The Query Management Facility

by J. J. Sordi

Data from a relational data base can be displayed in reports, changed, and otherwise controlled using a program called Query Management Facility (QMF). An overview of this program is presented and is followed by a discussion comparing equivalent forms of various queries expressed in two distinctly different languages. Both languages are designed for use with relational data and are supported by QMF.

The Query Management Facility (QMF)[1,2] provides a full-screen, interactive environment to query, define, and control relational data contained in either the IBM Database 2 (DB2)[3] or the Structured Query Language/Data System (SQL/DS)[4] data base systems. QMF is used from a visual display terminal and is functional only when installed with either DB2 or SQL/DS.

Relational data contained in DB2 and in SQL/DS is perceived to exist in table form as shown in Table 1. A considerable understanding of QMF can be obtained knowing no more than that about relational data; however, an understanding of some of the more complex applications of QMF requires some additional background. Other papers in this issue of the *IBM Systems Journal* address the DB2 data base system and provide some of that background. G. Sandberg's "A primer on relational data base concepts"[5] and C. J. Date's book, *An Introduction to Database Systems*,[6] are other good sources.

QMF provides a means of retrieving and displaying data and of inserting, deleting, and updating data using either a form of the Query-by-Example (QBE) language[7-10] conceived by M. M. Zloof at IBM's Thomas J. Watson Research Center or the Structured Query Language (SQL)[3,4,11-13] developed at IBM's San Jose Research Center. Although both QBE and SQL are designed specifically for use with relational data, they offer the QMF user a distinct choice between two languages that are very different in form and syntax.

This paper provides a QMF overview and discusses the types of queries that may be created using either QBE or SQL. It also shows the equivalent QBE and SQL syntax that may be used to create such queries. See References 1 and 2 for more detailed descriptions of QMF.

QMF overview

Every QMF session begins (and ends) with the QMF home panel shown in Figure 1. QMF items can be viewed by pressing the associated PF keys shown at the bottom of the display. Messages from QMF such as "OK, you may enter a command," will appear on the message line, just above the COMMAND line.

QMF retrievals. Using the QUERY PF key displays the last query created during a session. If none has been created (as would be the case upon first viewing the home panel), a blank panel that accommodates the entry of either a QBE query or an SQL query is displayed. A blank QBE panel is shown in Figure 2.

Figure 1 QMF home panel

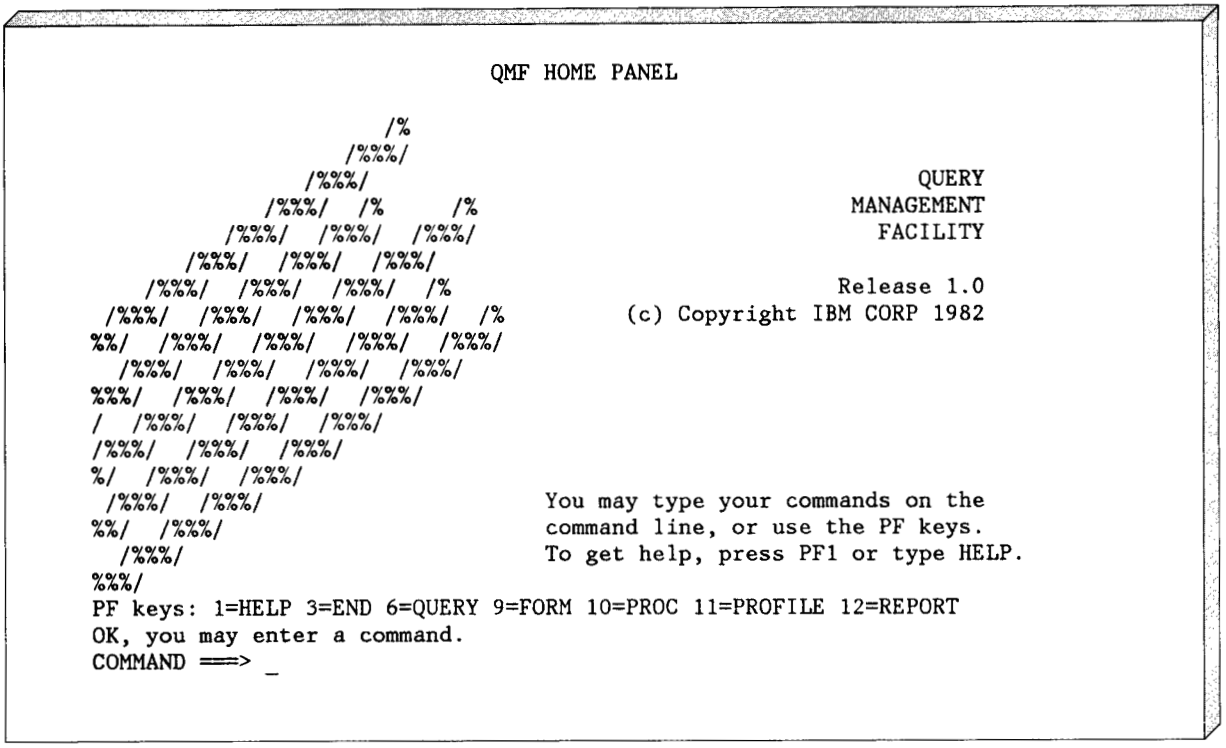


Table 1 The employee table

NAME	ENO	DNO	MGR	SAL	OTIME
PETRA DAVIS	775	46	299	10750	500
FRANK CANTRELL	529	77	182	9650	800
CAROL CAMPBELL	299	46	182	17500	—
JANET HILL	349	12	182	8000	1200
ANGELO DIVISIERO	284	51	299	6800	1500
WILLIAM JAMES	209	87	238	5500	250
RICHARD GOMEZ	182	51	014	25500	—
ANN MARIE SWANSON	125	46	182	13500	1340
RANDOLPH DREW	362	23	299	11700	1100
JANET GOULD	238	77	182	21000	—
TIMOTHY HAROLDSON	579	12	299	12500	—
STEVE WHITE	382	87	238	16500	—
JAMES VANZANT	234	46	182	11000	780
JOSEPH RUSSELL	787	87	299	18000	2300
PAMELA GRIFFITH	361	23	238	8500	1280
VERNON COLLINS	948	73	299	12400	650
SHARON ROSTEGE	271	51	238	6100	40

The intersection of every row and column of a table contains a data element or a null. A null signifies the absence of a value and is represented by a dash (—).

Figure 2 Blank QBE query panel

```
QBE QUERY                                                    LINE 1

*** END ***




1=HELP 2=RUN 3=END 4=ENL 5=REDUCE 6=DRAW 7=BA 8=FO 9=FORM 10=LE 11=RI 12=REPORT
OK, This is an empty QBE QUERY panel.
COMMAND ===>                                          SCROLL ===> PAGE
```

Figure 3 QBE retrieval query

```
QBE QUERY                                          MODIFIED     LINE 1

EMP    | NAME       |  ENO   | DNO | MGR |  SAL    | OTIME  |
-------+------------+--------+-----+-----+---------+--------|
P.     |            |        | AO. |     | >12000  |        |

*** END ***




1=HELP 2=RUN 3=END 4=ENL 5=REDUCE 6=DRAW 7=BA 8=FO 9=FORM 10=LE 11=RI 12=REPORT

COMMAND ===> _                                        SCROLL ===> PAGE
```

In order to create the QBE retrieval in Figure 3 from a blank panel, the DRAW command is used to obtain a "skeleton" of the EMP table by keying in EMP on the command line and pressing the DRAW PF key (or by keying in DRAW EMP on the command line and pressing ENTER). A skeleton of the EMP table will appear on the display, and the user completes the query by keying in the "P.", "AO.", and ">12000" at the appropriate locations in the skeleton.

The PAGE option in the lower right corner of the display shows the scrolling increment. That is, if the query were more than one page (screen-full) long, pressing PF key 8 would scroll forward by one page. The scroll option may be changed simply by keying over PAGE. For example, one may key in HALF for a half-page scroll, or 10 for a scroll of 10 lines at a time. The line number in the upper right corner indicates which line of the query is at the top of the display. For example, if the top nine lines were scrolled off the top of the display, 10 would appear as the first query line number on the display.

The query in Figure 3 asks for the name, employee number, department number, manager number, salary, and overtime pay of each employee who has a salary greater than $12 000. It also indicates that the results should appear in ascending order by department number.

The MODIFIED notation on the first line appears when the query is modified from the way it first appeared on the display. In this case, simply making entries into the blank QBE panel causes the appearance of the MODIFIED notation.

When the query is run by pressing the RUN PF key, the report in Figure 4 containing the results of the retrieval replaces the query on the display. The POS notation in the upper right corner indicates the left- and right-character position of the report on the display. In this case, the leftmost position is 1 and the rightmost position is 79. If a report is wider than the display, it may be scrolled left and right.

Using the QUERY PF key from the home panel may produce a blank SQL panel rather than a blank QBE panel. The SQL query in Figure 5, entered into a blank SQL panel, is semantically equivalent to the preceding QBE query and will produce an exact duplicate of the report shown in Figure 4.

