# TABLE OF CONTENTS

I. INTRODUCTION ........................................... 1
II. ENVIRONMENT ........................................... 2
III. EXPLANATION OF THE QBE SYSTEM ................. 3
IV. LOGICAL AND PHYSICAL DESIGN .................... 6
V. ADDING A NEW RELATION TO THE QBE SYSTEM ..... 9
VI. DELETING AN EXISTING RELATION FROM THE QBE SYSTEM 10
VII. SORTING ALGORITHM .................................. 11
VIII. ENHANCEMENTS ...................................... 12
IX. COMMON BASIC BUGS ................................... 14

APPENDIX A. QUERY BY EXAMPLE SYSTEM SPECIFICATIONS 15
APPENDIX B. EXAMPLE RELATIONS ......................... 21
APPENDIX C. QUERY BY EXAMPLE QUERY EXAMPLES ..... 22
APPENDIX D. QUERY BY EXAMPLE DDL/DML TABLES ..... 28
APPENDIX E. TERMINOLOGY ................................ 30
APPENDIX F. QUERY BY EXAMPLE COMMAND/FUNCTION SUMMARY 31
APPENDIX G. QBE SYSTEM MESSAGES .................... 32
APPENDIX H. DATA DICTIONARY .......................... 41
APPENDIX I. SCROLL EXAMPLE ............................ 44
APPENDIX J. EXECUTION TIMES OF SORT PROGRAMS AND AND QUICKSORT PROGRAMS 45
APPENDIX K. LEXICAL PARSER EXAMPLES ............... 48
APPENDIX L. QUERY BY EXAMPLE SYSTEM PROGRAM CHARTS 49
APPENDIX M. SCREEN TABLE COMMANDS ................ 51
APPENDIX N. THE EVALUATION OF THE 'AND' INTERNAL LINKED QUERY 52
APPENDIX O. QBE RETRIEVAL SEARCH ALGORITHM .... 54
APPENDIX P. LISTING OF THE QBE SYSTEM PROGRAMS .... 55

BIBLIOGRAPHY .............................................. 56
INTRODUCTION

This final report outlines the general details of the completed Graduate Project. The Graduate Project simulates a subset of IBM's Query By Example (viz. the Graduate Proposal for a complete description of the Project and Date (3)). It should be noted that Query By Example uses a two-dimensional syntax and that this syntax allows the user to make powerful queries using a simple Query syntax. Appendix F. outlines the various QBE elements (i.e. example elements, constants, relational operators and functions). Appendix E. outlines the terminology that is used in Relational Data Bases.

The final Graduate Project deviates from the Graduate Proposal in two areas. First, the process of creating a new relation in the Query By Example system is taken from Zloof (12) rather than Date (3, 4). The author uses an abbreviated form of Zloof's DDL table (viz. Appendix D.1). The author's DDL table allows the user to create a new relation in one step. That is, all Data Base parameters concerning the new relation can be entered at the same time (as oppose to entering the relational name and then entering its parameters at a later time). Finally, the completed Graduate Project does not include any update commands (i.e. 'U.'), but the author feels that the final Project adequately demonstrates the environment of Query By Example.
ENVIRONMENT

The host computer for this project is the IBM Personal Computer with two (2) disk drives. This project is written for single-sided, single-density disk drives. A minimum of 64k is required in order to run the OBE system programs.

This project is written in the IBM Personal Computer BASIC, Version D1.00. It should be noted that this BASIC Interpreter can access only 64k bytes of information (this is standard with most BASIC interpreters written by the Microsoft Co.).
EXPLANATION OF THE QBE SYSTEM PROGRAMS

The Query By Example System is composed of the following programs:
1. PROGRAM.1 - Create a new relation
2. PROGRAM.2 - Save the new relation
3. INIT.1 - Initialize a QBE system diskette
4. QBEMAIN.1 - Initialization program
5. QBEMAIN.2 - Initialization program
6. XLINK.1 - linked program
7. QBEDE.35 - QBE Add/Delete program
8. QBESORT.1 - QBE sort program
9. QBERETR.1 - QBE retrieval program
10. PRINT.F - Retrieval print program(function)
11. PRINT.PI - Retrieval print program(print/ internal)
12. EXTLINK.1 - Retrieval External program
13. DUMPFILE.S - Utility program

PROGRAM.1 is the QBE system program that allows the user to create a new relation in the QBE system. This program uses a DDL table that is more similar to Zloof's(12) DDL table than Date's(3) DDL table. The DDL table utilized in this project is an abbreviated form of Zloof's; and, therefore, there are many options that are not utilized. This includes the following: integrity constraints, time, float, fix and date pictures(types), and security constraints.

