R E L A T E^{TM}/3000

RELATIONAL DATA BASE MANAGEMENT SYSTEM

R E F E R E N C E M A N U A L

COMPUTER RESOURCES INCORPORATED
5333 Betsy Ross Drive
P.O. Box 58004
Santa Clara CA 95052
(408) 980-9898

RM-1007

NOTICE

TABLE OF CONTENTS

INTRODUCTION

This manual introduces RELATE(tm)/3000, CRI's Relational Database Management System, and provides information on how to use it. The RELATE/3000 system is designed to manipulate disk files on Hewlett-Packard's HP3000 series of computers under the Multi-Programming Executive (MPE) operating system.

RELATE/3000 is built on the premise that any user, whether technically experienced or beginning, should have the ability to manipulate computerized data files. RELATE/3000 consists of both interactive terminal commands and interface routines callable from BASIC, FORTRAN, SPL, and COBOL.

Relate commands are easy to understand and remember because all of the commands have an English-like structure.

This manual is divided into eight major sections:

1. **Definition of Terms**— This section explains terms that are crucial to understanding RELATE/3000 commands. This section also includes syntax conventions and a glossary.

2. **Commands Reference**— This section contains all commands available in RELATE/3000, and their options. Complete functional descriptions as well as examples are given for each command.

3. **Programmatic Interface**— The Programmatic Interface section lists the subroutines callable from BASIC FORTRAN, SPL, and COBOL, and describes how to use them.

4. **File System Descriptions**— This section deals with IMAGE/3000, KSAM, and MPE files. Restrictions on their usage with RELATE/3000 and general overviews on their manipulation are listed.

5. **Transaction Processing**— This section defines a transaction and describes the deferred update mechanism.

6. **Updating Views**— A discussion of the requirements which must be met to allow views to be updated.

7. **Security**— This section describes how the SECURITY commands operate and what restrictions are placed on them.

8. **RELATE/3000 Internals**— The RELATE/3000 Internals section describes what methods RELATE uses to solve various problems as well as how the software functions in general.

GETTING STARTED

Log on to the system with the **HELLO** command. When a colon (":") is displayed, enter **RELATE.** RELATE/3000 will display the current date and version number, and then prompt with a command number and a right parenthesis (")"). RELATE/3000 is now expecting the user to enter a command.

If an error is encountered during the execution of a command, an error number will be displayed between two asterisks (e.g., *6*). If any key other than RETURN is pressed, the error message corresponding to the error number given will be printed. If only a RETURN is entered, no error message will be printed.

SECTION 1

C O N C E P T S

NOTATION

The following notation is used to define the syntax of the RELATE/3000 commands and subroutines described in this manual:

CAPITALIZED BOLD WORDS	are the names of commands or subroutine names.
CAPITALIZED WORDS	identify words that have specific meanings to RELATE. These are sometimes referred to as keywords.
lower-case words	identify words that are names or labels to be specified by the user.
[ ] (Square Brackets)	are used to indicate that the enclosed item is optional and may be omitted. The brackets may be nested such that if the inner items are used the outer items must be used.
| | (Vertical Bars)	between items indicate that one of the items must be chosen.
... (Ellipsis)	indicates that the immediately preceding item may occur once, or any number of times in succession.

Commands may be entered in any combination of upper and lower case characters. They are processed as if only upper case characters had been used (except for information in double quotes). Commands may be up to 1500 characters in length and span up to 100 lines. Each line of the command may be up to 250 characters long. If an error occurs in a command, the command may be edited (see REDO) and then resubmitted.

All command names and most keywords may be abbreviated to as few characters as are required to uniquely identify the word. For example, the OPEN command may be abbreviated to O since no other commands begin with the same letter. The ADD command may only be shortened to AD since the ALLOW command also begins with an "A".

When RELATE is used in a batch mode or from the Host Language Interface routines it is recommended that keywords be spelled out fairly completely. This will ensure that ambiguities will not arise because of new features in subsequent releases of the software. Obviously, an error such as this is easy to correct at the terminal (since the new choices are displayed) but may take some time to correct in a job stream, procedure file, or program.

Commands in RELATE are generally in the following format:

```
[range] VERB [parameters] [FOR condition]
```

The commands must be entered in this format. That is, if a range is required it must come first, the command verb follows, then the parameters to the command followed by any condition. Some latitude is allowed in the entry of the parameter portion of the command. In general, some of the parameters are required while others are optional. The required parameters generally appear first. They must usually be entered in the order indicated in the command's description. Optional items can usually be rearranged as desired.

Punctuation and spacing are crucial when entering commands. Punctuation is performed by delimiters. Delimiters include a space (" "), comma (","), equal sign ("="), or semicolon (";"). Only the first space between items is important; extra spaces may be added to improve readability. Each command describes the punctuation required for proper operation. Generally, however, keywords are delimited by spaces (or equal signs), lists (field names, user names, etc.) are delimited by commas, and keyword sequences delimited by equal signs are separated with semicolons.

SPECIAL CHARACTERS

& (ampersand)	An ampersand entered as the last character on a line indicates that the input for the current line continues on the next line. The prompt for the continuation line will then be an "&)".
\ (backslash)	A backslash can be used to separate multiple commands, or responses, on a single line. A maximum of 1500 characters may be entered in this way. When multiple responses are entered, and an error occurs, all unused information is discarded and prompting is returned to the terminal.
//	This terminates the current input stream. It is also used to exit the system. If a "//" is actually desired as data it must be enclosed in quotes.
Control-H	(or Backspace) Characters can be deleted by using either a Control-H or a Backspace. One character is deleted each time the Control-H or backspace is used. If a hard-copy terminal is being used, the carriage will advance one line and then backspace a single position for each character deleted. If a CRT is being used, the cursor will normally backspace one position per character deleted.
Control-Q	The Control-Q resumes output suspended by the Control-S.
Control-S	The Control-S suspends output to the terminal. Output may be resumed by entering a Control-Q.
Control-X	Use Control-X to delete an entire line. The system responds with three exclamation points, a carriage return, and a line feed. No prompt is printed by the system and data or commands may immediately be entered.
Control-Y	A Control-Y can be used to terminate lengthy printout or cancel the execution of a command. The text "‹Control-Y›" is printed when this key is used. If a procedure file is executing when the key is struck, the user will have the opportunity to continue with or terminate the procedure file.

aggregate - A function that returns summary information on a file. For more information see the SELECT command.

alphabetic - A string of any characters, letters, digits, or punctuation.

assignment - An expression that is evaluated to set a new value. See also the EXPRESSION EVALUATION section.

command - A request by the user for RELATE/3000 to perform some desired action. It normally starts with a verb and is followed by additional parameters.

condition - An expression that is evaluated as TRUE or FALSE. A condition may not include aggregates. See also the EXPRESSION EVALUATION section.

data - Information.

date field - An unsigned, double, or real field given a date format when the file was created or with the MODIFY FIELD command; or an alpha field of the format M/D/Y.

DBA - A Data Base Administrator is the person in charge of the data base system.

double integer - A whole number in the range -2147483647 to +2147483647. A double integer is stored in two words.

expression - Some combination of constants, variables, functions, and operators. See also the EXPRESSION EVALUATION section.

field - The smallest item of information that can be accessed. Fields have many properties, including name, type, print length, and value. Fields are analogous to the blanks one fills in on an employment application. For example, one field could be YOUR_NAME. The field's name would be YOUR_NAME, its type would be Alphabetic, and its value might be "Jim Jones." When the term "field" is used, it will be qualified to indicate which property is being discussed.

fieldname - A name provided by the user to reference a column of information. A fieldname may be up to ten characters long. It must start with a letter and contain only letters, digits, or underscores ("_").

fieldlist - One or more fieldnames separated by commas.

file - A file is composed of one or more records (see "record") of the same type.

filename - A name provided by the user to reference all records of a given type (i.e., a file). A RELATE filename may be up to eight characters long. It must start with a letter and contain only letters or digits.

global switches - These are switches that apply to an entire command and, if used, are always found appended to the first word of the command name. For example, in "MODIFY:K FIELD", "K" is a global switch. See also "switch".

index – A method of organizing and accessing information. See the INDEXES AND KEYS section for more information. An index consists of keys. If a file is ordered by LAST_NAME, for example, the file would have an index containing the field LAST_NAME.

index number – A quick method of referring to an index. An index number may be from 0 to 9. Zero is reserved by the system to reference the index containing line numbers in ascending order. Indexes one through nine are user-defined. See the INDEXES AND KEYS section for more information.

integer – A whole number in the range -32767 to 32767. An integer is stored in one word.

key – A key is one or more fields used to sequence a file. For example, the part number in an Inventory File could be considered a key. Key values comprise the contents of an index.

keywords – Words that have special meanings to RELATE.

line number – Each line (record) of information entered into a RELATE, MPE, or KSAM file is assigned a sequential number, starting at one. These numbers can be used to indicate to RELATE/3000 which records should be processed by a command. See also the RANGE section.

local switches – These are switches that, if used, are attached to the items in a command line. For example, in "CHANGE PARTNO:P", "P" is a local switch. See also "switch".

logical – See "Unsigned".

long – A number, which may contain decimal places, in the range -1.157921×10^{76} to $+1.157921\times10^{76}$. A long number is stored in four words and has an accuracy of about 17 digits.

packed – A number containing up to 28 digits, including the sign, and an optional decimal point. A packed number is stored four (4) digits per word.

printlen – A value indicating the number of characters displayed when a value is printed. A print length is an integer optionally followed by a decimal point and another integer. The first integer portion indicates the total number of characters that will print, including the sign commas the decimal point, fractional digits, and so forth. The integer to the right of the decimal point indicates how many of the characters that are printed will be placed to the right of a decimal point.

procedure file – An ASCII MPE file, which may be an EDITOR file containing RELATE commands.

range - An indication of which records RELATE/3000 should choose to perform a command upon. See also the RANGE section.

real – A number, which may contain decimal places, in the range -1.157921×10^{76} to $+1.157921\times10^{76}$. A real number is stored in two words and has an accuracy of about seven (7) digits.

record – A record is composed of one or more fields. Think of an employment application as a record. Its fields might include NAME, ADDRESS, SALARY, and HOBBIES.

relation – A relation is a file or table containing information in tabular (two-dimensional) format. Files are placed into this format by normalization.

switch – A series of characters appended to an item in a command line other than the range. Switches are a means of expressing options. A switch is preceded by a colon (":").

Blanks are not allowed between the item and the switch or between any of the switches on a single item. Only the first character after each colon is recognized as being a switch. For example, "PRINT:S" could be entered as "PRINT:SUPPRESS".

When a "#" switch is indicated, it may be any single digit from 1 through 9. If a zero is entered, it is ignored. Only the last non-zero numeric switch on an item is recognized; all other numeric switches on the same item are ignored.

Multiple switches are allowed on a single item, as in "PRINT:P:S".

Switches that are not defined for a particular command are ignored. If superfluous switches are included, they will not generate an error.

Switches that indicate conflicting options generate an error.

The order of alphabetic switches on an item is not important.

The same letter may indicate different options on different commands, or on the same command if it is used as both a local and a global switch or on different items in the command line.

type – Each field in RELATE/3000 contains a single type of information. The types recognized by RELATE are: Alphabetic, Zoned, Integer, Double Integer, Real, Long, Packed, and Unsigned.

unary index – An index where the value of the key may not be duplicated. For example, in an Employee File indexed by EMP_NO, there should be no duplicate employee numbers.

unsigned – A whole number in the range 0 to 65535. If its value is 0, its value can also be called FALSE. Any other value can also be called TRUE. A logical is stored in one word.

zoned – A number containing up to 28 digits, plus a sign and a decimal point, if desired. A zoned number is stored two digits per word.

RANGE

Many RELATE/3000 commands allow the user to specify which records should be used in a particular operation. Records may be specified with a condition (described in the EXPRESSION EVALUATION section) or a range. A range is a shorthand method of specifying a condition. Any range can be translated to a condition.

The RELATE/3000 command interpreter interprets a range as a set of index values that indicate the data records to be used in a command. When a file is initially accessed, the system uses index number zero for the evaluation of the range. Index zero corresponds directly to the line or record numbers in the data file.

When a SET INDEX command is properly executed, the current index is changed to that indicated in the command. Furthermore, until access to the file is terminated, or another SET INDEX or PURGE INDEX command is executed for the current index, the range parameter must correspond to the format of that index. If a SELECT command with a BY clause is issued, the fields in the BY clause are used as the current index.

When an index has N fields, up to N+1 values may be included in the range. The N+1st value is the line number and exists in all indexes. Each of the values in the range must be separated by a colon (":"). If less than N+1 values are supplied, the remainder of the range defaults to either the largest or smallest value of the field, depending on the portion not supplied and whether the fields are in ascending or descending order.

A range consists of one or more of the following separated by commas:

value	A single value of the first field in the current index. All records having this value will be returned. If the field is alphabetic or date formatted, the value must be enclosed in quotes.
[stval]/[endval]	A starting value and an ending value. All records having values greater than or equal to stval and less than or equal to endval are returned. If stval is not specified, all records less than or equal to endval are returned. If endval is not specified, all records greater than or equal to stval are returned.

EXAMPLES:

If index #1 is composed of the fields PART and BIN, where PART is an alphabetic field (4 characters long), and BIN is an integer field:

```
"BOLT" PRINT
```

Prints all records (if any) where the part is listed as "BOLT".

```
"BOLT":5 PRINT
```

Prints all records where the part is listed as "BOLT" and the BIN is 5.

```
"BOLT":5:6 PRINT
```

Prints the record that has PART="BOLT", BIN=5, and a line number of 6.

```
"B"/"C" PRINT
```

Prints all records that have a part field greater than or equal to "B" and less than or equal to "C".

```
"BOLT","NUT":1 PRINT
```

All records that have PART="BOLT", and all records that have PART="NUT" in BIN 1, will be printed. Multiple ranges can be strung together, as above, by separating them with commas.

EXPRESSION EVALUATION

Throughout the text, several references are made (especially in the Command Reference section) to "expressions", "assignments", and "conditions". In general, all are made up of the same elements. An "expression" is some combination of constants, variables, functions, and operators. An "assignment" consists of a variable on the left, an equal sign, and an expression on the right, and results in the variable name obtaining the value of the evaluated expression. A "condition" is an expression that is evaluated to either TRUE (non-zero) or FALSE (zero) and qualifies records for use in a command.

Examples:

ASSIGNMENTS	CONDITIONS
	$MATCH(STRING,"A")
	LOGICALVAR
A=B	A=B
TOTAL=PART1+PART2	TOTAL=PART1+PART2
NUM=(OTHER/2.4)+$SIN(ITEM)	NUM=(OTHER/2.4)+$SIN(ITEM)
	NAME="SMITH" AND AGE<47
	CHAR>"W"
	PART1+PART2<=$COS(OTHER)*2

FOR condition vs. WHERE condition

Conditions following a WHERE keyword may contain aggregates (described in the SELECT command). Conditions following a FOR keyword may not contain aggregates.

Data Types

Any of the data types recognized by RELATE can be used in expressions. There are eight data types recognized by RELATE: ALPHABETIC, ZONED, INTEGER, DOUBLE INTEGER, REAL, LONG, PACKED, and UNSIGNED. Refer to the glossary for a description of each of the types.

Constants

Constants are elements in an expression which have a fixed value. Constants contained in quotes are ALPHABETIC in type. To determine the type of numeric constants, see the Type Conversion paragraph.

Examples: These are all constants:

```
"A"
"LMP*247"
3982
4.65
```

Variables or Fields

Variables (usually referred to as Fields) are elements in an expression which have different values depending on the current record.

Examples: These are all variables:

```
A
LMP
X247
FIRST_NAME
A.FRED
```

Type Conversion

If more than one data type appears in an expression, RELATE automatically converts the items to be consistent with one another. If constants are used in an expression, RELATE determines the type of the constant based on the types of the values surrounding the constant. For example, if A=B+1.23 is used, the 1.23 would be converted to the same data type as B. The operands would be added together, after which A or B would be converted. The following table summarizes the conversion operations:

		SURROUNDING VALUE							
	-	A	I	U	D	R	L	P	Z
	A	-	I	U	D	R	L	P	Z
	I	I	-	D	D	R	L	P	Z
	U	U	D	-	D	R	L	P	Z
ORIGINAL	D	D	D	D	-	R	L	P	Z
VALUES	R	R	R	R	R	-	L	P	Z
	L	L	L	L	L	L	-	P	Z
	P	P	P	P	P	P	P	-	Z
	Z	Z	Z	Z	Z	Z	Z	Z	-

The system attempts to preserve as much accuracy as possible by always converting to a representation that allows more significant digits or a more direct representation. If the automatic conversion done by RELATE seems incorrect, the user can explicitly state the data type preferred by using the $INTEGER, $DOUBLE, $REAL, $LONG, $PACKED, $ZONED, and $UNSIGNED functions.

Operators

An expression obtains meaning by connecting constants, variables, and functions with various operators. The standard operators available in RELATE are as follows:

NAME	SYMBOL	EXAMPLE	MEANING
unary plus	+	+5	Positive five.
unary minus	-	-5	Negative five.
plus	+	3+A	Add three and the value of A together.
minus	-	3-A	Subtract the value of A from three.
times	*	3*A	Multiply three times the value of A.
divide	/	3/A	Divide 3 by the value of A.
exponent	**	A**2	A squared.
NOT	NOT	NOT A	If A is not true (was zero or contained only blanks).
AND	AND	A AND B	If A and B are both true.
OR	OR	A OR B	If either A or B is true (was non-zero or contained characters other than blanks).

Conditions and assignments may, in addition, contain any of the following logical operators:

NAME	SYMBOL	EXAMPLE	MEANING
less than	<	A<3	True if the value of A is less than three.
less than or equal to	<=	A<=3	True if the value of A is less than or equal to three.
equal	=	A=3	True if value of A is equal to three.
greater than	>	A>3	True if the value of A is greater than three.
greater than or equal to	>=	A>=3	True if the value of A is greater than or equal to three.
not equal to	<>	A<>3	True if the value of A is not equal to three.

"Assignments" must contain the assignment operator:

equal	=	A=3	Set A to equal three.

The hierarchy of expression evaluation is as follows:

1) unary +, -
2) NOT
3) **
4) *, /
5) +, -
6) <, >, <=, >=, =, <>
7) AND
8) OR

Operators at the same level are not guaranteed to be performed in any particular order. Parentheses may be used to override the above hierarchy or to emphasize the order of evaluation. If nested parentheses are encountered, the inner expressions are evaluated first. Comparisons and the operators AND, OR, and NOT result in an integer value of one (for true) or zero (for false).

When comparisons of alphabetic fields or constants are performed, the shorter string is assumed to be padded with blanks up to the length of the longer string.

TYPE CONVERSION FUNCTIONS

RELATE/3000 will automatically convert between various data types when expressions are evaluated. The rules for conversion are given in the Expression Evaluation section. In some cases the default conversion precedence is not correct. The functions listed below allow the user to override the conversion order normally used by RELATE.

The printlen and decimal value must be specified as constants. If the type conversion function results in an integer, double, or unsigned field, the value is rounded to the nearest whole number.

$ASCII(expression [,printlen])
Transforms the value of the expression to an alphabetic value with the indicated print length. If $ASCII is performed on a field with a date FORMAT, the result will be the date in the field's defined format.

$DOUBLE(expression [,printlen])
Transforms the value of the expression to a double integer with the indicated printlength.

$INTEGER(expression [,printlen])
Transforms the value of the expression to an integer with the indicated printlength. Fractional amounts will be truncated.

$LONG(expression [,printlen [,decimals]])
Transforms the value of the expression to a long number with the indicated number of decimal positions included in its print length.

$PACKED(expression [,printlen [,decimals]])
Transforms the value of the expression to a packed number with the indicated number of decimal positions included in its print length.

$REAL(expression [,printlen [,decimals]])
Transforms the value of the expression to a real number with the indicated number of decimal positions included in its print length.

$UNSIGNED(expression [,printlen])
Transforms the value of the expression to an unsigned number with the indicated print length.

$ZONED(expression [,printlen [,decimals]])
Transforms the value of the expression to a zoned number with the indicated number of decimal positions included in its print length.

CHARACTER MANIPULATION FUNCTIONS

$APPEND(string [,...])
Returns an alphabetic field comprised of all of the parameters concatenated together. No deblanking is performed.

$CONCAT(string [,...])
Returns an alphabetic field comprised of the deblanked strings concatenated together. If a constant is used, the constant is not deblanked. Fields and variables, however, will be deblanked. The function results in a field large enough to contain all of the strings. The $ASCII function can be used to force the creation of a field containing fewer characters.

$DEB(string)
Returns the string without any leading or trailing blanks.

$DOWNS(string)
Returns the alphabetic field with all uppercase characters downshifted.

$HEAD(string [,separator])
Returns a substring of the first string starting at the beginning of the string and ending with the character immediately before the first occurrence of the indicated separator. The separator must be a field or constant of no more than 40 characters. If the separator is not given, the space character is used. If no separator is found in the string, the entire string is returned.

$LENGTH(string)
Returns the position of the last non-blank character in the field as an integer value.

$LOWERCASE(string)
Returns a string the same length as the input string. The string will be converted to lowercase with the first letter of each word capitalized. Any letter immediately following a blank, dash, period, slash ("/"), pound sign ("#"), comma, or left parenthesis is considered to be the first letter of a word.

$ROMAN(amount)
Returns a character string containing the value of amount in lower case Roman numerals. Amount may be negative. Decimal fractions are truncated.

$SUBSTR(string, start [,length])
Returns the portion of the field beginning with start for a count of length characters. If length is not specified the remainder of the field is returned. The first character in the string is 1.

$TAIL(string [,separator])
Returns a substring of the first string starting with the first character after the first occurrence of the separator and ending at the end of the string. The separator must be a field or constant of no more than 40 characters. If the separator is not given, the space character is used. If no separator is found, a string of zero length is returned.

$UPS(string)
Returns the alphabetic field with all lowercase characters upshifted.

$WORD(amount [,length [,sequence]])
Returns a string containing the amount written out in all capital letters as a number on a check (e.g. $WORD(1.23)= "ONE DOLLAR AND 23 CENTS"). The second parameter specifies the length of the resulting string. The default length is 80 characters. If the resulting string is longer than the specified length, then the rest of the string may be obtained by specifying a sequence number of 2 (e.g. sequence of 1 returns the first 80 characters, sequence=2 returns the next 80 characters, and so forth). If sequence is not specified, the default is one.

ARITHMETIC FUNCTIONS

$ABS(expression)
Returns the absolute value of the expression. The type of data returned by the function is determined by the type of the expression.

$CASE(expression, case1, value1, [case2, value2,]...defaultvalue)
Evaluates the expression and compares the result to each case. If a match is found, the following value is returned. If no match is found, the defaultvalue is returned.

$EXP(expression)
Returns the mathematical value "e" raised to the value of the expression. The type of data returned by the function is determined by the type of the expression.

$FACT(expression)
Returns the factorial of the specified expression. The value of the expression is first truncated to an integer ($FACT(4)=4*3*2*1=24).

$IF(expression, truevalue, falsevalue)
Returns truevalue if the expression is non-zero or non-blank. Returns falsevalue if the expression is zero or blank. The type of data returned by the function is determined by the truevalue and falsevalue expressions.

$LOG(expression [,base])
Returns the log of the expression with the indicated base. Base is optional. The default is the natural log. The type of data returned by the function is determined by the type of the expression.

$MAXIMUM(exp1, exp2, ...expN)
Returns the largest of the passed expressions. The type of data returned by the function is determined by the types of the expressions.

$MINIMUM(exp1, exp2, ...expN)
Returns the smallest of the passed expressions. The type of data returned by the function is determined by the types of the expressions.

$MOD(expression, modulo)
Divides the expression by the modulo and returns the remainder. The type of data returned by the function is determined by the type of the expression.