Pressing the QUERY PF key causes the last query created during a session to be displayed. If none has

Figure 4 Query report

```
REPORT                                              LINE 1      POS 1     79

  NAME              ENO  DNO  MGR  SAL    OTIME
------------------  ---  ---  ---  -----  -----
TIMOTHY HAROLDSON   579  12   299  12500    -
CAROL CAMPBELL      299  46   182  17500    -
ANN MARIE SWANSON   125  46   182  13500   1340
RICHARD GOMEZ       182  51   014  25500    -
VERNON COLLINS      948  73   299  12400    650
JANET GOULD         238  77   182  21000    -
STEVE WHITE         382  87   238  16500    -
JOSEPH RUSSELL      787  87   299  18000   2300
*** END ***

1=HELP 3=END 4=PRINT 6=QUERY 7=BACKWARD 8=FORWARD 9=FORM 10=LEFT 11=RIGHT
This is the report from your RUN command.
COMMAND ===> _                                          SCROLL ===> PAGE
```

Figure 5 SQL retrieval query

```
SQL QUERY                                              MODIFIED       LINE 1

SELECT NAME,ENO,DNO,MGR,SAL,OTIME
FROM EMP
WHERE SAL > 12000
ORDER BY DNO

*** END ***


1=HELP 2=RUN 3=END 4=PRINT 5=RESET 7=BACK 8=FOR 9=FORM 10=INS 11=DEL 12=REPORT

COMMAND ===>                                                  SCROLL ===> PAGE
```

Figure 6 QMF form

```
FORM

Column Descriptions:                    Page width is now: 48
  NUM  COLUMN HEADING                          USAGE    INDENT  WIDTH  EDIT
  ---  ---------------------------------------  -------  ------  -----  ----
    1  NAME                                              2       18     C
    2  ENO                                               2       3      L
    3  DNO                                               2       3      L
    4  MGR                                               2       3      L
    5  SAL                                               2       5      L
    6  OTIME                                             2       4      L
*** END ***

Control Break Text:                          Do you want OUTLINE? ===> YES
    1 ===>                              2 ===>
    3 ===>                              4 ===>
    5 ===>                              6 ===>

PAGE HEADING ===>
PAGE FOOTING ===>

1=HELP 3=END 4=PRINT 5=RESET 6=QUERY 7=BACKWARD 8=FORWARD 12=REPORT
OK, FORM is displayed.
COMMAND ===>                                                  SCROLL ===> PAGE
```

been created, as when the QMF home panel first appears at the beginning of a session, a blank QBE or SQL panel appears, depending upon the choice indicated in a user's profile.

A user's profile can be displayed by pressing the PROFILE PF key from the home panel, and the language choice can be changed by simply keying in QBE or SQL over the current choice. The change applies only for the current session, unless the SAVE command is used to save the modified profile.

An empty QBE or SQL panel may also be obtained at any time during a session by using the RESET command. If no language is given in the RESET command, a blank panel will appear according to the profile choice. A choice may be given in the RESET command. For example,

```
RESET QUERY (LANGUAGE = QBE)
```

will give a blank QBE panel, no matter what is specified in the user's profile.

The report form. QMF produces a form that describes the format of the report that was produced as the result of a retrieval. The form may be viewed by pressing the FORM PF key after the report is produced. For the report in Figure 4, the form will appear on the display as shown in Figure 6.

The INDENT column gives the number of blanks that precede each column. The EDIT column gives the edit characteristics of each column—either character (C) or numeral with no decimal places (L) in this example. The USAGE, Control Break Text, and OUTLINE are options for which there are no entries in the form produced for the report in Figure 4. They may be used to produce a modified report (as will be discussed shortly). The column names, indentations, etc., may be changed to produce a modified report

Figure 7 Modified QMF form

```
FORM                                                              MODIFIED

Column Descriptions:                  Page width is now: 73
  NUM  COLUMN HEADING                          USAGE    INDENT  WIDTH  EDIT
  ---  ---------------------------------------  -------  ------  -----  ----
   1   EMPLOYEE_NAME                                    2       18     C
   2   ENO                                      OMIT    2       3      L
   3   DEPARTMENT_NUMBER                                10      10     L
   4   MANAGER                                          2       8      L
   5   SALARY                                           2       6      L
   6   OVERTIME                                         2       8      L
*** END ***

Control Break Text:                        Do you want OUTLINE? ===> YES
    1 ===>                              2 ===>
    3 ===>                              4 ===>
    5 ===>                              6 ===>

PAGE HEADING ===> EMPLOYEES MAKING OVER 12000
PAGE FOOTING ===>

1=HELP 3=END 4=PRINT 5=RESET 6=QUERY 7=BACKWARD 8=FORWARD 12=REPORT
OK, FORM displayed.
COMMAND ===>                                              SCROLL ===> PAGE
```

by simply keying in the changes on the FORM panel, as for example in Figure 7.

After keying in the changes to produce this modified form, the REPORT PF key may be used to display data from the last retrieval run, as shown in Figure 8.

Other types of reports can be produced from the same data. For instance, BREAK1 can be keyed in under USAGE and next to DNO. This will cause a break to occur after each department number change. If SUM is entered next to SAL, a summary of salaries by department will appear at each control break along with any control break text given in the form. If OUTLINE is set to YES, the break values for DNO will not be repeated for each detail line. These and other changes to the preceding form would produce a summary report like the one in Figure 9. See Reference 2 for more detailed information on the form options and types of reports that can be produced using QMF.

If different data is needed to produce the desired report, pressing the QUERY PF key will display the last retrieval, which may then be modified and run to retrieve different data.

The current query, form, and retrieved data may be saved at any time by using the SAVE command. For instance, after a satisfactory report is produced, one may choose to save the form, the retrieved data, and the query that produced the data. A second form (or third, etc.) may then be produced and saved to view the same data in a different report. At a later time, the saved query may be run again to retrieve current data, and the saved forms used to produce reports from the current data. The ERASE command may be used at any time to erase items when they are no longer useful.

Insert, update, and delete queries. Either QBE or SQL can be used to insert or delete rows in tables or to update the contents of a particular row and column. In the following query, for example, QBE is used to insert three new employees in the employee table:

QBE Query

EMP	NAME	ENO	DNO	MGR	SAL
I.	'HECTOR URESTI'				12000
I.	'DONALD WURM'				8000
I.	'JAMES MOON'				8700

*** END ***

Figure 8 Report using a modified form

```
REPORT                                          LINE 1      POS 1     79
                    EMPLOYEES MAKING OVER 12000
EMPLOYEE                  DEPARTMENT
NAME                      NUMBER      MANAGER  SALARY  OVERTIME
------------------        ----------  -------  ------  --------
TIMOTHY HAROLDSON         12          299       12500    -
CAROL CAMPBELL            46          182       17500    -
ANN MARIE SWANSON         46          182       13500  1340
RICHARD GOMEZ             51          014       25500    -
VERNON COLLINS            73          299       12400   650
JANET GOULD               77          182       21000    -
STEVE WHITE               87          238       16500    -
JOSEPH RUSSELL            87          299       18000  2300
*** END ***

1=HELP 3=END 4=PRINT 6=QUERY 7=BACKWARD 8=FORWARD 9=FORM 10=LEFT 11=RIGHT
OK, Report is displayed.
COMMAND ===>                                          SCROLL ===> PAGE
```

Figure 9 Report with a control break

```
REPORT                                                LINE 1      POS 1     79
                    EMPLOYEES MAKING OVER 12000

DEPARTMENT  EMPLOYEE
NUMBER      NAME                 MANAGER          SALARY
----------  -------------------  -------          --------
12          TIMOTHY HAROLDSON    299               $12,500
                                                   --------
                    SUM OF SALARIES IN DEPT 12     $12,500

46          CAROL CAMPBELL       182               $17,500
            ANN MARIE SWANSON    182               $13,500
                                                   --------
                    SUM OF SALARIES IN DEPT 46     $31,000

73          VERNON COLLINS       299               $12,400
                                                   --------
                    SUM OF SALARIES IN DEPT 73     $12,400

77          JANET GOULD          182               $21,500
                                                   --------
                    SUM OF SALARIES IN DEPT 77     $21,500

87          STEVE WHITE          238               $16,500
            JOSEPH RUSSELL       299               $18,000
                                                   --------
                    SUM OF SALARIES IN DEPT 87     $34,500

                                                   ========
                                                   $111,900

                SUM OF SALARIES BY DEPARTMENT                    PAGE 1
*** END ***
1=HELP 2=RUN 3=END 4=PRINT 6=DRAW 7=BACKWARD 8=FORWARD 9=FORM 10=INSERT
11=DELETE 12=REPORT

COMMAND ===>                                              SCROLL ===> PAGE
```

The ENO, DNO, and MGR columns are set to null for each new row entered into the table because no values were given for these columns. Even though the OTIME column was not named in the query, it too gets a null for each new row.

Examples of QBE and SQL update, insert, and delete queries appear later in this paper.

QMF procedures. In QMF, a series of commands that constitute a procedure may be created using a PROC panel, and the entire procedure may be saved and run periodically. Depressing the PROC PF key provides the user with a blank PROC panel similar to a blank QBE or SQL panel. Figure 10 is the monthly report procedure, MONTHLY, that may have been keyed in or previously saved and now displayed.

When this procedure is run, it runs a saved retrieval called ED that uses a saved form called EMP_DEPT to produce a report. This report is printed before the results of the retrieval are saved in a table called EMPDEPT. Then it runs a saved retrieval called DL using a saved form called DIV_LOC to produce a report. This report is then printed before the results of the retrieval are saved in a table called DIVLOC.

Only the owner of a table may initially grant authority to others to retrieve, update, delete, or insert data in the table.

Table definitions and user authority. In the interest of saving space, SQL and QBE queries in this and the remaining parts of the paper will appear out of the context of the QMF display.

The preceding queries in this paper presume the existence and the accessibility of the tables being queried. Other QMF queries may be used to create or drop table definitions and to grant or revoke users' authority to manipulate data in such tables. Only the SQL syntax is available to express these types of functions.

The following SQL query, which is entered on a panel such as the one in Figure 5, grants Jones authority to delete rows from the DEPT table:

```
GRANT DELETE ON DEPT TO JONES
```

Only the owner of a table may initially grant authority to others to retrieve, update, delete, or insert data in the table. A table name in a query must be qualified by the owner's name, unless the person creating the query is also the owner. For example, if Smith owns the DEPT table, Jones can delete rows from the table only if Smith has granted delete authority to Jones. Even so, Jones must qualify the name of the table with the owner's name, as in the following query, in which the sales department is deleted from the DEPT table:

SMITH.DEPT	DNAME
D.	SALES

The name of the table, DEPT, is qualified by the owner's name, SMITH.