Appendix D.1 gives two examples of typical Query By Example DDL tables. An '*' in any attribute column signifies that that column is to be a component of the primary key. If no '*' is placed in any of the attribute columns(as in table #2) then all of the attribute columns will make up the primary key(this is standard QBE). The limits row in table #2 states the following: that all students with a status equal to 'graduates' must have a GPA greater than a 3. The numeric constant(i.e. the 3) must be a positive integer number. The irreflexive condition in table #2 states the following: that at any time a student may not be a professor. The '<<>' must be associated with the second constant element. (It should be noted that standard QBE uses an example element instead of a constant element.) Finally, if no picture is specified for any particular domain, then the default is 'CHAR(20)'. The QBE system messages for PROGRAM.1 are outlined in Appendix G.1. PROGRAM.1 will accept only valid input(i.e. relations) and sends this information to PROGRAM.2. PROGRAM.2 does the actual saving of the new relation into the QBE system.
PROGRAM.2 also does additional OBE system checks. These include checking for enough Data Base space on the Data Base diskette and checking for consistent Domain pictures (types). Appendix A.1 outlines the OBE system specifications. PROGRAM.2 performs specification checks outlined in this Appendix. The OBE system messages for PROGRAM.2 are outlined in Appendix G.2.

INIT.1 prepares a formatted IBM diskette to be used by the OBE system. This program creates eight (8) relational files, one (1) domain file, one (1) control file and one (1) relational name file. All the OBE system files are initialized to blanks/zeroes. A linked list is built in each of the eight (8) relational files. Appendix A outlines the OBE system files.

OBEMAIN.1 and OBEMAIN.2 are DML initialization programs. OBEMAIN.1 accepts input from the screen (e.g. name of a relation) and retrieves the OBE system information from the appropriate relational file. This information is sent to the appropriate program (e.g. OBEDE.35 or XLINK.1). XLINK.1 removes all the OBE system information before chaining to the appropriate OBE system program (e.g. INIT.1, PROGRAM.1 or DUMPFIL.E.S). OBEMAIN.2 performs the same functions as OBEMAIN.1 except that it chains to the OBE retrieval program -- OBERE.TR.1. The OBE system messages for OBEMAIN.1/OBEMAIN.2 are outlined in Appendix G.3.

OBEDE.35 allows the user to add/delete data to/from the OBE system. Appendix D.2 gives four examples of valid OBE ‘insert’ and ‘delete’ commands. The relations for this appendix are outlined in Appendix B. (These relations were taken from Data’s(3) Data Base book.) OBEDE.35 will accept only the ‘I.’ and ‘D.’ OBE commands. The OBE system messages are outlined in Appendix G.4. If any data has been add/deleted to/from the OBE system then the program will automatically chain to the OBESORT.1 program. This sort program generates the inversion keys for each attribute column in the current relation (i.e. the relation utilized in OBEDE.35). The inversion keys are stored in records 11 through 11+n-1, where n is the total number of records (viz. Appendix A.2). The OBE system messages for OBESORT.1 are outlined in Appendix G.5.
OBERETR.1 allows the user to retrieve data from the OBE system. This retrieval program is partitioned into four (4) major modules:

1. OBERETR.1 - mainline retrieval program
2. PRINT.F - prints out the OBE function query result
3. PRINT.PI - prints out the OBE print/internal query result(s)
4. EXTLINK.1 - evaluates a OBE external query and prints out the result(s)

Appendix F. outlines the various OBE system commands. It should be noted that an 'x' is used to denote an example element (standard OBE example elements are underlined). Various OBE query examples are given in Appendix C. The relations utilized are outlined in Appendix B. Appendices G.6, G.7, G.8 and G.9 outlined the OBE retrieval messages.

DUMPFILE.S is a utility program that allows the user to display/print-out the contents of all the OBE system files.