$PI
Returns the value of pi (3.14159) as a 64 bit floating point number.

$ROUND(expression [,decimals])
Returns a number rounded off to the specified number of decimal positions. If decimals is omitted then zero is assumed. Examples:

```
$ROUND(124.381,0)=124.000
$ROUND(124.381,1)=124.400
$ROUND(124.381,2)=124.380
$ROUND(124.381,-1)=120.000
```

$SIGN(expression)
Returns a -1 if the expression is less than zero, a 0 if the expression is zero and 1 if the expression is greater than zero. The type of data returned by the function is determined by the type of the expression.

$SQRT(expression)
Returns the square root of the expression. A 32 bit floating number is returned.

TRIGONOMETRIC FUNCTIONS

All trigonometric functions return real results regardless of the expression type.

$ACOS(expression [,1])
Returns $\cos^{-1}$(expression). The result is in radians if 1 is not specified; in degrees if it is.

$ASIN(expression [,1])
Returns $\sin^{-1}$(expression). The result is in radians if 1 is not specified; in degrees if it is.

$ATAN(expression [,1])
Returns $\tan^{-1}$(expression). The result is in radians if 1 is not specified; in degrees if it is.

$COS(expression [,1])
Returns the cosine of the expression. The expression must be in radians if 1 is not specified; in degrees if it is.

$SIN(expression [,1])
Returns the sine of the expression. The expression must be in radians if 1 is not specified; in degrees if it is.

$TAN(expression [,1])
Returns the tangent of the expression. The expression must be in radians if 1 is not specified; in degrees if it is.

DATE MANIPULATION FUNCTIONS

Only UNSIGNED, DOUBLE, or REAL fields or ALPHABETIC fields or constants represent valid date types. RELATE is aware of how these values are stored internally and can manipulate them appropriately. If an ALPHABETIC field or constant is used, it must be in "M/D/Y" or "M/D/C" format. An alphabetic field cannot, however, be given a date format with the MODIFY command.

$DAY(date)
Returns an integer representing the day of the month for the date.

$DAY_DIFF(startdate, enddate)
Returns the number of days between the two dates as a DOUBLE data type. If the enddate falls before the startdate, a negative value will be returned.

$DAY_WEEK(date [,option])
Returns the day of the week of the date passed. If option is not included, value is returned as an integer. If an undefined option is specified, blanks are returned. The options are as follows:

0 Day number (1-7) returned as an integer representing Sunday thru Saturday.
1 The first three characters of the day in uppercase ASCII characters.
2 The complete day name in upper and lowercase ASCII characters.

$FORMAT_TIME(time [,option])
Returns a string containing a formatted time. The time is an integer in a 24 hour (HHMM) format. If no option is specified, zero is assumed. If an undefined option is specified, blanks are returned. The options are as follows:

0 1:34 PM
1 13:34
2 1334

$JULIAN(date)
Returns an integer representing the julian portion of the date.

$LAST_DAY(date)
Returns a date (month, day, and year) equal to the last day of the month in the month and year of the original date.

$MONTH(date [,option])
Returns an indication of the month of the passed date in an alphabetic field. If option is not included, value is returned as an integer. If an undefined option is specified, blanks are returned. If no option is specified, zero is assumed. The options allowed are as follows:

0 Month number (1-12) returned as an integer.
1 The first three characters of the month name in uppercase ASCII characters.
2 The complete month name in upper and lowercase ASCII characters.

$NEW_DATE(originaldate, days)
The $NEW_DATE function returns a date the indicated number of days before or after the original date. The days parameter may be either positive or negative.

$YEAR(date)
Returns an integer representing the year of the date. The year will contain the century as in 1982.

INTER-RECORD FUNCTIONS

$LAST(expression [,key1,...])
The $LAST function can be used to move information from a record to the next record. The function operates by holding the value of the expression for the first record until the second record. The first use of the function returns a zero.

If any key fields are present, the function resets to zero when they change. The key fields do not sort the files as is the case with an aggregate. The user must guarantee that the records are fed through the function in the correct order. This function can safely be used in assignments and conditions. The function should not be used in a SELECT command unless a BY clause is given and the clause is satisified by a sorted index on one of the base files. The function should not be used in a view definition.

$RTOTAL(expression [,key1,...])
The $RTOTAL function returns the running total of the expression within a key. The function operates similarly to the $LAST function.

PATTERN MATCHING

In several places in RELATE, lists can be shortened by specifying a particular pattern that the items in the list should match. In addition, the $MATCH function is provided to make use of RELATE's pattern matching capabilities over the contents of a field in a file. Pattern matching may be used in:

$MATCH function
PRINT command fieldlist
SUM command fieldlist
CONSOLIDATE command fieldlist
FIELDS= in CREATE and OPEN FILE commands

The elements of pattern matching are:

%text	Search for all items beginning with the indicated text.
text$	Search for all items ending with the indicated text.
text?text	Match the indicated text but allow any character in the position(s) indicated by question mark(s).
char*	Zero or more consecutive occurrences of the preceding character.
[chars] [range of chars]	Indicates a character class. If any one of the characters in the class occurs, then a match is found (see example in $MATCH, following). The character class may include a slash as the first character to indicate NOT in this character class.
@char	Used when you want an actual %, $, ?, *, [, or] to indicate that the character should be read as is.

$MATCH(string, matchstring)

string	A character literal (a string enclosed in quotes), or a fieldname, whose contents will be searched for a match to the matchstring. If the string is not a constant it will have trailing blanks removed before being used.
matchstring	A character literal (a string enclosed in quotes), or a fieldname whose characters are described below, which $MATCH will attempt to find in the original string. If the matchstring is not a constant it will have trailing blanks removed before being used.

The $MATCH function returns a number indicating the position of the first character in the string that satisfies the matchstring. If a match cannot be found, zero is returned.

If string or matchstring are non-alphabetic fields, they will be converted to characters before the match is performed.

SAMPLE	EXPLANATION
$MATCH(TEXT,"A")	Find the position of the letter "A" in the TEXT field in all records.
$MATCH(TEXT,"%A")	Find all records where the first character of the field TEXT is the letter "A".
$MATCH(TEXT,"A$")	Find all records where the last character of the field TEXT is the letter "A".
$MATCH(TEXT,"A?B")	Find the first position of an "A", followed by any character, followed by a "B" in the field TEXT in all records. Both "ANB" and "XA&B" will be found.
$MATCH(TEXT,"?A???X?")	Find the position in each TEXT where there is a string of the format: any character followed by an "A", 3 other characters, and an "X". This will find "BAD AXE" but not "AD AXE" since there is no character immediately before the "A".
$MATCH(TEXT,"A?*B")	Find the position in each TEXT where there is a string of the format: "A", followed by any number of characters, followed by a "B". This would find "AB", "AQB", and "AS ANYBODY".
$MATCH(TEXT,"A??*B")	Find the position in each TEXT where there is an "A" followed by at least one other character followed by a "B".
$MATCH(TEXT,"A*")	Find the position in each TEXT where there are zero or more consecutive "A"'s (this would naturally find all records).
$MATCH(TEXT,"[ABC]X")	Find the position in each TEXT where there is either an "A", "B", or "C" followed by an "X".
$MATCH(TEXT,"[a-z][a-z]")	Find the position in each TEXT with two consecutive lowercase letters.
$MATCH(TEXT,"[/a-z]")	Find all texts that do not contain any lowercase letters.

$MATCH(TEXT,"@%@@")	Find the position in each TEXT where there is a "%" followed by an "@".

EXAMPLES:

The $MATCH function returns a number indicating the position of the requested match string. Since conditions are evaluated as true if the result is non-zero, this can be useful in FORs and WHEREs.

```
)OPEN FILE CUST
)SELECT NAME, X=$MATCH(NAME,"X")
)PRINT

NAME                    X

HASLETREX INC.          9
DEXMACH, INC.           3
CUPERCO                 0
AMERICAN TIRE CO.       0
FINCH, FINCH, & OTTO    0
NATIONAL AIRLINES       0
ALEXANDER HALE & CO.    4
PERFECT SOUND           0

8 LINES PRINTED.
)SELECT NAME WHERE $MATCH(NAME,"X")
)PRINT

NAME

HASLETREX INC.
DEXMACH, INC.
ALEXANDER HALE & CO.

3 LINES PRINTED.
)
```

SYSTEM DEFINED FIELDS

RELATE/3000 has several pre-defined fields available to the user. These fields allow access to the current date, time of day, port number, user name, and group name. The fields can be used in any expression, condition, targetlist or qualification. They can be used in security constraints in order to limit access on a time of day, user name or port number basis.

The following system defined fields exist:

$ACCOUNT	Returns the account name to which the user is logged in as an 8 character alphabetic field.
$DATE	Returns the current date as a double integer.
$ERROR	Returns the error number of the latest RELATE error. This is reset to zero whenever an IGNORE ERROR is encountered.
$GROUP	Returns the group name to which the user is logged in as an 8 character alphabetic field.
$PORT	Returns the terminal device number to which the user is logged on. A zero is returned if the function is evaluated in a job.
$TIME	Returns the current time of day as an integer. The time is returned in a 24 hour (HHMM) format.
$USER	Returns the user name as an 8 character alphabetic field.

INDEXES AND KEYS

When the user accesses information, he or she may need to be able to access it in more than one manner. Indexes assist in or speed up this operation. An index is an organized collection of fields by which the data in a file is referenced. A key is that field or fields.

If one thinks of a file cabinet as a file, the information in it can be sorted in only one way, usually alphabetically. If the information is sorted only by LAST NAME, with no attention paid to any other information, then we say that the KEY in the index is LASTNAME. We could also say "the file is indexed by [the key] LASTNAME".

Usually, however, other information is used to help access the information. For example, if duplicate LASTNAMEs are found, then checking proceeds to FIRSTNAME. If duplicates exist there, then the checking could proceed to an ADDRESS. In this case, the key would be said to be LASTNAME, FIRSTNAME, ADDRESS. One could also say "the file is indexed by [the key] LASTNAME, FIRSTNAME, ADDRESS", or "the index contains LASTNAME, FIRSTNAME, ADDRESS".

In RELATE/3000 a data file can have several indexes, although only one index can be used explicitly by the user at one time.

One index is supplied automatically by RELATE. This is the line number. As each record is added to a file, it is given its own unique line number. Nothing special need be done to make this index the current method of access. When a file is opened, it is automatically accessed by the line number.

The user will usually want to access information in a different order than line number (i.e., by LASTNAME and then FIRSTNAME). In that case, the command "CREATE INDEX BY LASTNAME, FIRSTNAME" can be used to sort the file in that order. When the command is completed, the system will say "INDEX #1 HAS BEEN CREATED". This means that from now on there will be an INDEX #1 consisting of the key LASTNAME, FIRSTNAME (until it is purged using the PURGE INDEX command).

Thereafter, when the user opens the file, it will be sorted by line number. But if the user wants it ordered by LASTNAME and FIRSTNAME, the command "SET INDEX 1" can be entered.

Up to thirty indexes may be defined in this manner on RELATE type files. The user may flip from one to the other, as often as desired, using the SET INDEX command. The current index defines the field that comprises the range of a command.

NORMALIZATION

For a relational database system to function correctly, all of the files used by the system must be normalized. The process of normalization removes repeating groups (COBOL OCCURS clauses) and decomposes files in such a way that the contents of each field in a file contains a value that depends only on the primary key of the record. This is a simplified explanation of normalization; for a more thorough description, the user is directed to An Introduction to Database Systems by C. J. Date, published by Addison-Wesley.

Although normalization sounds complex, the operations used to create the normalized files are really based on common sense. An example using a hypothetical job cost and employee information file should serve to illustrate the problems of working with unnormalized data.

Assume a file with the following fields:

```
WORK:     EMP_NO, EMP_NAME, SALARY,
          (PROJ_NO, HOURS, END_DATE)*10
```

The PROJ_NO, HOURS, and END_DATE fields are repeated ten times for each record.

To place the file into "first" normal form, the repeating group must be eliminated. This can be done by duplicating the EMP_NO, EMP_NAME and SALARY fields for each PROJ_NO entry. After the repeating group is removed, the file can be used by RELATE. Unfortunately, the file contains many undesirable features. Some of the undesirable properties that are exhibited by the new format include:

1) To change an employee's salary, many records may need to be changed depending on the number of projects that he has worked on.

2) An employee's name and salary cannot be entered until he has been assigned a project. Likewise, a project completion date cannot be specified until an employee is assigned to the project.

3) It contains a large amount of redundant data.

These problems occur because data exists in the record that is not directly associated with the primary key (in this case EMP_NO). To correct the problem, the file must be divided into two files:

```
EMP:      EMP_NO, EMP_NAME, SALARY

WORK:     EMP_NO, PROJ_NO, HOURS, END_DATE
```

Each field in the EMP file is now directly associated with the primary key (EMP_NO). Each field in the WORK file is not, however, directly related to the key (which is EMP_NO, PROJ_NO). The process performed on the original WORK file must now be repeated on the new WORK file:

```
WORK:     EMP_NO, PROJ_NO, HOURS

PROJ:     PROJ_NO, END_DATE
```

After this second transformation, the data is in a correctly normalized form. Notice that it is possible to:

1) Change an employee's salary or name by updating a single record.

2) Add a completion date for a project without assigning any employees and add an employee without assigning him a project.

3) Delete a project or an employee without destroying any other information.

The result of the normalization process is a set of two-dimensional tables. The format or content of these tables is easily understood by non-technical people, and the operations that must be performed on these tables by an application is much simpler to describe.

"TO filename"

When the "TO filename" clause is used in a command, RELATE will attempt to open the file for the duration of the command whether or not the file has already been explicitly opened. All options listed in the OPEN FILE command that are needed to access the file must be appended to the filename. If the file is correctly opened, the system continues.

If the file cannot be opened, RELATE will attempt to create the file. In this case, the options available on the CREATE FILE command may be specified after the filename. If the file cannot be created, an error results.

After the file is located, opened, or created, the path name that the file has assumed for the duration of the command is checked against the names of any other files that are also used in the command. If the name is duplicated, an error occurs and the command terminates.

When an output file is used, RELATE will automatically match fieldnames from the input file(s) with those in the output file(s). The order in which fields are matched is described in each command. RELATE will automatically perform data type conversion as well as alphanumeric field expansion and truncation during any copy operation.

If information is converted from a numeric format to an alphabetic format the resulting field will reflect all format options in effect at the time. For example, if the source field is formatted as a date field, the destination field will be filled with a date. Any type conversion errors that occur are ignored and result in zeroes or blanks in the destination field.

EXAMPLES:

```
OPEN DATABASE XDB; TYPE=IMAGE; PASSWORD="xxx"; MODE=3
OPEN SET LONGSET; PATH=L; DATABASE=XDB
OPEN FILE SFILE
COPY LONGSET.NAME=SFILE.TITLE  TO LONGSET;DATABASE=XDB

CREATE FILE ABC; RETENTION=TEMP; FIELDS=(NAME,A,20); PATH=A
SET PATH SFILE
COPY A1.NAME=SFILE.TITLE TO ABC; RETENTION=TEMP; PATH=A1
```

EXECUTION OF RELATE/3000 IN A JOB

When the RELATE/3000 Command Interpreter is executed in a job all commands and prompts are echoed to $STDLIST (RDBOUT). The result is identical to the output that would be obtained if RELATE/3000 was run at a hard copy terminal. RELATE/3000 commands and functions execute the same way in a job as in a session, except for the following:

1) The continuation character ("&"), if used, must be the last non-blank, non-null character on the line.

2) All errors are treated as fatal errors. The error number is printed followed by its description. A message is then generated indicating that the job will be terminated. RELATE then sets the system Job Control Word (JCW) bit 0 high and terminates. The job is then cancelled by MPE.

SECTION 2

C O M M A N D S

:mpecommand

Executes an MPE command directly from RELATE.

mpecommand	Must be an MPE command that can be executed programmatically, or the RUN or EDITOR command. See the MPE Intrinsics Reference Manual (HP part number 30000-90010) for a complete list of the commands that may be used.

EXAMPLES:

Any MPE command that can be executed from a program can also be executed from RELATE. The command name must be preceded by a colon (":").

```
    ):SHOWME
    USER: #S453,DOC79.RDB,DOC79        (NOT IN BREAK)
    MPE VERSION: HP32002C.N0.A3
    CURRENT: THU, MAR 18, 1982,  2:14 PM
    LOGON:   THU, MAR 18, 1982, 10:54 AM
    CPU SECONDS: 86           CONNECT MINUTES: 201
    $STDIN LDEV: 41           $STDLIST LDEV: 41
    )
```

ABORT [ENTIRE] TRANSACTION

Causes all file changes made at the current transaction level to be ignored and reduces the transaction level by one.

ENTIRE	Optional. If the transaction level is greater than 1 all levels can be ABORTed by using this keyword.

EXAMPLES:

The ABORT TRANSACTION command informs RELATE to ignore any file modification requests at the current transaction level. In most cases, this terminates the entire transaction. In some instances, transactions may be nested and the ABORT will cause the nesting level to be decreased by one (1) thus undoing a portion of a transaction.

```
)BEGIN TRANSACTION
THE TRANSACTION LEVEL IS NOW 1.
)PRINT NAME,ST

$LINE NAME                 ST

    1 HASLETREX INC.       CA
    2 DEXMACH, INC.        CA
    3 CUPERCO              CA
    4 AMERICAN TIRE CO.    CA
    5 FINCH, FINCH, & OTTO CA
    6 NATIONAL AIRLINES    CA
    7 ALEXANDER HALE & CO. NJ
    8 PERFECT SOUND        VA

8 LINES PRINTED
)DELETE FOR ST="CA"
6 LINES DELETED
```

Because a transaction is in progress the changes have not yet been made on the file.

```
)PRINT NAME,ST

$LINE NAME                 ST

    1 HASLETREX INC        CA
    2 DEXMACH, INC         CA
    3 CUPERCO              CA
    4 AMERICAN TIRE CO     CA
    5 FINCH, FINCH, & OTTO CA
    6 NATIONAL AIRLINES    CA
    7 ALEXANDER HALE & CO. NJ
    8 PERFECT SOUND        VA
```

```
8 LINES PRINTED.
```

If the operations performed in the transaction are incorrect they can be discarded by ABORTing the transaction

```
)ABORT TRANSACTION
ALL TRANSACTIONS HAVE BEEN ABORTED.
```

The contents of the file are unchanged.

```
)PRINT NAME,ST

$LINE NAME                 ST

    1 HASLETREX INC.       CA
    2 DEXMACH, INC.        CA
    3 CUPERCO              CA
    4 AMERICAN TIRE CO.    CA
    5 FINCH, FINCH, & OTTO CA
    6 NATIONAL AIRLINES    CA
    7 ALEXANDER HALE & CO  NJ
    8 PERFECT SOUND        VA

8 LINES PRINTED.
)
```

ADD [SEPARATOR="character"]

Adds data to the current file.

:I	Optional global switch. If the ADD command is used in a procedure, data is requested from the user's terminal (from $STDINX) unless a global "I" switch is used. If the global "I" switch is used, data is obtained from the procedure file. The switch is ignored if a procedure is not executing.
:L	Optional global switch. If specified, the line number will be displayed after each record is added. The line number will not be displayed if records are added to a view.
SEPARATOR	Optional. If a separator other than a comma is desired during data entry, it must be included in the command line. The separator must be a single non-blank character enclosed in quotes ("). Backslashes (") may not be used.

The ADD command will prompt for data input by printing a fieldname and a question mark. In general, it will prompt for input in the order in which the fields were described during the creation of the file (for exceptions, see the Data Entry Levels paragraph). One value may be entered, in which case the next field will be prompted for. More than one value may be entered, separated by commas or the user-defined separator, in which case the values will be assigned to the fields in the proper order and then the next field for which no data was found will be used as the prompt.

Data Entry Levels

Fields with a data entry level of zero are set to zeroes or blanks depending on type. These fields cannot be entered. Fields with a data entry level of two will be prompted for only once at the beginning of the command. All fields with a data entry level greater than two will be prompted for continually until "//'s" are entered. When the command begins, all fields with a data entry level greater than one are listed in increasing order of data entry level and within each level in the order in which they were structured. After all fields that need to be entered are printed, the first field is prompted for. The user may then enter as many fields as exist on the first level, separating them with commas or the user-specified separator.

Entering Adjacent Separators, //, RETURN, or "."

If two adjacent separators are found or if only a RETURN is entered, the corresponding field is set to blanks or zero depending on its type.

If a "//" is entered in the first field in a level, the user will be prompted for the next lower level that has enterable fields. If the user was at a level three, or no fields with a lower level exist, the command is terminated.

If a "//" is entered for any field on a level except for the first field, the data previously entered on that level is discarded and the user is prompted for the level again.

A "." may be entered to carry the previous value of a field down to the current line.

Input for alphabetic fields

Data for alphabetic fields may consist of any printable characters. If more data is entered than can be placed into an alphabetic field a warning is issued and the information is right truncated.

Checking keys

For RELATE files, as each field is entered, the index structure is checked to determine if the current field is the last field required to complete a unary key. If it is, the key is created and checked against the data file. If a duplicate key is found, an error occurs and the field is prompted for again.

Invalid Values

If an invalid value is entered into a field, the remaining input is discarded and the user will be prompted for the field again.

Trailing Spaces and Zeroes

It is not necessary to enter trailing spaces or zeroes. RELATE/3000 will place them into the file and provide them on printed output when appropriate. Leading zeroes are ignored on all numeric fields.

Trailing Separators

A trailing separator will be treated as though blanks or zeros, depending on the type of the following field, were entered after the separator.

Filling the File to Capacity

If the current file is filled during the command, the command is terminated with a message to that effect. No further additions are allowed until the file is expanded. This can easily be accomplished with a COPY or REORGANIZE command.

EXAMPLES:

A file must be current in order to ADD to it. Get the INVOICE file, permanently change the entry levels (see the MODIFY command) and then add some data to the file.

```
    )SET PATH INVOICE
```

```
)MODIFY:K FIELD NAME,NUMBER,ST; LEVEL=3
)MODIFY:K FIELD INVNO,AMOUNT,TAX; LEVEL=4
)MODIFY FIELD SALES_MAN; LEVEL=2
)ADD
ENTER SALES_MAN,NAME,NUMBER,ST,INVNO,AMOUNT,TAX

SALES_MAN? 36
NAME? NATIONAL AIRLINES
NUMBER? 1000
ST? CA
INVNO? 10221
AMOUNT? 627.01
TAX? 88.07

INVNO? 10455,335,40.2

INVNO? //

NAME? CUPERCO
NUMBER? 400
ST? CA
INVNO? 2738,948.6,32.40

INVNO? //

NAME? //
```

See what the file contains.

```
)PRINT

$LINE  INVNO NAME                     NUMBER ST   AMOUNT    TAX SALES

    1  10221 NATIONAL AIRLINES          1000 CA $ 627.01  88.07    36
    2  10455 NATIONAL AIRLINES          1000 CA $ 335.00  40.20    36
    3   2738 CUPERCO                     400 CA $ 948.60  32.40    36

3 LINES PRINTED.
```

Change the separation character. This enables a comma to be entered as part of a NAME.

```
)ADD SEPARATOR=";"
ENTER SALES_MAN;NAME;NUMBER;ST;INVNO;AMOUNT;TAX

SALES_MAN? 99
NAME? DEXMACH, INC.
NUMBER? 100
ST? CA
INVNO? 33;348.7
TAX? 18

INVNO? //
```

```
NAME? //

)PRINT

$LINE   INVNO NAME                          NUMBER ST    AMOUNT     TAX SALES

    1   10221 NATIONAL AIRLINES               1000 CA $ 627.01   88.07    36
    2   10455 NATIONAL AIRLINES               1000 CA $ 335.00   40.20    36
    3    2738 CUPERCO                          400 CA $ 948.60   32.40    36
    4      33 DEXMACH, INC.                    100 CA $ 348.70   18.00    99

4 LINES PRINTED.
)
```

ADD FIELD fieldname, type, printlength [.decimals] [;options]

Adds a new field to the structure of the current file.

fieldname	Required. A fieldname must be between 1 and 10 characters in length, contain only capital letters, digits, or underscores ("_") and start with a letter. Fieldnames must be unique within a file. Keywords such as TO, ERRORS, WITH, USING, and BY should not be used as fieldnames.
type	Required. The type of the field. The type must be one of the following: ALPHABETIC, DOUBLE, INTEGER, LONG, PACKED, REAL, UNSIGNED, ZONED
printlength	Required. The printlength determines the printing width of the field. For ALPHABETIC, PACKED, and ZONED fields, it also determines the number of words stored in the file. The limitations on the size of each field appears in the FIELD SIZE LIMITATIONS table in the CREATE FILE command. The printlength is the total print length of the field, including the sign, decimal point, and decimal positions. **[.decimals]** is an optional number of decimal positions within the total print length.
options	Optional. Any options described in the FIELD OPTIONS table in the CREATE FILE command may be specified.