Table definitions may be created using a CREATE query; however, a typical user may create a new table and table definition by saving data resulting from a retrieval. The table definition for such a new table is automatically created and saved by QMF, using the definitions of the retrieved data. Other users, a data base administrator for example, might originate a new table definition by using an SQL CREATE as follows:

```
CREATE TABLE EMP
      (NAME   CHAR(18),
       ENO    CHAR(3),
       DNO    CHAR(3),
       MGR    CHAR(3),
       SAL    INTEGER,
       OTIME  INTEGER)
```

Query and procedure variables. One may pass values to a query or procedure that replaces variable names appearing in a query or procedure. Typically, a previously created and saved procedure or query is run periodically with different values passed for each run. For example, the following query may have been saved as INSERT3:

EMP	NAME	ENO
I.	&NAME1	&NO1
I.	&NAME2	&NO2
I.	&NAME3	&NO3

The variable names begin with &. Variables may be passed as part of the RUN command, or entered into a QMF prompt panel. For example, the command RUN INSERT3, with no variables, will result in a prompt panel as shown in Figure 11. The user may then key in the parameters and press the ENTER key.

Other QMF functions. A number of QMF topics are not addressed in this overview, including

- HELP panels and prompts to help a user correct errors or to provide tutorial information about user-selected topics
- Views and synonyms
- The user profile options
- The export and import of items between users

Figure 10 Procedure on PROC panel

```
PROC                              MONTHLY                          LINE 1

RUN ED (FORM = EMP_DEPT)
PRINT REPORT
SAVE DATA AS EMPDEPT
RUN DL (FORM = DIV_LOC)
PRINT REPORT
SAVE DATA AS DIVLOC
*** END ***



1=HELP 2=RUN 3=END 4=PRINT 6=QUERY 7=BACKWARD 8=FORWARD 9=FORM
10=INSERT 11=DELETE 12=REPORT
OK, PROC is displayed.
COMMAND ===>                                          SCROLL ===> PAGE
```

Figure 11 Parameters prompt panel

```
                RUN COMMAND PROMPT -- VALUES OF VARIABLES

      Your RUN command runs a query or procedure with variables
      that need values.  Fill in the value for each variable named
      below, after the arrow.  Then press ENTER.  The message at the
      bottom gives the name of the first variable that needs a value.

  &NAME1          ===> _
  &NO1            ===>
  &NAME2          ===>
  &NO2            ===>
  &NAME3          ===>
  &NO3            ===>


1=HELP 3=END, or type HELP or END ===>
Variable &name1 needs a value.
COMMAND ===>                                          SCROLL ===> PAGE
```

Table 2 Simple retrieval result

NAME	DNO	SAL
PETRA DAVIS	46	10750
FRANK CANTRELL	77	9650
CAROL CAMPBELL	46	17500
JANET HILL	12	8000
ANGELO DIVISIERO	51	6800
WILLIAM JAMES	87	5500
RICHARD GOMEZ	51	25500
ANN MARIE SWANSON	46	13500
RANDOLPH DREW	23	11700
JANET GOULD	77	21000
TIMOTHY HAROLDSON	12	12500
STEVE WHITE	87	16500
JAMES VANZANT	46	11000
JOSEPH RUSSELL	87	18000
PAMELA GRIFFITH	23	8500
VERNON COLLINS	73	12400
SHARON ROSTEGE	51	6100

- The use of comments in queries

The reader should refer to Reference 2 for details on these topics.

Classification of QBE and SQL query types

The QMF user must be able to identify every application with a particular query type and then choose either QBE or SQL to create the query.

The task becomes much easier if the user has an understanding of the different classifications of query types and of how either the SQL or QBE language may be used to create a query. The remainder of this paper classifies the different types of queries that may be created using either QBE or SQL, and shows many semantically equivalent QBE and SQL examples. The tables referred to in the examples appear in the Appendix.

Simple retrievals. A *simple retrieval* retrieves data from and references a single table. As an example, here we are to retrieve the names, department numbers, and salaries of those employees in the EMP table:

```
SELECT NAME, DNO, SAL
FROM EMP
```

EMP	NAME	DNO	SAL
P.			

The names of columns in a table from which data is to be retrieved may appear in any order in the query. The information in Table 2 is produced as a result of the preceding retrieval.

The QBE query may also be created using P.'s under the column names as follows:

EMP	NAME	ENO	DNO	MGR	SAL	OTIME
	P.		P.		P.	

If the P. appears under the table name, EMP, as in the first example of a simple retrieval, data elements for all of the named columns are displayed. If the P. appears under column names as given above, only data elements for those columns are displayed.

Typically, a user will DRAW a table, causing all of the column names for the table to appear on the display. The user then simply enters the P.'s under the appropriate columns.

Duplicate rows. Data retrieved from a table may contain duplicate rows. A row is a *duplicate* of another row if the data elements in respective columns of both rows are identical. Using DISTINCT or UNQ. will retrieve data from rows that have no duplicates and from single copies of rows that do have duplicates.

Here we retrieve the division numbers and locations of all departments in the DEPT table, excluding duplicate rows from the report:

```
SELECT DISTINCT DIV,LOC
FROM DEPT
```

DEPT	DIV	LOC
P.UNQ.		

This eliminates the duplicates from the report as follows:

DIV	LOC
510	SJ
404	SA
121	LG
131	LG
302	LG
201	MI
111	SJ
111	LG

Duplicate rows may be retained in the report by omitting the DISTINCT and UNQ. qualifiers as below:

```
SELECT DIV,LOC
FROM DEPT
```

DEPT	DIV	LOC
P.		

This retains the duplicate rows:

DIV	LOC
510	SJ
404	SA
404	SA
510	SJ
121	LG
121	LG
131	LG
302	LG
201	MI
201	MI
201	MI
111	SJ
111	SJ
131	LG
111	LG

Conditional retrieval of data. In the preceding examples, data was selectively retrieved from columns by naming only those columns. One may also limit the data retrieved from the rows of tables. We now retrieve the names, salaries, and department numbers of those employees in Department 46 with salaries greater than $15 000:

```
SELECT NAME, SAL, DNO
FROM EMP
WHERE SAL >15000
AND DNO = 46
```

EMP	NAME	SAL	DNO
P.		>15000	46

Conditions in QBE may also be expressed using a CONDITIONS box obtained by keying COND into the command line and pressing the DRAW PF key. The conditions are then keyed into the CONDITIONS box as in the following example. The names, salaries, and overtime pay of those employees with overtime incomes greater than $1200 or salaries greater than $20 000 are to be retrieved:

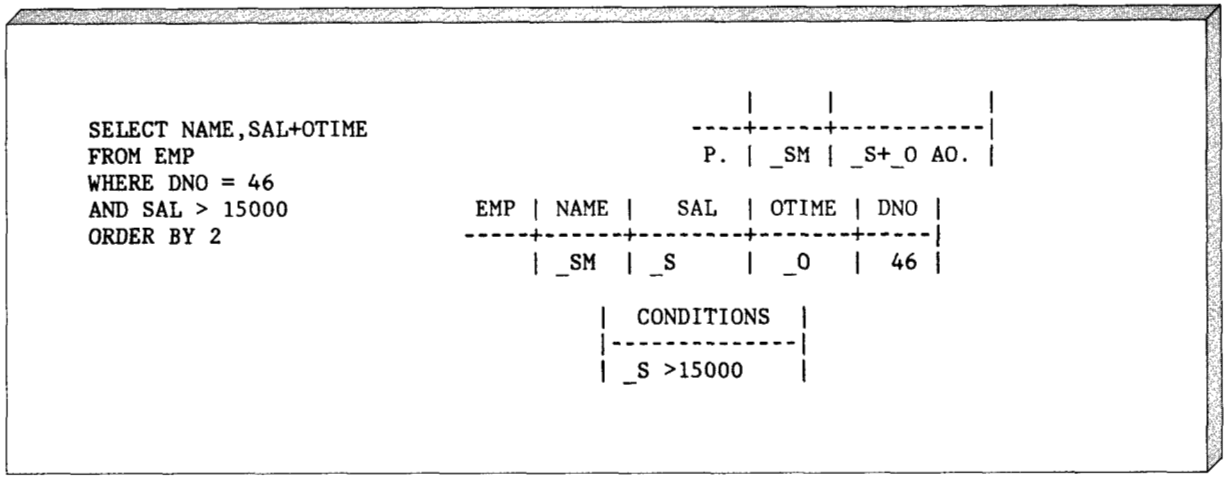
```
SELECT NAME, SAL, OTIME
FROM EMP
WHERE OTIME > 1200
OR SAL > 20000
```

EMP	NAME	SAL	OTIME
P.		_21K	_1250

CONDITIONS
_1250 > 1200 OR _21K > 20000

The names _21K and _1250 in the QBE query represent the data elements in the SAL and OTIME columns. They are also examples of the data elements that they represent. That is, 21K is an example of the selected SAL data elements and 1250 is an example of the selected OTIME elements. These names, which begin with an underscore (_), are called *example element* names and are originated by the creator of the query.

Figure 12 Retrieval using expressions



When an example element name is not part of a condition in a table skeleton, it may be used anywhere else to identify the row and column of the table skeleton in which it appears.

Use of sort in simple retrievals. Retrieved rows of data in a report may be ordered by data in one or more of the columns in the retrieval. Below we first retrieve the names, salaries, and department numbers of those employees with salaries greater than $20 000, then sort the report by department number in descending order and by name in ascending order:

Retrieved rows of data in a report may be ordered by data in one or more of the columns in the retrieval.

```
SELECT NAME, SAL, DNO
FROM EMP
WHERE SAL > 20000
ORDER BY DNO DESC, NAME
```

EMP	NAME	SAL	DNO
P.	AO(2).	>20000	DO(1).

