURSE =
          ( SELECT COURSE
            FROM COURSES
            WHERE CREDITS = '4' )
      AND ID =
          SELECT ID
          FROM FACULTY
          WHERE NAME = 'BILL GRANT'
```

It is not unusual for there to be more than one way to write a query. Any way that works is acceptable. If any of the solutions given here differ from yours, see the instructor to confirm the correctness of your query.

```
6.4   SELECT SUM(SIZE)
      FROM TEACHING
      WHERE ID =
          SELECT ID
          FROM FACULTY
          WHERE NAME = 'ANNE HALL'

6.5   SELECT NAME
      FROM FACULTY
      WHERE ID =
          SELECT ID
          FROM TEACHING
          WHERE COURSE = 'ECON105'
              OR COURSE = 'POLSCI115'


7.1   SELECT ID
      FROM TAKING
      WHERE COURSE = 'ECON113'
  UNION
      SELECT ID
      FROM TEACHING
      WHERE COURSE = 'ECON113'

7.2   SELECT COURSE
      FROM TAKING
      WHERE ID = '19'
  INTERSECT
      SELECT COURSE
      FROM TEACHING
      WHERE ID = '247'

7.3   SELECT HEAD
      FROM DEPARTMENT
  INTERSECT
      SELECT COMHEAD
      FROM FACULTY
```

Remember, if there is no WHERE clause all the values in the SELECTed column are selected.

```
7.4  SELECT DEPT
     FROM DEPARTMENT
     WHERE BUILDING = 'KEYNES'
   MINUS
     SELECT MAJOR
     FROM STUDENT
     WHERE HOMESTATE = 'OHIO'

7.5  SELECT HEAD
     FROM DEPARTMENT
   MINUS
     SELECT COMHEAD
     FROM FACULTY

7.6  SELECT DEPT
     FROM DEPARTMENT
   MINUS
     SELECT DEPT
     FROM FACULTY
```

(The top SELECT yields all the departments in the school, the bottom SELECT yields all the departments with faculty members. The total query prints the departments having no faculty members.)

```
8.1  SELECT DEPT
     FROM FACULTY
     GROUP BY DEPT
     WHERE COUNT(ID) < '10'

8.2  SELECT COURSE
     FROM TEACHING
     GROUP BY COURSE
     HAVING SUM(SIZE) < '10'

8.3  SELECT COURSE, COUNT(SECTION)
     FROM TEACHING
     GROUP BY COURSE

8.4  SELECT COURSE, COUNT(SECTION)
     FROM TEACHING
     GROUP BY COURSE
     HAVING SUM(SIZE) > '100'

8.5  SELECT MAJOR
     FROM STUDENT
     WHERE HOMESTATE = 'OHIO'
     GROUP BY MAJOR
     HAVING COUNT(NAME) > '10'
```

```
9.1  SELECT ID
     FROM TEACHING
     GROUP BY ID
     HAVING SET(COURSE) CONTAINS
        SELECT COURSE
        FROM TEACHING
        WHERE DEPT = 'ART'
```
Notice that both FROM clauses use the same table, TEACHING.

```
9.2  SELECT DEPT
     FROM FACULTY
     GROUP BY DEPT
     HAVING SET(ID) IN
        SELECT ID
        FROM FACULTY
        WHERE SEX = 'F'
```

```
9.3  SELECT ID
     FROM TAKING
     GROUP BY ID
     HAVING SET(COURSE) =
        SELECT COURSE
        FROM TEACHING
        WHERE DEPT = 'ART'
```

```
10.1 SELECT NAME, COURSE
     FROM STUDENT, TAKING
     WHERE STUDENT.ID = TAKING.ID
        AND HOMESTATE = 'OHIO'
```

```
10.2 SELECT COURSES.COURSE, TITLE, SECTION
     FROM COURSES, TEACHING
     WHERE COURSES.COURSE = TEACHING.COURSE
       AND SIZE >= LIMIT
```
(COURSES.COURSE in the SELECT clause could be replaced by TEACHING.COURSE because they both have the same value due to the join.)

```
10.3 SELECT NAME, COURSE
     FROM STUDENT, TAKING
     WHERE STUDENT.ID = TAKING.ID
        AND SEX = 'F'
        AND HOMESTATE = 'OHIO'
```

```
10.4 SELECT A.NAME, B.NAME
     FROM FACULTY A, FACULTY B
     WHERE A.COMHEAD = B.ID
        AND A.SALARY > B.SALARY
```
(Do you see why it is A.SALARY > B.SALARY?)

```
11.1 SELECT COURSE, AVG(SALARY)
     FROM TEACHING, FACULTY
     WHERE TEACHING.ID = FACULTY.ID
     GROUP BY COURSE
```

```
11.2 SELECT COURSE, AVG(SALARY)
     FROM TEACHING, FACULTY
     WHERE TEACHING.ID = FACULTY.ID
       AND SEX = 'M'
     GROUP BY COURSE

11.3 SELECT NAME
     FROM FACULTY
     WHERE FACULTY ID =
        SELECT ID
        FROM TEACHING
        GROUP BY ID
        HAVING SET(COURSE) CONTAINS
           SELECT COURSE
           FROM TEACHING
           WHERE DEPT = 'ART'
     (This is based on Problem 9.1.)

11.4 SELECT NAME
     FROM FACULTY
     WHERE ID =
        SELECT COMHEAD
        FROM FACULTY
        GROUP BY COMHEAD
        HAVING COUNT(ID) > 30

11.5 SELECT NAME
     FROM FACULTY
     WHERE ID =
          (SELECT COMHEAD
           FROM FACULTY
           GROUP BY COMHEAD
           HAVING COUNT(ID) > 30
        UNION
           SELECT HEAD
           FROM DEPARTMENT
           WHERE DEPT =
              SELECT DEPT
              FROM FACULTY
              GROUP BY DEPT
              HAVING COUNT(ID) > 30)

11.6 SELECT DEPT
     FROM DEPARTMENT
  MINUS
     SELECT MAJOR
     FROM STUDENT
     WHERE HOMESTATE = 'OHIO'

12.1 ARTFACULTY ← SELECT NAME
                  FROM FACULTY
                  WHERE DEPT = 'ART'
     (ARTFACULTY is an arbitrary name, any name will do.)
```

12.2 SEATS(COURSE,SECTION,NUMBER) ← SELECT COURSE,SECTION,LIMIT-SIZE
FROM TEACHING

12.3 MAXSIZE(DEPT,MAXSIZE) ← SELECT DEPT, MAX(SIZE)
FROM TEACHING
GROUP BY DEPT

```
SELECT ID, COURSE, SECTION, SIZE
FROM TEACHING, MAXSIZE
WHERE MAXSIZE.DEPT = TEACHING.DEPT
   AND MAXSIZE = SIZE
```

A P P E N D I X B

TABLET MANUAL

TABLE OF CONTENTS

Lesson 1. Introduction to tables and data bases.

You probably know that computers can store large amounts of information. This information is of little importance to people unless they can get the specific pieces of information they want. Much science fiction such as Star Trek and Star Wars present computers that respond beautifully to spoken questions:

Captain Kirk - "Where is the Klingon third fleet?"

Computer - "Alpha Centauri, first quadrant, fourth sector."

Unfortunately, at present it is not possible to communicate with a computer in this way.

We are going to learn a language for use in asking questions of a computer. This sort of language is called a query language and is used to retrieve information from a computer. The purpose of this course is twofold:

1. To aid anyone who may have need of the computer in the future.
2. To test the reaction of people to the language in order to improve the language.

Your participation in this course will result in your learning about a useful tool and will aid in the development of that tool.

The language we will study was designed for use by people who have no knowledge of computers. The language uses a stilted form of written English to communicate with the computer. This language allows us to state exactly what the computer is to do. The computer is a big, fast, expensive machine. It will do exactly what you ask it to do and this leads to a problem. It is much like the fairy tale about King Midas. King Midas asked that

everything he touched turn to gold. He forgot to exclude food, water, his daughter, etc. The computer is much the same - you get what you ask for.

The first thing we must do is find out what information is in the machine and what it looks like. The information may be of any type and oriented towards any use. Many people with diverse interests use the computer to store information. Historians, sociologists, scientists, librarians and businessmen all store information in computers. Though the particular information is different for the various users it is all stored in simple tables.

The following table contains information about students in some college:

STUDENT

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

The table has a name: STUDENT. Each column in the table also has a name - ID, NAME, SEX, HOMESTATE and MAJOR. There is a row in this table for each student in the college. The actual table in the computer would have many rows, the few rows shown above are just samples.

Often it takes more than one table to store all the information required. For example, the college may also want to store information about the courses each student is currently taking:

TAKING

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

The table named TAKING has three columns - ID, COURSE and SECTION. Each row of this table contains the information about a course being taken by a student. The sample rows shown above show that student number 1 is taking 3 courses. The course name and section number are given in addition to the student ID. From the ID given in the STUDENT table we see that John Jones is taking the three courses, HIST101, HIST102 and POLSCI115. The rows in this table are not really stored in any order, they are shown in a certain order above just as an example.

You might ask why we don't store all the information in one table such as:

STUDENT-TAKING

ID	NAME	SEX	HOMESTATE	MAJOR	COURSE	SECTION
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	HIST101	1
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	HIST102	2
1	JOHN JONES	M	MASSACHUSETTS	HISTORY	POLSCI115	1
2	JANE DOE	F	OHIO	ECONOMICS	ECON105	3
2	JANE DOE	F	OHIO	ECONOMICS	ECON201	1
2	JANE DOE	F	OHIO	ECONOMICS	MATH100	1

This table contains all the information from the STUDENT and TAKING tables. There is a row in the table for each course a student is taking. There are several problems with this table:

1. If John Jones took a semester off, he would not take any classes so there would be no rows in the STUDENT-TAKING table for him. Since this table would be the only one used, none of his personal information (such as homestate and major) would be in the computer. The STUDENT table is separate from the TAKING table so even if he took no courses his row in the STUDENT table would remain.
2. If Jane Doe were to change her major it is necessary to change the value in the MAJOR column for her in three rows of the STUDENT-TAKING table. Only one change is required in the STUDENT table because she only has her information in one row of that table.
3. The STUDENT-TAKING table has a lot of repeated information. Computers can store large amounts of information but there is always a limit. The STUDENT-TAKING table wastes space.

A collection of data such as the above college example is called a data base. This data base consists of only two tables but it is possible for data bases to contain many tables. When we want information from a data base we "query the data base" using our query language.

It is important to remember the exactness that the computer requires. When we state our query we will use the table and

column names contained in the data base. We must use the names exactly as they are spelled in the data base. For example, the column HOMESTATE must be spelled correctly. A different spelling or use of an equivalent word is not allowed. Neither HMOESTATE or BIRTHPLACE are allowed. Often we will ask the computer to print all the values in the column, you must not use the plural form, HOMESTATES, the column name is still HOMESTATE even if you are talking about many values. These requirements hold for all table and column names and must be scrupulously followed.

Perhaps it is time for another analogy. Say that at 9 A.M. every morning you ride your bike to a certain building on campus. Then, one morning you have to go someplace else. You get on your bike and start to ride. The next thing you know - you're at the usual building. Out of habit you got to the wrong place. The point is - you will not blame your bike for not taking you to the right place. You realize that your bike is a machine and does only what you physically direct it to do. Your bike does not know what you want it to do.

It is often hard to realize that the computer is a machine, but it is. Again using the HOMESTATE column as an example, you might think that BIRTHPLACE is equivalent. The computer stores the word "HOMESTATE" in its memory and when you use a column name it looks in its memory for the exact word that you used. It is not enough to be nearly correct. You must be perfectly correct. Later we will see that this exactness is required in all aspects of your dealings with the computer. As in the bicycle analogy the computer does not know what you want.

Several data bases will be presented in this course. This is the first one and will be used through the entire text. We will always use the following form to define a data base:

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

This time we have even named the data base itself. Often a few sample entries will be shown. Remember that the data base will really have many more entries than the few that are shown.

Most of the table and column meanings are probably obvious to you. Here is a bit more explanation:

STUDENT - Contains information about students. There is one row in this table for each student in the school. The REPID column contains the ID of each student's representative in the student senate. The representative is also a student.

TAKING - Contains the courses each student is taking this semester. There is one row for each course each student is taking.

DEPARTMENT - Contains information about the academic departments in the college. The HEAD column contains the ID of the department head.

FACULTY - Contains information about faculty members. There is one row for each faculty member. Each faculty member must be on a college committee. The COMHEAD column contains the ID of the head of a faculty member's committee. The committee head is also a faculty member.

TEACHING - Contains information about the courses being taught this semester. There is one row in this table for each section of each course taught this semester. LIMIT contains the limit on enrollment for a section, SIZE contains the actual size of the class.

COURSES - Contains information about all courses in the college, whether or not they are being taught this semester. There is one row for each course.

This data base is just an example. Each table could have many more columns and there could easily be more tables in the data base. Even so, these are the only tables in our data base. All the queries in this text will be based on these columns and tables.

This data base is used because it contains information familiar to us all. All the data bases in this course will be this sort of familiar data base. Generally, a data base can contain any sort of information. The methods presented in this course are applicable to any data base. At present we are interested in how to work with a data base and not the information stored in it. In terms of the bicycle analogy, it is necessary to learn to ride before you can really go anywhere.

Lesson 2. Queries using only one table.

All our queries will be written in the query language called TABLET. The language and concepts will be presented mainly through examples. The examples will first show the query in English and then give the TABLET version. You will need to use your copy of the sample data base to understand the examples and to do the problems. It is very important that you understand the information in the data base. Whenever you do a problem you should refer to the sample data base so that you will get the table and column names right.

Example 2.1

English: Print John Jones' home state and major.

This query concerns information contained in the NAME, HOMESTATE and MAJOR columns of the STUDENT table. The first step in the TABLET query is to form a new table containing only the columns of interest:

```
FORM JONESINFO
  FROM NAME, HOMESTATE, MAJOR OF STUDENT
```

We now have a new table defined as:

JONESINFO

NAME	HOMESTATE	MAJOR
JOHN JONES	MASSACHUSETTS	HISTORY
JANE DOE	OHIO	ECONOMICS

JONESINFO is the name of this new table. A table name must start with a letter and contain only letters and numbers. No blank spaces may be in the name. The table really could be called anything: XZ17U would be acceptable as the name. Simple names

like A and B are often used, too. Usually, the names should have some meaning just to be easier to use. You must not use names of tables in the database. For example, you cannot use STUDENT as a name for a new table.

JONESINFO has many rows, we only want the row for John Jones. The command:

KEEP ROW OF JONESINFO WHERE NAME = 'JOHN JONES'

will keep that row where the NAME column contains 'JOHN JONES'. Now the JONESINFO table contains only the row:

JOHN JONES MASSACHUSETTS HISTORY

We want to print the HOMESTATE and MAJOR so we say:

PRINT HOMESTATE, MAJOR

Actually, since the KEEP ROW command is referring to the table from the previous FORM command, the OF JONESINFO is not needed. The KEEP ROW command is simply:

KEEP ROW WHERE NAME = 'JOHN JONES'

The complete query is

TABLET:

```
FORM JONESINFO
  FROM NAME, HOMESTATE, MAJOR OF STUDENT
KEEP ROW WHERE NAME = 'JOHN JONES'
PRINT HOMESTATE, MAJOR
```

This TABLET query consists of three TABLET commands - the FORM command, the KEEP ROW command and the PRINT command.

The FORM command forms a new table that contains only the columns of interest. We are interested in the HOMESTATE and MAJOR columns because we want to print this information. The

NAME column is needed for the KEEP ROW command. These three columns are all in the STUDENT table.

The KEEP ROW command tells which row of the table we are interested in. We could also use the plural KEEP ROWS command because the computer understands either, KEEP ROW or KEEP ROWS. We want the row in which the NAME column contains 'John Jones'. We write this in TABLET as: WHERE NAME = 'JOHN JONES'.

The phrase contained in the WHERE clause (in this case NAME = 'JOHN JONES') is called an attribute. An attribute is a quality exhibited by something. For example an attribute of MacIntosh apples is that they are red. In this query language we are talking about the attribute of rows in a table. In Example 2.1 we want to print the row with the attribute that 'JOHN JONES' is in the NAME column. Attributes of rows correspond to attributes of actual things. A row in the student table that has 'OHIO' in the HOMESTATE column corresponds to a student from Ohio.

The quotes around 'JOHN JONES' are necessary to show that this is actually information that is stored in the table. Whenever we refer to a value that is stored in a table we put quotes around it, for example - 'ECONOMICS', '2', 'JANE DOE', 'OHIO' and 'ECON101'.

Actually, numbers do not have to be in quotes, the quotes are optional. The following are all correct

3 '3'

45 '45'

3.8 '3.8'.

Each command must be as shown above. The column names and table name may be different for different queries but the FORM is always

FORM table name

FROM column names OF table name

The KEEP ROWS is always

KEEP ROW(S) WHERE attribute

The PRINT is

PRINT columns

All the queries in this lesson and the next few lessons will use only one table. You can recognize a query that uses only one table by determining which columns are to be printed (they will be used in the PRINT command) and the columns involved in the attribute (used in the KEEP ROWS). If you can find a table containing all these columns, you can use the methods presented in this and the next few lessons.

Now you try one.

Problem 2.1

English: Print Mary Smith's ID number. (Notice that Mary Smith's information is not in the small sample data base given. This is normally the case - if you knew the answer you wouldn't ask the question.)

TABLET:

The answers to the problems are at the end of the text. Try to solve the problem before looking up the solution.

Example 2.2

English: What building is the Economics department in?

The DEPARTMENT table contains the BUILDING column (for the PRINT command) as well as the DEPT column (for the attribute in the KEEP ROWS command).

TABLET:

```
FORM ECONBLDG
  FROM DEPT, BUILDING OF DEPARTMENT
KEEP ROWS WHERE DEPT = 'ECONOMICS'
PRINT BUILDING
```

In this case the attribute is DEPT = 'ECONOMICS'. We want to print the building name contained in the row having a DEPT value of 'ECONOMICS'.

Problem 2.2

English: What department does John Milton teach in and what is his salary?

TABLET:

Problem 2.3

English: What's Bill Williams' major? (Bill is a student.)

TABLET:

All the queries written so far have been assumed to return information from one row of the table. Actually, this assumption

could be wrong. Take Example 2.1

```
FORM JONESINFO
  FROM NAME, HOMESTATE, MAJOR OF STUDENT
KEEP ROW WHERE NAME = 'JOHN JONES'
PRINT HOMESTATE, MAJOR
```

This attribute (NAME = 'JOHN JONES') could be exhibited by several rows, there may be more than one John Jones in the college. If there were three John Jones's we might get the response:

```
MASSACHUSETTS  HISTORY
   NEW YORK    BIOLOGY
     MAINE     ENGLISH.
```

The fact that many people may have the same name is the reason for the ID columns in the tables. If we said:

```
FORM JONESINFO
  FROM ID, HOMESTATE, MAJOR OF STUDENT
KEEP ROWS WHERE ID = '1'
PRINT HOMESTATE, MAJOR
```

and each person had a unique ID, we would be assured of getting:

```
MASSACHUSETTS  HISTORY.
```

Often we do wish to obtain information from many rows of a table.

Example 2.3

English: List the names of all history majors.

TABLET:

```
FORM HISTNAMES
  FROM NAME, MAJOR OF STUDENT
KEEP ROWS WHERE MAJOR = 'HISTORY'
PRINT NAME
```

This query will list the names of all history majors in the school. Many rows have the attribute that the value in the MAJOR column is 'HISTORY'. Again, we see that the columns (NAME, MAJOR) required in the query come from one table (STUDENT).

Problem 2.4

English: List the courses taken by the student whose ID is 2.

TABLET:

Example 2.4

English: What are the majors of students from Massachusetts?

TABLET:

```
FORM MASSMAJORS
  FROM HOMESTATE, MAJOR OF STUDENT
KEEP ROWS WHERE HOMESTATE = 'MASSACHUSETTS'
PRINT MAJOR
```

Obviously, this table will contain many majors for printing. Even if many students have the same major, each major will be printed only once. MASSMAJORS may have many rows in which the MAJOR is 'ENGLISH', but the PRINT command eliminates duplicates from output. We might get a response like:

HISTORY

BIOLOGY

FRENCH

ENGLISH

.

.

We would not get:

HISTORY

BIOLOGY

ENGLISH

HISTORY

FRENCH

BIOLOGY

.

.

.

The underlined majors are duplicates and would not be printed. Obviously, repeated values are really just wasted. The information needs to be printed only once. TABLET deletes duplicates from all lists.

Problem 2.5

English: List the courses and their titles for courses taught in the economics department.

TABLET:

All the TABLET queries have looked like:

```
FORM table name
  FROM column names OF table name
KEEP ROW(S) WHERE column name = 'value'
PRINT column names
```

The FORM command creates a table containing those columns of interest. The KEEP ROWS command keeps those rows satisfying the attribute. The PRINT command prints the information from the specified columns.

The KEEP ROWS command is not always needed:

```
FORM FACNAMES
  FROM NAME  OF FACULTY
PRINT NAME
```

will print the names of all faculty members. A FORM command is always required.

Problem 2.6

English: List the names and majors of all students.

TABLET:

Lesson 3. More on attributes.

All of the attributes have been of the form:

column name = 'value'

It is possible to use comparisons other than =, these are:

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
≠	not equal.

Example 3.1

English: List the names of all faculty members who make less than $11,000.

TABLET:

```
FORM FACSAL
  FROM NAME, SALARY OF FACULTY
KEEP ROWS WHERE SALARY < '11000'
PRINT NAME
```

The attribute of these faculty members is that they make less than $11,000. This attribute is written in TABLET as: SALARY < '11000'. Notice that we do not use the dollar sign ($) or comma (,) in the value '11000'. The computer looks at the value purely as a number with no interpretation of its meaning, so the $ is not used. Commas are just not used in numbers in this language.

This query will print the names of faculty members making less than $11,000, but not the names of people making exactly $11,000. If we want to include those people making exactly $11,000 the attribute would be : SALARY <= '11000'.

Problem 3.1

English: Print the courses worth 4 credits or more.

TABLET:

Problem 3.2

English: List the courses with more than two sections. (Assume that if a course has a section 3 or higher it must have sections 1 and 2.)

TABLET:

Now for another slightly different type of problem.

Example 3.2

English: List the faculty members' ID, course and section number for all full classes.

A class is full if the LIMIT and SIZE columns of the TEACHING table have the same value in the same row. In the sample data base MATH100 is full. We want to print the faculty ID, course and section number. All of this information is in the TEACHING table.

TABLET:

```
FORM FULLCLASS
  FROM ID, COURSE, SECTION, SIZE, LIMIT OF TEACHING
KEEP ROWS WHERE SIZE = LIMIT
PRINT ID, COURSE, SECTION
```

This query will print the desired information for rows having the attribute that the SIZE value and the LIMIT value are the same. Usually, just to be safe, we would say

SIZE >= LIMIT

in case someone counted wrong and allowed 41 students into a class whose limit is 40. If = were used alone, the class would not be printed as full and it certainly should be printed.

Notice that LIMIT is not in quotes. Column names are never quoted. If the attribute were SIZE >= 'LIMIT', the computer would look for the value 'LIMIT' in the SIZE column. That would be incorrect. Also if we were to write the attribute COURSE = ECON105 (instead of COURSE = 'ECON105') the computer would try to find a column named ECON105, which does not exist. Remember, quotes are put around values but not around column names.

Problem 3.3

English: Some departments are located in a building with the same name as the department (engineering for example). List out the department names and building names for which the names are not the same.

TABLET:

At this point we have seen two types of attributes:

1. column name = 'value'
2. column name = column name

In addition, we can use > , <, >=, <=, and ≠ instead of =.

Let's try some more complicated attributes.

Example 3.3

English: What are the ID's of students taking section 2 of ECON119?

Let's state this query in terms of the attributes a row in the TAKING table must have in order to be printed: Print the ID if the section number is 2 and the course is ECON119.

TABLET:

```
FORM ECONID
  FROM ID, COURSE, SECTION OF TAKING
KEEP ROWS WHERE SECTION = '2'
        AND COURSE = 'ECON119'
PRINT ID
```

The AND says that both attributes must apply to this row. In other words the row must have both SECTION = '2' and COURSE = 'ECON119' in order to be printed. The order of the two attributes does not matter. The following WHERE clause has the same effect as the one above:

```
WHERE COURSE = 'ECON119'
  AND SECTION = '2'.
```

Problem 3.4

English: List the names of history majors from Ohio.

TABLET:

Sometimes we may have more than just two attributes that must hold. When this happens we can add more AND's followed by

the other attributes. A KEEP ROWS command with 4 attributes all of which must hold would look like:

```
KEEP ROWS WHERE attribute1
          AND attribute2
          AND attribute3
          AND attribute4
```

Problem 3.5

English: List the names of women economics faculty members making more than $15,000.

TABLET:

Now what if we wanted to print a row if any one or more of a set of attributes holds?

Example 3.4

English: List the names of all students from Ohio and all students from Iowa.

Again let's state the problem with respect to a particular student: Print the name of a student if he is from Ohio or if he is from Iowa.

TABLET:

```
FORM OHIOIOWA
  FROM NAME, HOMESTATE OF STUDENT
KEEP ROWS WHERE HOMESTATE = 'OHIO'
  OR HOMESTATE = 'IOWA'
PRINT NAME
```

The OR means that a person qualifies if he is from Ohio or Iowa.

In terms of rows, we print a row if that row has 'OHIO' or 'IOWA' in the HOMESTATE column.

Notice that the English statement of the query used 'and' but that we used OR in the TABLET query. When a query has many attributes it is necessary to give some thought as to whether AND or OR is to be used. The English statement of the problem is often misleading.

In Example 3.4 it happens that both attributes concerned the same column, HOMESTATE. The attributes do not have to use the same columns.

Problem 3.6

English: List the name of all faculty members who are in the Economics department or who make less than $10,000.

TABLET:

Do you see the difference between AND and OR? AND is used when every attribute must hold, OR is used if any of the attributes may hold. Many attributes may be used with OR. If we had 4 attributes we would write:

```
KEEP ROWS WHERE attribute1
              OR attribute2
              OR attribute3
              OR attribute4.
```

A row would be printed if any of the attributes held.

Lesson 4. Arithmetic and functions.

It is often useful to perform arithmetic in a query.

Example 4.1

English: The economics faculty has decided to aid the finances of the college by having every member take a $500 pay cut per year. List the ID, name and new yearly pay for members of the economics faculty.

TABLET:

```
FORM PAYCUT
  FROM ID, NAME, DEPT, SALARY-500 (AS NEWSAL) OF FACULTY
KEEP ROWS WHERE DEPT = 'ECONOMICS'
PRINT ID, NAME, NEWSAL
```

We have formed a new column containing the employees salary minus $500. This column must have a name so we can refer to it later. AS NEWSAL says to call this column NEWSAL. Our table is:

PAYCUT

ID	NAME	DEPT	NEWSAL
312	BILL GRANT	ECOMOMICS	19500

The possible arithmetic operations are:

*	MULTIPLICATION
/	DIVISION
+	ADDITION
-	SUBTRACTION

Problem 4.1

English: It has been decided to let 10 more people into all the full classes. List the course name, section number and new limit of enrollment for the full classes.

TABLET:

Other calculations are also possible. The following functions are available:

MAX(column name) - the largest (maximum) value in the named column.

MIN(column name) - the smallest (minimum) value in the named column.

AVG(column name) - the average of the values in the named column.

SUM(column name) - the sum (total) of the values in the named column.

COUNT(column name) - the count of the number of elements in the named column.

Parentheses are used to enclose the name of the column on which the function is working. Only one column name may be inside the parentheses. Parentheses are always used with these functions.

Earlier it was said that duplicate values are eliminated from the result of a query. When these functions are used duplicate values are not eliminated from the named column, for example:

Example 4.2

English: Print the average salary of teachers in the art department.

We see that the average salary is not stored in any column. If the DEPARTMENT table had an AVGSAL column containing the average

salary for each department, we would simply say:

```
FORM ARTAVG
  FROM DEPT, AVGSAL OF DEPARTMENT
KEEP ROWS WHERE DEPT = 'ART'
PRINT AVGSAL
```

Since there is no AVGSAL column, we must use the other information stored in the data base to generate the average salary. The point is that if you want to use information that is not explicitly stored in the data base you can use functions (and arithmetic) to generate that information.

TABLET:

```
FORM ARTAVG
  FROM DEPT, SALARY OF FACULTY
KEEP ROWS WHERE DEPT = 'ART'
PRINT AVG(SALARY)
```

After the KEEP ROWS command we have this table:

ARTAVG

DEPT	SALARY

The SALARY column contains the salaries of all faculty members in the art department. In the PRINT we say to calculate the average salary. Duplicates are not eliminated from the SALARY column when the average is calculated.

If the art department has 5 employees, the salaries might be:

10000

12000

16000

16000

12000

with an average of 13200. This is the correct answer. If duplicates were eliminated, we would get 12666, which is incorrect.

The functions MAX, MIN, AVG, and SUM always work on columns that contain numbers. (What would AVG(NAME) do? Nothing meaningful.) COUNT just counts the number of elements of any kind in a column.

Example 4.3

English: How many people are on the faculty of the geography department?

TABLET:

```
FORM GEOGNUM
  FROM NAME, DEPT OF FACULTY
KEEP ROWS WHERE DEPT = 'GEOGRAPHY'
PRINT COUNT(NAME)
```

This function (COUNT) will still work correctly even if several members of the geography faculty had the same names. It would even work in the unlikely, but possible, case that everyone in the department had the same name. Does it make sense that duplicates are not eliminated?

The COUNT in the above could also be COUNT(DEPT) since we are really just counting the number of rows in the table and duplicates are not eliminated. A short form is often used - COUNT(*) - which just means to count the rows. The * is only used with the COUNT. It makes no sense to use the * with other functions because the column must be specified. For example, when using the AVG function you must specify which column you are averaging.

Lesson 5. Adding columns to a table.

What happens if we want to print from columns of one table but the attribute involves columns from another?

Example 5.1

English: List the courses being taught by Bill Grant.

There is no table containing all the necessary information. Course names are in TEACHING, but faculty names are not. The TEACHING table does contain the ID of the person teaching each course. The FACULTY table contains the ID of each person on the faculty. The TEACHING table connects to the FACULTY table through the ID column in each table. We want to create one table that has both the COURSE and NAME columns in it.

First, FORM a table from the COURSE and ID columns of the TEACHING table:

```
FORM COURSENAME
  FROM COURSE, ID OF TEACHING
```

Now, we add the NAME column of the FACULTY table to the COURSENAME table:

```
ADD COLUMN NAME
    OF FACULTY
    BY ID = ID
```

This ADD command adds the NAME column of the FACULTY table to the COURSENAME table. For each row in the COURSENAME table take the value for ID and find a row in FACULTY having this value. Then take the NAME value of that FACULTY row and add it to the COURSENAME row.

Obviously, we will get:

ECON105

MATH101

It is also possible to form the table in the other way:

```
FORM NAMECOURSE
  FROM NAME, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE NAME = 'BILL GRANT'
PRINT COURSE
```

The TAKING and COURSES tables also contain the course names but we chose the TEACHING table because it links to the FACULTY table. Think about how you would answer this query just by using the tables. Wouldn't you match up the ID values as done above?

Problem 5.1

English: List the name of the person teaching section 2 of ECON105.

TABLET:

Example 5.2

English: List the names of teachers of 4 credit courses.

Teachers names are only in the FACULTY table. The TEACHING table contains the courses taught by the faculty members. The COURSES table contains the credits for the courses. FACULTY

links to TEACHING by ID. TEACHING links to COURSES by COURSE, so:

TABLET:

```
FORM TEACHFOUR
  FROM ID, NAME OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
ADD COLUMN CREDITS
    OF COURSES
    BY COURSE = COURSE
KEEP ROWS WHERE CREDITS = '4'
PRINT NAME
```

The table is:

TEACHFOUR

NAME	ID	COURSE	CREDITS

Do you see why we need the COURSE column?

Problem 5.2

English: List the names of students taking 1 credit courses.

TABLET:

It is not necessary that the linking columns in the two tables have the same column name. Up to this point they all have had the same names, but that was by coincidence.

Example 5.3

English: Print the names of students whose major department is located in the Keynes building.

TABLET:

```
FORM KEYNESTUDENTS
  FROM NAME, MAJOR OF STUDENT
ADD COLUMN BUILDING
    OF DEPARTMENT
    BY MAJOR = DEPT
KEEP ROWS WHERE BUILDING = 'KEYNES'
PRINT NAME
```

We want to ADD the BUILDING column. We match up values in the MAJOR column (of KEYNESTUDENTS) and the DEPT column (of DEPARTMENT) and add the BUILDING name when the MAJOR and DEPARTMENT values are equal.

Problem 5.3

English: List the name and salary of the head of the economics department.

TABLET:

Example 5.4

English: What is the name of the head of the committee that John Milton is on?

We want to create a table having both the name of each faculty member as well as the name of his committee head. All faculty member names are in the NAME column of the FACULTY table.

TABLET:

```
FORM JOHNHEAD
  FROM NAME, COMHEAD OF FACULTY
ADD COLUMN NAME (AS HEADNAME)
    OF FACULTY
    BY COMHEAD = ID
KEEP ROWS WHERE NAME = 'JOHN MILTON'
PRINT HEADNAME
```

Let's look at the tables after the FORM

JOHNHEAD

NAME	COMHEAD
BILL GRANT	216
JOHN MILTON	312
ANNE HALL	192

FACULTY

ID	NAME
312	BILL GRANT
152	JOHN MILTON
172	ANNE HALL
192	JOE SMITH

After the ADD command we have a new column in the JOHNHEAD table. We must change the NAME column to HEADNAME so we don't have two columns called NAME in the JOHNHEAD table:

JOHNHEAD

NAME	COMHEAD	HEADNAME
JOHN MILTON	312	BILL GRANT
ANNE HALL	192	JOE SMITH

Do you see why JOHNHEAD was FORMed from the NAME and COMHEAD columns of FACULTY? We want the FACULTY members name and a link to his committee head. COMHEAD is that link.

Problem 5.4

English: Who is John Jones's representative?

TABLET:

Lesson 6. Combinations.

This review lesson will present new combinations of the queries that we have already seen. There are many possible combinations of forms. The purpose of this lesson is to show you how some forms are combined. In the tests you may see combinations that are not presented here.

Example 6.1

English: List the names of teachers who make more than $20,000 and teach MATH100.

TABLET:

```
FORM RICHMATH
  FROM NAME, SALARY, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE COURSE = 'MATH100'
              AND SALARY > '20000'
```

Here we have one attribute from the FACULTY table (SALARY > '20000') and the other from the TEACHING table (COURSE = 'MATH100'). We simply create a table containing all the columns for printing as well as for the attributes and KEEP the desired rows.

Now you try a problem that combines different features than the one just shown.

Problem 6.1

English: List the names of the faculty members whose classes would overflow (be over their limit) if 10 more people signed up.

TABLET:

Example 6.2

English: What is the average salary of teachers of ECON105?

TABLET:

```
FORM AVGSAL
  FROM SALARY, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE COURSE = 'ECON105'
PRINT AVG(SALARY)
```

Here we have combined ADDing a column with a function (AVG).

Problem 6.2

English: How many credits is Jane Doe taking?

TABLET:

Example 6.3

English: What is the total enrollment of courses worth 4 credits or more?