The current file must be a RELATE/3000 file to which the user has exclusive access. To add a field to a file in a secured group, the user must be the file's creator or the account librarian.

The new field cannot be added if the current file already contains 127 fields. The addition of a field is a logical operation which is performed very quickly. Individual records are not adjusted until some other operation forces them to be rewritten. The addition of fields to a file may cause the file to fill before its stated capacity is reached. To increase the physical space alloted to the file by MPE the file should be REORGANIZEd.

EXAMPLES:

Add a date field to the customer file.

```
    )OPEN FILE CUST
```

```
)SHOW STRUCTURE

FILE NAME               =CUST.DOC79.RDB

                  T          C             L A U
                  Y PRINT    O             E D P  INT  BEG  END
#   NAME          P  LEN  $  M SPECIAL     V D D SIZE WORD WORD

  1 NAME          A  20                    3 Y Y  20B    0    9
  2 NUMBER        I   5                    3 Y Y   1W   10   10
  3 ADDRESS       A  26                    3 Y Y  26B   11   23
  4 CITY          A  14                    3 Y Y  14B   24   30
  5 ST            A   2                    3 Y Y   2B   31   31
  6 PHONE         A   8                    3 Y Y   8B   32   35

PRINT LINE WIDTH = 86 CHARACTERS.
)ADD FIELD STARTDATE,D,8;FORMAT="M/D/Y"
)SHOW STRUCTURE

FILE NAME               =CUST.DOC79.RDB

                  T          C             L A U
                  Y PRINT    O             E D P  INT  BEG  END
#   NAME          P  LEN  $  M SPECIAL     V D D SIZE WORD WORD

  1 NAME          A  20                    3 Y Y  20B    0    9
  2 NUMBER        I   5                    3 Y Y   1W   10   10
  3 ADDRESS       A  26                    3 Y Y  26B   11   23
  4 CITY          A  14                    3 Y Y  14B   24   30
  5 ST            A   2                    3 Y Y   2B   31   31
  6 PHONE         A   8                    3 Y Y   8B   32   35
  7 STARTDATE     D   8        M/D/Y       3 Y Y   2W   36   37

PRINT LINE WIDTH = 95 CHARACTERS.
)
```

```
ALLOW functionlist
   [BY userlist]
   [IN grouplist]
   [;DEFAULT]
```

Creates or adds to a capability matrix that indicates what operations a user can perform in a particular group.

functionlist	Required. Specifies the functions that can be performed by the given users in the given groups. The following functions may be allowed:

ALL	Allow all functions.
SF-PERMANENT	Create permanent or temporary files.
SF-TEMPORARY	Create temporary files.
IA-CI	Interactive Command Interpreter access.
IA-PROG	Interactive programmatic access.
BA-CI	Batch Command Interpreter access.
BA-PROG	Batch programmatic access.

BY userlist	Optional. Specifies the user names to which operations will be granted. If the BY keyword is used, the list must contain one or more user names separated by commas. If the keyword is not used, an atsign ("@") is assumed.
IN grouplist	Optional. Specifies the groups in which user can perform the given functions. If the IN keyword is used, the list must contain one or more group names separated by commas. If not used, an atsign ("@") is assumed.
DEFAULT	Optional. If specified, each user-name/group-name combination listed is adjusted so that their access to the specified functions matches the general case ("@") for their group/user.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group.

When functional restrictions are applied, the following search order is used:

1) An entry with a matching user name and group name is searched for.

2) An entry with an atsign as the group name is searched for.

3) An entry with an atsign as the user's name is searched for.

4) An entry with an atsign as both a user name and a group name is searched for.

5) If a particular capability has not been ALLOWed, it is DISALLOWed.

The DISALLOW command can be used to remove capabilities that have previously been ALLOWED.

Command	RDBDD person	RDBDD allow
ALLOW RUN	@	Y
ALLOW RUN BY FRED	@	Y
	FRED	Y
DISALLOW RUN BY JOHN	@	Y
	FRED	Y
	JOHN	N
DISALLOW RUN BY FRED	@	Y
	FRED	N
	JOHN	N
ALLOW RUN BY FRED; DEFAULT	@	Y
	JOHN	N

EXAMPLES:

The command functions by searching for each user-name group-name combination in the RDBDD file. If a record is found, all functions specified are enabled. If a record is not found, a new record is added. If the BY or IN keywords are not supplied, they represent a user named "atsign" and a group named "atsign", respectivley. Thus, the following command would attempt to locate a record with a user group combination of "@", "DEV", and, failing to find such a record, would add a new record to the file.

```
)ALLOW ALL IN DEV
```

The system allows the DBA to create a default function list for each group or user. These default capabilities can then be overridden either positively or negatively (DISALLOW) on an as needed basis. For example:

```
)ALLOW SF-PERMANENT BY USER1 IN DEV1
)ALLOW SF-TEMPORARY,IA-CI BY USER1
```

After the first ALLOW command, USER1 can create permanent files in the DEV1 group. After the second ALLOW command, USER1 can create temporary files and access the Command Interpreter interactively in all groups but can still only create permanent files in the DEV1 group. If the DBA wants to allow USER1 to perform the other functions in DEV1, an ALLOW containing the DEFAULT keyword could be used to remove the first entry or another ALLOW could be used specifically for the functions in the group DEV1.

BEGIN TRANSACTION

Begins a logical transaction.

Once a transaction begins, all data modifications are saved in a holding file until the transaction completes. If the transaction is ABORTed then the changes are discarded. If the transaction is COMMITted the changes are posted to the appropriate files.

BEGIN TRANSACTION commands may be nested in order to allow sub-transactions within a larger transaction. This nesting allows a transaction to be checkpointed such that a correctable error in one part of the transaction does not force the entire transaction to be repeated. Each BEGIN TRANSACTION command must subsequently be followed by a COMMIT or an ABORT TRANSACTION command.

If a BEGIN TRANSACTION command has never been issued the transaction level is zero. For each BEGIN TRANSACTION command the level increases by one. For each COMMIT command the level decreases by one. When the level drops from one to zero the files taking part in the transaction are updated. If an ABORT TRANSACTION command is entered all changes made at the current transaction level (or deeper levels embedded in the current level) are ignored and the transaction level is decreased by one.

For more information on transactions see the Transaction Processing section.

EXAMPLES:

The BEGIN TRANSACTION command informs RELATE to place all file update requests into a logging file until the issuer can be sure of the validity of the complete transaction. Once the validity of the transaction is determined it can be COMMITed (which makes the changes a part of the data base) or ABORTed (which causes the changes to be ignored).

When the BEGIN TRANSACTION command is issued, the transaction level is increased by one (1) and any pending locks are acquired.

```
)LOCK FILE CUSTTMP
)SHOW FILES

L
O
C
K  FILE NAME          DATABASE NAME

   CUSTTMP            CUSTTMP.DOC79 RDB
```

```
)BEGIN TRANSACTION
THE TRANSACTION LEVEL IS NOW 1.
)SHOW FILES

L
O
C
K FILE NAME           DATABASE NAME

  CUSTTMP             CUSTTMP.DOC79.RDB
```

File changes are now logged into a temporary holding area until the transaction completes.

```
)PRINT NAME

$LINE NAME

     1 HASLETREX INC.
     2 DEXMACH, INC.
     3 CUPERCO
     4 AMERICAN TIRE CO.
     5 FINCH, FINCH, & OTTO
     6 NATIONAL AIRLINES
     7 ALEXANDER HALE & CO.
     8 PERFECT SOUND

8 LINES PRINTED.
)3 DELETE
1 LINE DELETED.
```

A transaction can operate on several files (if the user can lock them at the same time) and can perform several operations on each file.

```
)ADD
ENTER NAME,NUMBER,ADDRESS,CITY,ST,PHONE

NAME? INTERIOR LIGHTING
NUMBER? 1100
ADDRESS? 6952 OAK ROAD,SANTA PAULA,CA
PHONE? 662-1259

NAME? //

)PRINT NAME

$LINE NAME

     1 HASLETREX INC.
     2 DEXMACH, INC.
     3 CUPERCO
     4 AMERICAN TIRE CO.
     5 FINCH, FINCH, & OTTO
```

```
        6 NATIONAL AIRLINES
        7 ALEXANDER HALE & CO.
        8 PERFECT SOUND

    8 LINES PRINTED.
```

When the transaction is COMMITed the changes are reflected in the data base.

```
    )COMMIT TRANSACTION
    ALL TRANSACTIONS HAVE BEEN POSTED.
    )PRINT NAME

    $LINE NAME

        1 HASLETREX INC.
        2 DEXMACH, INC.
        4 AMERICAN TIRE CO.
        5 FINCH, FINCH, & OTTO
        6 NATIONAL AIRLINES
        7 ALEXANDER HALE & CO.
        8 PERFECT SOUND
        9 INTERIOR LIGHTING

    8 LINES PRINTED
    )
```

BEGIN

[range] CHANGE [fieldspecs] [FOR condition]

Selectively modifies data in a file. The user need not have exclusive access to the file to use this command.

range	Optional. If used, only records in the specified range are made available for changes. See the RANGE section.
:I	Optional global switch. If a CHANGE command is used in a procedure, the changes are requested from the user's terminal ($STDINX) unless a global "I" switch is used. If the global "I" switch is used, data is obtained from the procedure file(s). The switch is ignored if a procedure is not executing.
:L	Optional global switch. Prints the line number regardless of the current key.
:S	Optional global switch. Suppresses the printing of the current key.
fieldspecs	An optional list of fields whose values are to be made available for changes. If not supplied, or all fields in the fieldlist contain a local "P" switch, all fields are available for changes.

Each fieldspec in the list is of the format:

```
field
    or
(field [:PROMPT="text"] [:DEFAULT=YES/NO])
```

PROMPT	If specified, the keyword must be followed by text enclosed in quotes. This text, rather than the fieldname, will be used as a prompt when a new value is requested.
DEFAULT	If specified, the keyword must be followed by either YES, indicating that the existing value of the field will appear as the default value in the prompt; or NO, for no default value. If DEFAULT is not specified, YES is assumed.

LOCALS	Optional switch which, if used, is appended to one or more of the field names in the fieldlist.
	:P Prints the value of the field and does not request a new value.
FOR condition	Optional. If used, only records satisfying the specified condition are made available for changes. See the EXPRESSION EVALUATION section.

The command functions by locating the first record to be changed and displaying the key value (if a global :S is not used) or the line number (if a global :L is used). The system then prompts for the new value of the first field. If a RETURN is entered, the field will retain its original value.

Entering "//" returns the user to the command mode, ignoring any changes made to the current line and terminating the change command.

For a RELATE file, as each field is entered, the index structure is checked to determine if the current field is the last field required to complete a unary key. If it is, the key is created and checked against the data file. If a duplicate key is found, an error occurs and the field is re-prompted.

If an alphabetic field needs to be blanked out (set to spaces), enter a space and then RETURN.

EXAMPLES:

A file must be current in order for its data to be changed. List the current data. Generate change prompts for customers beginning with the letters "D" through "M" who operate out of the state of "CA". The local "P" switch on the CITY field will force it to print (along with NAME which is the current key) even though we only want to change the ADDRESS field.

```
)SET PATH CUST
)SET INDEX NAME
INDEX #2 IS NOW THE CURRENT INDEX.
)PRINT NAME, ADDRESS, CITY, ST

NAME                    ADDRESS                         CITY             ST

ALEXANDER HALE & CO.    83A SAN PEDRO                   ATLANTIC CITY    NJ
AMERICAN TIRE CO.       7052 EL CAMINO REAL             MOUNTAIN VIEW    CA
CUPERCO                 10802 WILKINSON AVENUE          CUPERTINO        CA
DEXMACH, INC.           PO BOX 1567                     SAN JOSE         CA
FINCH, FINCH, & OTTO    87 NORTH FIRST, SUITE 243C      LOS ANGELES      CA
HASLETREX INC.          89 BEST WAY                     SUNNYVALE        CA
NATIONAL AIRLINES       SAN FRANCISCO INTL AIRPORT      BURLINGAME       CA
PERFECT SOUND           415 FAIR OAKS AVENUE            LAWRENCE         VA

8 LINES PRINTED.
)"D"/"M" CHANGE CITY:P, ADDRESS FOR ST="CA"
DEXMACH, INC.           SAN JOSE
ADDRESS [PO BOX 1567]? BOX 877 RD1
FINCH, FINCH, & OTTO LOS ANGELES
ADDRESS [87 NORTH FIRST, SUITE 243C]?
HASLETREX INC.          SUNNYVALE
ADDRESS [89 BEST WAY]?
```

```
)"D"/"M" PRINT NAME, ADDRESS, CITY, ST

NAME                 ADDRESS                    CITY          ST

DEXMACH, INC.        BOX 877 RD1                SAN JOSE      CA
FINCH, FINCH, & OTTO 87 NORTH FIRST, SUITE 243C LOS ANGELES   CA
HASLETREX INC.       89 BEST WAY                SUNNYVALE     CA

3 LINES PRINTED.
```

When requesting changes from a procedure file, the ability to specify a more descriptive prompt than the fieldname is especially useful, as is the ability to suppress the printing of the current value of the field (the default).

```
)"AA"/"AZ" CHANGE (NUMBER; PROMPT="NEW CUSTOMER #"),&
&)        (PHONE;DEFAULT=NO)
ALEXANDER HALE & CO.
NEW CUSTOMER # [700]?
PHONE?
AMERICAN TIRE CO.
NEW CUSTOMER # [500]?
PHONE?
)
```

```
CLOSE
[DATABASE database]
[FILE filename[;DATABASE=databasename]]
[PATH pathname]
```

Closes paths, files, or databases so that further access is not allowed. If no parameters are supplied, all paths, files, and databases are closed.

DATABASE	Optional. If used, this keyword should be followed by the name of the database to be closed. Before closing the database, the system will close any files in the database and any paths to those files. If a CLOSE DATABASE is issued from the Host Language Interface routines, the database may not actually be closed if other cursors have files open in the database.
FILE	Optional. If used, this keyword should be followed by the name of the file or set to be closed. If any paths are open on the file, these are closed first. If a CLOSE FILE is issued from the Host Language Interface routines, the file will no longer be accessible from the passed cursor but may not physically be closed if it is referenced by some other cursor. If an IMAGE dataset is being closed, the DATABASE in which the set resides must be specified.
PATH	Optional. Indicates that the path with the following name should be closed. If the path closed is the last path on a file, the file is also closed.

The CLOSE command will cause any pending SELECT command to be cancelled. If the CLOSE command is issued from the Host Language Interface routines, only paths and files associated with the current cursor are closed.

A file cannot be closed while a transaction is in progress.

EXAMPLES:

Open files are closed automatically by RELATE when RELATE is exited. However, it may be necessary or desirable to close a file while still in RELATE. If a CLOSE PATH is executed and more than one path exists for that file only the indicated path will be closed. If, however, a CLOSE FILE is executed while more than one path is open for that file, all paths for that file will be closed.

```
)SHOW PATH

PATH NAME         FILE NAME          DATABASE NAME

DATEMASTER        DATE-MASTER        INVDB.DOC79.RDB (CURRENT PATH)
```

```
INV                 INVOICE             INVOICE.DOC79.RDB
INVOICE             INVOICE             INVOICE.DOC79.RDB
C                   CUST                CUST.DOC79.RDB
CUST                CUST                CUST.DOC79.RDB
CUST1               CUST1               CUST1.DOC79.RDB
MPECUST             MPECUST             MPECUST.DOC79.RDB

)CLOSE PATH CUST
)SHOW PATH

PATH NAME           FILE NAME           DATABASE NAME

DATEMASTER          DATE-MASTER         INVDB.DOC79.RDB (CURRENT PATH)
INV                 INVOICE             INVOICE.DOC79.RDB
INVOICE             INVOICE             INVOICE.DOC79.RDB
C                   CUST                CUST.DOC79.RDB
CUST1               CUST1               CUST1.DOC79.RDB
MPECUST             MPECUST             MPECUST.DOC79.RDB

)CLOSE FILE INVOICE
)SHOW PATH

PATH NAME           FILE NAME           DATABASE NAME

DATEMASTER          DATE-MASTER         INVDB.DOC79.RDB (CURRENT PATH)
C                   CUST                CUST.DOC79.RDB
CUST1               CUST1               CUST1.DOC79.RDB
MPECUST             MPECUST             MPECUST.DOC79.RDB

)
```

CLOSE RDBLIST

Completes a set of multiple outputs to the file RDBLIST.

Causes all output directed to the file RDBLIST since the previous OPEN RDBLIST command to print to RDBLIST (usually the printer).

After this command has been executed, all additional output to RDBLIST will be handled as it was before OPEN RDBLIST was executed.

EXAMPLES:

Normally, output to the printer is spooled immediately and a message is displayed that it has been done so. When RDBLIST is OPENed, however, output is grouped and not spooled until RDBLIST is CLOSEd.

```
)OPEN FILE CUST
)PRINT:P
THE OUTPUT HAS BEEN PLACED IN SPOOL FILE #090.
8 LINES PRINTED.
)OPEN RDBLIST
)PRINT:P
8 LINES PRINTED.
)OPEN FILE INVOICE
)PRINT:P
9 LINES PRINTED.
)CLOSE RDBLIST
THE OUTPUT HAS BEEN PLACED IN SPOOL FILE #091
)
```

```
CLOSE RDBLIST
```

COMMIT TRANSACTION

Forces the changes made during the current transaction level to be saved in the holding file and decreases the transaction level by one.

If the transaction level decreases from one to zero all file changes that have been COMMITted will be made to the data file.

EXAMPLES:

The COMMIT TRANSACTION command informs RELATE to make any pending changes to the data base permanent if the transaction level is one (1). If the transaction level is higher, that portion of the transaction is sealed off.

```
)BEGIN TRANSACTION
THE TRANSACTION LEVEL IS NOW 1.
)PRINT NAME

$LINE NAME

    1 HASLETREX INC.
    2 DEXMACH, INC.
    3 CUPERCO
    4 AMERICAN TIRE CO
    5 FINCH, FINCH, & OTTO
    6 NATIONAL AIRLINES
    7 ALEXANDER HALE & CO.
    8 PERFECT SOUND

8 LINES PRINTED.
)3 DELETE
1 LINE DELETED.
```

Because a transaction is in progress, record three still exists in the data base. If the transaction is COMMITted at this point only record three would be affected. If this change represented some logically complete portion of a larger transaction a sub-transaction should now be started.

```
)BEGIN TRANSACTION
THE TRANSACTION LEVEL IS NOW 2.
)5 DELETE
1 LINE DELETED
```

If an error is made in the next section of the transaction it can safely be ABORTed without affecting the prior operations.

```
)ABORT TRANSACTION
TRANSACTION LEVEL 2 HAS BEEN ABORTED.
```

When the transaction is COMMITted only record three will be deleted.

```
)COMMIT TRANSACTION
ALL TRANSACTIONS HAVE BEEN POSTED.
)PRINT NAME

$LINE NAME

     1 HASLETREX INC.
     2 DEXMACH, INC.
     4 AMERICAN TIRE CO.
     5 FINCH, FINCH, & OTTO
     6 NATIONAL AIRLINES
     7 ALEXANDER HALE & CO.
     8 PERFECT SOUND

7 LINES PRINTED.
)
```

COMPARE fieldlist WITH filename1[;options] [MATCHES [TO] filename2[;options] [ERRORS [TO] filename3[;options]] [BY keylist]

Compares two RELATE or KSAM files by key and outputs a file containing matching records and/or a file containing invalid records.

fieldlist — Required. A list of fields to be compared. Any fields listed here, which do not have local switches, must exist in both the current file and the WITH file. These fields are compared when a matching key is found. The fields must be of the same data type and in the case of alphabetic fields the same size.

LOCALS — Optional switches to be used on one or more items in the fieldlist. All fields with local switches must exist in filename3.

:E — Will contain a generated error message or error number. The local "E" switch may appear on a field of any type. If it appears on an alphabetic field, a text error message is generated and placed in the field. If it is on a numeric field, a number indicating the error will be generated. A numeric error code greater than zero is the field number that caused the error. An "E" switch may appear on an alphabetic and a numeric field at the same time. The possible errors are detailed in the table below.

Numeric	Alphabetic
-2	NOT IN MASTER
-1	NOT IN WITH
>0	fieldname DIFFERS

:F — Will contain the file name or number that the error occurred on. The local "F" switch may appear on a field of any type. If it appears on an alphabetic field, the filename that contained the error is output. If it appears on a numeric field, the file number (1 for the current file and 3 for the "WITH" file) is output. The local "F" switch may appear on an alphabetic and a numeric field at the same time.

:I — Will contain the index number on which the error occurred. The local "I" switch may only be used on a numeric field. If either the current file or the "WITH" file is an MPE file, zero will be output in error records from that file.

:R — Will contain the record number in which the error occurred. The local "R" switch may only be used on a numeric field.

WITH filename1 — Required. File to which current file is compared. Filename1 must exist when the command is given. It must be a RELATE or KSAM file containing an index with at least either the fields in the current index or those in the "BY" clause. Any options listed in the OPEN command needed to access the file must be appended to the filename, whether or not the file is already open.

MATCHES TO filename2
Optional. File to which matching records are output. If specified and it does not exist, it will be created in the same format as the current file. It may be any type of file. Any options given in the "TO filename" section needed to access the file must be appended to the filename, whether or not the file is already open.

ERRORS TO filename3
Optional. The file to which non-matching records are output. If specified and it does not exist, it will be created in the same format as the current file. It may be any type of file. Any options given in the "TO filename" section needed to access the file must be appended to the filename, whether or not the file is already open.

BY keylist — Optional. An ordered set of the fields in the current index to be used as the key. A key value should not be duplicated within each of the files. The fields used as the key must be of the same data type and, in the case of an alphabetic, zoned, or packed field, the same size in both the current file and the "WITH" file.

The command functions by reading serially down both the current file and the "WITH" file. The keys are compared and, if unequal, the record with the smaller of the two keys is output to the "ERRORS TO" file. The file that contained the key in error is read again and the new keys are compared. When a matching pair of keys is found the fields in the fieldlist that don't contain any local switches are compared. If any differ, error records are output and both input files are read again. If all fields have the same value, the records are output to the "MATCHES TO" file and both input files are read again.

The values for fields in the "MATCHES TO" file and the "ERRORS TO" file are obtained from the current file, if the fieldnames match. Fields not contained in the current file are located in the "WITH" file. The fields not located in either file are set to zeroes or blanks depending on type.

Either a "MATCHES TO" file or an "ERRORS TO" file, or both, should be used, as there is no default for either. If neither is specified, no output will be produced.

The current file should be set to the index containing the fields that will be used as keys. If no BY clause is specified, all fields in the current index will be used. The BY clause can only specify a subset (starting with the most major key field) of these fields. Both files must contain sorted indexes. This command will not function on IMAGE datasets or MPE files. However, several SELECT command sequences may be used to obtain similar information on all file types.