Appendix H. outlines the major program variables that are used in the OBE system.
LOGICAL AND PHYSICAL DESIGN

The Logical and Physical Design encompasses the utilization of Random Access files, Linked lists and Inversion keys. Appendix A outlines the general description of all the OBE system files. A general program outline is given in Appendix L. The OBE system programs are divided into two (2) main sections. The first section (which is on one (1) diskette) contains the OBE programs that allows the user to create a new relation, add/delete data to/from a relation, initialize a new diskette, and display/print-out the contents of the OBE system files. The second section (which is on one (1) diskette) allows the user to retrieve records from the OBE system. A brief description of the logical design will be given for the following major OBE system programs: A. PROGRAM.1 B. OBEDE.35 C. OBESORT.1 D. OBERETR.1

PROGRAM.1 is the OBE system program that allows the user to create a new relation. A user may enter a relation with anywhere from two (2) to five (5) attribute columns. Example DDL tables are given in Appendix D.1. The screen input is stored in two-dimensional tables (i.e. table$(6,2)$, table2$(6,3)$, table4$(6,2)$). Appendix M. outlines the valid screen input commands that the user may use. After all the input has been entered, the program performs a lexical check on the new relation’s parameters (viz. Appendix A.). Once accepted, the parameters are stored in the ‘ParseTable$(6,5)$’ and sent to PROGRAM.2. PROGRAM.2 is responsible for saving the new OBE system relation’s parameters.
QBEDE.35 is the QBE system program that allows the user to add/delete data to/from the any particular QBE relation. All new records are stored in arrays (i.e., D1*(100)...D5*(100)). The primary keys (if any) are loaded in Pkeys*(100), and these keys are used to check for duplicate primary keys (since all primary keys in QBE must be unique and non-nulled). If the program is aborted then the new records are not added to the current relation. The deletion of records involve a more complex algorithm. First, the first attribute column value is used as the search argument in locating a record in the current relation. That is, the first attribute column is binary search using that attribute column value as the search argument. If a match is found, then the Pivot variable is set to that location. Once a Pivot location is found, two "search/delete" subroutines ('single-step' and 'backtrack') are used to delete all the appropriate record(s). These subroutines dynamically delete the record(s) from the relational disk file. Since a linked list is used in connecting the records in each relation, these two subroutines must adjust the links when a record is deleted. The arrays 'UNKS*(100), LDC2*(100)' are used by these two subroutines. These arrays contain dynamic 'doubly links'. Therefore, when a record is deleted it is a simple matter in finding the record that comes before it and after it. It should be noted that the author does not recommend the use of linked lists in the building of a Relational Data Base system. Most Relational Data Base systems utilize the Inverted file structure (e.g., Inversion keys).

Finally, if the program is aborted then there is no way of un-deleting the record(s). The user must re-enter the deleted records. (The author did not have time to add a subroutine that back-ups the current relation at initialization time and then re-writes this back-up file to the current relation at abort time.)

The input from the screen is loaded into a two-dimensional table (i.e., 'TABLE*(7,13,2)'). What is seen on the screen is what is actually in the two-dimensional table. Therefore, relatively simple procedures are used in expanding a particular attribute column and in scrolling a particular table (viz., Appendix I.). It should be noted that a particular input is not saved in the table until the user hits the "return" key. Appendix M. outlines the valid screen commands.

The QBESORT.1 program generates the Inversion keys for each particular relation (via PROGRAM.2). The Quicksort program was chosen since it is the fastest general purpose internal sorting program today (viz., explanation on the choice of this sort program). QBESORT.1 uses the sort flags (viz., Appendix A.3) to determine which attribute columns to sort. If the sort flag is less then one (1) then that particular attribute column is sorted. The Inversion keys are saved in records 11 through 11+n-1, where n is equal to the total number data records in the relation.
The OBERETR.1 program allows the user to retrieve records from the OBE system. Appendix L.2 outlines the general program layout for the OBE retrieval programs. The input routine is identical to the input routine in QBEDE.35 except that the OBERETR.1 program uses a three-dimensional screen table (i.e. TABLE$(row, column, rn)) where rn = relational number. The data for a particular table is loaded in memory (i.e. DI$(110)$...DS$(110)$) whenever that screen table is being queried. Therefore, this speeds up the execution of the search routines in satisfying a particular query. It should be noted that all data is loaded in their relative locations. That is record #10 is loaded in array location 10.