```
TABLET:
     FORM TOTALIN
       FROM SIZE, COURSE OF TEACHING
     ADD COLUMN CREDITS
         OF COURSES
         BY COURSE = COURSE
     KEEP ROWS WHERE CREDITS >= '4'
     PRINT SUM(SIZE)
```

Example 6.4

English: How many history majors from Ohio are taking ECON105?

```
TABLET:
     FORM HISTOHIO
     FROM NAME, MAJOR, HOMESTATE, ID OF STUDENT
     ADD COLUMN COURSE
         OF TAKING
         BY ID = ID
     KEEP ROWS WHERE MAJOR = 'HISTORY'
                 AND HOMESTATE = 'OHIO'
                 AND COURSE = 'ECON105'
     PRINT COUNT(NAME)
```

Problem 6.3

English: Which 4 credit courses are taught by Bill Grant?

```
TABLET:
```

Problem 6.4

English: How many students is Anne Hall teaching?

TABLET:

Problem 6.5

English: List the names of people teaching ECON105 and the names of people teaching POLSCI115.

TABLET:

We could also say:

```
KEEP ROWS OF TEACHERS WHERE ID IN
    ID OF STUDENTID
PRINT ID
```

Do you see why? (Look at the table above.)

The entire query is:

TABLET:

```
FORM STUDENTID
  FROM ID, MAJOR OF STUDENT
KEEP ROWS WHERE MAJOR = 'ECONOMICS'
FORM TEACHERS
  FROM ID, DEPT OF FACULTY
KEEP ROWS WHERE DEPT = 'ECONOMICS'
KEEP ROWS OF STUDENTID WHERE ID IN ID OF TEACHERS
PRINT ID
```

Problem 7.2

English: List the courses taken by student 19 that are taught by faculty member 247. (The student does not have to be in the section taught by the faculty member, just has to be taking a course which is taught by that faculty member.)

TABLET:

Problem 7.3

English: Print the ID's of department heads who are also committee heads.

TABLET:

Lets's try another type of attribute in the KEEP ROWS command.

Example 7.3

English: List the courses taken by John Jones that are not taught by John Milton.

TABLET:

```
FORM JONES
  FROM NAME, ID OF STUDENT
ADD COLUMN COURSE
    OF TAKING
    BY ID = ID
KEEP ROWS OF JONES WHERE NAME = 'JOHN JONES'
FORM MILTON
  FROM NAME, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE NAME = 'JOHN MILTON'
KEEP ROWS OF JONES WHERE COURSE NOT IN
    COURSE OF MILTON
PRINT COURSE
```

Let's look at the tables before the last KEEP ROWS:

JONES

NAME	ID	COURSE
JOHN JONES	1	HIST101
JOHN JONES	1	HIST102
JOHN JONES	1	POLSCI115

MILTON

NAME	ID	COURSE
JOHN MILTON	152	HIST102
JOHN MILTON	152	HIST155
JOHN MILTON	152	HIST301

We see that

HIST101

POLSCI115

will be PRINTed because John Milton does not teach them but John Jones is taking them.

If we had said:

KEEP ROWS OF MILTON WHERE COURSE NOT IN COURSE OF JONES

We would get the answer to the question: What courses are taught by John Milton that are not taken by John Jones? The PRINT would yield -

HIST155

HIST301

- because these courses are taught by John Milton but not taken by John Jones.

Problem 7.4

English: List the names of departments located in Keynes hall that have no students from Ohio as majors. (With this simple data base we have to use some pretty stupid queries to illustrate all the points.)

TABLET:

Problem 7.5

English: Which department heads are not also committee heads?

TABLET:

Problem 7.6

English: List the departments having no faculty members. (Remember, only departments with faculty members will be in the FACULTY table.)

TABLET:

Example 7.4

English: List the ID's and majors of students from Ohio as well as the ID's and departments of faculty members making more than $10,000.

TABLET:

```
FORM STUTABLE
  FROM ID, MAJOR, HOMESTATE OF STUDENT
KEEP ROWS WHERE HOMESTATE = 'OHIO'
KEEP COLUMNS ID, MAJOR
FORM BOTHINFO
  FROM ID, DEPT, SALARY OF FACULTY
KEEP ROWS WHERE SALARY > '10000'
KEEP COLUMNS ID, DEPT
ADD ROWS OF STUTABLE TO BOTHINFO
PRINT ID, DEPT
```

We must have the corresponding columns of the STUTABLE and BOTHINFO tables in the same order, this is achieved by the KEEP COLUMNS commands. Now, when we combine the tables we will get the same type of information in each column. Notice that we print ID, DEPT not ID, MAJOR. The result of the ADD ROWS is in the BOTHINFO table which has ID and DEPT as the names of its columns.

Since the order of the columns in the two KEEP COLUMNS is the same - first the ID followed by the department (major). We would see:

```
247  ECONOMICS
116  HISTORY
  9  GEOGRAPHY
  .
  .
  .
```

Now the attributes of a KEEP ROWS command can be:

1. column name = 'value'
2. column name = column name
3. column name = AVG(column name)
4. column name IN column name OF table
5. column name NOT IN column name of table

In addition, > , < , >= , <= and ≠ may be used instead of =. Also, SUM, COUNT, MAX and MIN may be used instead of AVG.

Lesson 8. Groups.

Example 8.1

English: List the ID's of students taking more than 5 courses.

The TAKING table contains the courses each student is taking. Let's form a table of the ID's and COURSES from the TAKING table:

FORM WORKER

FROM ID, COURSE OF TAKING

We want to count the courses taken by each student and print the ID's of those students taking more than five. Each student has an individual ID. We want the rows grouped so that within any group all the rows have the same ID number:

WORKER

ID	COURSE
1	HIST101
1	HIST102
1	POLSCI115
2	ECON105
2	ECON219
2	ECON201
2	MATH101
2	PE117
2	PE116
3	LOGIC12

The sets of rows between the lines are called groups. When we used the functions we studied earlier (COUNT, AVG, MAX, MIN and SUM) with groups they yield one value for each group.

Here is what the query looks like:

```
TABLET:
     FORM WORKER
       FROM ID, COURSE OF TAKING
     GROUP BY ID
     KEEP GROUPS WHERE COUNT(COURSE) > 5
     PRINT ID
```

The

```
     GROUP BY ID
```

says that the rows of WORKER are grouped to have identical ID values within each group, as shown in the table above. The WHERE clause in a KEEP GROUPS command specifies group attributes. The function in the WHERE clause works within each group, so

```
     KEEP GROUPS WHERE COUNT(COURSE) > '5'
```

counts the number of courses within each group and specifies that we only want those groups with more than 5 courses in them. We print the ID's (PRINT ID). Duplicates are eliminated.

Usually the column named in the PRINT command is also the one in the GROUP BY. In this example we know that there is one row for each course taken by a student. We group the course by ID and count the courses. Then we print the ID's for the groups remaining after the KEEP GROUPS command.

Problem 8.1

English: List the departments having fewer than 10 faculty members.

TABLET:

Example 8.2

English: List the departments in which the average faculty member makes more than $20,000.

TABLET:

```
FORM HIGHPAY
  FROM DEPT, SALARY OF FACULTY
GROUP BY DEPT
KEEP GROUPS WHERE AVG(SALARY) > '20000'
PRINT DEPT
```

Problem 8.2

English: List the courses with total enrollment (for all the sections) less than 10.

TABLET:

Functions may also be used in the PRINT command.

Example 8.3

English: List the total payroll for each department.

TABLET:

```
FORM PAYROLL
  FROM DEPT, SALARY OF FACULTY
GROUP BY DEPT
PRINT DEPT, SUM(SALARY)
```

In this example the department was not explicitly asked for, but if the DEPT was left out of the PRINT command we would just get a list of sums without knowing which department each sum was for.

Problem 8.3

English: List the number of sections of each course being taught.

TABLET:

We can have functions in both the KEEP GROUPS and the PRINT command.

Example 8.4

English: List the department and average salary for the departments in which the top salary is more than $20,000.

TABLET:

```
FORM BIGSAL
  FROM DEPT, SALARY OF FACULTY
GROUP BY DEPT
KEEP GROUPS WHERE MAX(SALARY) > '20000'
PRINT DEPT, AVG(SALARY)
```

Problem 8.4

English: List how many sections there are for those courses in which the total enrollment (for all sections) is greater than 100.

TABLET:

A KEEP ROWS may also be used in a query using a GROUP BY. The KEEP ROWS gives the attributes of the rows in the table and the KEEP GROUPS specifies group attributes.

Example 8.5

English: List the departments in which the average salary for women is more than $20,000.

TABLET:

```
FORM WOMENSAL
  FROM DEPT, SEX, SALARY OF FACULTY
KEEP ROWS WHERE SEX = 'F'
GROUP BY DEPT
KEEP GROUPS WHERE AVG(SALARY) > '20000'
PRINT DEPT
```

The KEEP ROWS says that only those rows with the attribute SEX = 'F' are to be kept. So the result is only for women faculty members. The GROUP BY says that the women are to be grouped in departments. The KEEP GROUPS says that only the groups with the given attribute - AVG(SALARY) > '20000' - are to be left in the table.

Problem 8.5

English: List the majors having more than 10 people from Ohio in them.

TABLET:

Let's try to see when we do and when we don't need to use GROUP BY.

Example 8.6

English: How many people are on the faculty of the geography department? (Same as Example 4.3)

Here we want the COUNT(ID) but only for the geography department. We don't have to use GROUP BY because we are only considering the geography department and will only get one value printed.

TABLET:

```
FORM GEOGNUM
  FROM ID, DEPT OF FACULTY
KEEP ROWS WHERE DEPT = 'GEOGRAPHY'
PRINT COUNT(ID)
```

Example 8.7

English: How many people are on the faculty of each department?

Now we want the COUNT(ID) for each department. We must use GROUP BY to achieve this.

TABLET:

```
FORM DEPTNUMS
  FROM ID, DEPT OF FACULTY
GROUP BY DEPT
PRINT DEPT, COUNT(ID)
```

We PRINT the DEPT column so we can see which department goes with each faculty count. There are several rows printed as a result of this query.

Notice that the use of the function was dictated by the form of the data base. If the DEPARTMENT table had a SIZE column containing the number of faculty members in each department, the COUNT function would not have been needed for these queries. The solution to Example 8.6 would have been:

```
FORM GEOGNUM
  FROM DEPT, SIZE OF DEPARTMENT
KEEP ROWS WHERE DEPT = 'GEOGRAPHY'
PRINT SIZE
```

The solution to Example 8.7 would have been:

```
FORM DEPTNUMS
  FROM DEPT, SIZE OF DEPARTMENT
PRINT DEPT, SIZE
```

Obviously, the way information is stored in the data base affects your queries.

Lesson 9. More about groups.

Example 9.1

English: List the ID's of students who are taking all the courses taught by the art department.

The query:

```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
```

produces the names of all courses being taught by the art department. (The COURSES table is not used because it may contain courses not being taught this semester.) Let's think of these course names as a set of course names.

If a new table is formed from the ID and COURSE columns of the TAKING table and grouped by ID, each group contains all the courses being taken by a particular student:

```
FORM IDCOURSES
  FROM ID, COURSE OF TAKING
GROUP BY ID
```

We want to print out the ID's of students whose set of courses (the courses in his group) contains those courses taught by the art department.

TABLET:

```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
FORM IDCOURSES
  FROM ID, COURSE OF TAKING
GROUP BY ID
KEEP  GROUPS WHERE  COURSE OF  IDCOURSES CONTAINS
       COURSE OF ARTCOURSES
PRINT ID
```

Let's say that a particular student's group is:

9	ART101
9	PHIL202
9	ART201
9	ART303
9	CRAFTS113

If the art department is teaching:

ART101

ART201

ART303

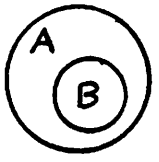
this semester, then this student qualifies to have his ID printed because his COURSE column CONTAINS the courses taught by the art department.

A student whose group is:

16	ART201
16	PHIL202
16	PHIL303
16	ART303

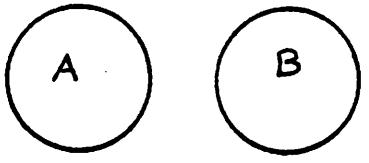
would not qualify for printing because he is not taking ART101 so his group does not contain all the courses offered by the art department.

If we say that circle A represents the courses taken by a student and circle B represents the courses taught by the art department then we are looking for the cases when the following diagram represents the situation:

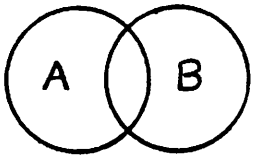


The circle A CONTAINS the circle B. We print the ID's for people whose course diagrams look like the one above.

If a person were taking no art courses the diagram would be:



Obviously A does not contain B. If a person were taking a few but not all of the art courses the diagram would be:



Still, A does not contain B.

Problem 9.1

English: List the ID's of faculty members who teach all the courses taught by the art department.

TABLET:

Another possible query is:

Example 9.2

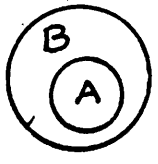
English: List the ID's of students taking only art classes. (This is different from Example 9.1.)

We want to print the ID's of students whose courses are contained in the courses taught by the art department.

TABLET:

```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
FORM IDCOURSES
  FROM ID, COURSES OF TAKING
GROUP BY ID
KEEP GROUPS WHERE COURSE OF IDCOURSES IN
    COURSE OF ARTCOURSES
PRINT ID
```

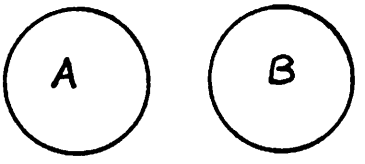
If A is all the courses taken by a student and B is all the courses taught by the art department, then ID's will be printed for students whose diagram is:



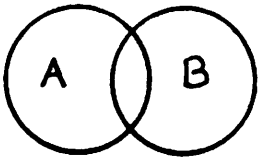
The student may not be taking all the courses taught by the art department but he is taking only courses in art.

IN in the TABLET query means that the courses taken by the student must be completely contained in the set of courses offered by the art department for a group to be kept. By the diagram, A is IN B. Notice that A IN B is the opposite of A CONTAINS B.

A student taking no art courses would have this diagram



Obviously A is not in B. A student taking art classes as well as some other classes would have the diagram



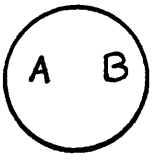
Neither of the two above cases would cause the student's ID to be printed.

Problem 9.2

English: List the departments that have only women faculty members.

TABLET:

Another type of query takes care of the situation where A and B must be identical

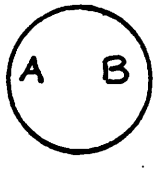


Everything in A is in B and everything in B is in A. We use the = symbol for this case.

Example 9.3

English: Does any department have both only women faculty members and also have all the women faculty members in the college?

This query might be asked to see if there is blatant sex discrimination in the college. Perhaps home economics is the only department with any women in it at all and it is entirely women. If A is the women in a department and circle B is all the women teachers in the college, we will print the department name if:



TABLET:

```
FORM WOMEN
  FROM ID,  SEX OF FACULTY
KEEP ROWS WHERE SEX = 'F'
FORM ALLWOMEN
  FROM ID, DEPT OF FACULTY
GROUP BY DEPT
KEEP GROUPS WHERE ID =
    ID OF WOMEN
PRINT DEPT
```

Problem 9.3

English: List the ID's of students taking only art courses but who are taking all the art courses offered.

TABLET:

Lesson 10. More practice with ADDing columns.

Example 10.1

English: List the names of the department heads and the buildings their departments are located in.

We want to create a table containing the department heads' names and building. The name is in the FACULTY table and the BUILDING is in the DEPARTMENT table. We need to link the ID to the HEAD column.

TABLET:

```
FORM HEADBLDG
  FROM NAME, ID OF FACULTY
ADD COLUMN BUILDING
    OF DEPARTMENT
    BY ID = HEAD
PRINT NAME, BUILDING
```

The ADD COLUMN command takes the following two tables:

HEADBLDG

NAME
BILL GRANT
JOHN MILTON
ANNE HALL

DEPARTMENT

ID	HEAD	BUILDING
312	172	BILLINGS
152	295	BILLINGS
172	312	KEYNES

The result is:

HEADBLDG

NAME	ID	BUILDING
BILL GRANT	312	KEYNES
ANNE HALL	172	BILLINGS

Finally, we PRINT the NAME and BUILDING so we get:

BILL GRANT KEYNES

ANNE HALL BILLINGS

The sample data base does not show John Milton's building. His building would be in the entire data base and would be printed. This is just a sample printing.

Example 10.2

English: List the names of the teachers, the course and the section for all full classes.

TABLET:

```
FORM FULL
  FROM NAME, ID OF FACULTY
ADD COLUMNS COURSE, SECTION, SIZE, LIMIT
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE SIZE >= LIMIT
PRINT NAME, COURSE, SECTION
```

For Example 10.2 we will get at least:

```
BILL GRANT      MATH101        2
JOHN MILTON     HIST101        1
```

The sample data base shows that these are both full classes. Bill Grant is teaching both ECON105 and MATH101, but only MATH101 is full so MATH101 is printed but ECON105 is not. It may be helpful to draw lines and circles on your sample data base as was done for Example 10.1. Do you see why

```
JOHN MILTON     HIST101        1
```

is printed?

Problem 10.1

English: Print the names of students from Ohio and the courses they are taking.

TABLET:

Problem 10.2

English: List the course name, title and section for all full courses.

TABLET:

Problem 10.3

English: List the names of all women students from Ohio and the courses they are taking.

TABLET:

Let's try another variation.

Example 10.3

English: List each student name along with the name of his representative.

Remember, representatives are also students so their names are in the STUDENT table. The student names are also in the STUDENT table. We know that we will print -

JOHN JONES JANE DOE

- because the REPID value in John Jones's row is 2 which is Jane Doe's ID number. Jane Doe is John Jones's representative. We need to create a table containing each student's name as well as his representative's name in the same row.

TABLET:

```
FORM REPRESENTATIVES
  FROM NAME, REPID OF STUDENT
ADD COLUMN NAME (AS REPNAME)
    OF STUDENT
    BY REPID = ID
PRINT NAME, REPNAME
```

Here we end up with the table:

REPRESENTATIVES

NAME	REPID	REPNAME
JOHN JONES	2	JANE DOE

Do you see why the FORM uses REPID, not ID?

Problem 10.4

English: List the names of faculty members whose salary is larger than their committee head's salary. Print the names of the committee heads, too.

TABLET:

Lesson 11. Combinations.

Example 11.1

English: List the names of the faculty members and the average size of the courses they teach.

The faculty member names are in the FACULTY table but the course sizes are in the TEACHING table. This means we are printing from two tables and must form a table containing columns from both the FACULTY and TEACHING tables. We also want to print the average class size which means we must have a grouped table.

TABLET:

```
FORM FACAVG
  FROM NAME, ID OF FACULTY
ADD COLUMN SIZE
    OF TEACHING
    BY ID = ID
GROUP BY ID
PRINT NAME, AVG(SIZE)
```

Problem 11.1

English: List the course names and the average salary of people teaching those courses.

TABLET:

Example 11.2

English: List the names of women faculty members and the average size of the courses they teach.

TABLET:

```
FORM WOMENAVG
  FROM NAME, SEX, ID OF FACULTY
ADD COLUMN SIZE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE SEX = 'F'
GROUP BY ID
PRINT NAME, AVG(SIZE)
```

Problem 11.2

English: List the course names and average salaries of men teaching those courses.

TABLET:

Here's another combination.

Problem 11.3

English: List the names of faculty members teaching all the courses taught in the art department.

TABLET:

Example 11.3

English: List the names of department heads who are also committee heads.

Do you see why the following won't work?

```
FORM HEADNAMES
  FROM HEAD OF DEPARTMENT
ADD COLUMN NAME
    OF FACULTY
    BY ID = HEAD
FORM COMNAMES
  FROM COMHEAD OF FACULTY
ADD COLUMN NAME
    OF FACULTY
    BY ID = COMHEAD
KEEP ROWS OF COMNAMES WHERE COMNAME IN
    NAME OF HEADNAME
PRINT NAME
```

We could have two different faculty members with the same name. If one of them is a committee head and the other is a department head we would print out that name. This is because the KEEP ROWS may find the same name in both the HEADNAMES and COMNAMES tables, but they are not the same person (do not have the same ID).

TABLET:

```
FORM HEADID
  FROM HEAD OF DEPARTMENT
FORM COMHEADID
  FROM COMHEAD OF FACULTY
KEEP ROWS OF HEADID WHERE HEAD IN
    COMHEAD OF COMHEADID
ADD COLUMN NAME
    OF FACULTY
    BY HEAD = ID
PRINT NAME
```

After the first FORM command, table HEADID contains the ID's of department heads. After the KEEP ROWS command, HEADID contains the ID's of those department heads who are also committee heads. We then ADD the NAME column to the HEADID table using the unique ID's. Finally, we PRINT the names.

Problem 11.4

English: List the names of heads of committees with more than 30 people on them.

TABLET:

Problem 11.5

English: List the names of committee heads and the names of department heads having more than 30 people in their committee or in their department. (This is a tricky one.)

TABLET:

Example 11.4

English: List the departments having no women on their faculties.

At first glance it might seem that the following would work:

```
FORM NOWOMEN
  FROM DEPT, ID, SEX OF FACULTY
KEEP ROWS WHERE SEX = 'F'
GROUP BY DEPT
KEEP GROUPS WHERE COUNT(ID) = '0'
PRINT DEPT
```

The KEEP ROWS says that the table has only women in it, therefore the groups will consist only of departments having women in them. But, the departments with no women in them have no groups in the table and nothing will be in the table for printing.

TABLET:

```
FORM ALLDEPTS
  FROM DEPT OF DEPARTMENT
FORM WOMDEPTS
  FROM SEX, DEPT OF FACULTY
KEEP ROWS WHERE SEX = 'F'
KEEP ROWS OF ALLDEPTS WHERE  DEPT
    NOT IN DEPT OF WOMDEPTS
PRINT DEPT
```

In this case the top FORM selects all departments in the school. The second FORM selects those departments having women in them. The KEEP ROWS ... NOT IN results in the departments having no women being kept in the ALLDEPTS table.

Problem 11.6

English: List the departments having no students from Ohio as majors.

TABLET:

Lesson 12. One more way to ADD columns.

Example 12.1

English: Create a table of the names and homestates of students in the college.

This is simple, we just need to say:

TABLET:

```
FORM NAMESTATE
  FROM NAME, HOMESTATE OF STUDENT
```

Problem 12.1

English: Create a table containing the names of faculty members in the art department.

TABLET:

Example 12.2

English: Create a table with faculty member names, their salary and the dollar amount of a 10% raise.

The simple way to do this is:

```
FORM FACRAISE
  FROM NAME, SALARY, .10 * SALARY (AS RAISE) OF FACULTY
```

The column containing .10 * SALARY must be given a name because all columns must have names. (AS RAISE) says that the new column is named RAISE.

Another way is:

TABLET:

```
FORM FACRAISE
  FROM NAME, SALARY OF FACULTY
ADD COLUMN .10 * SALARY (AS RAISE)
```

We do not need a BY or OF clause because we are merely adding a column that is calculated from the columns of FACRAISE itself. We will use a variation of this query shortly.

In either case we end up with the table:

FACRAISE

NAME	SALARY	RAISE
BILL GRANT	20000	2000
JOHN MILTON	14000	1400
ANNE HALL	19000	1900

Problem 12.2

English: Create a table containing course names, section numbers and the number of students needed to fill the class (the number of places left in the class).

TABLET:

Example 12.3

English: List the name and salary of the person making the most money in each department.

```
TABLET:
     FORM NAMEMAX
       FROM NAME, SALARY, DEPT OF FACULTY
     GROUP BY DEPT
     ADD COLUMN MAX(SALARY) (AS MAXSAL)
     KEEP ROWS WHERE SALARY = MAXSAL
     PRINT NAME, SALARY
```

As has been said before the MAX(SALARY) calculates a value for each group. This is an extension of Example 12.2. We are calculating MAXSAL from the groups in table NAMEMAX. The resulting table is:

NAMEMAX

NAME	SALARY	DEPT	MAXSAL
JACK LEE	14000	ECONOMICS	22000
BILL GRANT	20000	ECONOMICS	22000
JILL EVERS	22000	ECONOMICS	22000
JOHN MILTON	14000	HISTORY	20000
JOE KRAFT	20000	HISTORY	20000
CHRIS WILLS	20000	HISTORY	20000

Then we KEEP ROWS for people making the maximum salary and PRINT:

```
JILL EVERS   22000
JOE KRAFT    20000
CHRIS WILLS  20000
```

Problem 12.3

English: List the ID of the teacher, the course, section and enrollment for the class (or classes) in each department with the largest enrollment.

TABLET:

Problem solutions.

2.1
```
FORM MARYID
  FROM NAME, ID OF STUDENT
KEEP ROWS WHERE NAME = 'MARY SMITH'
PRINT ID
```

2.2
```
FORM MILTONINFO
  FROM NAME, DEPT, SALARY OF FACULTY
KEEP ROWS WHERE NAME = 'JOHN MILTON'
PRINT DEPT,SALARY
```

2.3
```
FORM BILLSMAJOR
  FROM NAME, MAJOR OF STUDENT
KEEP ROWS WHERE NAME = 'BILL WILLIAMS'
PRINT MAJOR
```

2.4
```
FORM TWOCOURSES
  FROM ID, COURSE OF TAKING
KEEP ROWS WHERE ID = '2'
PRINT COURSE
```

2.5
```
FORM ECONCOURSES
  FROM COURSE, DEPT, TITLE OF COURSES
KEEP ROWS WHERE DEPT = 'ECONOMICS'
PRINT COURSE, TITLE
```

2.6
```
FORM NAMEMAJOR
  FROM NAME, MAJOR OF STUDENT
PRINT NAME, MAJOR
```

3.1
```
FORM FOURCREDIT
  FROM COURSE, CREDITS OF COURSES
KEEP ROWS WHERE CREDITS >= '4'
PRINT COURSE
```
(The attribute for this query could also be written as CREDITS > 3 and the query would still be correct. It is possible to have correct solutions to the examples and problems in this text that are different from the solutions presented. If your solution to a problem or example is different from the one given, check its correctness with your instructor.)

3.2
```
FORM MANYSECTION
  FROM COURSE, SECTION OF TEACHING
KEEP ROWS WHERE SECTION > '2'
PRINT COURSE
```

3.3
```
FORM SAMEBLDG
  FROM DEPT, BUILDING OF DEPARTMENT
KEEP ROWS WHERE DEPT ≠ BUILDING
PRINT DEPT, BUILDING
```

3.4
```
FORM HISTOHIO
  FROM NAME, HOMESTATE, MAJOR OF STUDENT
KEEP ROWS WHERE MAJOR = 'HISTORY'
              AND HOMESTATE = 'OHIO'
PRINT NAME
```

3.5
```
FORM WOMENFAC
  FROM NAME, SEX, DEPT, SALARY OF FACULTY
KEEP ROWS WHERE SEX = 'F'
              AND DEPT = 'ECONOMICS'
              AND SALARY > '15000'
PRINT NAME
```

3.6
```
FORM POORECON
  FROM NAME, DEPT, SALARY OF FACULTY
KEEP ROWS WHERE DEPT = 'ECONOMICS'
               OR SALARY < '10000'
PRINT NAME
```

4.1
```
FORM BIGGER
  FROM COURSE, SECTION, SIZE, LIMIT OF TEACHING
KEEP ROWS WHERE SIZE  >=  LIMIT
PRINT COURSE, SECTION, LIMIT+10
```

4.2
```
FORM NINECOURSES
  FROM ID, COURSE OF TAKING
KEEP ROWS WHERE ID = '9'
PRINT COUNT(COURSE)
```

4.3
```
FORM AVGSIZE
  FROM ID, SIZE OF TEACHING
KEEP ROWS WHERE ID = '312'
PRINT AVG(SIZE)
```

4.4
```
FORM BIGSAL
  FROM NAME, SALARY OF FACULTY
KEEP ROWS WHERE SALARY+5000  >  '30000'
PRINT NAME
```
(You could say SALARY > '25000' by doing the arithmetic youself, but the way shown above uses the ideas from the lesson.)

4.5
```
FORM BIGSAL
  FROM NAME, SALARY OF FACULTY
KEEP ROWS WHERE SALARY = MAX(SALARY)
PRINT NAME
```

4.6
```
FORM BIGCOURSE
  FROM COURSE, SECTION OF TEACHING
KEEP ROWS WHERE SECTION = MAX(SECTION)
PRINT COURSE
```

```
4.7   FORM MATHSIZE
        FROM DEPT, SIZE OF TEACHING
      KEEP ROWS WHERE DEPT = 'MATH'
      FORM SMALL
        FROM COURSE, SECTION, SIZE OF TEACHING
      KEEP ROWS WHERE SIZE < MIN(SIZE OF MATHSIZE)
      PRINT COURSE, SECTION

5.1   FORM ECON105TEACHER
        FROM ID, NAME OF FACULTY
      ADD COLUMNS COURSE, SECTION
          OF TEACHING
          BY ID = ID
      KEEP ROWS WHERE COURSE = 'ECON105'
                  AND SECTION = '2'
      PRINT NAME

5.2   FORM ONECREDIT
        FROM NAME, ID OF STUDENT
      ADD COLUMN COURSE
          OF TAKING
          BY ID = ID
      ADD COLUMN CREDITS
          OF COURSES
          BY COURSE = COURSE
      KEEP ROWS WHERE CREDITS = '1'
      PRINT NAME

5.3   FORM HEADSAL
        FROM NAME, SALARY, ID OF FACULTY
      ADD COLUMN DEPT
          OF DEPARTMENT
          BY ID = HEAD
      KEEP ROWS WHERE DEPT = 'ECONOMICS'
      PRINT NAME, SALARY

5.4   FORM JONESREP
        FROM NAME, REPID OF STUDENT
      ADD COLUMN NAME (AS REPNAME)
          OF STUDENT
          BY REPID = ID
      KEEP ROWS WHERE NAME = 'JOHN JONES'
      PRINT REPNAME

6.1   FORM ALMOSTFULL
        FROM NAME, ID OF FACULTY
      ADD COLUMNS  SIZE, LIMIT
          OF TEACHING
          BY ID = ID
      KEEP ROWS WHERE SIZE+10 > LIMIT
      PRINT NAME
(The computer accepts COLUMN and COLUMNS.)
```

6.2
```
FORM JANECREDITS
  FROM NAME, ID OF STUDENT
ADD COLUMN COURSE
    OF TAKING
    BY ID = ID
ADD COLUMN CREDITS
    OF COURSES
    BY COURSE = COURSE
KEEP ROWS WHERE NAME = 'JANE DOE'
PRINT SUM(CREDITS)
```

6.3
```
FORM GRANTFOUR
  FROM NAME, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
ADD COLUMN CREDITS
    OF COURSES
    BY COURSE = COURSE
KEEP ROWS WHERE NAME = 'BILL GRANT'
            AND CREDITS = '4'
PRINT COURSE
```

6.4
```
FORM HALLSTUDENTS
  FROM NAME, ID OF FACULTY
ADD COLUMN SIZE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE NAME = 'ANNE HALL'
PRINT SUM(SIZE)
```

6.5
```
FORM TABLE
  FROM NAME, ID OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
KEEP ROWS WHERE COURSE = 'ECON105'
            OR COURSE = 'POLSCI115'
PRINT NAME
```

7.1
```
FORM STUDENTID
  FROM ID, COURSE OF TAKING
KEEP ROWS WHERE COURSE = 'ECON113'
KEEP COLUMN ID
FORM FACULTYID
  FROM ID, COURSE OF TEACHING
KEEP ROWS WHERE COURSE = 'ECON113'
KEEP COLUMN ID
ADD ROWS OF FACULTYID TO STUDENTID
PRINT ID
```

```
7.2   FORM COURSE19
        FROM ID, COURSE OF TAKING
      KEEP ROWS WHERE ID = '19'
      FORM COURSE247
        FROM ID, COURSE OF TEACHING
      KEEP ROWS WHERE ID = '247'
      KEEP ROWS OF COURSE19 WHERE COURSE IN
          COURSE OF COURSE247
      PRINT COURSE

7.3   FORM HEADS
        FROM HEAD OF DEPARTMENT
      FORM COMMHEADS
        FROM COMHEAD OF FACULTY
      KEEP ROWS OF HEADS WHERE HEAD IN COMHEAD
          OF COMHEADS
      PRINT HEAD
```

(We want all the department and committee heads so we do not need any other KEEP ROWS commands.)