EXAMPLES:

Compare the CUST file's NAME and ADDRESS to those in the file CUST1. Output matching records to a file called CUSTMAT which has not yet been created. Output mismatches to a file called CUSTERR which will also contain information on the error. The current file must have a current index other than the line number. The "WITH" file must have available to it an identical index, in this case, "NAME".

```
)CREATE FILE CUSTERR
ENTER FIELDNAME,TYPE,LENGTH[.DECIMALS]

  1? NAME,A,26
  2? ADDRESS,A,26
  3? CITY,A,14
  4? ERRNUM,I,3
  5? FILENA,A,8
  6? //
THE "CUSTERR" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
)SET PATH CUST1
)CREATE INDEX BY NAME
INDEX #1 HAS BEEN CREATED.
6 LINES INDEXED.
INDEX #1 IS NOW THE CURRENT INDEX.
)PRINT NAME, ADDRESS, CITY

NAME                    ADDRESS                   CITY

ALEXANDER HALE & CO.    83A SAN PEDRO             ATLANTIC CITY
AMERICAN TIRE CO.       7052 EL CAMINO REAL       MOUNTAIN VIEW
CUPERCO                 10802 WILKERSON AVENUE    CUPERTINO
DEXMACH, INC.           PO BOX 1568               SAN JOSE
HASLETREX, INC.         89 BEST WAY               GRUMPTON
NATIONAL AIRLINES       SAN FRANCISCO INTL AIRPORT SOUTH SF

6 LINES PRINTED.
)SET PATH CUST
)SET INDEX NAME
INDEX #2 IS NOW THE CURRENT INDEX.
```

```
)PRINT NAME, ADDRESS, CITY

NAME                    ADDRESS                      CITY

ALEXANDER HALE & CO.    83A SAN PEDRO                ATLANTIC CITY
AMERICAN TIRE CO.       7052 EL CAMINO REAL          MOUNTAIN VIEW
CUPERCO                 10802 WILKINSON AVENUE       CUPERTINO
DEXMACH, INC.           BOX 877 RD1                  SAN JOSE
FINCH, FINCH, & OTTO    87 NORTH FIRST, SUITE 243C   LOS ANGELES
HASLETREX INC.          89 BEST WAY                  SUNNYVALE
NATIONAL AIRLINES       SAN FRANCISCO INTL AIRPORT   BURLINGAME
PERFECT SOUND           415 FAIR OAKS AVENUE         LAWRENCE

8 LINES PRINTED.
)COMPARE NAME, ADDRESS, ERRNUM:E, FILENA:F WITH CUST1 &
&)MATCHES TO CUSTMAT ERRORS TO CUSTERR
THE "CUSTMAT" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
8 MASTER LINES READ.
6 WITH LINES READ.
6 VALID LINES OUTPUT.
8 ERROR LINES OUTPUT.
```

The CUSTMAT file contains ALL records which matched on the indicated comparison fields (NAME and ADDRESS) which means that there will be one record from each of the current and "WITH" files for each match found; eg, each record will be duplicated in NAME and ADDRESS although other fields may differ.

```
)OPEN FILE CUSTMAT
)PRINT NAME, ADDRESS

$LINE NAME                  ADDRESS

    1 ALEXANDER HALE & CO.  83A SAN PEDRO
    2 ALEXANDER HALE & CO.  83A SAN PEDRO
    3 AMERICAN TIRE CO.     7052 EL CAMINO REAL
    4 AMERICAN TIRE CO.     7052 EL CAMINO REAL
    5 NATIONAL AIRLINES     SAN FRANCISCO INTL AIRPORT
    6 NATIONAL AIRLINES     SAN FRANCISCO INTL AIRPORT

6 LINES PRINTED
```

The local "E" switch on the ERRNUM field will print the error number of the mismatch into ERRNUM. The local "F" switch prints the name of the file in which the error occurred.

```
)SET PATH CUSTERR
)PRINT:S NAME, ADDRESS, ERRNUM, FILENA

NAME                         ADDRESS                      ERR FILENA

CUPERCO                      10802 WILKINSON AVENUE         3 CUST
CUPERCO                      10802 WILKERSON AVENUE         3 CUST1
DEXMACH, INC.                BOX 877 RD1                    3 CUST
DEXMACH, INC.                PO BOX 1568                    3 CUST1
FINCH, FINCH, & OTTO         87 NORTH FIRST, SUITE 243C    -1 CUST
```

```
HASLETREX INC.              89 BEST WAY                       -1 CUST
HASLETREX, INC              89 BEST WAY                       -2 CUST1
PERFECT SOUND               415 FAIR OAKS AVENUE              -1 CUST

8 LINES PRINTED.
)
```

COMPILE CATALOG source INTO destination
[;WARN]
[;OLD=oldmaster]

Allows the user to create a message catalog in his own language.

source	Required. The name of an EDITOR file containing the source catalogue to be compiled. The file should contain records of 88 bytes. The file should be a numbered EDITOR file.
destination	Required. The name to give the file into which the compiled catalogue will be placed.
WARN	Optional. If specified, prints information about how the catalog is being compiled in cases that may not be standard. Errors will always be printed.
OLD	Required. If compiling a non-master catalog for any system other than RELATE (eg. accounting package catalogs), must be followed by the name of the current compiled Master Catalog for that system. If not specified, the currently open catalog will be used.

In order to set the language with the SYSTEM command, the user must add the language to his catalog, before compilation, in two steps. First, add the name of the language to SET 6, message zero (0). Message zero should look like this initially:

```
0B 1,"ENGLISH",0,-1
```

For each language that will have a catalog, add the language before the -1 as follows:

```
num,"language",0,
```

where "num" is a positive integer between 1 and 99, previously unused in message zero, and "language" is the name of the language enclosed in quotes. A catalog that could access English, German, and French, might have:

```
0B 1,"ENGLISH",0,2,"GERMAN",0,3,"FRENCH",0,-1
```

Then, add the name of the compiled catalog ("destination"), with group and account if needed, to SET 6. To ascertain what message number to give it, multiply "num" (from step 1) by 100. Hence:

```
$SET6
0B 1,"ENGLISH",0,2,"GERMAN",0,3,"FRENCH",0,-1
$ English catalog
100 RDBECAT.PUB.SYS
$ German catalog
200 RDBGCAT.PUB.SYS
$ French catalog
300 RDBFCAT.PUB.SYS
```

Translation procedures

When translating the catalog source file into another language, some rules must be followed.

1) Nothing enclosed in quotes should be translated.

2) Messages may be lengthened or shortened in the translation. Any additional lines of a message must be indented at least as far as the message number plus one space. For example:

```
3206      THIS IS THE FIRST LINE.
        NOTHING SHOULD BE FURTHER LEFT THAN THIS.
                  THIS LINE IS OK.
      THIS IS TOO FAR LEFT.
```

   If a continuation character ("&") appears at the end of a message line, the system will attempt to move words of the message around to fill each line based on the width of the output device.

3) Do not add any additional messages or remove any existing messages.

4) The source file must be a numbered 88-character width EDITOR file. In order to achieve this, when first entering the EDITOR to create the new source catalog, give the commands:

```
SET LENGTH=80
SET RIGHT=80
```

5) All lines with a $ in the first column are either:
   a) comments, which do not have to be translated, as the user will never see them, but should be included in the catalog for readability.

   b) SET numbers, which MUST be included in the new catalog.

Accessing the New Language

After the appropriate entries have been made to SET 6 of the catalog and the catalog has been compiled, use the $LANGUAGE parameter in the SYSTEM command in order to access the new language.

What the Catalog Is

The message catalog is a collection of data used by RELATE/3000. This data includes error messages, comments, and HELP messages, (which may be translated) and command names and device specifications (which cannot be translated). The catalog is divided into SETs of data, each set having its own purpose. A SET beginning is denoted by a $SETn line. A copy of this catalog is provided with RELATE as RDBCAT.

When the catalog is compiled, all of this data is packed into a format easily read by RELATE. The catalog is also given a directory based on set number and message number that enables RELATE to have virtually direct access so that a call for any data (e.g. an error message) takes almost no time. The compiled version provided with RELATE is called RDBECAT.

The catalog cannot be used by RELATE unless it has been compiled.

**[range] CONSOLIDATE [fieldlist]
TO filename[;options] [BY keylist]
[FOR condition]**

Creates a summary of the current path.

range	Optional. If specified, only records in the specified range are used in the consolidation. See the RANGE section.
:D	Optional global switch. If used, deletes each record used in the consolidation.
fieldlist	Optional list of fields upon which to perform operations. All fields in the fieldlist must exist in the output file. Fields in the output ("TO") file which are not included in the fieldlist and exist in the current file default to an "F" switch (first value). Groups of fields may be specified using the pattern-matching feature. No more than 127 fields may be requested for the consolidation.
LOCALS	Optional switches which, if used, are appended to items in the fieldlist. If a switch is not used on a field in the list, numeric fields default to a "T" (total) and alphabetic fields default to an "F" (first) switch. "T", "A", and "C" may not be used on an alphabetic field. If the average is requested, the sum is calculated as a long number and then divided by the number of records in the current key. The result is then converted to the destination field's data type.

:A	Averages the field. May not be used on an alphabetic field.
:C	Counts the number of records used. May not be used on an alphabetic field.
:F	Takes the first value.
:G	Takes the greatest (maximum) value.
:L	Takes the last value.
:S	Takes the smallest (minimum) value.
:T	Totals the field. May not be used on an alphabetic field.

TO filename	Required. The file to which the input is consolidated. Any options given in the "TO filename" section needed to access the file must be appended to the filename, whether or not the file is already open.

BY keylist	Optional. If the "BY keylist" clause is omitted, the current key will be used to determine when record breaks will occur. If the "BY keylist" clause is provided, all fields in the list must exist in the current data file and should also be in the current index (unless it is known that the file is sorted by those fields). The fields provided will determine when record breaks will occur.
FOR condition	Optional. If used, only records meeting the specified condition will be used in the consolidation. See the EXPRESSION EVALUATION section.

The CONSOLIDATE command will only function correctly on sorted indexes or files and will not work on IMAGE datasets. To generate a consolidation on IMAGE datasets, a SELECT command containing aggregates or a BY clause must be used.

EXAMPLES:

Consolidate the INVOICE file to a previously non-existent file called SUM. The local "C" switch counts the number of lines used. The local "F" switch takes the first value of the field. The local "T" switch totals the field. Since the BY NUMBER clause is included and the file is indexed by NUMBER (index #3) , record breaks in the SUM file are by NUMBER.

```
)SET PATH INVOICE
)SET INDEX 3
INDEX #3 IS NOW THE CURRENT INDEX.
)PRINT

NUMBER  INVNO NAME                     ST   AMOUNT     TAX SALES

   100     33 DEXMACH, INC.            CA $ 348.70   22.67    99
   100    105 DEXMACH, INC.            CA $  86.32    5.61    83
   100    106 DEXMACH, INC.            CA $  76.40    4.97    87
   400   2738 CUPERCO                  CA $ 948.60   61.66    36
   400  10044 CUPERCO                  CA $ 500.00   32.50    99
   400  23557 CUPERCO                  CA $  37.00    2.40    86
   500    727 AMERICAN TIRE CO.        CA $   3.14    0.20    87
   500    747 AMERICAN TIRE CO.        CA $   0.97    0.06    83
   500   8663 AMERICAN TIRE CO         CA $  86.00    5.59    36
   700  10002 ALEXANDER HALE & CO.     NJ $ 999.99   90 00    45
   800     33 PERFECT SOUND            VA $ 677.77    0 00    83
  1000  10221 NATIONAL AIRLINES        CA $ 627.01   40.76    36
  1000  10455 NATIONAL AIRLINES        CA $ 335.00   21.77    36

13 LINES PRINTED.
)CONSOLIDATE INVNO:C,NUMBER:F,NAME:F,AMOUNT:T,TAX:T TO SUM &
&)BY NUMBER
THE "SUM" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
13 LINES INPUT.
  LINES OUTPUT.
```

```
)OPEN FILE SUM
)PRINT INVNO,NAME,NUMBER,AMOUNT,TAX

$LINE  INVNO NAME                  NUMBER    AMOUNT    TAX

  1        3 DEXMACH, INC.            100 $ 511.42   33.24
  2        3 CUPERCO                  400 $1485.60   96.56
  3        3 AMERICAN TIRE CO.        500 $  90.11    5.86
  4        1 ALEXANDER HALE & CO.     700 $ 999.99   90.00
  5        1 PERFECT SOUND            800 $ 677.77    0.00
  6        2 NATIONAL AIRLINES       1000 $ 962.01   62.53

6 LINES PRINTED.
)
```

[range] COPY [assignment[,...]] TO filename[;options] [FOR condition]

Copies information from the current path to the TO file.

range	Optional. If used, only records in the specified range are copied. See the RANGE section.
:D	Optional global switch. Deletes each record from the current file as it is copied. An assignment cannot be made to any fields in the current path, if this switch is used.
assignment	Optional. Assignments may contain fields from either the current path or the output ("TO") file. Fields in the output file must be qualified by the name of the path as specified with the TO file options. If any assignments are given, they are evaluated after the input record has been converted to the format of the output record.
TO filename	Required. The file into which the information is copied. Any options given in the "TO filename" section needed to access the file must be appended to the filename, whether or not it is already open.
FOR condition	Optional. If used, only records meeting the specified condition are copied. See the EXPRESSION EVALUATION section.

The command functions by reading records from the current file, copying information from fields with identical names from the input to the output, and then evaluating the assignments. Any type conversions required are done automatically. Any fields that exist in the output but not in the input are set to zeroes or blanks depending on type.

If any of the assignments are made to a field in the current path, each record is rewritten after the assignments are evaluated.

EXAMPLES:

Copy all fields that exist in both CUST and MPECUST from CUST to MPECUST. The NUMBER field should have 2000 added to it in the MPECUST file.

```
)SET PATH CUST
)PRINT NAME, NUMBER, ST

$LINE NAME                   NUMBER ST

    1 HASLETREX INC.            200 CA
    2 DEXMACH, INC              100 CA
    3 CUPERCO                   400 CA
    4 AMERICAN TIRE CO.         500 CA
    5 FINCH, FINCH, & OTTO      600 CA
    6 NATIONAL AIRLINES        1000 CA
```

```
      7 ALEXANDER HALE & CO.      700 NJ
      8 PERFECT SOUND             800 VA

 8 LINES PRINTED.
)COPY MPECUST.NUMBER=NUMBER+2000 TO MPECUST; TYPE=MPE; &
&)DOMAIN=TEMPORARY; FIELDS=(NAME,A,26),(NUMBER,I,6),&
&)(ADDRESS,A,26),(CITY,A,14),(DATE_UPD,R,8.0)
8 LINES COPIED
)SET PATH MPECUST
)PRINT NAME, NUMBER

$LINE NAME                      NUMBER

      1 HASLETREX INC.           2200
      2 DEXMACH, INC.            2100
      3 CUPERCO                  2400
      4 AMERICAN TIRE CO.        2500
      5 FINCH, FINCH, & OTTO     2600
      6 NATIONAL AIRLINES        3000
      7 ALEXANDER HALE & CO.     2700
      8 PERFECT SOUND            2800

 8 LINES PRINTED.
```

Now make a second copy of records whose state is "CA". Since the "TO" file was not erased, records are added to the file.

```
)SET PATH CUST
)COPY TO MPECUST FOR ST="CA"
6 LINES COPIED.
)SET PATH MPECUST
)PRINT NAME, NUMBER

$LINE NAME                      NUMBER

      1 HASLETREX INC.           2200
      2 DEXMACH, INC.            2100
      3 CUPERCO                  2400
      4 AMERICAN TIRE CO.        2500
      5 FINCH, FINCH, & OTTO     2600
      6 NATIONAL AIRLINES        3000
      7 ALEXANDER HALE & CO.     2700
      8 PERFECT SOUND            2800
      9 HASLETREX INC.            200
     10 DEXMACH, INC.             100
     11 CUPERCO                   400
     12 AMERICAN TIRE CO.         500
     13 FINCH, FINCH, & OTTO      600
     14 NATIONAL AIRLINES        1000

14 LINES PRINTED.
)
```

CREATE DICTIONARY

Creates the data dictionary used to store the security and view information used in a secure environment by RELATE/3000.

This command can only be executed by an account librarian executing in the PUB group. It must be done before the CREATE:D VIEW, PURGE:D VIEW, ALLOW, DISALLOW, ENABLE, DISABLE, PERMIT, or DENY commands will function.

The command creates a file called RDBDD with the following format:

FIELD	TYPE	SIZE	CONTENTS
TYPE	ALPHABETIC	12	Contains the type of an entry.
ITEM	ALPHABETIC	18	Contains a specific occurrence name for the TYPE.
QUAL	ALPHABETIC	18	Qualifies the ITEM to a specific case.
SEQ	INTEGER	3	A sequence number for entries that comprise more than one record.
DATA	ALPHABETIC	40	Contains the security information.

In order for the RELATE security system to function the account librarian must enable lock access to the PUB group of the account for any account users with the ALTGROUP command.

EXAMPLES:

Create the data dictionary for the account. The dictionary need only be created once.

```
)CREATE DICTIONARY
THE DATA DICTIONARY ("RDBDD") HAS BEEN CREATED.
)
```

```
        CREATE FILE filename [,keyfilename]
[;TYPE=RELATE|MPE|KSAM]
[;STRUCTURE=pathname]
[;RECORDS=recordcount]
[;RETENTION=PERMANENT|TEMPORARY|NONE]
[;CODE=filecode]
[;PATH=pathname]
[;FIELDS=fieldnamelist]
[;INDEXES=indexlist]
[;PRIVILEGED]
```

Explicitly creates a new RELATE/3000, KSAM/3000, or MPE file. Many of the commands in the Command Interpreter may also create new files as a result of the operation of the command.

:I	Optional global switch. If the CREATE FILE command is used in a procedure, data is requested from the user's terminal ($STDINX) unless a global "I" switch is used. If the global "I" switch is used, information on fields in the file will be taken from the prodedure file. The switch is ignored if a procedure file is not executing.
filename	Required. This is the name of the file to be created. The filename may include a lockword. Files may only be created in the user's log-on account. Filenames that duplicate RELATE keywords (such as TO, ERRORS, WITH, USING, and BY) should be avoided.
keyfilename	Required when a KSAM file is created. The parameter cannot be specified when a RELATE or MPE file is created. The file may not presently exist.
TYPE	Optional. If specified, a file of the given type will be created. By default, a RELATE file is created.
STRUCTURE	Optional. If specified, and the FIELDS keyword is not specified, the new file is created with the same format as the file referenced by the pathname. If the FIELDS keyword is specified, the format of the file is assumed to be that of the current file (if one exists), and STRUCTURE needs not be specified. If CREATE FILE is issued in the Host Language Interface routines, the pathname must exist in the passed cursor.
RECORDS	Optional. If specified, the new file will be created with room for the specified number of records. By default, a new file has room for 4096 data records. If a RELATE file is created, the record count is logically maintained by RELATE and will not correspond to the file limit as given in MPE. There is no practical limit to the number of records in a RELATE file. Extents are allocated only as the addition of data requires them so the file can be given a large limit without using excess disc space.

RETENTION

Optional. If specified, it determines the domain into which the file is saved after it is created. By default, a new file is saved permanently. The user may specify that the file should be saved temporarily or not at all. Temporary files are purged by MPE when the user logs off. Files with a retention of NONE are purged when the user closes the file. A file with the same name must not currently exist in the given domain.

CODE

Optional. This keyword can only be specified if a KSAM or an MPE file is being created. The file created will be given this MPE file code. Codes reserved by HP (those greater than 1024) and any code in the 620 to 639 range (these are used by RELATE) should be avoided.

PATH

Optional. If specified, the newly created file will be accessed through this path name. If not specified, the path name will be the filename excluding the group and account name. Each path name in the Command Interpreter must be unique. If CREATE FILE is called from the Host Language Interface routines, the path name must be unique in the passed cursor. The file cannot be created if the assumed or given path name duplicates an existing name.

FIELDS

Optional. If specified, the fieldnamelist must be of the format given in the SPECIFYING FIELDNAMES section. If the STRUCTURE of the current file is being used, groups of fields may be specified with the pattern-matching feature. In addition to the standard pattern matching, a minus sign ("-") may be included as the first character to indicate NOT fitting this pattern. If **not specified, the user is prompted for the fields that should be included in the file. One fieldname and its format should be specified in response to each prompt.** After all fields have been entered, a "//" will create the file. If the CREATE command is executed from the Host Language Interface routines or a KSAM file is being created, the STRUCTURE or the FIELDS keyword (or both) must be given.

INDEXES

Required if a KSAM file is created. Optional when a RELATE file is created. The keyword is ignored when an MPE file is created. The keyword must be followed by one or more index specifications (separated by commas) in the following formats:

```
        (fieldname [,...] [;UNARY])
                 number
                   @
```

If a fieldname is given the name must exist in the new file. If a number or an atsign is used, the STRUCTURE keyword must have been specified. When a number is given an index containing the fields from the index of the same number in the structure file is created. If the atsign is used, all indexes from the structure file are created.

PRIVILEGED

Optional. This keyword can only be specified if a RELATE file is being created by a user with Account Librarian (AL) capability. It is ignored for MPE and KSAM files. A file created as a PRIVILEGED file can only be accessed through RELATE.

A file cannot be created while a transaction is in progress.

Once the file has been created, the MODIFY FILE command can be used to alter defaults for data COMPRESSION, CRASH PROOFING, primary key CLUSTERing, and logical or physical DELETES.

SPECIFYING FIELDNAMES

Each item in the fieldnamelist must be in one of the following two formats:

```
[(]fieldname[,type,printlength][;options][)]
@
```

Each item describes a single field except for the atsign("@"). If the atsign is used the STRUCTURE keyword must have been specified. The atsign includes all fields from the structure file that have not yet been specified in the new file. Multiple fields may be specified, if separated by commas. A maximum of 127 fields, totaling no more than 512 words, may exist in any one file.

fieldname	Required. A fieldname must be between 1 and 10 characters in length, contain only capital letters, digits, or underscores ("_") and start with a letter. Fieldnames must be unique within a file. Keywords such as TO, ERRORS, WITH, USING, and BY should not be used as fieldnames. If the STRUCTURE keyword is used, only the name of the field need be specified. In this case, the name must not be in parenthesis and the type, printlength, and options are obtained from the existing field definition. If the STRUCTURE keyword is used, the special fieldname "@" may be used to obtain all fields from the structure file that are not yet in the new file.
type	Optional. The type of the field. The type must be one of the following: ALPHABETIC DOUBLE INTEGER LONG PACKED REAL UNSIGNED ZONED
printlength	Required if the type was specified. The printlength determines the printing width of the field. For ALPHABETIC, PACKED, and ZONED fields, it also determines the number of words stored in the file. The limitations on the size of each field appears in the FIELD SIZE LIMITATIONS table. The printlength is of the format **total length[.decimals]**, where **total length** is the total print length of the field, including the sign and decimal point, and **decimals** is the number of decimal positions.
options	Optional. Any options described in the FIELD OPTIONS table on the following pages may be specified.

FIELD SIZE LIMITATIONS

The table below summarizes the data types available in RELATE/3000. See the Data Types Interface description in Section 3 for comparisons with other data types.