The SQL sort column names in the ORDER BY clause may be followed by either ASC or DESC for ascending and descending order. If neither is used, ASC is assumed. The sort is done by the values in the first column named in the ORDER BY clause and then by the second column named.

The QBE sort columns are indicated by AO for ascending and DO for descending. The sort is done by the column that has the lowest integer value in a sort operator and then by the column that has the next highest value in a sort operator.

Rows may be ordered using only data that will be in the report. Since the employee number ENO is not in the report produced by the preceding retrieval, it may not be used as a sort field.

Using expressions for report data. Thus far, the data elements that have appeared in a report were data elements copied from a table. It is possible to use an arithmetic expression that refers to data elements from a table and to use the value obtained from an evaluation of the expression as a single entry in the report, as shown in Figure 12. In this example, we retrieve the name and the sum of the salary and overtime pay of each employee with a salary greater than $15 000 in Department 46 and order the report by the sum. The SQL ORDER BY clause must use an index to the select list when the sort is done on an expression.

QBE requires a skeleton that represents the report when any data element in the report is not derived from a single data element in a table. In this example, _SM, _S, and _O are used in the report skeleton to refer to the NAME, SAL, and OTIME columns of the EMP table.

Dependent retrievals. In simple retrievals, data retrieved from a row may depend upon conditions that refer to data in the same row. In a *dependent retrieval*, data retrieved from a row will depend upon data in other rows, usually in rows of other tables. In the following example, we retrieve the names, salaries, and department numbers of those employees whose departments are located in San Jose (SJ):

```
SELECT NAME,SAL,DNO
FROM EMP
WHERE DNO = ANY
  (SELECT DNUM
  FROM DEPT
  WHERE LOC = 'SJ')
```

EMP	NAME	SAL	DNO
P.			_D

DEPT	DNUM	LOC
	_D	SJ

In this case, the user recognizes that the data to be retrieved is in one table, just as in a simple retrieval. However, a simple retrieval will not do because the condition for retrieval is dependent upon data in other rows. In this case, the other rows are in the DEPT table.

The dependent query form of QBE is characterized by the link from the EMP row to the DEPT row forged by using the example element name, _D, in both rows. Using the same name indicates a link on the same values in both tables. That is, data is retrieved

from each row in EMP for which there is a row in DEPT with the same department number and a location of San Jose.

The dependent query form of SQL is characterized by the use of a subquery. The subquery is part of the condition expressed in the SQL WHERE clause. In this example, the subquery identifies a subset of rows from the DEPT table that indicate locations in San Jose.

The dependent query form of SQL is characterized by the use of a subquery.

Dependent retrievals using rows in the same table. Retrieval from the rows of a table can be dependent upon data in different rows of the same table. For example, here we retrieve the names, salaries, manager numbers, and employee numbers of those employees having salaries larger than those of their managers:

```
SELECT NAME,SAL,MGR,ENO
FROM EMP EMP1
WHERE MGR = ANY
  (SELECT ENO
  FROM EMP EMP2
  WHERE EMP1.SAL > EMP2.SAL)
```

EMP	NAME	SAL	MGR	ENO
P.		>_MS _MS	_M	 _M

In this case, the user recognizes that the data to be retrieved is in the same table and that the data on the manager's salary is in different rows of the table.

The SQL user must provide alternate names for EMP (EMP1 and EMP2) so that the distinction can be made between the employee's salary and the manager's salary. The distinction in QBE is clear from the location of the example element names in different rows.

Join retrievals. In simple and dependent retrievals, data is retrieved from only one table. One may retrieve data from more than one table by conceptually joining the tables to form a single table. Such a query is called *join retrieval.* Figure 13 depicts this type of query, in which the names, department names, and locations of those employees in Department 77 and Division 404 are retrieved.

The user must recognize that data to be retrieved is in different tables and that a join is needed to get data from both tables into the report.

In QBE, a join is implied when example element names that reference data elements in both EMP and DEPT appear in the same row of a report skeleton.

The SQL query implies a join of EMP and DEPT because both are named in the FROM clause. It is possible to force a join using SQL, even when a join table is not referenced in the SELECT and WHERE clauses, by simply naming the table in the FROM clause. Of course, one should not create such a query because the join is not necessary.

Union retrievals. Data elements from the columns of different tables may be retrieved into the same columns of a report by using a *union retrieval.* Figure 14 illustrates this type of retrieval, in which we retrieve the names and salaries of those employees in Department 46 and in Division 405, and order the report by salary. In this type of application, the user must recognize that the data to be retrieved into the same columns of the report is in different tables.

In essence, a *union* contains separate queries, the results of which are united to form a single report. A union, by definition, does not contain duplicates. That is, all duplicates in a report are removed as the result of a union, and the use of DISTINCT or UNQ. in the query is meaningless.

Equivalent join and dependent queries in SQL. In SQL, it is possible to create a join retrieval that performs the same function as a dependent retrieval. That is, an application requires that data be retrieved from each row of a single table, and retrieval is dependent upon data in other rows of the same or a different table. Rather than use a dependent retrieval, the table that would have appeared in a subquery is joined with the table from which the data is retrieved. If duplicates are not retained, either a join or dependent SQL retrieval will produce the same result. For example, in Figure 15 we retrieve the names,

salaries, and department numbers of those employees located in San Jose.

For performance reasons, when either an SQL join or an SQL-dependent query give the same result, it is best to choose the SQL join form. Care must be taken, however, when duplicates *are* retained, as will be discussed below.

Note that the equivalent QBE syntax is the same for either SQL query in the example of Figure 15. It is not necessary for the user to try to create an equivalent join form using QBE because the QBE-dependent form is optimized for best performance.

When duplicate rows are retained, using an SQL join query instead of an SQL dependent query may *not* produce the same results. For example, the following dependent query retrieves duplicate division numbers and division names from HQLOC of those divisions located in Scranton that have a department in Los Gatos (LG):

```
SELECT DV#,DIV
FROM HQLOC
WHERE LOCN = 'SCRANTON'
AND DV# = ANY
  (SELECT DIV
  FROM DEPT
  WHERE LOC = 'LG')
```

HQLOC	DV#	DIV	LOCN
	P._D	P.	SCRANTON

DEPT	DIV	LOC
	_D	LG

The following table is produced:

DV#	DIV
121	MANUFACTURING
302	PERSONNEL
121	MANUFACTURING
121	MANUFACTURING
302	PERSONNEL

The following query, in which duplicates are selected using a join, is *not* the equivalent of the preceding

Figure 13 Join retrieval

```
SELECT NAME,DNAME,LOC
FROM EMP, DEPT
WHERE DNO = 77
AND DIV = 404
AND DNO = DNUM
ORDER BY NAME
```

P.	_E AO.	_D	_L

EMP	NAME	DNO
_E		_M

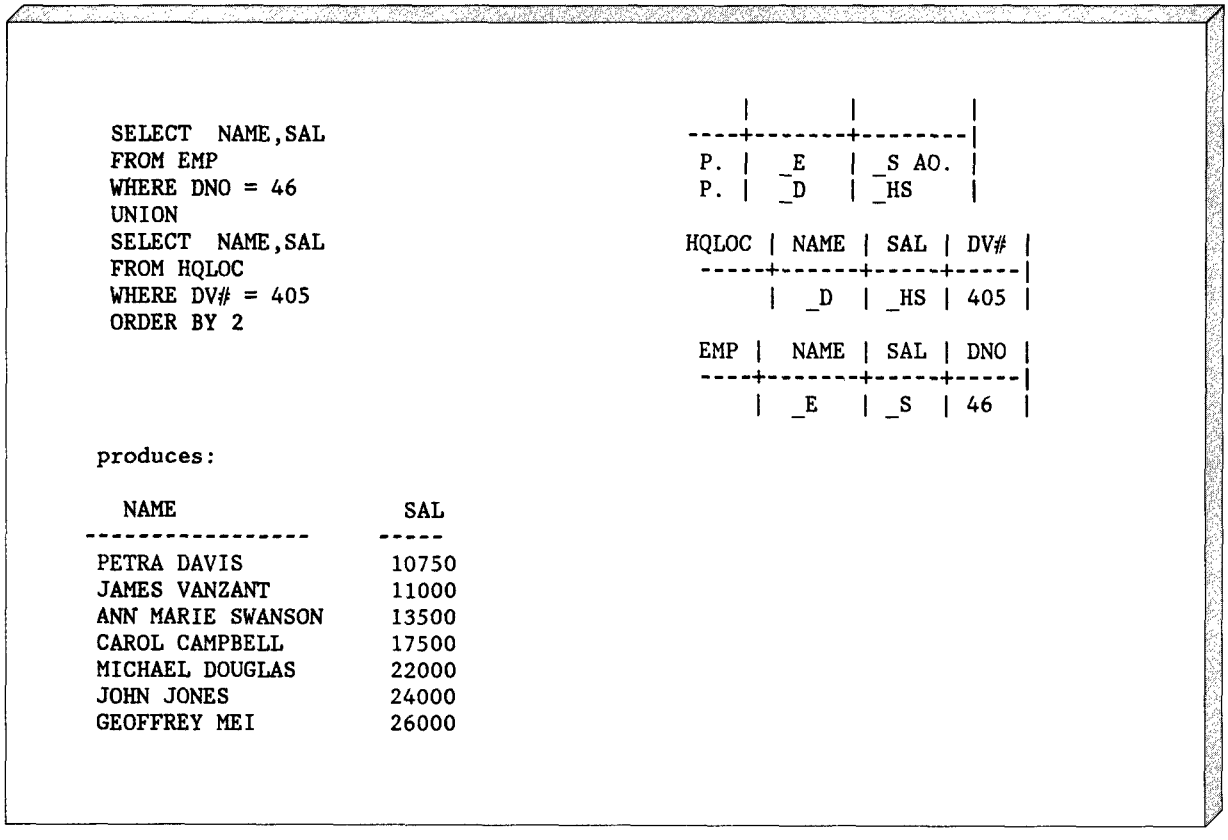
DEPT	DNUM	DIV	LOC	DNAME
	_M	404	_L	_D

CONDITIONS
_M = 77

produces the following:

NAME	DNAME	LOC
FRANK CANTRELL	ACCOUNTING	SA
JANET GOULD	ACCOUNTING	SA

Figure 14 Union retrieval



dependent query. That is,

```
SELECT DV#,X.DIV
FROM HQLOC X DEPT Y
WHERE LOCN = 'SCRANTON'
AND DV# = Y.DIV
AND LOC = 'LG'
```

P.	_D	_DV

HQLOC	DV#	DIV	LOCN
	_D	_DV	SCRANTON

DEPT	DIV	LOC
	_D	LG

produces the following result:

DIV	DNO
121	MANUFACTURING
121	MANUFACTURING
302	PERSONNEL
121	MANUFACTURING
121	MANUFACTURING
121	MANUFACTURING
121	MANUFACTURING
302	PERSONNEL

This report is a result of the join and the retention of duplicates produced from the join. If the DISTINCT qualifier is used to eliminate duplicates, all duplicates are excluded. That is,

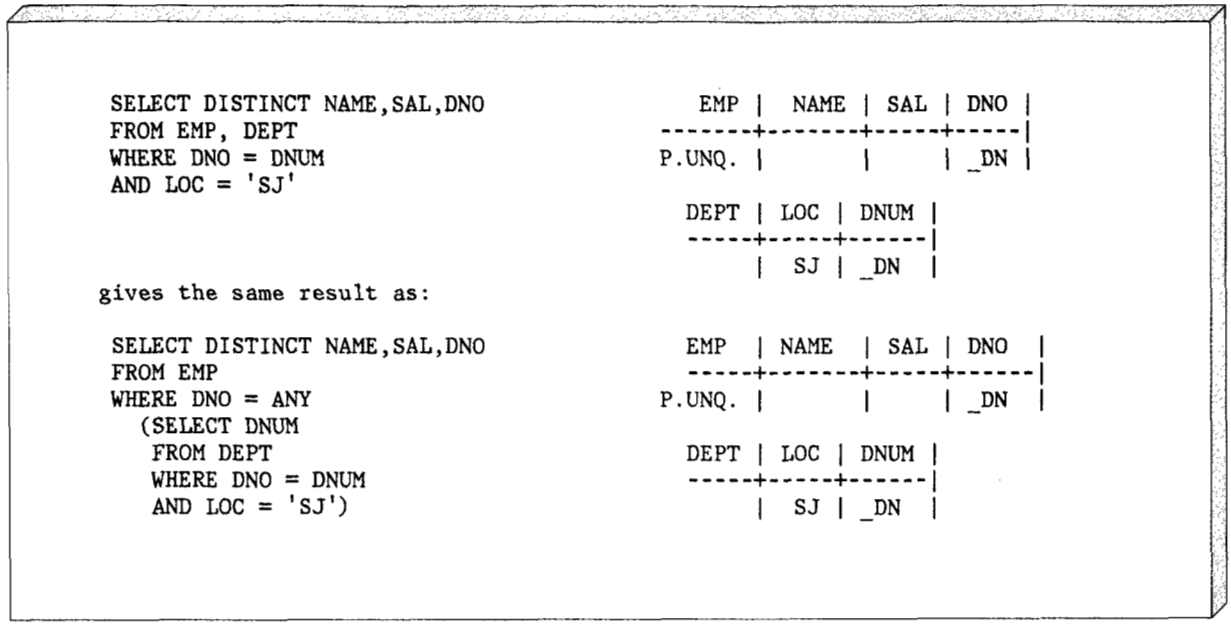
```
SELECT DISTINCT DV#,X.DIV
FROM HQLOC X DEPT Y
WHERE LOCN = 'SCRANTON'
AND DV# = Y.DIV
AND LOC = 'LG'
```

P.UNQ.	_D	_DV

HQLOC	DV#	DIV	LOCN
	_D	_DV	SCRANTON

DEPT	DIV	LOC
	_D	LG

Figure 15 Join equivalent of a dependent query



eliminates all duplicates:

DIV	DNO
121	MANUFACTURING
302	PERSONNEL

The result is still not the same as for the dependent query retrieving duplicates. Generally speaking, one should use a dependent retrieval instead of a join if duplicates are to be retained, unless one is familiar enough with the data to know that no unwanted duplicates will be formed as the result of a join. For example, the retrieval in Figure 15 includes the names of employees from the EMP table. Since there are no duplicate names in the EMP table, there will be no duplicates in the report produced from the query. Therefore, the DISTINCT and UNQ. qualifiers may be omitted, and the join form of the SQL retrieval will give the same report as the SQL-dependent form.

Grouping in retrieval queries. Thus far, data has been addressed in terms of the rows and columns that contain data elements. QMF also provides a means of partitioning tables into groups of rows and of referencing such groups using built-in functions. One or more columns may be named upon which grouping is to take place, and all rows in a table that have the same values in the grouping columns constitute a specific group or partition of the table. Grouping is indicated by use of the SQL GROUP BY clause or the QBE G. operator. Following is an example in which we retrieve the department number and average salary for each department:

```
SELECT DNO, AVG (SAL)
FROM EMP
GROUP BY DNO
```

P.	_D	AVG._S
EMP	SAL	DNO
	_S	_D G.

When grouping is used in a QMF retrieval, only group data may be retrieved. That is, the SELECT clause in SQL may contain only built-in functions and columns named in the GROUP BY clause; the report skeleton in QBE may contain only built-in functions and example elements for which there is a G. in a source table.

Note that a report skeleton is required in the QBE query because other than data elements from rows of a table are to appear in the report.

Conditional retrieval using built-in functions. In all our previous examples, conditions referred only to data elements. A QMF query that has grouping may also have conditions that contain built-in functions. In such instances, the grouping must be done before the value for a built-in function can be determined. Conditions that reference only data elements are evaluated first to retrieve rows that meet retrieval criteria. Retrieved rows are then grouped, and the conditions that apply to groups are evaluated last. SQL uses the WHERE and HAVING clauses to distinguish row from group conditions. In Figure 16, the department numbers and the sums of the salaries from Departments 46, 51, and 79 are to be retrieved if the average salary in the department is over $12 000. Only the salaries of those employees making more than $10 000 are to be used.

Grouping with more than one column. When more than one grouping column is specified in the same GROUP BY clause in SQL or in the same query row in QBE, data elements used for grouping from each row are treated as a single combined value. In the example below, we retrieve the department number, manager, and average salary for each department and manager:

```
SELECT DNO,MGR,AVG(SAL)
FROM EMP
GROUP BY DNO,MGR
```

P.	_D	_M	AVG._S
EMP	MGR	SAL	DNO
	G._M	_S	_D G.

Each group is formed on a combined MGR and DNO value. That is, all rows that have the same MGR and DNO values are members of the same group.

Grouping in a join retrieval. Grouping may be specified in a join retrieval. There is no significant difference in the way grouping is done for a join retrieval except that the grouping is done on the joined data. That is, the join is done on retrieved rows before the grouping is done. Figure 17 shows an example in which we are to retrieve the department numbers, locations, and sums of the salaries of employees in Division 404, grouping by department number and location.

In this example, EMP and DEPT are joined, and the DNO and LOC grouping columns are part of the join. That is, grouping takes place after the join and according to the combined DNO and LOC columns.

Grouping in a dependent retrieval. In dependent retrievals, grouping columns may reference the table from which data is retrieved or the table upon which

Figure 16 Conditional selection of groups

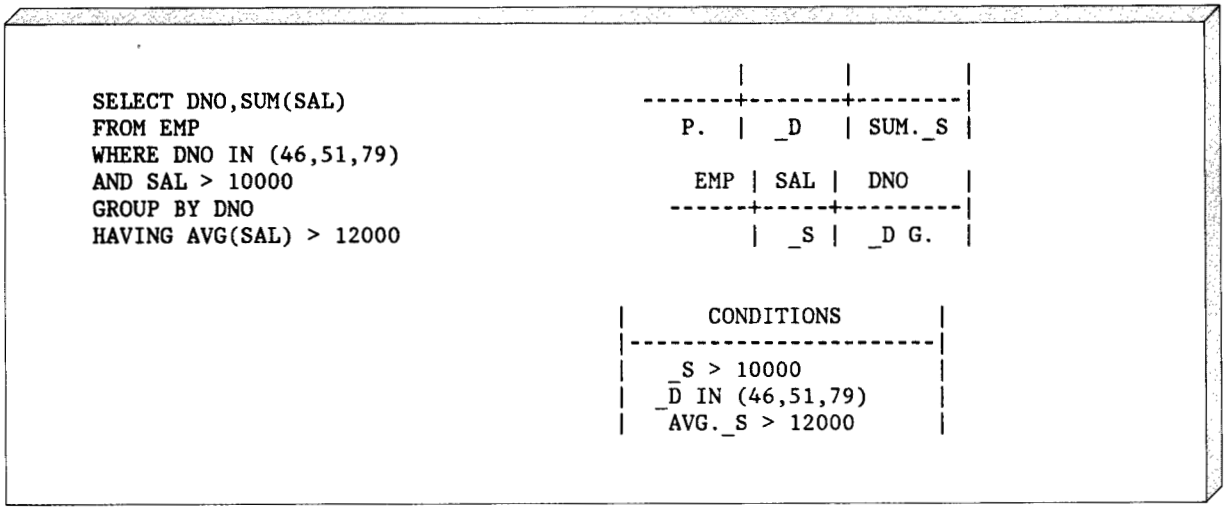
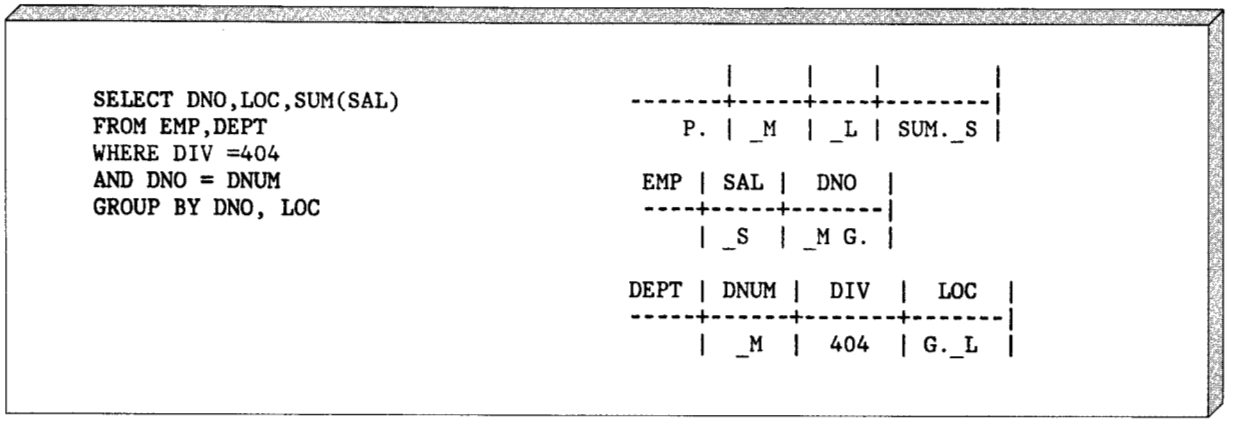