```
7.4   FORM KEYNESDEPT
        FROM DEPT, BUILDING OF DEPARTMENT
      KEEP ROWS WHERE BUILDING = 'KEYNES'
      FORM OHIO
        FROM HOMESTATE, MAJOR OF STUDENT
      KEEP ROWS WHERE HOMESTATE = 'OHIO'
      KEEP ROWS OF KEYNESDEPT WHERE DEPT
          NOT IN MAJOR OF OHIO
      PRINT DEPT

7.5   FORM HEADS
        FROM HEAD OF DEPARTMENT
      FORM COMHEADS
        FROM COMHEAD OF FACULTY
      KEEP ROWS OF HEADS WHERE HEAD NOT IN
          COMHEAD OF COMHEADS
      PRINT HEAD

7.6   FORM DEPTS
        FROM DEPT OF DEPARTMENT
      FORM HASFACULTY
        FROM DEPT OF FACULTY
      KEEP ROWS OF DEPTS WHERE DEPT
          NOT IN DEPT OF HASFACULTY
    PRINT DEPT
```

(If a department has a row in the DEPARTMENT table but no rows in the FACULTY table, it has no faculty members.)

```
8.1   FORM SMALLDEPT
        FROM NAME, DEPT OF FACULTY
      GROUP BY DEPT
      KEEP GROUPS WHERE COUNT(NAME) < 10
      PRINT DEPT
```

8.2
```
FORM SMALLCOURSE
  FROM COURSE, SIZE OF TEACHING
GROUP BY COURSE
KEEP GROUPS WHERE SUM(SIZE) < 10
PRINT COURSE
```

8.3
```
FORM HOWMANY
  FROM COURSE, SECTION OF TEACHING
GROUP BY COURSE
PRINT COURSE, COUNT(SECTION)
```

8.4
```
FORM BIGCOURSE
  FROM COURSE, SECTION, SIZE OF TEACHING
GROUP BY COURSE
KEEP GROUPS WHERE SUM(SIZE) > 100
PRINT COURSE, COUNT(SECTION)
```

8.5
```
FORM OHIOMAJORS
  FROM NAME, HOMESTATE, MAJOR OF STUDENT
KEEP ROWS WHERE HOMESTATE = 'OHIO'
GROUP BY MAJOR
KEEP GROUPS WHERE COUNT(NAME) > 10
PRINT MAJOR
```

9.1
```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
FORM ARTTEACHERS
  FROM ID, COURSE OF TEACHING
GROUP BY ID
KEEP GROUPS WHERE COURSE CONTAINS
    COURSE OF ARTCOURSES
PRINT ID
```

9.2
```
FORM WOMEN
  FROM ID, SEX OF FACULTY
KEEP ROWS WHERE SEX = 'F'
FORM WOMENDEPTS
  FROM DEPT, ID OF FACULTY
GROUP BY DEPT
KEEP GROUPS WHERE ID IN
    ID OF WOMEN
PRINT DEPT
```

9.3
```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
FORM ARTLOVER
  FROM ID, COURSE OF TAKING
GROUP BY ID
KEEP GROUPS WHERE COURSE =
    COURSE OF ARTCOURSES
PRINT ID
```

```
10.1  FORM OHIOSTUDENTS
        FROM ID, NAME, HOMESTATE OF STUDENT
      ADD COLUMN COURSE
          OF TAKING
          BY ID = ID
      KEEP ROWS WHERE HOMESTATE = 'OHIO'
      PRINT NAME, COURSE

10.2  FORM FULLCOURSE
        FROM COURSE, SECTION, SIZE, LIMIT OF TEACHING
      ADD COLUMN TITLE
          OF COURSES
          BY COURSE = COURSE
      KEEP ROWS WHERE SIZE >= LIMIT
      PRINT COURSE, TITLE, SECTION

10.3  FORM WOMENOHIO
        FROM NAME, SEX, HOMESTATE, ID OF STUDENT
      ADD COLUMN COURSE
          OF TAKING
          BY ID = ID
      KEEP ROWS WHERE SEX = 'F'
                  AND HOMESTATE = 'OHIO'
      PRINT NAME, COURSE

10.4  FORM TABLE
        FROM NAME, SALARY, COMHEAD OF FACULTY
      ADD COLUMN NAME (AS HEADNAME), SALARY (AS HEADSAL)
          OF FACULTY
          BY COMHEAD = ID
      KEEP ROWS WHERE SALARY > HEADSAL
      PRINT NAME, HEADNAME

11.1  FORM COURSEAVG
        FROM COURSE, ID OF TEACHING
      ADD COLUMN SALARY
          OF FACULTY
          BY ID = ID
      GROUP BY COURSE
      PRINT COURSE, AVG(SALARY)

11.2  FORM MENAVG
        FROM SEX, SALARY, ID OF FACULTY
      ADD COLUMN COURSE
          OF TEACHING
          BY ID = ID
      KEEP ROWS WHERE SEX = 'M'
      GROUP BY COURSE
      PRINT COURSE, AVG(SALARY)
```

11.3
```
FORM ARTCOURSES
  FROM COURSE, DEPT OF TEACHING
KEEP ROWS WHERE DEPT = 'ART'
FORM ARTNAMES
  FROM ID, NAME OF FACULTY
ADD COLUMN COURSE
    OF TEACHING
    BY ID = ID
GROUP BY ID
KEEP GROUPS WHERE COURSE CONTAINS
    COURSE OF ARTCOURSES
PRINT NAME
```

11.4
```
FORM BIGCOMHEAD
  FROM COMHEAD, NAME OF FACULTY
GROUP BY COMHEAD
ADD COLUMN NAME
    OF FACULTY
    BY COMHEAD = ID
KEEP GROUPS WHERE COUNT(NAME) > 30
PRINT NAME
```

11.5
```
FORM BIGCOMHEAD
  FROM COMHEAD, NAME OF FACULTY
GROUP BY COMHEAD
KEEP GROUPS WHERE COUNT(NAME) > 30
ADD COLUMN NAME
    OF FACULTY
    BY COMHEAD = ID
KEEP COLUMN NAME
FORM BIGDEPTHEAD
  FROM HEAD, DEPT OF DEPARTMENT
ADD COLUMN NAME
  OF FACULTY
  BY DEPT = DEPT
GROUP BY HEAD
KEEP GROUPS WHERE COUNT(ID) > 30
KEEP COLUMN NAME
ADD ROWS OF BIGDEPTHEAD TO BIGCOMHEAD
PRINT NAME
```

11.6
```
FORM ALLDEPTS
  FROM DEPT OF DEPARTMENT
FORM OHIOMAJORS
  FROM MAJOR, HOMESTATE OF STUDENT
KEEP ROWS WHERE HOMESTATE = 'OHIO'
KEEP ROWS OF ALLDEPTS WHERE DEPT
    NOT IN MAJOR OF OHIOMAJORS
PRINT DEPT
```

12.1
```
FORM ARTFAC
  FROM NAME, DEPT OF FACULTY
KEEP ROWS WHERE DEPT = 'ART'
KEEP COLUMN NAME
```

12.2
```
FORM ROOM
  FROM COURSE, SECTION, LIMIT, SIZE OF TEACHING
ADD COLUMN LIMIT-SIZE (AS ROOMLEFT)
KEEP COLUMNS COURSE, SECTION, ROOMLEFT
```

12.3
```
FORM CLASSINFO
  FROM ID, COURSE, DEPT, SECTION, SIZE OF TEACHING
GROUP BY DEPT
ADD COLUMN MAX(SIZE) (AS MAXSIZE)
KEEP ROWS WHERE SIZE = MAXSIZE
PRINT ID, COURSE, SECTION, SIZE
```

A P P E N D I X C

EXPERIMENTAL MATERIALS

TABLE OF CONTENTS

To: Freshmen and sophomore students in the School of Business Administration.

Accounting 297A/397A and Management 297A/397A (Special Topics)

Spring Semester 1978

These identical courses are open to all undergraduate students - especially freshmen and sophomores.

You probably know that computers are widely used in business. One of the prime uses of computers is to store payroll, inventory, billing and other information. When information like this is stored on the computer it is called a database. It is vital that people in management positions have this information readily available in order to determine the present situation and plan the future direction of their business. However, management people should not have to learn the details of how computers work in order to obtain this information.

Languages have been developed for people with no computer training to learn and use. These languages allow a person to obtain any information from the database without having to become a computer expert. These languages are called query languages. As the use of databases grows, so does the use of query languages.

These courses will teach you to use a query language. The courses are worth 1 credit and you can sign up for any one of them. Usually, a 1 credit course meets once a week for an entire semester. However, these courses will meet several times a week but only for the first part of the semester. The exact meeting times and the length of the course will be decided at the sign up meetings (see below) by student vote. The courses will be over by mid-semester. No computer experience is needed or wanted of students taking these courses.

Schedule numbers: Accounting 397A - 036951
Management 397A - 519504
297A - available at sign up meetings

Sign up meetings: You may come to any one of the following meetings; bring an add card with you. These meetings will determine how often the classes meet each week.

Tuesday Jan. 31 or Thursday Feb. 2	2:30 PM	SBA 100
Tuesday Jan. 31 or Thursday Feb. 2	3:35 PM	SBA 100
Tuesday Jan. 31 or Thursday Feb. 2	4:40 PM	SBA 100
Wednesday Feb. 1 or Friday Feb. 3	3:35 PM	DKSN 114
Wednesday Feb. 1 or Friday Feb. 3	4:40 PM	DKSN 114

Accounting 297A/397A and Management 297A/397A

Course Information

Course goals - The primary goals of this course are twofold:

1. To teach you to use a computer information system.
2. To test the learnability of the system.

Course requirements - In order to achieve the above goals it will be necessary to have many quizzes during the course. There will be about 12 short quizzes (15 - 20 minutes each), one at each class meeting. For the test of learnability to be complete, every student must take every quiz. There will also be two mandatory exams, one immediately at the end of the course and one two weeks later. There will be about one hour of reading required outside of class for each of the 12 quizzes.

Grading - Due to the conscientiousness required of students in this course, grades will be high. In order to enforce this conscientiousness, every quiz missed (and not made up before the next quiz) will cause a heavy grade penalty. The exam immediately at the end of the course will determine your grade. The other exam (two weeks later) is mandatory but will not affect your grade and will require no studying. There will be a heavy grade penalty if the second exam is missed and not made up in a reasonable amount of time.

Note: DON'T BE OVERLY CONCERNED ABOUT THE PENALTIES. IF YOU ARE CONSCIENTIOUS, THEY WILL NOT AFFECT YOU. IT IS ONLY FAIR TO TELL YOU ABOUT THEM NOW, RATHER THAN LATER.

Prerequisite - There are no prerequisites. This course is designed for people with no knowledge of computers.

Text - The text will be available from your instructor at the first class meeting. Bring $3.00 to purchase the text to the first class meeting.

Schedule numbers:

Accounting 397A	036951
Accounting 297A	034766
Management 397A	519504
Management 297A	516845

ACCOUNTING 297A/397A and MANAGEMENT 297A/397A

Personal Information

Name:

Address:

Phone:

Student number:

Class:

Major:

Choice of class meeting 1.
2.
3.
4.

Available (Circle YES if you are available at the given time, circle NO if you are not):

	2:30		3:35		4:40	
Monday			YES	NO	YES	NO
Tuesday	YES	NO	YES	NO	YES	NO
Wednesday			YES	NO	YES	NO
Thursday	YES	NO	YES	NO	YES	NO
Friday			YES	NO	YES	NO

Which course do you wish to take?

Accounting 397A _____
Accounting 297A _____
Management 397A _____
Management 297A _____

What experience do you have with computers? (None is required.)

Math background:

What are the meanings of the following symbols? (This is just a background question; you do not need to know these symbols.)

> ∪ (union)

< ∩ (intersect)

=

Database - 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

Most of the table and column meanings are probably obvious to you. Here is a bit more explanation:

STUDENT - Contains information about students. There is one row in this table for each student in the school. The REPID column contains the ID of each student's representative on the student senate. The representative is also a student.

TAKING - Contains the courses each student is taking this semester. There is one row for each course each student is taking.

DEPARTMENT - Contains information about the academic departments in the college. The HEAD column contains the ID of the department head.

FACULTY - Contains information about faculty members. There is one row for each faculty member. Each faculty member must be on a college committee. The COMHEAD column contains the ID of the head of a faculty member's committee. The committee head is also a faculty member.

TEACHING - Contains information about the courses being taught this semester. There is one row in this table for each section of each course taught this semester. LIMIT contains the limit on enrollment for a section, SIZE contains the actual size of the class.

COURSES - Contains information about all courses in the college, whether or not they are being taught this semester. There is one row for each course.

Database - MANUFACTURER

DEPT

DNAME	LOCATION
VEHICLE	BOSTON
STEREO	SEATTLE
TOYS	AMHERST

PRODUCT

PRODNO	PRODNAME	PRICE	ONHAND	MAKEMORE
1	BICYCLE	89.50	200	100
2	MOPED	450.00	100	100
3	TURNTABLE	105.00	137	100
4	RECEIVER	400.00	100	100

EMPLOYEE

ID	NAME	SEX	DNAME	MGRID	JOB	SALARY	SENIORITY
1	BILL JONES	M	VEHICLE	3	MACHINIST	16000	7
2	JILL SMITH	F	STEREO	9	DESIGNER	18000	8
3	MARY LEE	F	VEHICLE	12	ENGINEER	20000	5

PARTS

PARTNO	PARTNAME	PRICE	ONHAND	REORDER
101	BEARING	.04	10000	5000
102	WHEEL	8.00	400	400
103	TRANSISTOR	.30	1000	500

USES

PRODNO	PARTNO	QUANTITY
1	101	24
1	102	2
2	101	30
2	102	2
3	101	30
4	103	12

PRODUCES

DNAME	PRODNO	QUANTITY
VEHICLE	1	400
VEHICLE	2	400
STEREO	3	150
STEREO	4	100
TOYS	1	150

WORKSON

ID	PRODNO
1	1
1	2
2	3

DEPT - Contains information about the departments in the company. There is one row for each department. The columns contain the department name (DNAME) and its LOCATION.

PRODUCT - Contains information about the products manufactured by this company. There is one row for each product. The columns contain the product identification number (PRODNO), the product name (PRODNAME), its PRICE, the number ONHAND, and the MAKEMORE level. When the number ONHAND is less than or equal to the MAKEMORE value, more of this product should be made.

EMPLOYEE - Contains information about the employees of the company. There is one row for each employee. The columns contain the employee's ID, NAME, SEX, the name of the department he works in (DNAME), the ID of his manager (MGRID, the manager is also an employee), his JOB type, SALARY and SENIORITY level (10 is the highest and 1 is the lowest).

PARTS - Contains information about the parts used in making the products. There is one row for each part. The columns contain the part number (PARTNO), the PARTNAME, its PRICE, the number ONHAND and the REORDER level. When the number ONHAND is less than or equal to the REORDER level, the part must be reordered.

USES - Contains information about the parts making up each product. The PRODNO column contains the product number and the PARTNO column contains the part number of a part that is used in the manufacture of that product. QUANTITY contains the quantity of the part used in making one copy of the product. There is a row for each part in each product.

PRODUCES - Contains information on which department produces which products. DNAME gives the department name and PRODNO gives the product number of a product produced by that department. QUANTITY is the quantity of the product produced by the department in one month. There is one row for each product produced by each department.

WORKSON - Contains information about which employee works on which product. ID gives the employee ID and PRODNO gives the product number of a product he works on.

ACCOUNTING 397A - QUIZ 1

1. Where is the vehicle department located?

2. What is Jill Smith's job and what is her salary?

3. List the names of people working for manager 9.

4. List the product names of all the products.

ACCOUNTING 397A - QUIZ 1

1. Where is the vehicle department located?