TYPE	MAXIMUM	DECIMALS	COMMENTS
ALPHABETIC	250	NO	The system allocates two characters per word. If an odd number of characters is requested, an extra blank is added by the system.
DOUBLE	250	NO	A double may contain whole number values from -2147483648 to 2147483647. Two words are used to store a double integer; the leftmost bit of the first word is 1 for negatives.
INTEGER	250	NO	An integer may contain whole number values from -32767 to 32767. One word is used to store an integer number; the leftmost bit is 1 for negatives.
LONG	250	YES	A long field will maintain up to 16 digits of accuracy. Four words are used to store a long value with the format: one sign bit and a nine-bit exponent followed by a 54-bit mantissa.
PACKED	28	YES	A packed decimal field will maintain up to 28 digits of accuracy. Four digits are stored per word. An additional digit position is required for the sign. If the printsize requested plus one (for the sign) is not a multiple of four, an extra word is allocated. All packed numbers are signed.
REAL	250	YES	A real field will maintain up to 7 significant digits of accuracy. Two words are used to store a real value with the format: one sign bit and a nine-bit exponent followed by a 22-bit mantissa.
UNSIGNED	250	NO	An unsigned field may contain whole number values from 0 through 65535. One word is used to store an unsigned quantity.
ZONED	28	YES	A zoned decimal field will maintain up to 28 digits of accuracy. Two digits are stored per word. If an odd number of digits is requested, an extra word is allocated. All zoned numbers are signed.

FIELD OPTIONS

Additional field options include:

```
[;FORMAT=formatnumber|"datestring"|UPPERCASE|LOWERCASE]
[;INTERNAL=internal#]
[;LEVEL=entrylevel]
[;DOLLAR=FIXED|FLOAT|NONE]
[;COMMAS=YES|NO]
[;NAME=fieldname]
[;TYPE=type]
[;SIZE=printsize]
```

FORMAT	If specified, determines the printed appearance of the field. A format number cannot be specified for ALPHABETIC fields.
formatnumber	0=leading sign, only minus prints (default). Cancels any existing FORMAT. 1=leading sign, forced. 2=trailing sign, only minus prints. 3=trailing sign, forced. 4="( )" around negative numbers. 5=CR for negative numbers. 6=CR for negative, DR for positive numbers.
datestring	Can only be specified for DOUBLE, UNSIGNED or REAL fields. The field cannot have DOLLAR signs, COMMAS, or decimal places in the printsize when the date type is assigned. Unsigned date fields are stored internally in the same format as system dates (bits 0-6 are the year, bits 7-15 are the Julian date). Real date fields are stored as YYMMDD. Double date fields are stored as: word (1).(1:12)=year with century, word (2).(0:4)=month, word (2).(4:8)=day. The print length of a field is automatically adjusted to fit the indicated date format. The datestring consists of some combination of the following characters:

C	Indicates the year position, including the century. Printed as CCCC.
D	Indicates day-of-the-month position. Printed as DD.
J	Indicates Julian date position. Printed as JJJ.
M	Indicates month position. Printed as MM.
N	Indicates the name of the month position. Printed as NNN.
Y	Indicates year position. Printed as YY.
Z	If used as the first character, leading zeroes will be forced on the following date element (M, D, or J).

The C and Y codes, M and N codes, and the M, D and J codes may not be used in the same date format.

If no D or J is provided in a date format, the day is assumed to be the first of the month.

A slash ("/"), dash ("-"), blank (" "), or a period (".") can be used to separate the types.

Once a field has been assigned a date format, all dates entered for that field must be valid in the format which has been assigned. All constants compared to dates should be enclosed in quotes. Date formatted fields should not be used arithmetically in expressions.

UPPERCASE/ LOWERCASE	The UPPERCASE and LOWERCASE keywords can only be specified for ALPHABETIC fields. UPPERCASE forces all information subsequently placed in the field to be shifted to uppercase. LOWERCASE leaves all data exactly as entered. The default is LOWERCASE.
INTERNAL	Sets the internal field number. This number must be between 0 and 32767, inclusive. Zero is the default. An internal number other than zero cannot be duplicated. The internal number can only be referenced from the Host Language Interface routines.
LEVEL	Determines the data entry level for this field (see ADD). This number must be 0, or between 2 and 9, inclusive. Three (3) is the default (see ADD).
DOLLAR	Determines whether or not a dollar sign should be printed. FIXED places the dollar sign in the left-most print position for the field. FLOAT places it immediately to the left of the first non-blank character. NONE, the default, specifies that no dollar sign should be printed. This keyword cannot be used on ALPHABETIC or date fields.
COMMAS	Determines whether or not commas should be used to format numbers (e.g., 23,489,327.84). NO is the default. This keyword cannot be used on ALPHABETIC or date fields.
NAME	Changes the fieldname. This keyword is only valid in the MODIFY command or the Host Language Interface BIND procedure.
TYPE	Changes the type of the field. This keyword is only valid in the Host Language Interface BIND procedure.
SIZE	Changes the printsize of the field. This keyword is only valid in the MODIFY command and the Host Language Interface BIND procedure.

EXAMPLES:

Create a RELATE/3000 file called SALES.

```
)CREATE FILE SALES; FIELDS=(NAME,A,20),(NUMBER,I,6),(SALES,I,5)
THE "SALES" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

Create a RELATE/3000 file called CUST.

```
)CREATE FILE CUST; TYPE=RELATE
ENTER FIELDNAME,TYPE,LENGTH[.DECIMALS]

   1? NAME,A,20
   2? NUMBER,I,6
   3? ADDRESS,A,26
   4? CITY,A,14
   5? ST,A,2
   6? PHONE,A,8
   7? //
THE "CUST" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

Create a temporary MPE file with the same structure as the CUST file without state or phone number and including a field called DATE_UPD.

```
)CREATE FILE MPECUST; TYPE=MPE; RETENTION=TEMP; STRUCTURE=CUST; &
&)FIELDS=NAME,NUMBER,ADDRESS,CITY,(DATE_UPD,R,8.0;FORMAT="MM/YY/DD")
THE "MPECUST" FILE HAS BEEN CREATED AS A TEMPORARY MPE FILE.
```

Create a RELATE file called INVOICE.

```
)CREATE FILE INVOICE
ENTER FIELDNAME,TYPE,LENGTH[.DECIMALS]

   1? INVNO,I,6
   2? NAME,A,20
   3? NUMBER,I,6
   4? ST,A,2
   5? AMOUNT,REAL,8.2;DOLLAR=FIXED
   6? TAX,R,6.2
   7? SALES_MAN,INT,5
   8? //
THE "INVOICE" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

Create a RELATE file called CUST1 with the same fields as CUST and a LOCATION field.

```
)CREATE FILE CUST1; STRUCTURE=CUST; FIELDS=@,(LOCATION,A,10)
THE "CUST1" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

Create a RELATE file called TMPCUST with retention=NONE containing fields from CUST that begin with the letter N (Name and Number). See the Pattern-matching section for further information.

```
)CREATE FILE TMPCUST; RETENTION=NONE; STRUCTURE=CUST; FIELDS=%N
THE "TMPCUST" FILE HAS BEEN CREATED AS AN OPEN TEMPORARY RELATE/3000
FILE.
)SHOW

FILE NAME              =TMPCUST.DOC79.RDB

            T
            Y  PRINT  INT
NAME        P   LEN  SIZE

NAME        A   20    20B
NUMBER      I    6     1W

PRINT LINE WIDTH = 33 CHARACTERS.
)
```

CREATE INDEX [number] BY fieldlist [;UNARY]

Creates a new index for the current file. The current file must be a RELATE/3000 file to which the user has exclusive access. If the file exists in a secured group, the user must be the creator to index it.

number	Optional. If the index number desired is not specified, the next available number is used. Zero is reserved for the line number. Up to thirty additional indexes, numbered 1 through 30, may be created for the file.
fieldlist	Required. A list of one or more fields by which the file should be indexed.
LOCALS	Optional switches appended to any fieldname in the fieldlist. :A Ascending field (default). :D Descending field.
UNARY	Optional. If specified, a key may not be dupicated in the index. Any attempt to add a record with a duplicate key will generate an error. If duplicate keys already exist, the unary index will not be created.

Each index may contain multiple fields and the fields need not be contiguous in the data record or of the same data type. An index can be a maximum of fifty words in length. A maximum of eight fields may be included in any one index. A field may exist in as many indexes as desired.

If the value of a field that exists in one or more indexes is changed, all appropriate indexes will be automatically updated to reflect the change.

Indexes may be created regardless of the amount of data in the current file. If a large file is being created it is more efficient to create the index structure after the file has been loaded.

EXAMPLES:

A file must be current to create an index. Index the INVOICE file by NAME and NUMBER. Create an index numbered 3 by NUMBER and INVNO. Show the indexes. Create a descending index by AMOUNT; if no index number is specified, the lowest available number is used.

```
)SET PATH INVOICE
)CREATE INDEX BY NAME, NUMBER
INDEX #1 HAS BEEN CREATED
13 LINES INDEXED
INDEX #1 IS NOW THE CURRENT INDEX
)CREATE INDEX 3 BY NUMBER, INVNO
```

```
INDEX #3 HAS BEEN CREATED.
13 LINES INDEXED.
INDEX #3 IS NOW THE CURRENT INDEX.
)SHOW INDEXES

FILE NAME             =INVOICE.DOC79.RDB

INDEX 1 BY NAME,NUMBER
  WORDS IN KEY (+NODE) : 13 (+1)
  DISTRIBUTION         : 2,2
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1
INDEX 3 BY NUMBER,INVNO (CURRENT INDEX)
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 2,1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1

)CREATE INDEX BY AMOUNT:D
INDEX #2 HAS BEEN CREATED.
13 LINES INDEXED.
INDEX #2 IS NOW THE CURRENT INDEX.
```

Create a unary index for customer NUMBER in the CUST file. If any duplicate NUMBERs exist, this would be disallowed. In the future, entry of any duplicates will be prohibited. Then create an index by NAME and print NAME and ADDRESS. (Note that NAME need not be specified as it is the current key and will print automatically.)

```
)SET PATH CUST
)CREATE INDEX BY NUMBER; UNARY
INDEX #1 HAS BEEN CREATED
8 LINES INDEXED.
INDEX #1 IS NOW THE CURRENT INDEX.
)CREATE INDEX BY NAME
INDEX #2 HAS BEEN CREATED.
8 LINES INDEXED.
INDEX #2 IS NOW THE CURRENT INDEX.
)PRINT ADDRESS

NAME                   ADDRESS

ALEXANDER HALE & CO.   83A SAN PEDRO
AMERICAN TIRE CO.      7052 EL CAMINO REAL
CUPERCO                10802 WILKINSON AVENUE
DEXMACH, INC.          PO BOX 1567
FINCH, FINCH, & OTTO   87 NORTH FIRST, SUITE 243C
HASLETREX INC.         89 BEST WAY
NATIONAL AIRLINES      SAN FRANCISCO INTL AIRPORT
PERFECT SOUND          415 FAIR OAKS AVENUE

8 LINES PRINTED.
)
```

CREATE VIEW viewname
viewcommands

Creates a view and stores it permanently.

:I	Optional global switch. If a CREATE VIEW command is used in a procedure, the view commands are requested from the user's terminal ($STINDX) unless a global "I" switch is used, view commands are obtained from the procedure file(s). The switch is ignored if a procedure is not executing.
:D	Optional global switch. If used, the VIEW is added to the RDBDD file. Otherwise, it is added to the log-on group as a file. This switch can only be specified by an account librarian executing in the PUB group.
viewname	Required. The name of the view. The name must be from 1 to 8 characters long, start with a letter, and contain only letters and digits. The name cannot duplicate the name of an existing view or file.
viewcommands	Required. A sequence of one or more OPEN commands followed by a SELECT command. The REDO command can be used to edit a viewcommand which caused an error. The REDO command does not become part of the view definition.

A view is a logical file created by a user or the DBA to simplify access to data. VIEWS can also be used to impose additional security restrictions on IMAGE datasets and to isolate users and applications from most changes to a database format.

A view is composed of one or more OPEN commands followed by a SELECT command. The OPEN commands are executed when the view is created. The SELECT command is checked for proper syntax and to ensure that the required path names exist. The files opened during the creation of the view are closed when the view is complete if the global D switch has not been specified.

Virtually no time is taken constructing a view since no data is accessed until a view is used. When a view is OPENed or CLOSEd, the files underlying the view are OPENed or CLOSEd. When the view is queried, the files that comprise the view are referenced.

When the DBA defines a view, any security provisions attached to the underlying files are ignored when the view is accessed. Thus, the DBA can allow a user access to different information than would normally be allowed by a user s security provisions. When a user creates a view the security conditions from the underlying files are enforced.

A view is a dynamic window on the database and is not a copy of its contents. Changes to the database immediately affect the contents of the view. To access data through a view the OPEN FILE command is used.

EXAMPLES:

Create a view using fields from the INVOICE and CUST files. We want to know the SALES_MAN, the customer NUMBER, the customer NAME and PHONE number, and the INVOICE number. We link the files using the customer NUMBER.

```
)CREATE VIEW ACTIVE
-OPEN FILE INVOICE; PATH=IN
-OPEN FILE CUST; PATH=CU
-SELECT IN.SALES_MAN, CU.NUMBER, CU.NAME, CU.PHONE, &
&- IN.INVNO UNIQUE BY NUMBER, INVNO &
&-  WHERE CU.NUMBER=IN.NUMBER
THE "ACTIVE" VIEW HAS BEEN CREATED.
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

ACTIVE             ACTIVE                  (CURRENT PATH)
CUST               CUST               CUST.DOC79.RDB
CUST1              CUST1              CUST1.DOC79.RDB

)SHOW CURRENT

SELECTION

   MAXIMUM RECORDS          =4096
   FILE TYPE                =SELECTION
   RECORDS CANNOT BE ADDED, UPDATED, OR DELETED.

)PRINT

NUMBE  INVNO SALES NAME                   PHONE

  100     33    99 DEXMACH, INC.          800-2111
  100    105    83 DEXMACH, INC.          800-2111
  100    106    87 DEXMACH, INC.          800-2111
  500    727    87 AMERICAN TIRE CO.      941-0000
  500    747    83 AMERICAN TIRE CO.      941-0000
  500   8663    36 AMERICAN TIRE CO.      941-0000
  700  10002    45 ALEXANDER HALE & CO.   712-1305
 1000  10221    36 NATIONAL AIRLINES      UNLISTED
 1000  10455    36 NATIONAL AIRLINES      UNLISTED

9 LINES PRINTED.
)
```

[range] DELETE [FOR condition]

Deletes records from the current file.

range	Optional. If used, only records in the specified range will be deleted. See the RANGE section.
FOR condition	Optional. If used, only records meeting the specified condition will be deleted. See the EXPRESSION EVALUATION section.

Either a range or a condition must be specified if executed interactively from the Command Interpreter. Both the range and condition may be left off when the command is executed through the Host Language Interface routines.

Records are not physically removed from a RELATE/3000 file during a delete unless the DELETE=PHYSICAL option has been specified with the MODIFY FILE command. Otherwise, records are flagged so that they are ignored in subsequent processing. These flagged records can be made usable again with the RECOVER command.

Records may not be deleted from MPE files.

EXAMPLES:

Delete records from the INVOICE file where NUMBER is between 500 and 1000 and TAX is less than one hundredth of the AMOUNT.

```
)SET PATH INVOICE
)SET INDEX 3
INDEX #3 IS NOW THE CURRENT INDEX.
)PRINT

NUMBER  INVNO NAME                   ST   AMOUNT     TAX SALES

   100     33 DEXMACH, INC.          CA $ 348.70   18.00    99
   100    105 DEXMACH, INC.          CA $  86.32    4.81    83
   100    106 DEXMACH, INC.          CA $  76.40    1.10    87
   400   2738 CUPERCO                CA $ 948.60   32.40    36
   400  10044 CUPERCO                CA $ 500.00   35.99    99
   400  23557 CUPERCO                CA $  37.00    3.00    86
   500    727 AMERICAN TIRE CO.      CA $   3.14    1.59    87
   500    747 AMERICAN TIRE CO.      CA $   0.97    0.00    83
   500   8663 AMERICAN TIRE CO.      CA $  86.00    1.32    36
   700  10002 ALEXANDER HALE & CO    NJ $ 999.99    9.99    45
   800     33 PERFECT SOUND          VA $ 677.77    3.33    83
  1000  10221 NATIONAL AIRLINES      CA $ 627.01   88.07    36
  1000  10455 NATIONAL AIRLINES      CA $ 335.00   40.20    36

13 LINES PRINTED.
```

```
)500/1000 DELETE FOR TAX<AMOUNT•.01
3 LINES DELETED.
)PRINT

NUMBER   INVNO NAME                   ST    AMOUNT     TAX SALES

   100      33 DEXMACH, INC.          CA $ 348.70   18.00    99
   100     105 DEXMACH, INC           CA $  86.32    4.81    83
   100     106 DEXMACH, INC.          CA $  76.40    1.10    87
   400    2738 CUPERCO                CA $ 948.60   32.40    36
   400   10044 CUPERCO                CA $ 500.00   35.99    99
   400   23557 CUPERCO                CA $  37.00    3.00    86
   500     727 AMERICAN TIRE CO.      CA $   3.14    1.59    87
   500    8663 AMERICAN TIRE CO.      CA $  86.00    1.32    36
  1000   10221 NATIONAL AIRLINES      CA $ 627.01   88.07    36
  1000   10455 NATIONAL AIRLINES      CA $ 335.00   40.20    36

10 LINES PRINTED.
)
```

DENY READ|DELETE|ADD|CHANGE|ALL ON file
[BY userlist]

Allows the DBA to revoke previously permitted file operations.

file	Required. The name of a RELATE, MPE, or KSAM file or the name of a view that exists in the log-on account.
BY userlist	Optional. If the BY keyword is used, this must be one or more user names separated by commas. If not included, an atsign ("@") is assumed.

The command can only be executed by an account librarian (a user with AL capability) executing in the PUB group.

If a DENY is issued for a file and user combination that does not exist, no error is given and the request is ignored.

The PERMIT command can be used to counteract the DENY command.

EXAMPLES:

The DENY command can be used in conjunction with the PERMIT command to create the desired security structure. For instance, to deny all users except MGR access to the file SECURE, both the PERMIT and DENY commands must be used. Note that if BY is not specified, the default is all ("@").

```
)DENY ALL ON SECURE
)PERMIT ALL ON SECURE BY MGR
```

In addition, the DENY command can be used to revoke previously PERMITted access. If specific users had been PERMITted certain access capabilities to a file, that user must be again specified in order to affect his access. For instance, the following command will have no effect, as ALL capabilities are already denied for all users, and the user MGR is not specified so his access is not changed. The second command would be needed to alter access capabilities for MGR.

```
)DENY CHANGE ON SECURE
)DENY CHANGE ON SECURE BY MGR
)
```

DISABLE DATA LOGGING
[IN grouplist]

Instructs RELATE to discontinue logging changes to RELATE files in the indicated groups.

IN grouplist	Optional. Specifies the groups in which logging will no longer be performed. When specified, this list must contain one or more group names separated by commas. If the keyword "IN" is not specified, logging in all groups will be discontinued.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

The command will not have any effect on any file previously opened.

For more detailed information on logging see the TRANSACTION PROCESSING section.

DISABLE EVENT LOGGING
[BY userlist]

Instructs RELATE to discontinue logging event data for the indicated users.

BY userlist	Optional. Specifies the users for which the events should no longer be logged. When specified, the userlist must contain one or more user names separated by commas. If the list is not provided the special items will be logged for all users.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

This command will not affect users currently running RELATE.

For more detailed information on logging see the TRANSACTION PROCESSING section.

DISABLE SECURITY
[IN grouplist]

Reinstates a non-secure environment in the groups specified. After the security system is disabled in a group, access to files in the group is no longer restricted in any way by RELATE.

IN grouplist	Optional. Specifies the groups in which security will be suspended. If the list is specified, it must contain one or more group names separated by commas. If the keyword is not specified, an atsign ("@") is assumed.

The command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

The command will not have any affect on users running in the groups until RELATE is restarted. The command takes effect the next time a particular group's security status is required.

Use of the DISABLE SECURITY command does not modify any lower level provisions issued by the ALLOW or PERMIT commands; it simply causes them to be ignored. This command can be used to suspend the provisions in a set of groups to allow file reorganization without regard to the capabilities ALLOWed individual users. After the reorganization has been completed, the ENABLE SECURITY command can be used to restore the secure environment.

EXAMPLES:

The DISABLE SECURITY command causes RELATE to ignore all security provisions created with the PERMIT, DENY, ALLOW, and DISALLOW commands. If no GROUP is specified, all ("@") are assumed to be DISABLEd. If a specific group was ENABLEd, its name must be specified to DISABLE it.

```
)DISABLE SECURITY
)DISABLE SECURITY IN THISONE
)
```

```
DISALLOW functions
       [BY userlist]
       [IN grouplist]
```

Removes capabilities from a matrix that indicates what operations a user can perform in a particular group.

functions — Required. Specifies the functions that can no longer be performed by the given users in the given groups. The following functions may be disallowed:

ALL	Disallows all functions.
SF-PERMANENT	Create permanent or temporary files.
SF-TEMPORARY	Create temporary files.
IA-CI	Interactive Command Interpreter access.
IA-PROG	Interactive programmatic access.
BA-CI	Batch Command Interpreter access.
BA-PROG	Batch programmatic access.

BY userlist — Optional. Specifies the user names for which the functions will be disallowed. If the BY keyword is used, the list must contain one or more user names separated by commas. If the keyword is not used, an atsign ("@") is assumed.

IN grouplist — Optional. Specifies the groups in which the users can no longer perform the given functions. If the IN keyword is used, the list must contain one or more group names separated by commas. If the keyword is not used, atsign ("@") is assumed.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group.

The record for each user-name group-name combination is searched for in the RDBDD.PUB file. If the record is not found, one is added that inhibits the functions specified. If the record is found, the record is changed so that all functions specified are disabled.

The DISALLOW command will not affect the users and groups specified until RELATE is restarted.

The ALLOW command can be used to reinstate capabilities which have been DISALLOWED.

EXAMPLES:

To ensure that a certain function will be available to a group or user, the ALLOW command is used.

```
    )ALLOW BA-PROG BY RADNOR
```

If GROUP is not specified, an atsign ("@") is assumed. The same is true for USER. In this example, batch programmatic access would be disallowed for everybody except for the User RADNOR, as the specified function capability for him overrides the general case.

```
    )DISALLOW BA-PROG
```

ENABLE DATA LOGGING [IN grouplist] TO logfile

Instructs RELATE to begin logging changes to permanent RELATE files in the indicated groups. The log can subsequently be used to recover files and changes to files after a system failure.

IN grouplist	Optional. Specifies the groups in which logging will be performed. If the list is specified it must contain one or more group names separated by commas. If the group list is not specified the system will log file changes in all groups.
TO logfile	Required. The name of the log file which should be used. The file must have previously been created using MPE commands.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

The command will not have any effect on users or files running in the specified groups until RELATE is restarted.

For more information on logging see the TRANSACTION PROCESSING section.

ENABLE EVENT LOGGING [OF events] [BY userlist] TO logfile

Instructs RELATE to begin logging the events specified for the users specified.

OF events	Optional. Specifies the special events which should be logged. If the items are specified the list must be composed of one or more of the following keywords:

COMMANDS	Log commands issued by the user or the user's application.
STARTUP	Logs initializations and termination of RELATE.
ACCESS	Logs the names of files and data bases which the user has accessed.

BY userlist	Optional. Specifies the users for which the events should be logged. The list must contain one or more user names separated by commas. If the list is not provided the special items will be logged for all users.
TO logfile	Required. The name of the log file which should be used. The file must have previously been created using MPE commands.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

The command will not have any effect on users currently running RELATE.

For more information on logging see the TRANSACTION PROCESSING section.

ENABLE SECURITY
[IN grouplist]

Creates a secure RELATE/3000 environment in the groups specified. After the security system has been enabled, access to all files in the group is restricted to users who have specifically been granted access.

IN grouplist	Optional. Specifies the groups in which a secure environment will be enforced. If the list is specified it must contain one or more group names separated by commas. If the keyword is not specified, atsign ("@") will be assumed.

This command can only be executed by an account librarian (a user with AL capability) executing in the PUB group. The RELATE dictionary must have previously been created.