Figure 17 Join retrieval using grouping



retrieval is dependent. Here we retrieve the names and salaries of those whose salaries are greater than the average salary of any department:

```
SELECT NAME, SAL
FROM EMP
WHERE SAL > ANY
  (SELECT AVG(SAL)
  FROM EMP
  GROUP BY DNO)
```

EMP	NAME	SAL
P.	_E	>AVG._S

EMP	SAL	DNO
	_S	G.

Conditions that contain group and row references. A condition that contains both group and row reference to the same rows of a table is ambiguous and will be rejected by QMF. The following three queries are parts of an example of mixed group and row references. The first is an invalid query:

```
SELECT NAME,SAL,DNO,OTIME
FROM EMP
WHERE OTIME > 500
GROUP BY DNO
HAVING SAL > AVG(SAL)
```

EMP	NAME	SAL	DNO	OTIME
P.	_N	_S	_D	>500

CONDITIONS
_S1 > AVG. _S2

This query could be interpreted to mean "retrieve data from the EMP table for those whose salaries are greater than the average salary of employees who make more than $500 in overtime":

```
SELECT NAME,SAL,DNO,OTIME
FROM EMP
WHERE SAL > ANY
  (SELECT AVG(SAL)
  FROM EMP
  WHERE OTIME > 500
  GROUP BY DNO)
```

EMP	NAME	SAL	DNO	OTIME
P.		_S1 _S2	G.	>500

CONDITIONS
_S1 > AVG. _S2

or it could mean "retrieve data from the EMP table for those who make more than $500 in overtime and whose salaries are greater than the average salary of any department":

```
SELECT NAME,SAL,DNO,OTIME
FROM EMP
WHERE OTIME >500
AND SAL > ANY
  (SELECT AVG(SAL)
  FROM EMP
  GROUP BY DNO)
```

EMP	NAME	SAL	DNO	OTIME
P.		_S1 _S2	G.	>500

CONDITIONS
_S1 > AVG. _S2

Figure 18 Join within a dependent retrieval

```
SELECT NAME,SAL,DNO
FROM EMP
WHERE DNO = ANY
  (SELECT DNUM
   FROM DEPT   , HQLOC
   WHERE   DIV =   DV#
   AND LOCN = 'SCRANTON')
```

EMP	NAME	SAL	DNO
P.	_E		_D

DEPT	DNUM	DIV
	_D	_E

HQLOC	DV#	LOCN
	_E	SCRANTON

Either of the latter two queries is valid in QMF, but the first of these three queries is invalid because of the ambiguity.

Mixed join and dependent retrievals. Previously, joins and dependent queries have been discussed as separate query types. It is possible to mix joins and dependent queries in the same QMF query. An example is shown in Figure 18, in which we retrieve the names, salaries, and department numbers of those employees whose departments are in a division located in Scranton.

Figure 19 is an example of the mixed join and dependent retrieval whereby we retrieve the names, salaries, and locations of those employees in departments not in a division located in Blakely.

For performance reasons, it is always better to use joins at lower levels in an SQL-dependent query than it is to use multiple levels of dependence. The SQL query in Figure 20 uses multiple levels of dependence (a subquery within a subquery). In this query, we retrieve the names, salaries, and department numbers of those employees whose departments are located in San Jose and whose division is located in Scranton. Figure 21 shows an equivalent query using a join in the SQL subquery. The result is the same no matter which SQL form is used, but performance may be better for the latter SQL form. Note that either SQL form has the same equivalent QBE form.

Multiple levels of dependence. There are three cases where it is sometimes necessary to use multiple levels of dependence in SQL:

1. When creating more than one group
2. When ALL is used with a comparison operator
3. When comparing group and row data

Release 1 of QMF does not provide QBE support for these cases. That is, Release 1 of QMF does not provide QBE support for ALL in a comparison operator, and the QBE equivalent form to those SQL queries that require multiple levels of dependence is not supported in Release 1 of QMF.

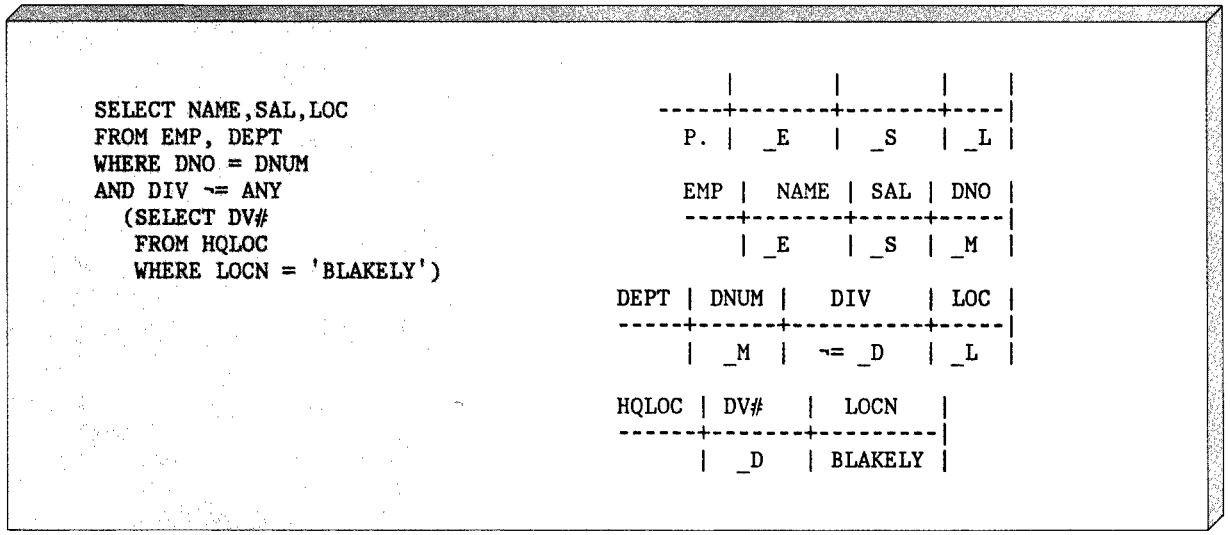
Figure 22 is an example of the first case. Here we retrieve the names, salaries, and department numbers of those employees in any department having an average salary greater than the average salary of any division. Although the equivalent QBE form exists, as shown in this example, it is not supported in Release 1 of QMF.

Performance considerations using retrieval queries. There are at least three situations where performance is a consideration when creating QMF queries:

1. Ordering of retrieved data
2. Elimination of duplicate rows
3. Using an equivalent join in place of a subquery

Ordering of data has an obvious impact upon performance because it means data must be sorted before it is displayed. Not so obvious, however, is the fact that elimination of duplicates causes a sort to be done. That is, data is sorted so that duplicate rows can be located and eliminated from a report. Generally speaking, then, elimination of duplicates

Figure 19 Mixed join and dependent retrieval



```
SELECT NAME,SAL,LOC
FROM EMP, DEPT
WHERE DNO = DNUM
AND DIV ¬= ANY
  (SELECT DV#
   FROM HQLOC
   WHERE LOCN = 'BLAKELY')
```