```
SELECT LOCATION
FROM DEPT
WHERE DNAME = 'VEHICLE'
```

2. What is Jill Smith's job and what is her salary?

```
SELECT JOB, SALARY
FROM EMPLOYEE
WHERE NAME = 'JILL SMITH'
```

3. List the names of people working for manager 9.

```
SELECT NAME
FROM EMPLOYEE
WHERE MGRID = '9'
```

4. List the product names of all the products.

```
SELECT PRODNAME
FROM PRODUCT
```

ACCOUNTING 397A - QUIZ 1

1. Where is the vehicle department located?

```
FORM VEHLOC
  FROM DNAME, LOCATION OF DEPT
KEEP ROW WHERE DNAME = 'VEHICLE'
PRINT LOCATION
```

2. What is Jill Smith's job and what is her salary?

```
FORM JILLSINFO
  FROM NAME, JOB, SALARY OF EMPLOYEE
KEEP ROW WHERE NAME = 'JILL SMITH'
PRINT JOB, SALARY
```

3. List the names of people working for manager 9.

```
FORM WORKFOR9
  FROM NAME, MGRID OF EMPLOYEE
KEEP ROWS WHERE MGRID = '9'
PRINT NAME
```

4. List the product names of all the products.

```
FORM ALLPRODS
  FROM PRODNAME OF PRODUCT
PRINT PRODNAME
```

ACCOUNTING 397A - QUIZ 2

1. Which employees make more than $15,000?

2. What are the part numbers of parts that need to be reordered?

3. Which machinists make more than $20,000?

4. List the names of engineers and machinists.

ACCOUNTING 397A - QUIZ 2

1. Which employees make more than $15,000?

```
SELECT NAME
FROM EMPLOYEE
WHERE SALARY > '15000'
```

2. What are the part numbers of parts that need to be reordered?

```
SELECT PARTNO
FROM PARTS
WHERE ONHAND <= REORDER
```

3. Which machinists make more than $20,000?

```
SELECT NAME
FROM EMPLOYEE
WHERE JOB = 'MACHINIST'
  AND SALARY > '20000'
```

4. List the names of engineers and machinists.

```
SELECT NAME
FROM EMPLOYEE
WHERE JOB = 'ENGINEER'
  OR JOB = 'MACHINIST'
```

ACCOUNTING 397A - QUIZ 2

1. Which employees make more than $15,000?

```
FORM A
  FROM NAME, SALARY OF EMPLOYEE
KEEP ROWS WHERE SALARY > '15000'
PRINT NAME
```

2. What are the part numbers of parts that need to be reordered?

```
FORM B
  FROM PARTNO, ONHAND, REORDER OF PARTS
KEEP ROWS WHERE ONHAND <= REORDER
PRINT PARTNO
```

3. What are the names of machinists who make more than $20,000?

```
FORM RICHMACH
  FROM NAME, JOB, SALARY OF EMPLOYEE
KEEP ROWS WHERE JOB = 'MACHINIST'
  AND SALARY > '20000'
PRINT NAME
```

4. List the names of engineers and machinists.

```
FORM ENGMACH
  FROM NAME, JOB OF EMPLOYEE
KEEP ROWS WHERE JOB = 'ENGINEER'
  OR JOB = 'MACHINIST'
PRINT NAME
```

ACCOUNTING 397A - QUIZ 3

1. Each department is supposed to produce 20% more of each product next month. List the department names, the product numbers and the increase in quantity to be produced.

2. List the part numbers of parts that would have to reordered if there were 10 less on hand.

3. How many employees work in the stereo department?

4. Which employees have the highest seniority level?

ACCOUNTING 397A - QUIZ 3

1. Each department is supposed to produce 20% more of each product next month. List the department names, the product numbers and the increase in quantity to be produced.

```
SELECT DNAME, PRODNO, .20 * QUANTITY

FROM PRODUCES
```

2. List the part numbers of parts that would have to be reordered if there were 10 less on hand.

```
SELECT PARTNO

FROM PARTS

WHERE ONHAND - 10 <= REORDER
```

3. How many employees work in the stereo department?

```
SELECT COUNT(NAME)

FROM EMPLOYEE

WHERE DNAME = 'STEREO'
```

4. Which employees have the highest seniority level?

```
SELECT NAME

FROM EMPLOYEE

WHERE SENIORITY =

  SELECT MAX(SENIORITY)

  FROM EMPLOYEE
```

ACCOUNTING 397A - QUIZ 3

1. Each department is supposed to produce 20% more of each product next month. List the department names, the product numbers and the increase in quantity to be produced.

```
FORM HIGHPROD
  FROM DNAME, PRODNO, .20 * QUANTITY (AS INCPROD) OF PRODUCES
PRINT DNAME, PRODNO, INCPROD
```

2. List the part numbers of parts that would have to be reordered if there were 10 less on hand.

```
FORM PARTREORDER
  FROM PARTNO, ONHAND, REORDER OF PARTS
KEEP ROWS WHERE ONHAND - 10 <= REORDER
PRINT PARTNO
```

3. How many employees work in the stereo department?

```
FORM STEREOCOUNT
  FROM NAME, DNAME OF EMPLOYEE
KEEP ROWS WHERE DNAME = 'STEREO'
PRINT COUNT(NAME)
```

4. Which employees have the highest seniority level?

```
FORM HIGHSEN
  FROM NAME, SENIORITY OF EMPLOYEE
KEEP ROWS WHERE SENIORITY = MAX(SENIORITY)
PRINT NAME
```

ACCOUNTING 397A - QUIZ 4

1. What are the names and salaries of people who work in departments located in Boston?

2. List the names of employees who work on speakers.

3. List the names of people whose manager is Pete Dunne.

ACCOUNTING 397A - QUIZ 4

1. What are the names and salaries of people who work in departments located in Boston?

```
SELECT NAME, SALARY
FROM EMPLOYEE
WHERE DNAME =
  SELECT DNAME
  FROM DEPT
  WHERE LOCATION = 'BOSTON'
```

2. List the names of employees who work on speakers.

```
SELECT NAME
FROM EMPLOYEE
WHERE ID =
  SELECT ID
  FROM WORKSON
  WHERE PRODNO =
    SELECT PRODNO
    FROM PRODUCT
    WHERE PRODNAME = 'SPEAKER'
```

3. List the names of people whose manager is Pete Dunne.

```
SELECT NAME
FROM EMPLOYEE
WHERE MGRID =
  SELECT ID
  FROM EMPLOYEE
  WHERE NAME = 'PETE DUNNE'
```

ACCOUNTING 397A - QUIZ 4

1. What are the names and salaries of people who work in departments located in Boston?

```
FORM BOSTNAMES
  FROM NAME, DNAME, SALARY OF EMPLOYEE
ADD COLUMN LOCATION
  OF DEPT
  BY DNAME = DNAME
KEEP ROWS WHERE LOCATION = 'BOSTON'
PRINT NAME, SALARY
```

2. List the names of employees who work on speakers.

```
FORM TABLENAMES
  FROM ID, NAME OF EMPLOYEE
ADD COLUMN PRODNO
  OF WORKSON
  BY ID = ID
ADD COLUMN PRODNAME
  OF PRODUCT
  BY PRODNO = PRODNO
KEEP ROWS WHERE PRODNAME = 'SPEAKER'
PRINT NAME
```

3. List the names of people whose manager is Pete Dunne.

```
FORM PETES
  FROM NAME, MGRID OF EMPLOYEE
ADD COLUMN NAME (AS MGRNAME)
  OF EMPLOYEE
  BY MGRID = ID
KEEP ROWS WHERE MGRNAME = 'PETE DUNNE'
PRINT NAME
```

ACCOUNTING 397A - QUIZ 5 (Review)

1. Which departments produce tricycles?

2. What is the average salary of machinists?

3. What is the price of product 5?

4. List the names of machinists with seniority of 7.

ACCOUNTING 397A - QUIZ 5 (Review)

1. Which departments produce tricycles?

```
SELECT DNAME
FROM PRODUCES
WHERE PRODNO =
  SELECT PRODNO
  FROM PRODUCT
  WHERE PRODNAME = 'TRICYCLE'
```

2. What is the average salary of machinists?

```
SELECT AVG(SALARY)
FROM EMPLOYEE
WHERE JOB = 'MACHINIST'
```

3. What is the price of product 5?

```
SELECT PRICE
FROM PRODUCT
WHERE PRODNO = 5
```

4. List the names of machinists with seniority of 7.

```
SELECT NAME
FROM EMPLOYEE
WHERE JOB = 'MACHINIST'
  AND SENIORITY = '7'
```

ACCOUNTING 397A - QUIZ 6

1. List the product numbers of products using part 152 as well as the product numbers of products produced by the stereo department.

2. List the ID's of people working in the vehicle department who also work on product 1.

3. List the part numbers of parts that cost more than $5.00 and are not used in product 1.

ACCOUNTING 397A - QUIZ 6

1. List the product numbers of products using part 152 as well as the product numbers of products produced by the stereo department.

```
     SELECT PRODNO
     FROM USES
     WHERE PARTNO = '152'
UNION
     SELECT PRODNO
     FROM PRODUCES
     WHERE DNAME = 'STEREO'
```

2. List the ID's of people working in the vehicle department who also work on product 1.

```
     SELECT ID
     FROM EMPLOYEE
     WHERE DNAME = 'VEHICLE'
INTERSECT
     SELECT ID
     FROM WORKSON
     WHERE PRODNO = '1'
```

3. List the part numbers of parts that cost more than $5.00 and are not used in product 1.

```
     SELECT PARTNO
     FROM PARTS
     WHERE PRICE > '5.00'
MINUS
     SELECT PARTNO
     FROM USES
     WHERE PRODNO = '1'
```

ACCOUNTING 397A - QUIZ 6

1. List the product numbers of products using part 152 as well as the product numbers of products produced by the stereo department.

```
FORM USES152
  FROM PRODNO, PARTNO OF USES
KEEP ROWS WHERE PARTNO = '152'
KEEP COLUMN PRODNO
FORM STEREOPRODS
  FROM PRODNO, DNAME OF PRODUCES
KEEP ROWS WHERE DNAME = 'STEREO'
KEEP COLUMN PRODNO
ADD ROWS OF USES152 TO STEREOPRODS
PRINT PRODNO
```

2. List the ID's of people working in the vehicle department who also work on product 1.

```
FORM VEHID
  FROM ID, DNAME OF EMPLOYEE
KEEP ROWS WHERE DNAME = 'VEHICLE'
FORM ID1
  FROM ID, PRODNO OF WORKSON
KEEP ROWS WHERE PRODNO = '1'
KEEP ROWS OF VEHID WHERE ID IN
  ID OF ID1
PRINT ID
```

3. List the part numbers of parts that cost more than $5.00 and are not used in product 1.

```
FORM MORETHAN5
  FROM PARTNO, PRICE OF PARTS
KEEP ROWS WHERE > '5.00'
FORM USEDIN1
  FROM PARTNO, PRODNO OF USES
KEEP ROWS WHERE PRODNO = '1'
KEEP ROWS OF MORETHAN5 WHERE PARTNO NOT IN
  PARTNO OF USEDIN1
PRINT PARTNO
```

ACCOUNTING 397A - QUIZ 7

1. List the product numbers of products using more than 7 different parts.

2. List the jobs in which the average salary is less than $8,000.

3. List the department name and the total annual payroll for each department.

4. List the departments and the number of women in the departments. (Do not list the departments having no women.)

ACCOUNTING 397A - QUIZ 7

1. List the product numbers of products using more than 7 different parts.

```
SELECT PRODNO
FROM USES
GROUP BY PRODNO
HAVING COUNT(PARTNO) > '7'
```

2. List the jobs in which the average salary is less than $8,000.

```
SELECT JOB
FROM EMPLOYEE
GROUP BY JOB
HAVING AVG(SALARY) < '8000'
```

3. List the department name and the total annual payroll for each department.

```
SELECT DNAME, SUM(SALARY)
FROM EMPLOYEE
GROUP BY DNAME
```

4. List the departments and the number of women in the departments. (Do not list the departments having no women.)

```
SELECT DNAME, COUNT(NAME)
FROM EMPLOYEE
WHERE SEX = 'F'
GROUP BY DNAME
```

ACCOUNTING 397A - QUIZ 7

1. List the product numbers of products using more than 7 different parts.

```
FORM MORETHAN7
  FROM PRODNO, PARTNO OF USES
GROUP BY PRODNO
KEEP GROUPS WHERE COUNT(PARTNO) > '7'
PRINT PRODNO
```

2. List the jobs in which the average salary is less than $8,000.

```
FORM LOWJOBS
  FROM JOB, SALARY OF EMPLOYEE
GROUP BY JOB
KEEP GROUPS WHERE AVG(SALARY) < '8000'
PRINT JOB
```

3. List the department name and the total annual payroll for each department.

```
FORM DEPTPAY
  FROM DNAME, SALARY OF EMPLOYEE
GROUP BY DNAME
PRINT DNAME, SUM(SALARY)
```

4. List the departments and the number of women in the departments. (Do not list the departments having no women.)

```
FORM COUNTWOMEN
  FROM NAME, SEX, DNAME OF EMPLOYEE
KEEP ROWS WHERE SEX = 'F'
GROUP BY DNAME
PRINT DNAME, COUNT(NAME)
```

ACCOUNTING 397A - QUIZ 8 (Review)

1. What is the total salary of all the machinists?

2. Which departments produce product 5?

3. List the product numbers of products using part 152 that are produced by the stereo department.

4. List the departments in which the average salary is greater than $18,000.

5. What are the product numbers of products that Bill Jones works on?

ACCOUNTING 397A - QUIZ 8 (Review)

1. What is the total salary of all the machinists?

```
SELECT SUM(SALARY)
FROM EMPLOYEE
WHERE JOB = 'MACHINIST'
```

2. Which departments produce product 5?

```
SELECT DNAME
FROM PRODUCES
WHERE PRODNO = '5'
```

3. List the product numbers of products using part 152 that are produced by the stereo department.

```
SELECT PRODNO
FROM USES
WHERE PARTNO = '152'
INTERSECT
SELECT PRODNO
FROM PRODUCES
WHERE DNAME = 'STEREO'
```

4. List the departments in which the average salary is greater than $18,000.

```
SELECT DNAME
FROM EMPLOYEE
GROUP BY DNAME
HAVING AVG(SALARY) > '18000'
```

5. What are the product numbers of products that Bill Jones works on?

```
SELECT PRODNO
FROM WORKSON
WHERE ID =
  SELECT ID
  FROM EMPLOYEE
  WHERE NAME = 'BILL JONES'
```

ACCOUNTING 397A - QUIZ 8 (Review)

1. What is the total salary of all the machinists?

```
FORM MACHSAL
  FROM SALARY, JOB OF EMPLOYEE
KEEP ROWS WHERE JOB = 'MACHINIST'
PRINT SUM(SALARY)
```

2. Which departments produce product 5?

```
FORM PROD5
  FROM DNAME, PRODNO OF PRODUCES
KEEP ROWS WHERE PRODNO = '5'
PRINT DNAME
```

3. List the product numbers of products using part 152 that are produced by the stereo department.

```
FORM USE152
  FROM PRODNO, PARTNO OF USES
KEEP ROWS WHERE PARTNO = '152'
FORM STEREO
  FROM DNAME, PRODNO OF PRODUCES
KEEP ROWS WHERE DNAME = 'STEREO'
KEEP ROWS OF STEREO WHERE PRODNO IN
  PRODNO OF USE152
PRINT PRODNO
```

4. List the departments in which the average salary is greater than $18,000.

```
FORM BIGAVG
  FROM DNAME, SALARY OF EMPLOYEE
GROUP BY DNAME
KEEP GROUPS WHERE AVG(SALARY) > '18000'
PRINT DNAME
```

5. What are the product numbers of products that Bill Jones works on?