The command will not have any affect on users running in the groups until RELATE is restarted. The command takes effect the next time a particular group's security status is required.

Because of security provisions within MPE, users who have not logged-on to the secured group or who do not have the secured group as their home group may be unable to access any information within the group. It may be necessary to alter the security provisions on the group to allow access to any user. Unfortunately, this then allows users who have a knowledge of the system to perform unauthorized operations (e.g., copying) with the files. This problem can be partially resolved by assigning different users different capabilities which can be screened by MPE.

The DISABLE SECURITY command will cancel the effect of ENABLE SECURITY.

EXAMPLES:

The ENABLE SECURITY command puts into effect all security provisions created with the PERMIT, DENY, ALLOW, and DISALLOW commands. If the ENABLE command is never executed, the data dictionary will never be checked and no security will be provided. The following command will ENABLE security in only the ONLYONE group.

```
)ENABLE SECURITY IN ONLYONE
)
```

END, EXIT, or //

Terminates access to RELATE/3000.

:C	Optional global switch. Prints the total CPU time used in this run.
:T	Optional global switch. Prints the connect time used since RELATE/3000 was run.

When END or EXIT is executed, the user is returned to the MPE executive where he can log off the system. If the command is executed in the Host Language Interface routines, all files are closed and all cursors are released.

When RELATE/3000 is executed in a job, the CPU time and connect time used are always printed when RELATE terminates.

ERASE FILE filename [;DATABASE=databasename]

Erases (deletes all of the records from) the indicated file. If the file is in a secure group, the user must be the creator of the file in order to erase it.

filename	The name of the file to be erased. The file must have an open path. An IMAGE master set can only be erased if no detail sets contain entries that reference the set. An MPE file can be erased even though records may not be deleted on an individual basis. A VIEW cannot be erased.
databasename	The database in which the file resides. This parameter must be specified to erase an IMAGE dataset.

The ERASE FILE command will cause any pending SELECT command to be cancelled.

A file cannot be erased while a transaction is in progress.

Once a file has been erased, the RECOVER command will no longer have any effect.

EXAMPLES:

Erase CUST1 and try printing it to verify that it is indeed empty.

```
)SET PATH CUST1
)PRINT NAME

$LINE NAME

    1 LSI AEROSPACE
    2 IOWA STORM DOOR CO.

2 LINES PRINTED.
)ERASE FILE CUST1
)PRINT NAME
NO LINES PRINTED.
)
```

EXECUTE filename
[;SHOW[=YES|NO|SAME]]

Executes RELATE/3000 commands from a file.

filename	Required. The file must exist in either the permanent or temporary domain, and must be a numbered or unnumbered ASCII MPE file. The file can be created with the HP EDITOR. The file should contain the RELATE/3000 commands to be executed. The command lines (not the entire command) should be a maximum of 250 characters.
SHOW	Optional. If specified, the keyword may be followed by YES, NO, or SAME. If YES is specified (or SHOW is not followed by a keyword) the commands will be shown as they are executed. If SAME is specified, SHOW will be set to the same status as the calling procedure file. If SHOW is not specified, NO is assumed.

EXECUTE commands may appear in procedure files. Files may be nested until memory space cannot be obtained for required buffers.

All informational and warning messages generated by RELATE/3000 are suppressed when a procedure is executing unless the SHOW option is used.

If an error occurs during a procedure, and the user did not request that the error be ignored, the error is reported and all procedure files are closed.

Commands that accept data from the user (ADD, CHANGE, CREATE FILE, and CREATE VIEW) will still request data at the standard input device unless these commands contain a global "I" switch.

Procedure files may also be invoked by enclosing them in braces ("{" and "}"). A procedure file may be invoked at any prompt. If the SHOW option is desired, it must appear inside the braces.

If a file called RDBIN exists in the user's log-on group, the commands in this file are executed before RELATE/3000 prompts the user for commands at the terminal.

A procedure file may be cancelled by entering a Control-Y.

Commands in procedure files may extend onto more than one line. As when using RELATE directly, each command line that will be continued on the next line must end with an ampersand ("&"). The HP EDITOR will allow the user to add a line with an ampersand if the last character on the line is a blank (eg., the last two characters are "& ").

Commands in a file may extend over up to 100 lines and contain up to 1500 characters. If the continuation lines are indented for readability, the spaces at the beginning of the line are included in the 1500-character count.

EXAMPLES:

Execute a procedure file containing RELATE commands. First, execute it using the EXECUTE statement and not showing the commands as they execute; then, execute it by enclosing it in braces and also show the commands as they execute.

```
):EDITOR
HP32201A.7.08 EDIT/3000  FRI, SEP 24, 1982, 10:36 AM
(C) HEWLETT-PACKARD CO. 1980
/TEXT DOIT
/LIST ALL
     1      NOTE PROCEDURE FILE BEGINS
     2      SHOW PATH
     3      NOTE PROCEDURE FILE ENDS
/END
)EXECUTE DOIT

PATH NAME           FILE NAME           DATABASE NAME

CUST                CUST                CUST.DOC79.RDB
CUST1               CUST1               CUST1.DOC79.RDB

){DOIT;SHOW}
)NOTE PROCEDURE FILE BEGINS
)SHOW PATH

PATH NAME           FILE NAME           DATABASE NAME

CUST                CUST                CUST.DOC79.RDB
CUST1               CUST1               CUST1.DOC79.RDB

)NOTE PROCEDURE FILE ENDS
)
```

FIX FILE filename|fileset [;CREATOR]

Converts files from the RELATE 4.4 format to the RELATE 4.5 format.

filename	Required if a fileset is not given. If specified, this must be the name of a file in the logon account. The filename may contain a lockword. The indicated file will be converted.
fileset	Required if a filename is not given. If specified, the fileset must be a valid MPE file template referencing files in the current account. All RELATE files in the template will be converted. Any file with a lockword will be skipped.
CREATOR	Optional. If specified, only files created by the current user will be converted. This keyword should be used when files converted in secured groups or application programs depend on specific creator names.

As each file is accessed, its name is displayed followed by the current MPE end of file and the name of the creator of the file. An asterisk ("*") is diplayed as successive tenths of the file are copied. As each index is created the index number is displayed. Finally, the sectors of storage used by the original file and the resulting file are displayed. Converted files generally use less disc space than the original if the original file had a wide record size, contained many records, or was indexed. If the original file was not indexed or the data could not be compressed significantly the new file will use more space than the original file.

EXAMPLES:

You may specify one file or a fileset to convert at one time. An asterisk is printed as each tenth of the file is converted, then the index numbers are displayed as they are converted.

```
)FIX FILE OLDCUST
                                   CREATOR                              OLD        NEW
FILENAME                   RECORDS NAME         PROGRESS           SECTORS    SECTORS
OLDCUST  .DOC79                253 DOC79        **********1             71         56
)
```

FIX FORMAT MAP=mapfile
;FROM=inputfile [;EBCDIC]
TO outputfile

The FIX command copies information from the input file to the output file and reformats each record according to the specifications in the mapfile. The purpose of the FIX command is to align and translate the data in existing tape, KSAM, or MPE files onto word boundaries.

MAP	Required.
mapfile	An editor file containing a description of the input file. One record should exist for each field or filler that exists in the input file. The format for each record should be:

[fieldname], datatype[, length[.decimals]]

fieldname	The fieldname should be the name of a field that exists in the output file. If the field does not exist a warning will be issued. If a fieldname is not given, the information occupying the space indicated by the data type and length will be ignored.
datatype	The datatype is the first letter of a RELATE/3000 data type.
length	The length should represent the number of bytes (or nibbles in the case of packed numbers) that comprise the input field. (This should ALWAYS be 2 for integer and unsigned, 4 for Double and Real, 8 for Long). If the data type is packed or zoned, the number of decimals should also be given. If not specified, the data type must be I, D, R, L, or U.

FROM	Required. Specifies the name of an existing MPE or KSAM file that will be reformatted. The file must contain fixed length records. The file may be a tape (serial) file.
EBCDIC	Optional. If specified, this causes alphabetic and zoned fields in the input to be translated from EBCDIC to ASCII.
TO	Required. Specifies the name of the file into which the inputfile will be translated. The file must already exist and may not be open in the current cursor.

EXAMPLES:

The FIX command copies information from a source file (which may be a tape file) into an existing RELATE file. The command can align fields that exist on byte boundaries, translate from EBCDIC to ASCII, skip filler, and do data type conversion.

To use the command an output file is required.

```
)CREATE FILE FIXOUT; RETENTION=TEMPORARY
ENTER FIELDNAME,TYPE,LENGTH[.DECIMALS]

   1? PART,A,4
   2? DESC,A,15
   3? QTY,I,4
   4? //
THE "FIXOUT" FILE HAS BEEN CREATED AS A TEMPORARY RELATE/3000
FILE.
)CLOSE FILE FIXOUT
```

A map file is required to inform RELATE of the relationship between the fields in the source file and those in the output file. The missing fieldname (in the second line of the map) indicates a filler item which should not exist in the output file.

```
:EDITOR

HP32201A.7.10 EDIT/30000   FRI, MAR 19, 1982, 11:03 AM
(C) HEWLETT-PACKARD CO. 1981
/ADD
     1       PART,A,3
     2       ,A,2
     3       DESC,A,12
     4       QTY,A,3
     5       //
  ...
/KEEP FIXMAP
/EXIT
```

The source file for this example is an unnumbered EDITOR file containing inventory information in an out of date format.

```
:EDITOR

HP32201A.7.10 EDIT/30000   FRI, MAR 19, 1982, 11:04 AM
(C) HEWLETT-PACKARD CO. 1981
/TEXT OLDINVEN
/LIST ALL
     1       10002VISE GRIPS   001
     2       10102HAMMER       003
     3       10302SCREWDRIVER  034
     4       10504LEVEL        023
/EXIT
```

HELP [commandname] [requests]
HELP ERROR errorrange

Displays information concerning RELATE/3000 commands or errors.

:F	Optional global switch. Form-feeds the output. The HELP command will attempt to prevent the output from overlapping page boundaries.
:P	Optional global switch. If used, output is directed to the file RDBLIST. If RDBLIST cannot be opened, the output is directed to the device class "LP". Also sets the global "F" switch.
commandname	Optional. A RELATE command about which information is desired. One or two keywords of the command name may be specified (eg: OPEN or OPEN FILE). If only one keyword is supplied and it is ambiguous, the user will be prompted for the second keyword.
requests	Optional. If a commandname is specified, one or more of the following may be requested in addition:

ALL	Enables SYNTAX, PURPOSE, KEYWORDS, EXAMPLES, and DESCRIPTION.
DESCRIPTION	Displays miscellaneous information about the command.
EXAMPLES	Displays examples of the command's use.
KEYWORDS	Displays explanations of the parameters in the command.
PURPOSE	Displays the purpose of the command.
SYNTAX	Displays the complete syntax of the command.

If a commandname is not specified, one of the following may be requested:

COMMANDS	Lists all RELATE/3000 command names.
FUNCTIONS	Displays the names of all functions and system-defined fields which can be used in RELATE.

ERROR	Optional. If used, the keyword must be followed by a RELATE error number or range of error numbers whose text should be displayed.

EXAMPLES:

The HELP command can be entered to obtain further information about user actions and command formats and functions.

```
)HELP
 FORMAT:
     HELP ERROR errornumberlist
               or
     HELP [commandname] [requests]

               where "requests" =
                   [SYNTAX]
                   [,PURPOSE]
                   [,KEYWORDS]
                   [,DESCRIPTION]
                   [,EXAMPLES]
                   [,ALL]
)HELP COMMANDS

ADD          ALLOW      BEGIN        CHANGE     CLOSE     COMMIT
COMPARE      COMPILE    CONSOLIDATE  COPY       CREATE    DELETE
DENY         DISABLE    DISALLOW     DRAW       ELSE      ENABLE
END          ENDIF      ERASE        EXECUTE    EXIT      FIX
HELP         IF         IGNORE       LABEL      LET       LIST
LOCK         MODIFY     NOTE         OPEN       PAUSE     PERMIT
PLOT         PRINT      PURGE        QUIZ       RECOVER   REDO
REORGANIZE   REPORT     SELECT       SET        SHOW      SORT
SUM          SYSTEM     TERMINAL     UNLOCK     UPDATE
```

If no requests are specified, SYNTAX is assumed.

```
)HELP ADD

                              ADD [SEPARATOR="character"]

OPTIONS: SYNTAX, PURPOSE, KEYWORDS, DESCRIPTION, EXAMPLES.
)HELP ADD PURPOSE

        Adds data to the current file

OPTIONS: SYNTAX, PURPOSE, KEYWORDS, DESCRIPTION, EXAMPLES
```

If the first keyword is ambiguous, the alternatives for the second keyword are displayed.

```
)HELP SET
OPTIONS: INDEX, PATH, DEFAULT, DEVICE, FRAME, SIZE, SPEED, UNITS,
WINDOW
)HELP SET PATH

                                   SET PATH pathname
```

```
OPTIONS: SYNTAX, PURPOSE, KEYWORDS, DESCRIPTION, EXAMPLES.
)HELP SET PATH KEYWORDS

                               previously been opened but might no
longer be the
          pathname             Required  This is the pathname of a file
that has
                               previously been opened but might no
longer be the
                               current file

OPTIONS: SYNTAX, PURPOSE, KEYWORDS, DESCRIPTION, EXAMPLES.
)HELP ERROR 1342/1345

1342 INVALID CHARACTER IN NUMBER.

1343 NUMBER TOO LARGE FOR ITS TYPE.

1344 NUMBER TOO LARGE FOR ITS FIELD.

1345 MISSING DIGIT(S) AFTER SIGN.

)
```

IF [condition]

Allows conditional execution of commands.

If a condition is not specified, the command attempts to read a record from the current path. If a condition is specified, the command can be interpreted as "IF there are any records where the condition is true". The condition may contain all items that are legal in the WHERE clause of the SELECT command. The command attempts to read a record from the path or file referenced by the condition, NOT from the current path or selection (if there is one). This may sometimes appear to give results inconsistent with what the user is trying to achieve.

If a record is found, execution of commands continues until an ELSE or ENDIF is found. If a record is not found, commands are ignored until an ELSE or ENDIF is found. IF commands may be nested.

ELSE

If a record was found which meets the IF conditions, commands between the ELSE and ENDIF are ignored. Otherwise, execution resumes with the command following the ELSE and continues until an ENDIF is found.

ENDIF

Marks the end of a set of commands to be executed conditionally. Commands following the ENDIF will be executed regardless of the result of the IF command. Exactly one ENDIF must exist for each IF.

EXAMPLES:

If there is a single value from some function that applies to the entire current path, you can evaluate that value as true or false.

```
)SET PATH CUST
)IGNORE ALL ERRORS
)COPY TO CUSTT; TYPE=MPE; RETENTION=NONE
THE "CUSTT" FILE HAS BEEN CREATED AS AN OPEN TEMPORARY MPE FILE.
8 LINES COPIED.
)IF $ERROR<>0
**** RECORD NOT FOUND. COMMAND EXECUTION SUSPENDED.
)   NOTE:D The CUSTT file couldn't be created.
)ENDIF
**** COMMAND EXECUTION RESUMED.
```

If you want to verify that your current path has at least one record in it, you can use the IF function with no parameters.

```
)SELECT @ WHERE ST="CA"
)IF
**** RECORD FOUND, COMMAND EXECUTION PROCEEDING.
)    COPY TO CALCUST; RETENTION=NONE
THE "CALCUST" FILE HAS BEEN CREATED AS AN OPEN TEMPORARY
RELATE/3000 FILE.
6 LINES COPIED.
)ELSE
**** COMMAND EXECUTION SUSPENDED.
)    NOTE:D There are no customers in California.
)ENDIF
**** COMMAND EXECUTION RESUMED.
```

The condition on the IF command has the effect of performing an internal SELECT with the condition in the WHERE clause, so any current SELECTion has no bearing on the evaluation of that condition.

```
)SELECT @ WHERE ST="NJ"
)PRINT

NAME                      NUMBE ADDRESS                          CITY
ST PHONE

ALEXANDER HALE & CO.        700 83A SAN PEDRO                    ATLANTIC
CITY   NJ 712-1305

1 LINE PRINTED.
)IF $MAX(NUMBER)>=1000
**** RECORD FOUND, COMMAND EXECUTION PROCEEDING.
)NOTE THIS HAS EVALUATED THE ENTIRE CUST FILE, NOT THE
    CURRENT SELECTION.
)ENDIF
)
```

IGNORE [ALL] ERROR[S] [errornumber]

Allows the user to ignore errors on the next command and proceed in RELATE.

ALL	Optional. If specified, all RELATE errors will be ignored on the following command only. If ALL is specified, no additional errornumber may be specified.
errornumber	Optional. If specified, this must be a RELATE error number to be ignored on the following command only.

Either ALL or an errornumber must be specified. If an ignored error occurs on the following command, the message "ERROR #num HAS BEEN IGNORED." is printed but no error condition results.

If an error was detected, the $ERROR system-defined field will be set to the error number of the ignored error. $ERROR will remain set to this number until another error is encountered or another IGNORE ERROR is executed. IGNORE ERROR resets $ERROR to zero.

EXAMPLES:

The IGNORE ERROR command will ignore any specified errors that might occur on the following command.

```
)HELP ERROR 15

  15 A CURRENT PATH DOES NOT EXIST (A FILE MAY NOT BE OPEN).

)IGNORE ERROR 15
)PRINT X
ERROR #15 HAS BEEN IGNORED.
)NOTE If this same command were to be entered again right
)NOTE now, error 15 would no longer be ignored.
)
```

IGNORE

- [range] LABEL [modifiers] USING formatfile[;options] [FOR condition]

Displays output in a user-specified format.

range	Optional. If specified, only lines in the specified range will have labels printed. See the RANGE section.
:I	Optional global switch. If used, a forms alignment will not be requested prior to labels being generated.
:P	Optional global switch. If used, output is directed to the file RDBLIST. If RDBLIST cannot be opened, the output is directed to the device class "LP".
modifiers	Optional. Can be any one or more of the following separated by semicolons (";"):

Modifier	Description	Default
ACROSS=num	Number of labels across the page.	1
DOWN=num	Number of labels down the page. If specified in conjunction with a global ":P", the FORMATTED option is assumed. If ":P" is not used, printing will stop after each set of labels to allow the operator to insert a new form. If ":P" is not used and FORMATTED is specified, output will be form-fed (instead of pausing) after each set of labels.	unlimited
LINES=num	Number of lines per label.	6
REPEAT=num	Number of times a data record is to be re-used.	1
WIDTH=num	Width of each label in characters.	40
SUPPRESS	Suppresses the printing of blank lines in labels.	not suppressed

FORMATTED	Formats the output in pages. If specified in conjunction with a global ":P", and DOWN is not specified, the value of DOWN is assumed to be the configured number of lines per page divided by the number of lines per label. If FORMATTED to the line printer, 3 lines will be left at the top and bottom of each page.	not formatted

The value of a modifier can be changed by records obtained from the format file or by inclusion of the modifier in the command line. Modifiers specified in the command line take precedence over those obtained from the format file.

The final value of all modifiers must be greater than or equal to one.

formatfile

Required. Any options listed in the OPEN command needed to access the file must be appended to the filename, whether or not the file is already open. The formatfile must contain output formatting specifications. No more than 70 specifications may be included. The first four fields of the file must be of the type indicated below:

FIELD	TYPE	USE
1	A	The first character indicates the code of the record. The code may be the first letter of any of the local modifier switches (except FORMATTED) or "C" or "F" indicating a constant or a field. A "C" code indicates that field number four contains a constant to be printed. An "F" code indicates that field four contains a field name and the data in that field is to be printed. See modifiers, above, for other codes.
2	I	The value of a modifier, or the line number on the label where the data will appear for "C" and "F" codes. If this is a line number, it must be less than or equal to the number of lines on the label.
3	I	The position on the label where the data will be placed for "C" and "F" codes. This field is ignored for all other codes.
4	A	Contains an alphabetic constant if the code was a "C", or a fieldname if the code was an "F". The field is ignored for all other codes.

The position field in the format file (field number three) may be any value less than or equal to the WIDTH modifier. It may also be zero or negative as described below:

VALUE	ACTION
0	The next data value is positioned one space after the last non-blank character.
-1	The next data value is positioned just after the last non-blank character.
-2	Same as 0, but the data has all leading blanks removed before it is used.
-3	Same as -1, but the data has all leading blanks removed before it is used.

The order of records in the format file is significant for the data that will be output on any given line. The system will place information on a line in the order in which the data is read from the format file.

FOR condition — Optional. If used, only records meeting the specified condition will have labels printed. See the EXPRESSION EVALUATION section.

The command attempts to align odd-sized forms by spacing past the first form. This simplifies form-loading by allowing odd-sized forms forms to be loaded as if they were 11 inches in length.

If the output file for the labels is an interactive device the user will be requested to align the forms prior to the labels being generated.

EXAMPLES:

Create a format file for address labels.

```
)CREATE FILE FORMAT
ENTER FIELDNAME,TYPE,LENGTH[,DECIMALS]

  1? CODE,A,1
  2? LINE,I,2
  3? POS,I,2
  4? DATA,A,20
  5? //
THE "FORMAT" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000
FILE
```

The format file will print "TO:" on line 1, position 1 of the address label, NAME on line 2, position 1 followed by NUMBER at position 22. ADDRESS will be on line 3, CITY on line 4, followed by a comma, then a space and STATE. The label WIDTH is specified as being 40 characters.

```
)ADD SEPARATOR="/"
ENTER CODE/LINE/POS/DATA

CODE? C/1/1/TO:

CODE? F/2/1/NAME

CODE? F/2/22/NUMBER

CODE? F/3/1/ADDRESS

CODE? F/4/1/CITY

CODE? C/4/-1/,

CODE? F/4/0/ST

CODE? W/40//

CODE? //

)PRINT

$LINE C LI PO DATA

    1 C  1  1 TO:
    2 F  2  1 NAME
    3 F  2 22 NUMBER
    4 F  3  1 ADDRESS
    5 F  4  1 CITY
    6 C  4 -1 ,
    7 F  4  0 ST
    8 W 40  0

8 LINES PRINTED
```

Now print the address labels using the information from the CUST file and the format in the FORMAT file. Print the labels two across, making 3 copies of each, and overriding the format file's WIDTH to 30. Print labels only for customers outside of California.

```
)SET PATH CUST
)PRINT NAME, ADDRESS, ST

$LINE NAME                 ADDRESS                     ST

    1 HASLETREX INC        89 BEST WAY                 CA
    2 DEXMACH, INC         BOX 877 RD1                 CA
    3 CUPERCO              10802 WILKINSON AVENUE      CA
    4 AMERICAN TIRE CO.    7052 EL CAMINO REAL         CA
    5 FINCH, FINCH, & OTTO 87 NORTH FIRST, SUITE 243C  CA
    6 NATIONAL AIRLINES    SAN FRANCISCO INTL AIRPORT  CA
```

```
     7 ALEXANDER HALE & CO. 83A SAN PEDRO            NJ
     8 PERFECT SOUND        415 FAIR OAKS AVENUE     VA

8 LINES PRINTED.
)LABEL ACROSS=2; REPEAT=3; WIDTH=30 USING FORMAT FOR ST<>"CA"
TO:                              TO:
ALEXANDER HALE & CO.      700    ALEXANDER HALE & CO.      700
83A SAN PEDRO                    83A SAN PEDRO
ATLANTIC CITY, NJ                ATLANTIC CITY, NJ


TO:                              TO:
ALEXANDER HALE & CO.      700    PERFECT SOUND             800
83A SAN PEDRO                    415 FAIR OAKS AVENUE
ATLANTIC CITY, NJ                LAWRENCE, VA


TO:                              TO:
PERFECT SOUND             800    PERFECT SOUND             800
415 FAIR OAKS AVENUE             415 FAIR OAKS AVENUE
LAWRENCE, VA                     LAWRENCE, VA


2 DATA LINES USED.
6 LABELS PRINTED.
)
)
```

[range] LET assignment[,...] [FOR condition]

Makes arithmetic or alphabetic assignments to fields in the current path.

range	Optional. If specified, only records in the range can be assigned. See the RANGE section.
assignment	Required. Has the format fieldname=expression, where the fieldname must exist in the current file and the expression must be a valid expression. To make multiple assignments in the same command, separate the assignments with a comma (","). Multiple assignments are evaluated from left to right.
FOR condition	Optional. If used, only records meeting the condition will be changed. See the EXPRESSION EVALUATION section.