The OBE queries are partitioned into four (4) groups: 1. Print Queries 2. Function Queries 3. Internal Queries and 4. External Queries. See Appendix C. for an example of each type of query. Each of these four sections have their own specialized parsing routine. For example, the external parser checks for the presence of 2 (two) example elements. In addition, to these specialized parsers there is a general purpose lexical parser (viz. Appendix K.) whose primary function involves the evaluation of the OBE queries. This parser perform two general functions. First, it checks for valid input. Secondly, it initializes all the appropriate option (e.g. 'A0.', 'P.', etc.) flags (via 0$(11, 5)$) and the appropriate relational operator flags (via 'COND$(5, 2)$', 'STACK$(5, 2)$', 'PS$(5, 2)$', and 'CP$(5, 2)$').

Refer to Appendix Q. for a general description of the searching technique utilized in the OBERETR.1/EXTLINK.1 programs. If any records are found then the OBERETR.1 program 'chains' to the appropriate 'print' program (i.e. PRINT.F, PRINT.PI, and EXTLINK.1). After the data is print out, the 'print' programs 'chain' back to OBERETR.1.
ADDING A NEW RELATION TO THE QBE SYSTEM

The process of physically adding a new relation to the QBE system involves the following steps:

1. Get valid Data Base parameters from the user(PROGRAM.1).
2. Save the new Data Base parameters:
   A. Search for the next available space in the relational name file (i.e., RELATNAME). If there is no available space then terminate the program.
      If there is available space in this file then save the name of the new relation (i.e., relationalname$) in that location(relationalloc). Increment the relational counter in the control file(CNTRL.1).
   B. Search for available space in the Domain file(DOMAINS).
      If there is no available space then terminate the program.
      If there is available space then save any new domains and record their location. If any particular domain already exist, then just record its location.
   C. Save the new Data Base parameters in the specified relational file (RELAT.N, where N = relationalloc). Refer to Appendix A for a complete description of the QBE system files.
DELETING AN EXISTING RELATION FROM THE OBE SYSTEM

The process of physically deleting an existing relation from the OBE system involves the following steps:

1. Delete every record in the relation (viz. Appendix D.2) via the OBE Add/Delete program(OBEDE.35).

2. Delete the relational name from the OBE system relational name file(RELATNAME). Therefore, this space is made available to the Data Base system.

3. Decrement the appropriate Domain counters in the OBE system domain file(DOMAIN). If any of the counters is equal to zero, then delete its corresponding domain name from this file. Therefore, this space is made available to the Data Base system.

4. Decrement the total number of relations counter in the OBE system control file(CNTRL.1).
SORTING ALGORITHM

The Quicksort program was chosen for this project since it is the fastest general purpose internal sort available today. There are many versions of C.A.R. Hoare's Quicksort program; and, therefore, the author selected the program in Wirth's(10) Data Structures book. Appendix J.2 and Appendix J.3 give two versions of the Quicksort program. The first version requires a stack size \( n \), where \( n \) is the total number of records. The second version requires a stack size \( 1.442695 \times \log(n) \), where \( \log(n) \) is the natural logarithm of \( n \) elements. The second version requires a smaller stack size since it stacks only the larger of the two partitions and continues sorting the smaller partition.

Appendix J.1 gives the execution times for various internal sort programs (from Wirth(10)). It is evident that the Quicksort is at least twice as fast as the Shellsort and several times faster than the other sort programs (with the exception of the Bubblesort when sorting ordered data). However, it should be noted that each of the six sort programs can be 'tuned' to increase its efficiency. For example, the Hibbard distance (Knuth(7)) can be used to decrease the number of data moves that the Shellsort performs. Sedgewick(8) gives a comprehensive study of the Quicksort program in the CACM and he includes many techniques for increasing the efficiency of the Quicksort program. For example, 'loop-unwrapping' may be used to speed up the sort program.

It should be noted that the first If-statement in the Quicksort program is not required to be 'If I<=J', but instead could have been 'If I<J'. However, using 'If I<J' keeps the inner loops fast. Knuth(7) attributes this observation to Singleton: "that 'If I<=J' is used to keep the inner loops fast and helps split the subfiles nearly in half when equal elements are present."