```
FORM BILLSPRODS
  FROM NAME, ID OF EMPLOYEE
ADD COLUMN PRODNO
  OF WORKSON
  BY ID = ID
KEEP ROWS WHERE NAME = 'BILL JONES'
PRINT PRODNO
```

ACCOUNTING 397A - QUIZ 9

1. List the departments producing all the products that use part 152.

```
FORM USES152
  FROM PRODNO, PARTNO OF USES
KEEP ROWS WHERE PARTNO = '152'
FORM ALL152
  FROM DNAME, PRODNO OF PRODUCES
GROUP BY DNAME
KEEP GROUPS WHERE PRODNO CONTAINS
  PRODNO OF USES152
PRINT DNAME
```

2. List the product numbers of products using only parts that need to be reordered.

```
FORM REORDERPARTS
  FROM PARTNO, ONHAND, REORDER OF PARTS
KEEP ROWS WHERE ONHAND <= REORDER
FORM TABLE
  FROM PRODNO, PARTNO OF USES
GROUP BY PRODNO
KEEP GROUPS WHERE PARTNO IN
  PARTNO OF REORDERPARTS
PRINT PRODNO
```

3. List the ID's of people who work on all products produced by the stereo department and work on no other products.

```
FORM STEREOPRODS
  FROM DNAME, PRODNO OF PRODUCES
KEEP ROWS WHERE DNAME = 'STEREO'
FORM MAKEPRODS
  FROM ID, PRODNO OF WORKSON
GROUP BY ID
KEEP GROUPS WHERE PRODNO =
  PRODNO OF STEREOPRODS
PRINT ID
```

ACCOUNTING 397A - QUIZ 10

1. Print the name of each employee and the location of his department.

2. List the name of the producing department and the product name for products we need to make more of.

3. List the names of employees and the names of their managers for employees that have higher seniority than their managers.

ACCOUNTING 397A - QUIZ 10

1. Print the name of each employee and the location of his department.

```
SELECT NAME, LOCATION

FROM EMPLOYEE, DEPT

WHERE EMPLOYEE.DNAME = DEPT.DNAME
```

2. List the name of the producing department and the product name for products we need to make more of.

```
SELECT DNAME, PRODNAME

FROM PRODUCES, PRODUCT

WHERE PRODUCES.PRODNO = PRODUCT.PRODNO

  AND ONHAND <= MAKEMORE
```

3. List the names of employees and the names of their managers for employees that have higher seniority than their managers.

```
SELECT A.NAME, B.NAME

FROM EMPLOYEE A, EMPLOYEE B

WHERE A.MGRID = B.ID

  AND A.SENIORITY > B.SENIORITY
```

ACCOUNTING 397A - QUIZ 10

1. Print the name of each employee and the location of his department.

```
FORM NAMELOC
  FROM NAME, DNAME OF EMPLOYEE
ADD COLUMN LOCATION
  OF DEPT
  BY DNAME = DNAME
PRINT NAME, LOCATION
```

2. List the name of the producing department and the product name for products we need to make more of.

```
FORM MOREPRODS
  FROM DNAME, PRODNO OF PRODUCES
ADD COLUMNS PRODNAME, ONHAND, MAKEMORE
  OF PRODUCT
  BY PRODNO = PRODNO
KEEP ROWS WHERE ONHAND <= MAKEMORE
PRINT DNAME, PRODNAME
```

3. List the names of employees and the names of their managers for employees that have higher seniority than their managers.

```
FORM EMPHIGH
  FROM NAME, MGRID, SENIORITY OF EMPLOYEE
ADD COLUMNS NAME (AS MGRNAME), SENIORITY (AS MGRSEN)
  OF EMPLOYEE
  BY MGRID = ID
KEEP ROWS WHERE SENIORITY > MGRSEN
PRINT NAME, MGRNAME
```

ACCOUNTING 397A - QUIZ 11 (Review)

1. Which departments are located in Seattle?

2. List the departments located in Boston that do not produce product 1.

3. List the average salary of women in each job.

4. List the names of departments producing all the products worked on by employee 1.

5. List the name of each employee along with the name of his manager.

ACCOUNTING 397A - QUIZ 11 (Review)

1. Which departments are located in Seattle?

```
SELECT DNAME
FROM DEPT
WHERE LOCATION = 'SEATTLE'
```

2. List the departments located in Boston that do not produce product 1.

```
SELECT DNAME
FROM DEPT
WHERE LOCATION = 'BOSTON'
MINUS
SELECT DNAME
FROM PRODUCES
WHERE PRODNO = '1'
```

3. List the average salary of women in each job.

```
SELECT JOB, AVG(SALARY)
FROM EMPLOYEE
WHERE SEX = 'F'
GROUP BY JOB
```

4. List the names of departments producing all the products worked on by employee 1.

```
SELECT DNAME
FROM PRODUCES
GROUP BY DNAME
HAVING SET(PRODNO) CONTAINS
  SELECT PRODNO
  FROM WORKSON
  WHERE ID = '1'
```

5. List the name of each employee along with the name of his manager.

```
SELECT A.NAME, B.NAME
FROM EMPLOYEE A, EMPLOYEE B
WHERE A.MGRID = B.ID
```

ACCOUNTING 397A - QUIZ 11 (Review)

1. Which departments are located in Seattle?

```
FORM SEATTLEDEPTS
  FROM DNAME, LOCATION OF DEPT
KEEP ROWS WHERE LOCATION = 'SEATTLE'
PRINT DNAME
```

2. List the departments located in Boston that do not produce product 1.

```
FORM PROD1DEPTS
  FROM DNAME, PRODNO OF PRODUCES
KEEP ROWS WHERE PRODNO = '1'
FORM BOSTDEPTS
  FROM DNAME, LOCATION OF DEPT
KEEP ROWS WHERE LOCATION = 'BOSTON'
KEEP ROWS OF BOSTDEPTS WHERE DNAME NOT IN
  DNAME OF PROD1DEPTS
PRINT DNAME
```

3. List the average salary of women in each job.

```
FORM WOMENAVG
  FROM JOB, SEX, SALARY OF EMPLOYEE
KEEP ROWS WHERE SEX = 'F'
GROUP BY JOB
PRINT JOB, AVG(SALARY)
```

4. List the names of departments producing all the products worked on by employee 1.

```
FORM EMP1
  FROM ID, PRODNO OF WORKSON
KEEP ROWS WHERE ID = '1'
FORM PRODEPT
  FROM DNAME, PRODNO OF PRODUCES
GROUP BY DNAME
KEEP GROUPS WHERE PRODNO CONTAINS
  PRODNO OF EMP1
PRINT DNAME
```

5. List the name of each employee along with the name of his manager.

```
FORM EMPMGR
  FROM NAME, MGRID OF EMPLOYEE
ADD COLUMN NAME (AS MGRNAME)
  OF EMPLOYEE
  BY MGRID = ID
PRINT NAME, MGRNAME
```

ACCOUNTING 397A - QUIZ 12

1. A federal judge has ruled that all women machinists in the company are to have their seniority level increased by 1. Create a table containing the ID, name and new seniority level for women machinists.

2. Print the names of employees and their jobs for employees who make the smallest salary in their job classification.

ACCOUNTING 397A - QUIZ 12

1. A federal judge has ruled that all women machinists in the company are to have their seniority level increased by 1. Create a table containing the ID, name and new seniority level for women machinists.

```
WOMACHSEN(ID,NAME,NEWSEN) ← SELECT ID, NAME, SENIORITY + 1
                            FROM EMPLOYEE
                            WHERE JOB = 'MACHINIST'
                              AND SEX = 'F'
```

2. Print the names of employees and their jobs for employees who make the smallest salary in their job classification.

```
SMALLSAL(JOB,MINSAL) ← SELECT JOB, MIN(SALARY)
                       FROM EMPLOYEE
                       GROUP BY JOB

SELECT NAME, JOB
FROM EMPLOYEE, SMALLSAL
WHERE EMPLOYEE.JOB = SMALLSAL.JOB
  AND SALARY = MINSAL
```

ACCOUNTING 397A - QUIZ 12

1. A federal judge has ruled that all women machinists in the company are to have their seniority level increased by 1. Create a table containing the ID, name and new seniority level for women machinists.

```
FORM WOMMACHSEN
  FROM ID, NAME, SEX, JOB, SENIORITY + 1 (AS NEWSEN)
  OF EMPLOYEE
KEEP ROWS WHERE SEX = 'F' AND JOB = 'MACHINIST'
KEEP COLUMNS ID, NAME, NEWSEN
```

2. Print the names of employees and their jobs for employees who make the smallest salary in their job classification.

```
FORM SMALLSAL
  FROM NAME, JOB, SALARY OF EMPLOYEE
GROUP BY JOB
ADD COLUMN MIN(SALARY) (AS MINSAL)
KEEP ROWS WHERE SALARY = MINSAL
PRINT JOB, NAME
```

Name:_____

How long did it take you to read each of the following lessons?

7._____ 8._____ 9._____

10._____ 11._____ 12._____

On the following scale evaluate the difficulty of each lesson. (1 means that you could comprehend the lesson immediately, 10 means that the lesson was as difficult to understand as anything you have ever read.)

	easy									hard
Lesson 7.	1	2	3	4	5	6	7	8	9	10
Lesson 8.	1	2	3	4	5	6	7	8	9	10
Lesson 9.	1	2	3	4	5	6	7	8	9	10
Lesson 10.	1	2	3	4	5	6	7	8	9	10
Lesson 11.	1	2	3	4	5	6	7	8	9	10
Lesson 12.	1	2	3	4	5	6	7	8	9	10

Are you presently taking a computer class other than Accounting 397A? If so, which one?

What computer courses have you taken before this semester?

Are you planning to take any more computer courses? If you are, which one do you plan to take and when?

If you have further comments about the course, write them on the back of the page.

Database - 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

This database contains information about a mailorder company. This company is old and established and sells only high quality items.

DEPARTMENT - Contains information about the departments in the company. There is one row for each department. FLOOR tells which floor of the building the department is on.

ITEM - Contains information about the items sold by the company. There is one row for each item. RETAIL contains the retail price of the item. The item is reordered if ONHAND is less than or equal to REORDER. ALTERNATE gives the item number of the item to be substituted for this one if the ONHAND value is 0. Obviously, ALTERNATE contains the item number of another item in the ITEM table.

SELLS - Contains information about which departments sell which items. QUOTA gives the quantity of the item that must be sold by this department each month. There is one row for each item sold by each department.

SUPPLIER - Contains information about the companies that supply items to this mailorder company. There is one row for each supplier company.

SUPPLIES - Contains information about the items supplied by each supplier. WHOLESALE is the wholesale price per item charged by the supplier. ONORDER tells how many of the item is presently on order from the supplier. There is one row for each item supplied by each company.

CHARGEACCTS - Contains information about the charge accounts held by customers of the company. There is one row for each account. TOTALBILL contains the total amount owed by the customer. LIMIT contains the customer's credit limit. A customer is over his limit if TOTALBILL is greater than LIMIT. RATING contains the customer's credit rating (1 is poor, 10 is excellent). Each customer must be referred by another charge customer in order to get his account. REFERREDBY contains the account number of the person who referred this customer to the company.

CHARGED - Contains information about the items charged by the customer. PERITEM is the price the customer paid per item charged; it is included because the customer may have purchased a sale item or purchased an item before its RETAIL price in the ITEM table went up. (If the customer returns the item, this is the amount he gets back.)

ACCOUNTING 397A - Take home review quiz.

This quiz uses the mailorder database. The mailorder database will be used for both the final and the retention tests. One purpose of this quiz is to give you experience with the mailorder database before the tests.

This quiz is different than the others you have taken in this course. You will be given the query written in the query language and you are to:

a. List sample output from the sample database. Since the database is only a sample, it may not be possible to write down the answer to some queries. If this is the case just write "not enough information".

b. Translate the query into English.

This quiz must be turned in before the final is given. It will not affect you grade unless it is not done. The solutions will be available at the optional review session and at the final.

PLEASE WORK ALONE ON THIS QUIZ. AFTER YOU HAVE FINISHED THE QUIZ YOU MAY DISCUSS IT WITH OTHER PEOPLE BUT DO NOT CHANGE YOUR ANSWERS. THE QUIZ WILL NOT AFFECT YOUR GRADE BUT YOUR ANSWERS WILL HELP TO DETERMINE THE ABILITY OF PEOPLE TO UNDERSTAND QUERIES WRITTEN IN THIS LANGUAGE.

Name _____

How long did it take you to do this quiz? _____

Example:

```
0.  SELECT DEPT
    FROM SELLS
    WHERE ITEMNO = '3'
```

a. Camping.

b. Which department sells item 3?

1.
```
SELECT LOCATION
FROM SUPPLIER
WHERE SUPPNAME = 'WARMTH, INC'
```

a.

b.

2.
```
SELECT NAME
FROM CHARGEACCTS
WHERE RATING > '8'
```

a.

b.

3.
```
SELECT DESCRIPTION
FROM ITEM
WHERE ONHAND <= REORDER
```

a.

b.

4.
```
SELECT SUPPNAME
FROM SUPPLIES
WHERE ITEMNO = '3'
  AND ONORDER > '400'
```

a.

b.

5.
```
SELECT SUPPNAME, ITEMNO
FROM SUPPLIES
WHERE WHOLESALE * ONORDER >'5000'
```

a.

b.

6.
```
SELECT COUNT(RATING)
FROM CHARGEACCTS
WHERE RATING = '10'
```

a.

b.

7.
```
SELECT FLOOR
FROM DEPARTMENT
WHERE DEPT =
  SELECT DEPT
  FROM SELLS
  WHERE ITEMNO = '3'
```

a.

b.

8.
```
SELECT LOCATION
FROM SUPPLIER
WHERE SUPPNAME =
  SELECT SUPPNAME
  FROM SUPPLIES
  WHERE ITEMNO =
    SELECT ITEMNO
    FROM ITEM
    WHERE DESCRIPTION = 'WOOL BLANKET'
```

a.

b.

9.
```
SELECT DESCRIPTION
FROM ITEM
WHERE ITEMNO =
  SELECT ALTERNATE
  FROM ITEM
  WHERE ITEMNO = '3'
```

a.

b.

10.
```
SELECT SUPPNAME
FROM SUPPLIER
WHERE LOCATION = 'BOSTON'
  AND SUPPNAME =
    SELECT SUPPNAME
    FROM SUPPLIES
    WHERE ITEMNO = '3'
```

a.

b.

11.
```
SELECT SUPPNAME
FROM SUPPLIER
WHERE LOCATION = 'BOSTON'
INTERSECT
SELECT SUPPNAME
FROM SUPPLIES
WHERE ITEMNO = '3'
```

a.

b.

12.
```
SELECT LOCATION
FROM SUPPLIER
GROUP BY LOCATION
HAVING COUNT(LOCATION) > '5'
```

a.

b.

13.
```
SELECT ACCTNO, AVG(PERITEM)
FROM CHARGED
GROUP BY ACCTNO
HAVING COUNT(*) > '10'
```

a.

b.

14.
```
SELECT ACCTNO
FROM CHARGED
GROUP BY ACCTNO
HAVING SET(ITEMNO) CONTAINS
  SELECT ITEMNO
  FROM SUPPLIES
  WHERE SUPPNAME = 'WARMTH, INC'
```

a.

b.

15.
```
SELECT DEPT, SELLS.ITEMNO, RETAIL
FROM SELLS, ITEM
WHERE SELLS.ITEMNO = ITEM.ITEMNO
```

a.

b.

ACCOUNTING 397A - Review exam solutions.

1.
```
SELECT LOCATION
FROM SUPPLIER
WHERE SUPPNAME = 'WARMTH, INC'
```

a. ANCHORAGE
b. Where is Warmth, Inc. located?

2.
```
SELECT NAME
FROM CHARGEACCTS
WHERE RATING >'8'
```

a. JAMES LEE
b. List the names of people with credit rating greater than 8.

3.
```
SELECT DESCRIPTION
FROM ITEM
WHERE ONHAND <= REORDER
```

a. SHEEPSKIN SLIPPER
WOOL BLANKET
b. List the descriptions of items that need to be reordered.

4.
```
SELECT SUPPNAME
FROM SUPPLIES
WHERE ITEMNO = '3'
  AND ONORDER > '400'
```

a. JIM'S SPORTS
b. List the names of suppliers who have more than 400 of item 3 on order.

5.
```
SELECT SUPPNAME, ITEMNO
FROM SUPPLIES
WHERE WHOLESALE * ONORDER > '5000'
```

a. JIM'S SPORTS 1
JIM'S SPORTS 3
b. List the supplier names and item numbers for which the bill for the quantity on order will exceed $5,000.

6.
```
SELECT COUNT(RATING)
FROM CHARGEACCTS
WHERE RATING = '10'
```

a. Not enough information.
b. How many charge accounts have a credit rating of 10?

7.
```
SELECT FLOOR
FROM DEPARTMENT
WHERE DEPT =
  SELECT DEPT
  FROM SELLS
  WHERE ITEMNO = '3'
```

a. 2
b. Which floor is the department that sells item 3 located on?

8.
```
SELECT LOCATION
FROM SUPPLIER
WHERE SUPPNAME =
  SELECT SUPPNAME
  FROM SUPPLIES
  WHERE ITEMNO =
    SELECT ITEMNO
    FROM ITEM
    WHERE DESCRIPTION = 'WOOL BLANKET'
```

a. BOSTON
ANCHORAGE

b. List the locations of suppliers who supply wool blankets.

9.
```
SELECT DESCRIPTION
FROM ITEM
WHERE ITEMNO =
  SELECT ALTERNATE
  FROM ITEM
  WHERE ITEMNO = '3'
```

a. THERMAL BLANKET

b. What is the description of item 3's alternate?

10.
```
SELECT SUPPNAME
FROM SUPPLIER
WHERE LOCATION = 'BOSTON'
  AND SUPPNAME =
    SELECT SUPPNAME
    FROM SUPPLIES
    WHERE ITEMNO = '3'
```

a. JIM'S SPORTS

b. Which suppliers located in Boston supply item 3?

11.
```
SELECT SUPPNAME
FROM SUPPLIER
WHERE LOCATION = 'BOSTON'
INTERSECT
SELECT SUPPNAME
FROM SUPPLIES
WHERE ITEMNO = '3'
```

a. JIM'S SPORTS

b. Which suppliers located in Boston supply item 3?

12.
```
SELECT LOCATION
FROM SUPPLIER
GROUP BY LOCATION
HAVING COUNT(LOCATION) > '5'
```

a. Not enough information.

b. List the cities in which more than 5 suppliers are located.

13.
```
SELECT ACCTNO, AVG(PERITEM)
FROM CHARGED
GROUP BY ACCTNO
HAVING COUNT(*) > '10'
```

a. Not enough information.
b. For the accounts that have charged more than 5 different items list the account number and the average price per item.

14.
```
SELECT ACCTNO
FROM CHARGED
GROUP BY ACCTNO
HAVING SET(ITEMNO) CONTAINS
  SELECT ITEMNO
  FROM SUPPLIES
  WHERE SUPPNAME = 'WARMTH, INC'
```

a. Not enough information.
b. List the account numbers that have charged all the items supplied by Warmth, Inc.

15.
```
SELECT DEPT, SELLS.ITEMNO, RETAIL
FROM SELLS, ITEM
WHERE SELLS.ITEM = ITEM.ITEMNO
```

a.
```
CLOTHES   1    23.00
CLOTHES   2    19.00
CAMPING   3    27.50
```
b. List the department, the item numbers of items they sell and the retail price of the items.

ACCOUNTING 397A - Take home review quiz.

This quiz uses the mailorder database. The mailorder database will be used for both the final and the retention tests. One purpose of this quiz is to give you experience with the mailorder database before the tests.

This quiz is different than the others you have taken in this course. You will be given the query written in the query language and you are to:

a. List sample output from the sample database. Since the database is only a sample, it may not be possible to write down the answer to some queries. If this is the case just write "not enough information".

b. Translate the query into English.

This quiz must be turned in before the final is given. It will not affect your grade unless it is not done. The solutions will be available at the optional review session and at the final.

PLEASE WORK ALONE ON THIS QUIZ. AFTER YOU HAVE FINISHED THE QUIZ YOU MAY DISCUSS IT WITH OTHER PEOPLE BUT DO NOT CHANGE YOUR ANSWERS. THE QUIZ WILL NOT AFFECT YOUR GRADE BUT YOUR ANSWERS WILL HELP TO DETERMINE THE ABILITY OF PEOPLE TO UNDERSTAND QUERIES WRITTEN IN THIS LANGUAGE.

Name _____

How long did it take you to do this quiz? _____

Example:

```
0.  FORM Z
      FROM DEPT, ITEMNO OF SELLS
    KEEP ROWS WHERE ITEMNO = '3'
    PRINT DEPT
```

a. Camping.

b. Which department sells item 3?

1.
```
FORM A
  FROM SUPPNAME, LOCATION OF SUPPLIER
KEEP ROWS WHERE SUPPNAME = 'WARMTH, INC'
PRINT LOCATION
```

a.

b.

2.
```
FORM B
  FROM NAME, RATING OF CHARGEACCTS
KEEP ROWS WHERE RATING > '8'
PRINT NAME
```

a.

b.

3.
```
FORM C
  FROM DESCRIPTION, ONHAND, REORDER OF ITEM
KEEP ROWS WHERE ONHAND <= REORDER
PRINT DESCRIPTION
```

a.

b.

4.
```
FORM D
  FROM SUPPNAME, ITEMNO, ONORDER OF SUPPLIES
KEEP ROWS WHERE ITEMNO = '3' AND ONORDER > '400'
PRINT SUPPNAME
```

a.

b.

5.
```
FORM E
  FROM SUPPNAME, ITEMNO, WHOLESALE, ONORDER OF SUPPLIES
KEEP ROWS WHERE WHOLESALE * ONORDER > '5000'
PRINT SUPPNAME, ITEMNO
```

a.

b.

6.
```
FORM F
  FROM RATING OF CHARGEACCTS
KEEP ROWS WHERE RATING = '10'
PRINT COUNT(RATING)
```

a.

b.

7.
```
FORM G
  FROM FLOOR, DEPT OF DEPARTMENT
ADD COLUMN ITEMNO
  OF SELLS
  BY DEPT = DEPT
KEEP ROWS WHERE ITEMNO = '3'
PRINT FLOOR
```

a.

b.

8.
```
FORM H
  FROM ITEMNO, DESCRIPTION OF ITEM
ADD COLUMN SUPPNAME
  OF SUPPLIES
  BY ITEMNO = ITEMNO
ADD COLUMN LOCATION
  OF SUPPLIER
  BY SUPPNAME = SUPPNAME
KEEP ROWS WHERE DESCRIPTION = 'WOOL BLANKET'
PRINT LOCATION
```

a.

b.

9.
```
FORM I
  FROM DESCRIPTION, ITEMNO OF ITEM
ADD COLUMN ITEMNO (AS OTHERITEMNO)
  OF ITEM
  BY ITEMNO = ALTERNATE
KEEP ROWS WHERE OTHER ITEMNO = '3'
PRINT DESCRIPTION
```

a.

b.

10.
```
FORM J
  FROM SUPPNAME, LOCATION OF SUPPLIER
ADD COLUMN ITEMNO
  OF SUPPLIES
  BY SUPPNAME = SUPPNAME
KEEP ROWS WHERE ITEMNO = '3' AND LOCATION = 'BOSTON'
PRINT SUPPNAME
```

a.

b.

11.
```
FORM K
  FROM SUPPNAME, LOCATION OF SUPPLIER
KEEP ROWS WHERE LOCATION = 'BOSTON'
FORM L
  FROM SUPPNAME, ITEMNO OF SUPPLIES
KEEP ROWS WHERE ITEMNO = '3'
KEEP ROWS OF K WHERE SUPPNAME IN SUPPNAME OF L
PRINT SUPPNAME
```

a.

b.

12.
```
FORM M
  FROM LOCATION OF SUPPLIER
GROUP BY LOCATION
KEEP GROUPS WHERE COUNT(LOCATION) > '5'
PRINT LOCATION
```

a.

b.

13.
```
FORM N
  FROM ACCTNO, PERITEM OF CHARGED
GROUP BY ACCTNO
KEEP GROUPS WHERE COUNT(*) > '10'
PRINT ACCTNO, AVG(PERITEM)
```

a.

b.

14.
```
FORM O
  FROM ITEMNO, SUPPNAME OF SUPPLIES
KEEP ROWS WHERE SUPPNAME = 'WARMTH, INC'
FORM P
  FROM ACCTNO, ITEMNO OF CHARGED
GROUP BY ACCTNO
KEEP GROUPS WHERE ITEMNO CONTAINS ITEMNO OF O
PRINT ACCTNO
```

a.

b.

15.
```
FORM Q
  FROM DEPT, ITEMNO OF SELLS
ADD COLUMN RETAIL
  OF ITEM
  BY ITEMNO = ITEMNO
PRINT DEPT, ITEMNO, RETAIL
```

a.

b.

ACCOUNTING 397A - Review quiz solutions.

1.
```
 FORM A
   FROM SUPPNAME, LOCATION OF SUPPLIER
KEEP ROWS WHERE SUPPNAME = 'WARMTH, INC'
PRINT LOCATION
```

a. ANCHORAGE
b. Where is Warmth, Inc. located?

2.
```
 FORM B
   FROM NAME, RATING OF CHARGEACCTS
KEEP ROWS WHERE RATING > '8'
PRINT NAME
```

a. JAMES LEE
b. List the names of people with credit rating greater than 8.

3.
```
 FORM C
   FROM DESCRIPTION, ONHAND, REORDER OF ITEM
KEEP ROWS WHERE ONHAND <= REORDER
PRINT DESCRIPTION
```

a. SHEEPSKIN SLIPPER
WOOL BLANKET
b. List the descriptions of items that need to be reordered.

4.
```
 FORM D
   FROM SUPPNAME, ITEMNO, ONORDER OF SUPPLIES
KEEP ROWS WHERE ITEMNO = '3' AND ONORDER > '400'
PRINT SUPPNAME
```

a. JIM'S SPORTS
b. List the names of suppliers who have more than 400 of item 3 on order.

5.
```
 FORM E
   FROM SUPPNAME, ITEMNO, WHOLESALE, ONORDER OF SUPPLIES
KEEP ROWS WHERE WHOLESALE * ONORDER > '5000'
PRINT SUPPNAME, ITEMNO
```

a. JIM'S SPORTS 1
JIM'S SPORTS 3
b. List the supplier names and item numbers for which the bill for the quantity on order will exceed $5,000.

6.
```
 FORM F
   FROM RATING OF CHARGEACCTS
KEEP ROWS WHERE RATING = '10'
PRINT COUNT(RATING)
```

a. Not enough information.
b. How many charge accounts have a credit rating of 10?

7.
```
FORM G
  FROM FLOOR, DEPT OF DEPARTMENT
ADD COLUMN ITEMNO
  OF SELLS
  BY DEPT = DEPT
KEEP ROWS WHERE ITEMNO = '3'
PRINT FLOOR
```

a. 2

b. Which floor is the department that sells itemn 3 located on?

8.
```
FORM H
  FROM ITEMNO, DESCRIPTION OF ITEM
ADD COLUMN SUPPNAME
  OF SUPPLIES
  BY ITEMNO = ITEMNO
ADD COLUMN LOCATION
  OF SUPPLIER
  BY SUPPNAME = SUPPNAME
KEEP ROWS WHERE DESCRIPTION = 'WOOL BLANKET'
PRINT LOCATION
```

a. BOSTON
ANCHORAGE

b. List the locations of suppliers who supply wool blankets.

9.
```
FORM I
  FROM DESCRIPTION, ITEMNO OF ITEM
ADD COLUMN ITEMNO (AS OTHERITEMNO)
  OF ITEM
  BY ITEMNO = ALTERNATE
KEEP ROWS WHERE OTHERITEMNO = '3'
PRINT DESCRIPTION
```

a. THERMAL BLANKET

b. What is the description of item 3's alternate?

10.
```
FORM J
  FROM SUPPNAME, LOCATION OF SUPPLIER
ADD COLUMN ITEMNO
  OF SUPPLIES
  BY SUPPNAME = SUPPNAME
KEEP ROWS WHERE ITEMNO = '3' AND LOCATION = 'BOSTON'
PRINT SUPPNAME
```

a. JIM'S SPORTS

b. Which suppliers located in Boston supply item 3?

11.
```
FORM K
  FROM SUPPNAME, LOCATION OF SUPPLIER
KEEP ROWS WHERE LOCATION = 'BOSTON'
FORM L
  FROM SUPPNAME, ITEMNO OF SUPPLIES
KEEP ROWS WHERE ITEMNO = '3'
KEEP ROWS OF K WHERE SUPPNAME IN SUPPNAME OF L
PRINT SUPPNAME
```

a. JIM'S SPORTS
b. Which suppliers located in Boston supply item 3?

12.
```
FORM M
  FROM LOCATION OF SUPPLIER
GROUP BY LOCATION
KEEP GROUPS WHERE COUNT(LOCATION) > '5'
PRINT LOCATION
```

a. Not enough information.
b. List the cities in which more than 5 suppliers are located.

13.
```
FORM N
  FROM ACCTNO, PERITEM OF CHARGED
GROUP BY ACCTNO
KEEP GROUPS WHERE COUNT(*) > '10'
PRINT ACCTNO, AVG(PERITEM)
```

a. Not enough information.
b. For the accounts that have charged more than 5 different items list the account number and the average price per item.

14.
```
FORM O
  FROM ITEMNO, SUPPNAME OF SUPPLIES
KEEP ROWS WHERE SUPPNAME = 'WARMTH, INC'
FORM P
  FROM ACCTNO, ITEMNO OF CHARGED
GROUP BY ACCTNO
KEEP GROUPS WHERE ITEMNO CONTAINS ITEMNO OF O
PRINT ACCTNO
```

a. Not enough information.
b. List the account numbers that have charged all the items supplied by Warmth, Inc.

15.
```
FORM Q
  FROM DEPT, ITEMNO OF SELLS
ADD COLUMN RETAIL
  OF ITEM
  BY ITEMNO = ITEMNO
PRINT DEPT, ITEMNO, RETAIL
```

a.
```
CLOTHES    1      23.00
CLOTHES    2      19.00
CAMPING    3      27.50
```
b. List the department, the item numbers of items they sell and the retail price of the items.

ACCOUNTING 397A - About the last two tests.

The final will determine your grade. It will be given on Wednesday, April 5 from 6:30 pm to 8:30 pm in SBA 100. THE FINAL IS OPEN BOOK; BE SURE TO BRING YOUR TEXT. If you cannot come at that time, see your instructor after class. If something comes up between now and then, call Charles Welty at 247-9357 (home); if no answer try 545-1500 (office). Leave your name and phone number if Charles Welty is not in his office and your call will be returned.

The retention test will not affect your grade unless you do not take it. The retention test will be given on Wednesday, April 26 from 6:30 pm to 8:30 pm in SBA 100. When you hand in your retention test you will be told your grade in the course. Contact Charles Welty as above if you must miss the scheduled time.

Note: If you have not paid for the text ($3.00) by the time grades are due, you will receive an incomplete in the course.

ACCOUNTING 397A - FINAL

This exam is quite long and you are not expected to finish. Go through the exam once doing all the problems you find easy, then go through the exam again doing the more difficult ones.

Name:_____

Time started: _____ Time finished:_____

FINAL

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. List the item numbers of items costing less than $5.00 that need to be reordered and have item 8 as their alternate.

5. 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.

ACCOUNTING 397A - Retention test reminder.

The retention test will be given on Wednesday April 26 at 6:30 PM in SBA 100. It will also be given on Thursday April 27 at 6:30 PM in SBA 100. If you cannot come at either of these times call Charles Welty first at 247-9357 (home). If no answer call 545-1500 (office), leave a message if Charles Welty is not in.

Course evaluation.

Please fill out the enclosed course evaluation form and bring it to the retention test.

ACCOUNTING 397A - Course evaluation.

Name (optional) _____

Which language did you study? SQL _____ TABLET _____

What aspect of the course did you like the most?

What aspect of the course did you like the least?

What parts of the language did you find easy?

What parts of the language did you find difficult?

Was the text helpful?

What was your motivation for taking the course?

What did you get out of the course?

Was the course too little work, the right amount of work or too much work?

Comments on the class meetings:

General comments:

ACCOUNTING 397A - RETENTION

This exam is quite long and you are not expected to finish. Go through the exam once doing all the problems you find easy, then go through the exam again doing the more difficult ones.

Name:_____

Time started: _____ Time finished: _____

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 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 supplies.

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.

A P P E N D I X D

SUBJECT BACKGROUNDS

Class	SQL	TABLET
Senior	5	4
Junior	5	5
Sophomore	14	6
Freshman	15	22
Major		
Business Administration	17	19
Accounting	5	7
Fashion Marketing	1	2
Marketing	1	1
Other	11	8
Computer Experience		
None	17	20
BASIC	10	10
FORTRAN	5	6
Other	3	1
Math Background		
Calculus	28	27
Pre-calculus	7	10
Familiarity with Math Symbols		
>	34	36
<	34	36
=	34	36
$\cup$	25	27
$\cap$	26	27