Performing an assignment on a field that is either in the current index or in the condition of the FOR clause may produce spurious results.

EXAMPLES:

Zero out all tax fields. Set tax to equal six and a half percent of AMOUNT for California, and 9% of AMOUNT for New Jersey. All other states should remain at zero.

```
)SET PATH INVOICE
)PRINT

$LINE  INVNO NAME                   NUMBER ST    AMOUNT     TAX SALES

    1  10221 NATIONAL AIRLINES        1000 CA $  627.01   88.07    36
    2  10455 NATIONAL AIRLINES        1000 CA $  335.00   40.20    36
    3     33 DEXMACH, INC.             100 CA $  348.70   18.00    99
    4   2738 CUPERCO                   400 CA $  948.60   32.40    36
    5  23557 CUPERCO                   400 CA $   37.00    3.00    86
    6  10044 CUPERCO                   400 CA $  500.00   35.99    99
    7    105 DEXMACH, INC.             100 CA $   86.32    4.81    83
    8    106 DEXMACH, INC.             100 CA $   76.40    1.10    87
    9  10002 ALEXANDER HALE & CO       700 NJ $  999.99    9.99    45
   10     33 PERFECT SOUND             800 VA $  677.77    3.33    83
   11    727 AMERICAN TIRE CO.         500 CA $    3.14    1.59    87
   12    747 AMERICAN TIRE CO.         500 CA $    0.97    0.00    83
   13   8663 AMERICAN TIRE CO.         500 CA $   86.00    1.32    36

13 LINES PRINTED.
)LET TAX=0
13 LINES ASSIGNED
)LET TAX=AMOUNT*.065 FOR ST="CA"
```

```
11 LINES ASSIGNED.
)LET TAX=AMOUNT*.09 FOR ST="NJ"
1 LINE ASSIGNED.
)PRINT

$LINE  INVNO NAME                        NUMBER ST    AMOUNT     TAX SALES

    1  10221 NATIONAL AIRLINES             1000 CA $ 627.01    40.76    36
    2  10455 NATIONAL AIRLINES             1000 CA $ 335.00    21.77    36
    3     33 DEXMACH, INC.                  100 CA $ 348.70    22.67    99
    4   2738 CUPERCO                        400 CA $ 948.60    61.66    36
    5  23557 CUPERCO                        400 CA $  37.00     2.40    86
    6  10044 CUPERCO                        400 CA $ 500.00    32.50    99
    7    105 DEXMACH, INC.                  100 CA $  86.32     5.61    83
    8    106 DEXMACH, INC.                  100 CA $  76.40     4.97    87
    9  10002 ALEXANDER HALE & CO.           700 NJ $ 999.99    90.00    45
   10     33 PERFECT SOUND                  800 VA $ 677.77     0.00    83
   11    727 AMERICAN TIRE CO.              500 CA $   3.14     0.20    87
   12    747 AMERICAN TIRE CO.              500 CA $   0.97     0.06    83
   13   8663 AMERICAN TIRE CO.              500 CA $  86.00     5.59    36

13 LINES PRINTED.
)
```

LIST COMMANDS [rangelist] [TO filename[;RECORDS=records][;WIDTH=width]]

Lists the indicated RELATE commands.

:P	Optional global switch. Directs output to the file RDBLIST (usually the printer). If RDBLIST cannot be opened, the output is directed to the device class "LP".
rangelist	Optional. Specifies a range of RELATE command numbers to be listed. If not specified, all commands executed thus far will be listed in the order of execution.
TO filename	Optional. If specified, the indicated RELATE commands will be listed to the file with this name. If the file does not already exist, it will be created as a permanent ASCII MPE file. Commands listed to a file will be listed without command numbers.
RECORDS	Optional. If specified, the keyword must be followed by an integer indicating the number of records to allocate for the file referenced by filename. If filename already exists, this parameter is ignored. If filename does not exist and RECORDS is not specified, the number of records defaults to 1024.
WIDTH	Optional. If specified, the keyword must be followed by an integer indicating the width in bytes (characters) of the records for the TO file. If filename already exists, this parameter is ignored. If filename does not exist and WIDTH is not specified, the record width defaults to 72 bytes.

This command can not be executed from the Host Language Interface routines.

EXAMPLES:

You can list previously executed commands either to your session or to a file. See the LIST FILE command for the contents of the output file.

```
)LIST COMMANDS 2/6

)NOTE ***** These commands demonstrate LIST COMMANDS.
):PURGE COMLIST
)CLOSE
)IGNORE ALL ERRORS
)OPEN FILE MPECUST; TYPE=MPE; STRUCTURE=CUST; FIELDS=&
&)         NAME, NUMBER, ADDRESS, CITY, (DATE_UPD,R,8)

)LIST COMMANDS 2/6 TO COMLIST;RECORDS=10
THE "COMLIST" FILE HAS BEEN CREATED AS A PERMANENT ASCII MPE FILE.
)
```

LIST FILE *filename*

Lists the contents of the indicated text file.

:P	Optional global switch. Directs the output to the file RDBLIST (usually the printer). If RDBLIST cannot be opened, the output is directed to the device class "LP".
filename	Required. The name of any file containing ASCII (text) information. The file may be numbered or unnumbered.

EXAMPLES:

The LIST FILE command can list the contents of any MPE text file (including EDITOR files). Here, we list the contents of the file created with the LIST COMMANDS command.

```
)LIST FILE COMLIST

NOTE ***** These commands demonstrate LIST COMMANDS.
:PURGE COMLIST
CLOSE
IGNORE ALL ERRORS
OPEN FILE MPECUST; TYPE=MPE; STRUCTURE=CUST; FIELDS=&
       NAME, NUMBER, ADDRESS, CITY, (DATE_UPD,R,8)

)
```

LOCK DATABASE databasename
LOCK FILE filename [;DATABASE=databasename]

Indicates to RELATE what resources should be locked for the next transaction. The LOCK command cannot be issued if a transaction is in progress.

databasename	Required if the DATABASE keyword is specified. This option will lock the entire database indicated. The option can only be used on IMAGE data bases.
filename	Required if the FILE keyword is specified. The name of a file that is currently open.
DATABASE	Can only be specified when the file listed exists in an IMAGE data base.

The LOCK command is used to inform RELATE of any files that will be changed in subsequent commands. When the command is issued, RELATE verifies that the requested locks can be obtained. The locks cannot be obtained if the file is not open with a locking option and is open with shared access or if multiple file locks are requested. The LOCK command cannot be issued if a transaction is in progress.

The locks are not actually obtained from the operating system until an attempt is made to read or write to one of the files for which locks are pending or until a BEGIN TRANSACTION command is executed.

Once locks are explicitly obtained through the LOCK command these must be released by the user. Locks obtained by RELATE are held only for the duration of a command unless the command is within a transaction. The locks automatically obtained are released when the transaction completes. If a transaction is just beginning, RELATE may obtain more locks. These locks are added to any the user may have requested and are not released until the user issues an UNLOCK command.

EXAMPLES:

The LOCK command allows the user to place locks on specific files prior to the execution of a sequence of commands.

```
)OPEN FILE CUST; MODE=SHARE
)LOCK FILE CUST
```

A SHOW FILES command can be used to determine what files have locks pending (which is indicated by a "P") and which files are presently locked (which is indicated by a "Y").

```
)SHOW FILES

L
O
C
K FILE NAME          DATABASE NAME

  INVOICE            INVOICE.DOC79.RDB

P CUST               CUST.DOC79.RDB
```

Once locks have been manually obtained the user must explicitly release them.

```
)UNLOCK
)SHOW FILES

L
O
C
K FILE NAME          DATABASE NAME

  INVOICE            INVOICE.DOC79.RDB

  CUST               CUST.DOC79.RDB

)
```

MODIFY FIELD fieldlist;formatlist

Alters the structure of a file by changing the format of existing fields.

:K	Optional global switch. Keeps the changes permanently. This switch can only be used on RELATE/3000 files. The user must be the creator of the file or an account librarian and have exclusive access to the file to use this switch.
fieldlist	Required. A list of one or more existing fieldnames, separated by commas, that are to be modified. An atsign ("@") may be used to represent all fields in the current file.
formatlist	Required list of one or more of the options described in the CREATE FILE's FIELD OPTIONS section with the following exceptions:

1) The TYPE cannot be changed and must not be specified.

2) The NAME and INTERNAL keywords must not be specified if more than one fieldname is included in the fieldlist.

3) A field cannot be changed to FORMAT="datestring" if it currently has a numeric format specification, decimal places in the print length, or DOLLAR or COMMAS specified. The reverse is also true.

4) If NAME is given, it must be a valid fieldname that does not already exist in the file.

5) If INTERNAL is given, it must be followed by an internal field number that is not already in the file unless zero is specified.

6) The SIZE cannot be changed on an alphabetic field.

EXAMPLES:

Change the name of the ST field to STATE. Change the print length of the NUMBER field from 6 characters to 5. Note that where the global "K" (keep) switch is not used, the changes will only be in effect until the CUST file is closed or RELATE is exited.

```
    )SET PATH CUST
```

```
)SHOW STRUCTURE

FILE NAME             =CUST.DOC79.RDB

                T            C           L A U
                Y PRINT      O           E D P  INT  BEG  END
#   NAME        P  LEN  $    M SPECIAL   V D D SIZE WORD WORD

  1 NAME        A  20                    3 Y Y  20B    0    9
  2 NUMBER      I   6                    3 Y Y   1W   10   10
  3 ADDRESS     A  26                    3 Y Y  26B   11   23
  4 CITY        A  14                    3 Y Y  14B   24   30
  5 ST          A   2                    3 Y Y   2B   31   31
  6 PHONE       A   8                    3 Y Y   8B   32   35

PRINT LINE WIDTH = 87 CHARACTERS.
)PRINT NAME, NUMBER, ST

$LINE NAME                        NUMBER ST

    1 HASLETREX INC.                 200 CA
    2 DEXMACH, INC.                  100 CA
    3 CUPERCO                        400 CA
    4 AMERICAN TIRE CO.              500 CA
    5 FINCH, FINCH, & OTTO           600 CA
    6 NATIONAL AIRLINES             1000 CA
    7 ALEXANDER HALE & CO.           700 NJ
    8 PERFECT SOUND                  800 VA

8 LINES PRINTED.
)MODIFY:K FIELD ST; NAME=STATE
)MODIFY FIELD NUMBER; SIZE=5
)SHOW STRUCTURE

FILE NAME             =CUST.DOC79.RDB

                T            C           L A U
                Y PRINT      O           E D P  INT  BEG  END
#   NAME        P  LEN  $    M SPECIAL   V D D SIZE WORD WORD

  1 NAME        A  20                    3 Y Y  20B    0    9
  2 NUMBER      I   5                    3 Y Y   1W   10   10
  3 ADDRESS     A  26                    3 Y Y  26B   11   23
  4 CITY        A  14                    3 Y Y  14B   24   30
  5 STATE       A   2                    3 Y Y   2B   31   31
  6 PHONE       A   8                    3 Y Y   8B   32   35

PRINT LINE WIDTH = 86 CHARACTERS.
)PRINT NAME, NUMBER, STATE

$LINE NAME                        NUMBE ST

    1 HASLETREX INC.                200 CA
    2 DEXMACH, INC.                 100 CA
```

```
          3 CUPERCO                      400 CA
          4 AMERICAN TIRE CO.            500 CA
          5 FINCH, FINCH, & OTTO         600 CA
          6 NATIONAL AIRLINES           1000 CA
          7 ALEXANDER HALE & CO.         700 NJ
          8 PERFECT SOUND                800 VA

8 LINES PRINTED.
)
```

```
 MODIFY FILE filename
[;CLUSTER=index]
[;COMPRESS=YES|NO]
[;CRASHPROOF=YES|NO]
[;DELETE=LOGICAL|PHYSICAL]
[;SCAN=blocks]
```

Alters the manner in which data is handled in a particular file.

filename	Required. The name of an open RELATE/3000 file. The parameters following the filename will be permanently adjusted to reflect the new values specified.
CLUSTER	Optional. If specified, the keyword must be followed by the number of an existing index. Data will be stored clustered by this index. Sequential retrieval of data through this index will be significantly quicker than all other indexes. The default clustering index is the first unary index created on the file if the user has not previously assigned a clustering index.
COMPRESS	Optional. If specified, the keyword must be followed by YES or NO. If YES is specified, data compression will be attempted on individual data records. The compression benefit will be greatest when data records contain large partially filled text fields or contain completely blank alphabetic fields or numeric fields containing zeroes. If NO is specified, data records will not be compressed. All session temporary (RETENTION=TEMP) and permanent (RETENTION=PERM) files default to YES. All open temporary files (RETENTION=NONE) default to NO.
CRASHPROOF	Optional. If specified, the keyword must be followed by YES or NO. If YES is specified, all updates are forced to disc on a record by record basis. If NO is specified, data is only forced to disc when a command completes. All session temporary (RETENTION=TEMP) and permanent (RETENTION=PERM) files default to YES. All open temporary files (RETENTION=NONE) default to NO.
DELETE	Optional. If specified, the keyword must be followed by LOGICAL or PHYSICAL. The mode may always be changed from PHYSICAL to LOGICAL. It may be changed from LOGICAL to PHYSICAL only if no logically deleted records exist in the file. If records are LOGICALLY deleted, they can be RECOVERED. To remove logically deleted records the file must be REORGANIZED. If records are PHYSICALLY deleted they are removed from the file and the space used by the record can be used by another record. This will have no effect on $LINE. The default is LOGICAL.

SCAN Optional. If specified, the keyword must be followed by a value between 0 and 100 inclusive. If a record cannot be clustered, this value (which is taken as a percentage) determines when a scan should be done to find a partially full data block. If a suitably large space cannot be found in the blocks scanned, the record is placed at the logical end of the file. Larger values of this parameter will encourage better utilization of file space at the expense of more disc activity during the addition (and in certain cases updating) of data. Smaller values increase speed during these operations. If zero is specified, no scanning is performed and records which cannot be clustered are immediately placed at the logical end of the file. In this case (and particularly if the key values in the clustering index are not random with respect to deleted records) periodic reorganizations will be required to reuse the file space efficiently. A value of 3 is used by default.

EXAMPLES:

This command is the only one that can alter crashproofing and data compression.

```
)SET PATH CUST
)SHOW CURRENT

FILE NAME                =CUST.DOC79.RDB

   FILE (OR SET) NAME     =CUST
   CURRENT RECORDS        =8 (EOF=8)
   MAXIMUM RECORDS        =4096
   FILE TYPE              =RELATE/3000
   DISPOSITION,RETENTION  =PERMANENT,PERMANENT
   ACCESS MODE            =EXCLUSIVE,NOLOCK
   CLUSTERING INDEX       =1
   DATA COMPRESSION       =YES
   CRASHPROOF ACCESS      =YES
   DELETE                 =LOGICAL
   SCAN                   =10
   RECORDS CAN BE         :DELETED,UPDATED,ADDED

)MODIFY FILE CUST; COMPRESS=NO; CRASH=NO
)SHOW CURRENT

FILE NAME                =CUST.DOC79.RDB

   FILE (OR SET) NAME     =CUST
   CURRENT RECORDS        =8 (EOF=8)
   MAXIMUM RECORDS        =4096
   FILE TYPE              =RELATE/3000
   DISPOSITION,RETENTION  =PERMANENT,PERMANENT
   ACCESS MODE            =EXCLUSIVE,NOLOCK
   CLUSTERING INDEX       =1
   DATA COMPRESSION       =NO
```

```
    CRASHPROOF ACCESS       =NO
    DELETE                  =LOGICAL
    SCAN                    =10
    RECORDS CAN BE          :DELETED,UPDATED,ADDED

)
```

NOTE [any text]

The NOTE command acts as a comment only. It performs no action whatsoever. Anything at all may be included on the same line as the NOTE command.

:D	Optional global switch. If used, the text following the NOTE command will be displayed. This only affects commands executing in a procedure with the SHOW option turned off.

The command is useful for documenting RELATE/3000 procedure files.

EXAMPLES:

The NOTE command acts as a comment only. It performs no action at all.

```
    )NOTE THIS IS A NOTE.
    )NOTE NOTES ARE USEFUL FOR DOCUMENTING PROCEDURE FILES.
    )NOTE EXIT
    )
```

NOTE

```
OPEN DATABASE databasename
   ;TYPE=IMAGE
   [;INFORMATION]
   [;PASSWORD=password]
   ;MODE=accessmode
```

Opens an IMAGE database so that sets can be accessed.

databasename	Required. The name of the IMAGE database to be opened.
TYPE	Required. Indicates the type of the database to be opened.
INFORMATION	Optional. If specified when an IMAGE database is opened, the names of the sets in the database will be displayed.
PASSWORD	Optional. The password must be specified when the requested IMAGE database has a password. If not specified, a null password is assumed. If the user is the creator of the database, a semicolon (";") can be used as the password. The semicolon must be enclosed within quotes.
MODE	Required for IMAGE databases. The accessmode must be specified by number when an IMAGE database is opened. The IMAGE modes are as follows:

MODE	TYPE	CONCURRENT ACCESS	LOCKING
1	Modify	Modify	Yes
2	Update	Update	
3	Modify	None	
4	Modify	Read	
5	Read	Modify	Yes
6	Read	Modify	
7	Read	None	
8	Read	Read	

EXAMPLES:

Open an IMAGE database. Use the INFORMATION parameter to obtain a list of available datasets within the database. Open the database with read access only and assume that this user was the creator (hence the password of ";"). Use the OPEN FILE|SET command to open one of the sets.

```
)OPEN DATABASE INVDB; TYPE=IMAGE; MODE=8; PASSWORD=";"; &
&)     INFORMATION

THE FOLLOWING SET(S) CAN BE ACCESSED:
DATE-MASTER

)OPEN SET DATE-MASTER; DATABASE=INVDB; INFORMATION

INDEX #0 UNARY (RECORD NUMBERS) 1 FIELD(S).
  $LINE


INDEX #1 UNARY (HASHED) 1 FIELD(S).
  DATE

)SHOW PATH

PATH NAME         FILE NAME          DATABASE NAME

DATEMASTER        DATE-MASTER        INVDB.DOC79.RDB (CURRENT PATH)
MPETXUN           MPETXUN            MPETXUN.DOC79.RDB
MPETEXT           MPETEXT            MPETEXT.DOC79.RDB
INV               INVOICE            INVOICE.DOC79.RDB
INVOICE           INVOICE            INVOICE.DOC79.RDB
C                 CUST               CUST.DOC79.RDB
CUST              CUST               CUST.DOC79.RDB
CUST1             CUST1              CUST1.DOC79.RDB
MPECUST           MPECUST            MPECUST.DOC79.RDB
```

```
OPEN FILE|SET filename
[;TYPE=RELATE|MPE|KSAM|IMAGE]
[;STRUCTURE=pathname]
[;DATABASE=databasename]
[;INFORMATION]
[;ERASE]
[;DOMAIN=PERMANENT|TEMPORARY]
[;RETENTION=PERMANENT|TEMPORARY|NONE]
[;PATH=pathname]
[;FIELDS=fieldlist]
[;MODE=accessmodes]
```

Opens a dataset or file.

filename	Required. This is either the name of an IMAGE dataset or a RELATE, KSAM, or MPE file as determined by the TYPE parameter.
TYPE	Optional. Indicates the type of the file to be opened. A RELATE file is the default.
STRUCTURE	Required when a KSAM or a MPE file is opened and FIELDS is not specified. The parameter cannot be specified when a RELATE or IMAGE dataset is opened. The newly opened file will have the format of the file referenced by the given pathname.
DATABASE	Required to access an IMAGE dataset. The parameter is used to specify the database in which the dataset is located. The database must have previously been opened.
INFORMATION	Optional. If specified when a file is opened, information regarding index structures and fieldname changes will be displayed. This should be used when accessing an IMAGE dataset, as RELATE will remove all special characters from the fieldnames and truncate the result to ten characters.
ERASE	Optional. If specified, the file or set is erased after it is opened. If an IMAGE master dataset is opened, the set is only erased if no detail entries referencing the set exist. IMAGE automatic masters, VIEWs, and MPE files cannot be erased. The file cannot be erased unless the appropriate access mode is used, and, if the file resides in a secured group, the user is the creator of the file. A file cannot be erased if a transaction is in progress.
DOMAIN	Optional. Indicates the present domain of the file. The file may be a temporary or permanent file. If not specified, the file is assumed to exist in the permanent domain.
RETENTION	Optional. If specified, the file is saved in the specified domain when it is closed. If NONE is specified, the file is purged. This parameter may not be specified for an IMAGE dataset. If not specified, the file is saved in the domain in which it currently exists. A file cannot be

moved from the PERMANENT domain to the TEMPORARY domain.

PATH	Optional. If specified, the newly opened file will be given this path name. If not specified, the path name will be the filename (or setname) excluding any group or account names supplied and excluding all special characters on IMAGE dataset names. Each path name in the Command Interpreter must be unique. If OPEN FILE is used from the Host Language Interface routines, the path name must be unique in the passed cursor. The file cannot be opened if the assumed or given path name duplicates an existing name.
FIELDS	Optional. This keyword may only be specified when the TYPE of the file being opened is MPE or KSAM. The keyword must be followed by a fieldlist of the format described in the Specifying Fields section of the CREATE command. If the STRUCTURE of the current file is being used, groups of fields may be specified with the pattern-matching feature. In addition to the standard pattern matching, a minus sign ("-") may be included as the first character to indicate NOT matching this pattern.
MODE	Optional. If specified, the file is opened in the access mode that follows. If a RELATE, KSAM, or MPE file is opened, the mode may be specified by number or by keywords. This is ignored for IMAGE datasets. The numeric modes are as follows:

MODE	TYPE	CONCURRENT ACCESS	LOCKING
1	Modify	Modify	Yes
2	Update	Update	
3	Modify	None	
4	Modify	Read	
5	Read	Modify	Yes
6	Read	Modify	
7	Read	None	
8	Read	Read	

The keywords available are as follows:

UPDATE	Allows records in the file to be changed but records cannot be added or deleted.
MODIFY	Records can be added, deleted, or changed.
READ	Records can only be read.

The user may also specify EXCLUSIVE, SEMI-EXCLUSIVE, or SHARED access. EXCLUSIVE access prevents any other user from accessing the file. SEMI-EXCLUSIVE access prevents another user from writing to the file but does not prevent reading. SHARED access allows other users to read from or write to the file.

If the access mode is SHARED or SEMI-EXCLUSIVE, the system will perform locking to ensure database integritv when changes are made. If the user has exclusive access to the file or database, the system assumes NOLOCK.

For RELATE, KSAM, and MPE files, the default mode is EXCLUSIVE, MODIFY.