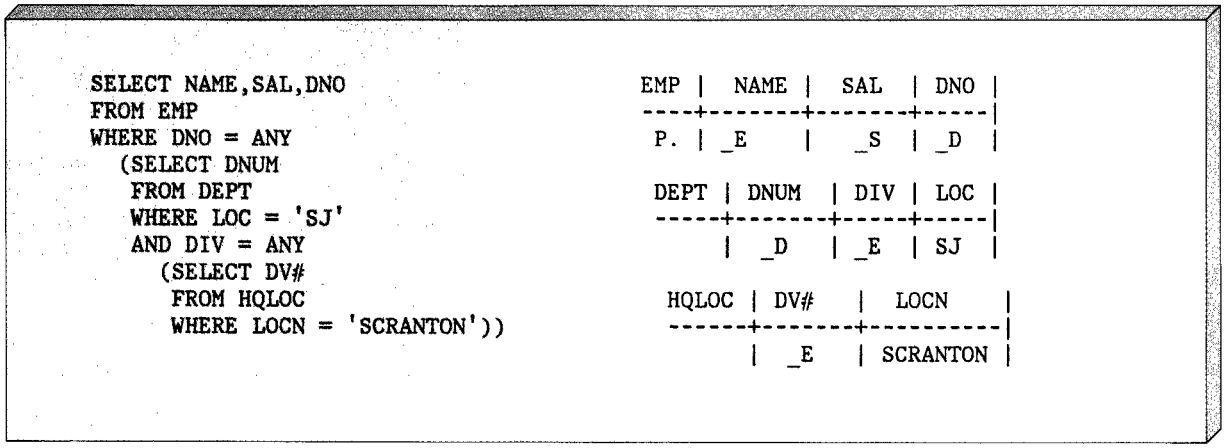
P.	_E	_S	_L

EMP	NAME	SAL	DNO
	_E	_S	_M

DEPT	DNUM	DIV	LOC
	_M	¬= _D	_L

HQLOC	DV#	LOCN
	_D	BLAKELY

Figure 20 Joins at levels of dependence



```
SELECT NAME,SAL,DNO
FROM EMP
WHERE DNO = ANY
  (SELECT DNUM
   FROM DEPT
   WHERE LOC = 'SJ'
   AND DIV = ANY
     (SELECT DV#
      FROM HQLOC
      WHERE LOCN = 'SCRANTON'))
```

EMP	NAME	SAL	DNO
P.	_E	_S	_D

DEPT	DNUM	DIV	LOC
	_D	_E	SJ

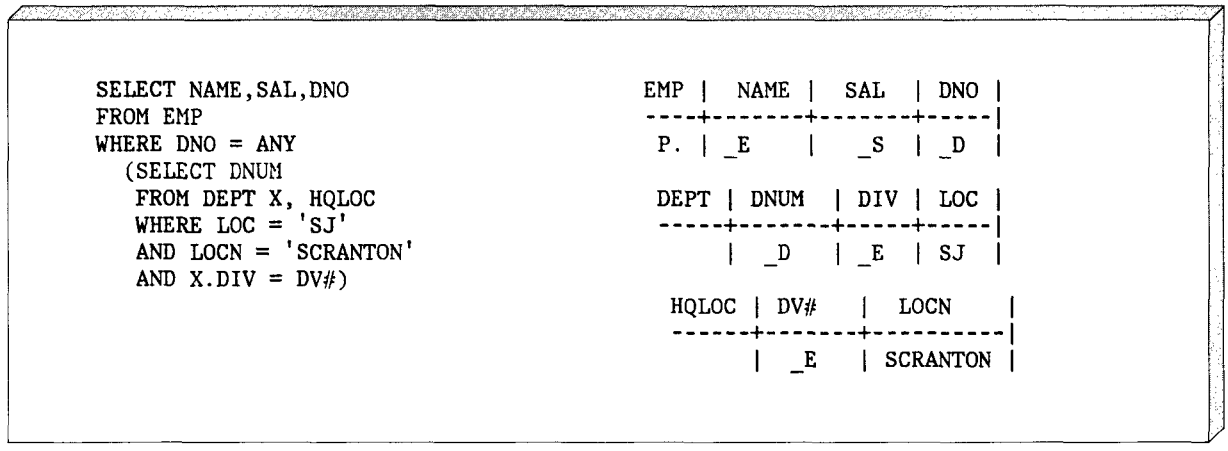
HQLOC	DV#	LOCN
	_E	SCRANTON

and the specification of ordering should be omitted unless necessary.

As discussed previously, performance may be better if a join is used instead of an equivalent dependent query. If, however, duplicates are retained in a join, unwanted duplicates may appear as the result of the join. Under those circumstances, if performance is a consideration, one has the choice of composing a dependent query with the duplicates retained, or of composing a join query with duplicates eliminated. Knowing something about the data has a bearing on the choice to be made. That is, if one knows from the queried data that there will be no unwanted duplicates, selection of duplicates in a join should be used instead of a dependent query when performance is a consideration.

Insert, update, and delete queries. QMF also includes the ability to insert, update, and delete data using either QBE or SQL. These types of queries are referred to collectively as maintenance queries.

Figure 21 Join replacing a level of dependence



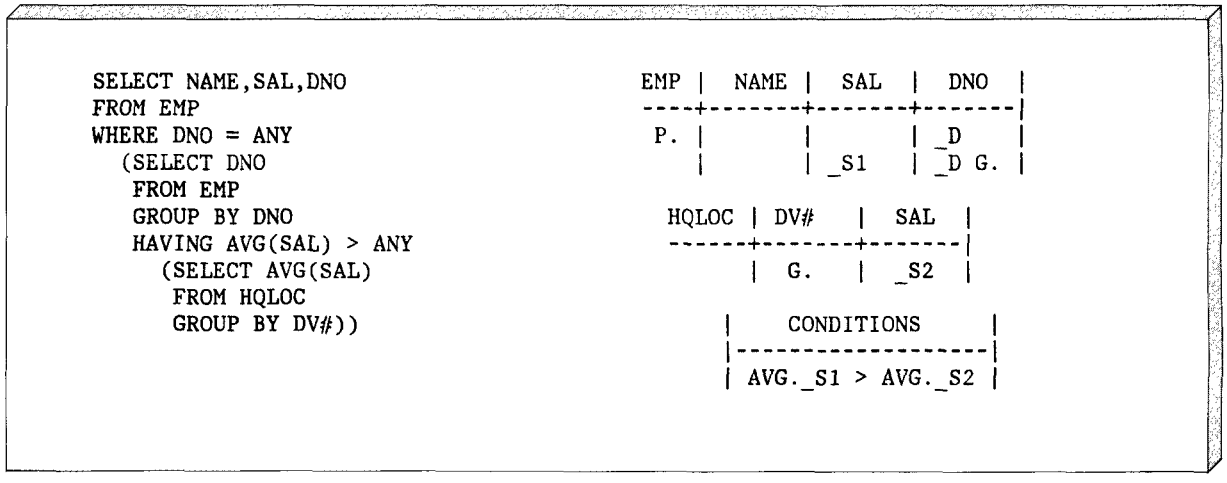
```
SELECT NAME,SAL,DNO
FROM EMP
WHERE DNO = ANY
  (SELECT DNUM
   FROM DEPT X, HQLOC
   WHERE LOC = 'SJ'
   AND LOCN = 'SCRANTON'
   AND X.DIV = DV#)
```

```
EMP |  NAME |  SAL  | DNO |
----+-------+-------+-----|
 P. | _E    |   _S  | _D  |

DEPT | DNUM  | DIV | LOC |
-----+-------+-----+-----|
     |  _D   | _E  | SJ  |

 HQLOC | DV#   |  LOCN     |
 ------+-------+-----------|
       |  _E   | SCRANTON  |
```

Figure 22 Separate groups in a multidependent query



```
SELECT NAME,SAL,DNO
FROM EMP
WHERE DNO = ANY
  (SELECT DNO
   FROM EMP
   GROUP BY DNO
   HAVING AVG(SAL) > ANY
     (SELECT AVG(SAL)
      FROM HQLOC
      GROUP BY DV#))
```

```
EMP |  NAME |  SAL  |  DNO  |
----+-------+-------+-------|
 P. |       |       | _D    |
    |       | _S1   | _D G. |

 HQLOC | DV#   |  SAL  |
 ------+-------+-------|
       |  G.   |  _S2  |

      |    CONDITIONS     |
      |-------------------|
      | AVG._S1 > AVG._S2 |
```

In QMF, one may not insert, update, or delete data into more than one table at a time; therefore, unions and joins are not relevant to maintenance queries. Also, the specification of a sort applies only to data as it appears in a report. Since maintenance queries do not produce a report, sorts are not valid in such queries.

Simple inserts. A simple insert is simpler than its retrieval counterpart because no conditions may appear in a simple insert. Only the constant values to be added as part of the newly inserted row are given. If no value is given for any column, a null is indicated for that column in the newly inserted row.

In the following example, if we insert the name and employee number for James Jones into the employee table, the newly inserted row will have nulls for the columns other than NAME and ENO. A null is not the same as a zero or a blank. A null means the absence of a value, and if a null is referenced in an expression, the expression produces a null. Also, if a column with nulls is specified as a grouping column, *each null* is treated as a separate group:

Figure 23 Dependent delete query

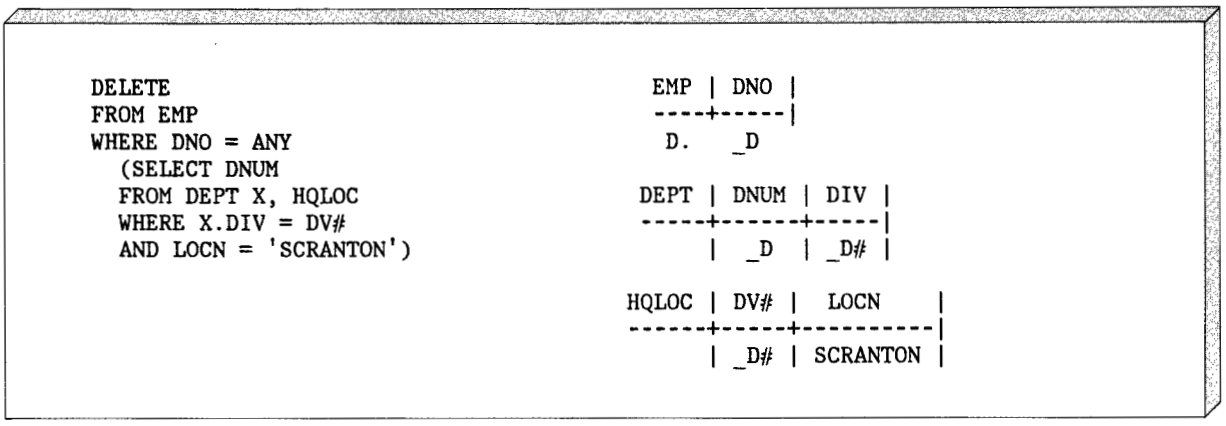
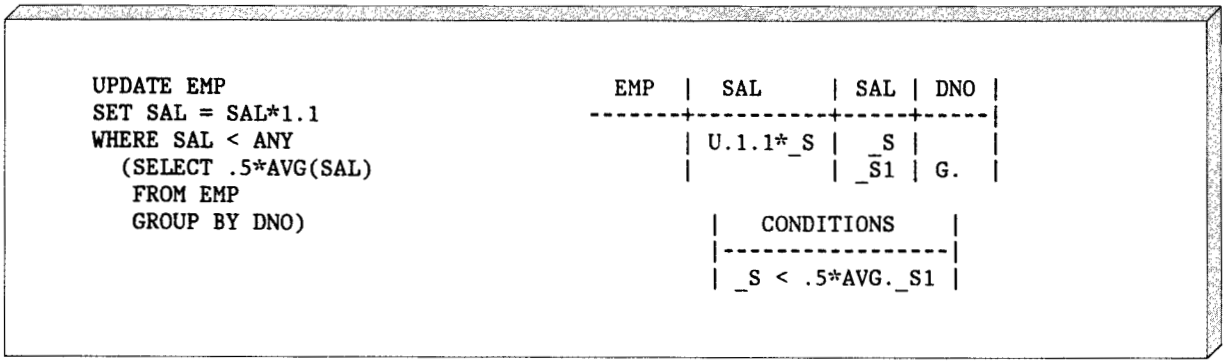


Figure 24 Dependent update query




```
INSERT INTO EMP (NAME,ENO)
VALUES (JAMES JONES', 579)
```

EMP	NAME	ENO
I.	'JAMES JONES'	579

Simple updates. Data elements within existing rows of a table may be updated to a new value (or to a value that replaces a null). As an example, we update employee number 579 to contain the new employee number 979. Also, we update the department number to 77 and the salary to $7200:

```
UPDATE EMP
SET DNO = 77,
  ENO = 979,
  SAL = 7200
WHERE ENO = 579
```

EMP	DNO	ENO	ENO	SAL
	U.77	579	U.979	U.7200

Note that the ENO column is repeated in the QBE query. Even though a column may be repeated in a query, reference is to the single column with the same name in the actual table.

Simple deletes. Simple deletes are similar in form to simple retrievals, except that rather than qualifying a row for display, the row is qualified for deletion. In the example below, we delete James Jones in department number 77 from the employee table:

```
DELETE
FROM EMP
WHERE DNO = 77
AND NAME = 'JAMES JONES'
```

EMP	NAME	DNO
D.	'JAMES JONES'	77

Dependent insert, update, and delete queries. Dependent insert, update, and delete queries are very

similar in form to dependent retrievals. In Figure 23, all employees in divisions located in Scranton are deleted.

Grouping with insert, update, and delete queries. Inserts, updates, and deletes are made on a row-by-row basis. That is, a group of rows cannot be specified for insert, update, or delete. However, grouping may occur in a dependent query. Figure 24 shows a case where each employee's salary that is less than half the average salary of any department is increased by ten percent.

Concluding remarks

One of the unique features of QMF is the dual syntax provided as part of the query facility. Users can accommodate their personal needs and preferences by choosing either SQL or QBE to compose a particular query. The result of a query is completely independent of the syntax used to formulate the query. Data from a retrieval, for instance, may be formulated into different reports, saved, modified, and otherwise manipulated in the same manner, no matter which syntax was used to retrieve the data.

Every QBE query in QMF is translated into an equivalent SQL form. That is, every query submitted to the relational data base manager by QMF is either an SQL query composed by the user or an SQL query translated from QBE to SQL.

There are a number of benefits derived from translating QBE to an equivalent SQL form:

- SQL queries are parsed by the relational data base manager. If QBE queries were not translated to SQL by QMF, a separate parser would have to be built into every relational data base manager supported by QMF for both QBE and SQL.
- SQL provides the standard set of query functions supported by QMF. To guarantee that QBE does not diverge from the standard, QBE (and any other language syntax that may be supported by QMF) must be translatable to SQL. (Since QBE is dependent upon SQL for the function that it may provide, the QBE in QMF is different from that originated and defined by Zloof.)
- The parsing of a QBE query can easily include translation to a "flattened" or linear form that can be made available to the user as an alternative to the graphic form. In QMF, it would be pointless to provide other than SQL as that linear form.

Acknowledgment

My thanks go to Pat Wilson and Steve Uhlir for their assistance in preparing this paper.

Appendix: Sample tables

The tables referenced in the examples in this paper are on the following page.

Cited references

1. *Query Management Facility, General Information Manual,* GC26-4071, IBM Corporation; available through IBM branch offices.
2. *Query Management Facility, User's Guide and Reference Manual,* SC26-4100, IBM Corporation; available through IBM branch offices.
3. *IBM Database 2 General Information,* GC26-4073, IBM Corporation; available through IBM branch offices.
4. *SQL/Data System General Information,* GH24-5012, IBM Corporation; available through IBM branch offices.
5. G. Sandberg, "A primer on relational data base concepts," *IBM Systems Journal* **20,** No. 1, 23–40 (1981).
6. C. J. Date, *An Introduction to Database Systems,* Addison-Wesley Publishing Company, Reading, MA (1977).
7. M. M. Zloof, "Query-by-Example," *AFIPS Conference Proceedings, National Computer Conference* **44,** 431–438 (1975).
8. M. M. Zloof, "Query-by-Example: A data base language," *IBM Systems Journal* **16,** No. 4, 324–343 (1977).
9. M. M. Zloof, "Query-by-Example: The invocation and definition of tables and forms." *Proceedings of the 1st International Conference on Very Large Data Bases* (September 1975).
10. *Query-by-Example Terminal User's Guide,* SH20-2078, IBM Corporation; available through IBM branch offices.
11. *IBM Database 2, Introduction to SQL,* GC26-4082, IBM Corporation; available through IBM branch offices.
12. D. D. Chamberlin et al., "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20,** No. 6, 560–575 (1976).
13. M. W. Blasgen et al., "System R: An architectural overview," *IBM Systems Journal* **20,** No. 1, 41–62 (1981).

Reprint Order No. G321-5214.

Joseph J. Sordi *IBM General Products Division, Santa Teresa Laboratory, P.O. Box 50020, San Jose, California 95150.* Mr. Sordi is a Senior Planner in the Advanced Language Products department at the Santa Teresa Laboratory. He joined IBM in 1966 at the IBM Gaithersburg facility, where he participated in the design and implementation of the Generalized Information System (GIS) and the early design of the Interactive Query Facility (IQF) language. He was the manager of IQF development and then of Advanced Text Management System (ATMS) development before joining his present department, where he did the design and implementation of the Query-by-Example (QBE) support in the Query Management Facility (QMF). Mr. Sordi has received an IBM Invention Achievement Award as coinventor of the algorithm used to translate QBE to the Structured Query Language (SQL) in QMF. He continues to work in the area of query languages at Santa Teresa. He received a B.S. in mathematics and philosophy from the University of Scranton in 1960.

EMP Table

NAME	ENO	DNO	MGR	SAL	OTIME
Petra Davis	775	46	299	10750	500
Frank Cantrell	529	77	182	9650	800
Carol Campbell	299	46	182	17500	—
Janet Hill	349	12	182	8000	1200
Angelo Divisiero	284	51	299	6800	1500
William James	209	87	238	5500	250
Richard Gomez	182	51	014	25500	—
Ann Marie Swanson	125	46	182	13500	1340
Randolph Drew	362	23	299	11700	1100
Janet Gould	238	77	182	21000	—
Timothy Haroldson	579	12	299	12500	—
Steve White	382	87	238	16500	—
James Vanzant	234	46	182	11000	780
Joseph Russell	787	87	299	18000	2300
Pamela Griffith	361	23	238	8500	1280
Vernon Collins	948	73	299	12400	650
Sharon Rostege	271	51	238	6100	40

DEPT Table

DNAME	DNUM	DIV	LOC
Sales	23	510	SJ
Payroll	46	404	SA
Accounting	77	404	SA
Publicity	18	510	SJ
Development	49	121	LG
Applications	33	121	LG
Systems	73	131	LG
Recruiting	44	302	LG
Communications	21	201	MI
Media	37	201	MI
Publications	12	201	MI
Printing	87	111	SJ
Classified	72	111	SJ
Systems Analysis	55	131	LG
Programming	51	111	LG

HQLOC Table

NAME	ENO	DIV	DV#	SAL	LOCN
Kenneth Jamison	014	Manufacturing	121	60000	Scranton
Harold Smith	888	Publicity	201	15750	New York
Peter Osgood	385	Personnel	302	16500	Scranton
Solomon Esterman	112	Accounting	404	42000	Blakely
Richard Meinhardt	912	Manufacturing	131	24000	Blakely
James Hershey	321	Marketing	510	23000	New York
Merian Stiller	989	Manufacturing	111	18000	Blakely
Ellen Goodman	111	Publicity	201	36000	New York
Salley Peterson	195	Marketing	510	14000	New York
Michael Douglas	141	Accounting	405	22000	Harrison
Michael Buckley	629	Manufacturing	121	19500	Scranton
Robert Hershfeld	438	Manufacturing	121	17500	Scranton
Peter Langan	666	Marketing	510	22000	New York
Daniel Magana	235	Manufacturing	111	48000	Blakely
Hunan Choy	181	Personnel	302	33000	Scranton
Evelyn Gallagher	004	Publicity	201	45000	New York
Geoffrey Mei	657	Accounting	405	26000	Harrison
John Jones	040	Accounting	405	24000	Harrison