EXAMPLES:

A file can be opened more than once if different path names are assigned to it. This is useful for recursive data structures such as a bill of materials file.

```
)OPEN FILE CUST
)OPEN FILE CUST; PATH=C
)OPEN FILE INVOICE
)OPEN FILE INVOICE; PATH=INV
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

INV                INVOICE            INVOICE.DOC79.RDB  (CURRENT PATH)
INVOICE            INVOICE            INVOICE.DOC79.RDB
C                  CUST               CUST.DOC79.RDB
CUST               CUST               CUST.DOC79.RDB
CUST1              CUST1              CUST1.DOC79.RDB
MPECUST            MPECUST            MPECUST.DOC79.RDB
```

Open an 80-byte MPE Editor file with 72 characters of text and an 8-digit line number on the right end of each record. This can be done either by giving the name of an open path with the same structure or by listing the field specifications explicitly.

```
)OPEN FILE RELTEXT
)SHOW

FILE NAME              =RELTEXT.DOC79.RDB

             T
             Y  PRINT  INT
NAME         P   LEN   SIZE

TEXT         A   72     72B
LINE_NUM     A    8      8B

PRINT LINE WIDTH = 87 CHARACTERS
```

```
)OPEN FILE MPETEXT; TYPE=MPE; STRUCTURE=RELTEXT
)CLOSE FILE RELTEXT
)CLOSE FILE MPETEXT
)OPEN FILE MPETEXT; TYPE=MPE; FIELDS=(TEXT,A,72),(NUM,A,8)
```

Open an MPE file with 72 bytes per record. This can be done either by listing the field specifications or by giving a structure file and listing the names of the desired fields. Notice we can use an open MPE file for the structure.

```
)OPEN FILE MPETXUN; TYPE=MPE; STRUCTURE=MPETEXT; FIELDS=TEXT
)SHOW

FILE NAME            =MPETXUN.DOC79.RDB

           T
           Y PRINT  INT
NAME       P  LEN  SIZE

TEXT       A  72    72B

PRINT LINE WIDTH = 78 CHARACTERS.
)
```

OPEN RDBLIST

Sets up the file RDBLIST for multiple output.

After the OPEN RDBLIST command has been executed, all RELATE output scheduled to go to the file RDBLIST (usually the printer) will be spooled into a single file until a CLOSE RDBLIST command is encountered or RELATE is terminated.

Normally when a command is executed with a global :P switch (direct output to printer), the output from the command is printed immediately. If an OPEN RDBLIST has been executed, however, no output will be printed until a CLOSE RDBLIST is executed. At that point, the output from all commands since the OPEN RDBLIST that were executed with a global :P will be printed.

EXAMPLES:

Normally, output to the printer is spooled immediately and a message is displayed that it has been done so. When RDBLIST is OPENed, however, output is grouped and not spooled until RDBLIST is CLOSEd.

```
)OPEN FILE CUST
)PRINT:P
THE OUTPUT HAS BEEN PLACED IN SPOOL FILE #088.
8 LINES PRINTED.
)OPEN RDBLIST
)PRINT:P
8 LINES PRINTED.
)OPEN FILE INVOICE
)PRINT:P
9 LINES PRINTED.
)CLOSE RDBLIST
THE OUTPUT HAS BEEN PLACED IN SPOOL FILE #089.
)
```

PAUSE ["comment"]

Causes RELATE to pause until RETURN is pressed.

:D	Optional global switch. If used, the text following the PAUSE command will be displayed and RELATE will pause. This only affects commands executing in a procedure with the SHOW option turned off.
comment	Optional. If used, this character string must be enclosed in quotes. This is for informational purposes only.

PAUSE functions only if the user is executing a procedure file with the SHOW option. PAUSE will not function from a job. The system will pause until the carriage return is pressed.

EXAMPLES:

The PAUSE command, if used in a procedure file with a SHOW option, will prevent RELATE from executing until RETURN is pressed.

```
)EXECUTE DOIT2;SHOW
)NOTE   PROCEDURE FILE BEGINS
)NOTE   This procedure file demonstrates the PAUSE command.
)PAUSE  "PRESS RETURN WHEN READY"
)NOTE   PROCEDURE FILE ENDS
)
```

```
PERMIT READ|DELETE|ADD|CHANGE|ALL ON file
                [;FIELDS=fieldlist]
                [BY userlist]
                [FOR condition]
```

Allows the DBA to authorize individual users to perform functions on files. In addition, the DBA can restrict access at the field (column) or record (row) level.

file	Required. The name of a RELATE, MPE, or KSAM file or the name of a view that exists in the log-on account.
FIELDS	Optional. If not included, all fields in the file may be referenced when a record is added, read, or changed. If included, this must be a list of one or more fieldnames separated by commas. The fields contained in the list are the only fields that the user can reference when records are read, added, or changed.
BY userlist	Optional. If the BY keyword is used, this must be a list of one or more user names separated by commas. If not included, an atsign ("@"), representing all groups, is assumed.
FOR	Optional. If specified, the condition following must be met when the indicated operations are performed on the file. The FOR clause may reference all of the fields in the file and is not limited by the FIELDS keyword.

The command can only be executed by an account librarian (a user with AL capability) executing in the PUB group.

The PERMIT command allows the DBA to apply different security provisions to different users, based on the function being performed. The security provisions are only effective if the file resides in a secured group. The DENY command can be used to revoke previously PERMITted operations.

The FIELDS keyword is used to restrict the columns that can be accessed. It can be specified when READ, ADD, CHANGE, or ALL permission is given. Fields can only be ADDed or CHANGEd if they can be READ.

The FOR clause restricts access on a record basis and can be used when any of the permissions are given. The user may only DELETE and CHANGE records that can be READ. Records may be ADDed that cannot later be READ.

The PERMIT command does not check for the existence of the file, users, or fields used in the command. When the security restriction is used later, any fields that do not exist in the file or that the user does not have READ access to are ignored.

EXAMPLES:

To illustrate how the PERMIT command controls access, a file and a typical set of security restrictions will be set up. The file will be called EMPS and contains some of the information that would exist in a payroll-personnel system: NAME, the employee name; MANAGER, the employee's manager; SALARY; REVIEW, review date; and EXT, the phone extension.

```
)CREATE FILE EMPS
ENTER FIELDNAME,TYPE,LENGTH[.DECIMALS]

  1? NAME,A,30
  2? MANAGER,A,30
  3? SALARY,R,10.2
  4? REVIEW,R,8;FORM="MM/DD/YY"
  5? EXT,I,4
  6? //
THE "EMPS" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
```

Each user will be responsible for maintaining specific portions of the database. Additionally, each user should be able to view as much information as would be reasonable for the position of the user. Briefly, the users are as follows:

(1) JOHN and RICK are department managers.
(2) PRES is the president of the company.
(3) PAYROLL is used by all of the payroll clerks.
(4) PRSUPER is the payroll supervisor.
(5) PHONE is the telephone operator.

The user PRES must be allowed to read any portion of the database. He is not allowed to make any modifications. This capability can be assigned with a single PERMIT as follows:

```
)PERMIT READ ON EMPS BY PRES
```

The user PHONE can only access the NAME and EXT fields. Additionally, the user should be allowed to change the EXT field.

These capabilities can be assigned with two PERMIT commands. The first authorizes read access to the two fields. The second authorizes the update of the EXT field.

```
)PERMIT READ ON EMPS; FIELDS=NAME, EXT BY PHONE
)PERMIT CHANGE ON EMPS; FIELD=EXT BY PHONE
```

The users JOHN and RICK should be allowed to access any record for employees that work in their departments. They should also be allowed to change the REVIEW dates in those records.

Two PERMIT commands must be issued for each manager. The first authorizes the read; the second, the update of the REVIEW field. A FOR clause is not required on the second PERMIT since a user can only update records that can be read.

```
)PERMIT READ ON EMPS BY JOHN FOR MANAGER="JOHN"
)PERMIT CHANGE ON EMPS; FIELD=REVIEW BY JOHN
```

The user PAYROLL can read, add, and update all records in the file except for the REVIEW, EXT and SALARY field of those earning more than $50,000 per year.

This capability can be assigned with a single PERMIT command:

```
)PERMIT ALL ON EMPS; FIELDS=NAME, MANAGER, SALARY &
&)BY PAYROLL &
&)FOR SALARY<50000
```

The user PRSUPER must be able to access all information in the file and change the NAME, MANAGER, and SALARY fields. This takes two PERMIT commands. The first command authorizes the user to read and delete any record in the file. The second allows the user to only update and add the NAME, MANAGER, and SALARY fields.

```
)PERMIT READ, DELETE ON EMPS BY PRSUPER
)PERMIT ADD, CHANGE ON EMPS; FIELDS=NAME, MANAGER, SALARY &
&)BY PRSUPER
```

[range] PRINT [fieldlist] [FOR condition]

Displays information from the current file.

range	Optional. If used, only records in the specified range are printed. See the RANGE section.
:num	Optional global switch. Skips a line after every "num" lines. If a global "num" switch is used with break fields, the line count is reset after any control break. If a global "num" switch is used with a global "F" or "P" switch, the line count is reset at the top of each page.
:C	Optional global switch. If used, the complete field name of each field will print at the top of each page. If not specified, fieldnames will be truncated to the print size of the field.
:D	Optional global switch. If used, the filename, index number and fieldnames, date, and the page number will be printed at the top of each new page.
:F	Optional global switch. Form-feeds the output. The print command enters report mode if used. The report mode attempts to format the output in pages matching the type of the user's terminal (as specified with the TERMINAL command).
:L	Optional global switch. Prints the line number.
:N	Optional global switch. Suppresses the printing of the fieldnames in the heading.
:P	Optional global switch. Directs the output to the line printer. Also sets the global "F" switch. Output is directed to the file RDBLIST. If RDBLIST cannot be opened, the output is directed to the device class "LP".
:S	Optional global switch. Suppresses the current key.
:T	Optional global switch. If used, grand totals are printed for all numeric fields that are printed. If local "T" switches are also used, only these fields are totaled.
fieldlist	An optional list of fields to be printed. If not specified, or all items in the list contain a local "B" switch, all fields will be printed in the order in which they were structured. The fieldname $LINE may be included in the fieldlist to print the line number at a position other than the far left of the line, if the file is not a SELECTION. Groups of fields may be specified using the pattern-matching feature.

LOCALS	Optional switches which, if used, appear on items in the fieldlist.

:num	Skips "num" lines on a control break.
:B	Causes this field to be used to generate a control break. This switch only affects the spacing of the output, not which fields are printed.
:F	Form-feeds on a control break. Also sets the global "F" switch if used on a break field.
:H	Indicates a field or text that should be used in the heading. It may appear on an alphabetic field or on an alphabetic constant enclosed in quotes. These items may also contain a local "C", "L", or "R" switch to center, left, or right justify the text. Left justification is the default. The system ensures that no data is lost when items are justified. Items that are centered or right justified cannot be longer than the width of the output line.
:S	Suppresses this field if it has not changed. If a local "S" switch appears on a break field, and the field will also be printed, values duplicated from one line to the next are suppressed (blanked out). When a new page is started, all fields are printed on the first line.
:T	Totals on the field or totals the field. If a local "T" switch appears on a break field (fields that contain a local "B" switch) subtotals will be generated when the field changes. If a local "T" switch appears on fields other than those with a local "B" switch, those fields will be totaled.

FOR condition	Optional. If used, only records meeting the condition are printed. See EXPRESSION EVALUATION section.

If a number is too large to be printed, the field is filled with pound signs ("#").

The fields that compose the current index are automatically printed on the left of any output unless a global "S" switch is used.

If the PRINT command is issued without a fieldlist (or if all fields in the list contain a local "B" switch), all fields in the file are printed.

The local "num", "F", and "S" switches are only recognized on those fields that also contain the local "B" switch. A total or subtotal is not printed if it is the total of only a single line.

When a total or subtotal is printed, the field that caused the control break is printed followed by an asterisk ("*").

EXAMPLES:

Print the INVOICE file by customer NUMBER. The automatic printing of the index fields is suppressed by the global "S" switch. The NUMBER field is used as a break field, suppressing printing until a break (change in value) occurs and then printing two spaces. The INVNO and AMOUNT fields are printed for each line. The local "B" switch does not cause the field to be printed, so NUMBER and NAME must be included again in the fieldlist.

```
)SET PATH INVOICE
)SET INDEX NUMBER, INVNO
INDEX #3 IS NOW THE CURRENT INDEX.
)PRINT:D:S NUMBER:B:S:2, NAME:B:S, NUMBER, NAME, INVNO, AMOUNT

INVOICE          MON, JAN 28, 1985, 12:16 PM

NUMBER NAME                      INVNO   AMOUNT

   100 DEXMACH, INC.                33 $ 348.70
                                   105 $  86.32
                                   106 $  76.40


   400 CUPERCO                    2738 $ 948.60
                                 10044 $ 500.00
                                 23557 $  37.00


   500 AMERICAN TIRE CO.           727 $   3.14
                                   747 $   0.97
                                  8663 $  86.00


   700 ALEXANDER HALE & CO.      10002 $ 999.99


   800 PERFECT SOUND                33 $ 677.77


  1000 NATIONAL AIRLINES         10221 $ 627.01
                                 10455 $ 335.00

13 LINES PRINTED.
```

Print NAME, AMOUNT, and TAX for all invoices where the SALES_MAN was number 36, total the AMOUNT and TAX fields, and do not print the fieldnames in the heading. The current key is not suppressed, so NUMBER and INVNO will print before each record.

```
)PRINT:N:T NAME, AMOUNT:T, TAX:T FOR SALES_MAN=36

   400    2738 CUPERCO                $ 948.60   32.40
   500    8663 AMERICAN TIRE CO.      $  86.00    1.32
  1000   10221 NATIONAL AIRLINES      $ 627.01   88.07
  1000   10455 NATIONAL AIRLINES      $ 335.00   40.20

                                      $1996.61  161.99


4 LINES PRINTED.
```

All functions (except aggregates) may be used in the condition of the PRINT command.

Print all companies whose name begins with "N".

```
)PRINT NUMBER,NAME FOR $MATCH(NAME,"%N")

 INVNO NUMBER NAME

 10221    1000 NATIONAL AIRLINES
 10455    1000 NATIONAL AIRLINES

2 LINES PRINTED.
```

Print all invoices for Cuperco, making sure you get the name whether it's in upper or lower case.

```
)PRINT NUMBER,NAME,AMOUNT FOR $UPS(NAME)="CUPERCO"

 INVNO NUMBER NAME                   AMOUNT

  2738     400 CUPERCO              $ 948.60
 10044     400 CUPERCO              $ 500.00
 23557     400 CUPERCO              $  37.00

3 LINES PRINTED.
)
```

PURGE FILE filename

Purges the given filename. If the file is in a secured group, the user must be the creator of the file.

filename	The name of the file to be purged. The file must have previously been opened. An IMAGE database or dataset cannot be purged from within RELATE.

A file cannot be purged while a transaction is in progress.

EXAMPLES:

Show all existing paths. Purge some files. A system LISTF would have to be done to verify that they have indeed been purged. The difference between a RELATE PURGE and a system PURGE is that the RELATE purge automatically purges any existing index files for the purged file, so the user need never touch the index files themselves.

```
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

SUM                SUM                SUM.DOC79.RDB
CUSTMAT            CUSTMAT            CUSTMAT.DOC79.RDB
CUSTERR            CUSTERR            CUSTERR.DOC79.RDB
INVOICE            INVOICE            INVOICE.DOC79.RDB (CURRENT PATH)
CUST               CUST               CUST.DOC79.RDB
CUST1              CUST1              CUST1.DOC79.RDB
MPECUST            MPECUST            MPECUST.DOC79.RDB

)PURGE FILE MPECUST
THE "MPECUST" FILE HAS BEEN PURGED.
)PURGE FILE SUM
THE "SUM" FILE HAS BEEN PURGED
)PURGE FILE CUSTMAT
THE "CUSTMAT" FILE HAS BEEN PURGED
)PURGE FILE CUSTERR
THE "CUSTERR" FILE HAS BEEN PURGED
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

INVOICE            INVOICE            INVOICE.DOC79.RDB (CURRENT PATH)
CUST               CUST               CUST.DOC79.RDB
CUST1              CUST1              CUST1.DOC79.RDB

)
```

PURGE INDEX number

Purges an existing index from the current file. Indexes can only be purged from RELATE/3000 files to which the user has exclusive access. If the file exists in a secured group, only the creator of the file may purge the index.

number	Required. The index number of the index to be purged.

Indexes may be purged regardless of the amount of data in the current file.

If the current index is purged, the line number becomes the current index.

An index cannot be purged if a path to the file, other than the current path, exists.

EXAMPLES:

Purge indexes from the INVOICE file. First set index 2 and show all the indexes available. Purge an index that is not the current index and view the remaining indexes. Purge the current index and note that the current index becomes the line number (index 0).

```
)SET PATH INVOICE
)SET INDEX 2
INDEX #2 IS NOW THE CURRENT INDEX
)SHOW INDEX

FILE NAME              =INVOICE.DOC79.RDB

INDEX 1 BY NAME,NUMBER
  WORDS IN KEY (+NODE)  : 13 (+1)
  DISTRIBUTION          : 2,2
  LEVELS IN TREE        : 1
  NUMBER OF USED NODES  : 1

INDEX 2 BY AMOUNT-D (CURRENT INDEX)
  WORDS IN KEY (+NODE)  : 4 (+1)
  DISTRIBUTION          : 1
  LEVELS IN TREE        : 1
  NUMBER OF USED NODES  : 1

INDEX 3 BY NUMBER,INVNO
  WORDS IN KEY (+NODE)  : 4 (+1)
  DISTRIBUTION          : 2,1
  LEVELS IN TREE        : 1
  NUMBER OF USED NODES  : 1

)PURGE INDEX 1
INDEX #1 HAS BEEN PURGED.
```

```
)SHOW INDEX

FILE NAME                =INVOICE DOC79.RDB

INDEX 2 BY AMOUNT:D (CURRENT INDEX)
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1

INDEX 3 BY NUMBER,INVNO
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 2,1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1

)PURGE INDEX 2
INDEX #2 HAS BEEN PURGED.
INDEX #0 IS NOW THE CURRENT INDEX.
)SHOW INDEX

FILE NAME                =INVOICE.DOC79.RDB

INDEX 3 BY NUMBER,INVNO
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 2,1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1

)
```

PURGE VIEW filename

Purges an existing view.

:D	Optional global switch. If used, the VIEW is purged from the RDBDD. Otherwise, it is purged from the log-on group.
filename	Required. The name of an existing view.

EXAMPLES:

```
)PURGE VIEW ACTIVE
THE "ACTIVE" VIEW HAS BEEN PURGED.
)
```

QUIZ[:P] reportname
[;SOURCE=datafile]
[;PARM=parm]
[;MAXDATA=maxdata]

Invokes the QUIZ* report writer to generate a report based on the data referenced by the current path.

:P	Optional global switch. Directs the output to the line printer. This switch causes a file equation of the form FILE QUIZLIST;DEV=LP to be issued.
reportname	The name of an existing QUIZ report. This filename is assigned the formal designator QUIZUSE by RELATE.
SOURCE	Optional. Indicates that the report normally expects input from a file other than QUIZDATA (see below for more information). If used, this keyword must be followed by the name of the file from which QUIZ will normally read information for a report.
PARM	Optional. If used, this keyword must be followed by a value indicating the size, in words, of the internal table available to QUIZ to build a report description. The default is 2000 words.
MAXDATA	Optional. If used, this keyword must be followed by a value indicating the maximum amount, in words, of virtual memory available to QUIZ. This value must always exceed the workspace by at least 2000 words. If sorting is required, MAXDATA must exceed the work area by at least 4000 words. The default is 12000 words.

The interface to QUIZ is accomplished through the use of file equations, message files, and process handling. RELATE/3000 first creates a message file in the temporary domain. This file is called QUIZDATA and contains space for 100 records of information. File equations are then issued for QUIZUSE and possibly QUIZLIST (if the global P switch was used) and QUIZDATA (if the SOURCE keyword was used). QUIZ is now started as a son process and RELATE suspends until the message file is opened by QUIZ. RELATE then begins placing information into the file while QUIZ is simultaneously reading the information and producing the report. When RELATE completes writing to the message file, the file is purged and RELATE suspends until QUIZ completes the generation of the report, at which time RELATE prompts the user for a new command.

* QUIZ is a product of COGNOS Incorporated.

RELATE expects QUIZ to exist as QUIZ.PUB.COGNOS. The actual actual location of the program cannot be adjusted by a file equation. The user must have executed access to the program.

If the report contains a sort option it is recommended that the keyword SORTED be used instead of the keyword SORT. A BY clause should be placed on the SELECT command to guarantee that RELATE will produce the data in the proper order.

EXAMPLES:

In order to use QUIZ a schema must be compiled that describes the elements and records comprising the application data base. To interface RELATE/3000 to QUIZ, MPE files must be defined that correspond to the format of the SELECT command output. This information can be obtained by executing a SHOW command after the SELECT command that will retreive the data.

```
)OPEN FILE CUST
)OPEN FILE INVOICE
)SELECT CUST.@, INVOICE.INVNO, INVOICE.AMOUNT
WARNING: ALL FILES IN SELECT COMMAND MAY NOT BE JOINED TOGETHER.
)SHOW

SELECTION

            T
            Y  PRINT   INT
NAME        P   LEN   SIZE

NAME        A   20     20B
NUMBER      I    5      1W
ADDRESS     A   26     26B
CITY        A   14     14B
ST          A    2      2B
PHONE       A    8      8B
INVNO       I    6      1W
AMOUNT      R    8.2    2W

PRINT LINE WIDTH = 96 CHARACTERS
```

The schema below contains the definition of the record indicated above as INPUT2. Under normal circumstances the schema would contain information on actual files in addition to the dummy files required by the interface mechanism.

```
:RUN QSCHEMA.PUB.COGNOS

Q S C H E M A   (1.04.F)

>
> SCHEMA "RELATE/3000 INTERFACE DEMONSTRATION"
>
> FILE INPUT1 TYPE MPE
> FILE INPUT2 TYPE MPE
>
> ELEMENT NUMBER          9(5) HEADING "CUSTOMER↑NUMBER"
> ELEMENT NAME            X(20) HEADING "CUSTOMER↑NAME"
> ELEMENT ADDRESS         X(26)
> ELEMENT CITY            X(14)
> ELEMENT STATE           X(2)
> ELEMENT PHONE           X(8)
> ELEMENT INVOICE         9(5) HEADING "INVOICE↑NUMBER"
> ELEMENT AMOUNT          9(8)V9(2) HEADING "INVOICE↑AMOUNT" &
>                           PICTURE "↑↑,↑↑↑.↑↑"
>
> RECORD INPUT1
>   ITEM NAME
>   ITEM NUMBER INTEGER
>   ITEM ADDRESS
>   ITEM CITY
>   ITEM STATE
>   ITEM PHONE
>
> RECORD INPUT2
>   ITEM NAME
>   ITEM NUMBER INTEGER
>   ITEM ADDRESS
>   ITEM CITY
>   ITEM STATE
>   ITEM PHONE
>   ITEM INVOICE INTEGER
>   ITEM AMOUNT REAL
>
> BUILD

END OF PROGRAM
```

Once the schema has been created the report can be defined. Because of the nature of the interface, the CHOOSE and SELECT commands available in QUIZ should not be used. Additionally, any sorting that is required should be done in RELATE instead of QUIZ. All SORT statements in QUIZ should be entered as SORTED.

```
:EDITOR

HP32201A.7.10 EDIT/3000  MON, MAR 22, 1982,   2:43 PM
(C) HEWLETT-PACKARD CO. 1981
/ADD
          ACCESS INPUT2

          DEFINE LOCATION STRING*30 = PACK(CITY+", "+STATE)

          REPORT &
            TAB 15 INVOICE &
            TAB 30 AMOUNT SCALE 2

          PAGE HEADING &
            "O U T S T A N D I N G   I N V O I C E S  " &
            "R E P O R T" SKIP 2

          SORTED ON NUMBER &
            HEADING &
                    NUMBER &
              TAB 10 NAME SKIP 1 &
              TAB 10 ADDRESS SKIP 1 &
              TAB 10 LOCATION SKIP 1 &
              TAB 10 PHONE SKIP 2 &
            FOOTING &
              TAB 10 "CUSTOMER TOTAL" &
              TAB 30 AMOUNT SUBTOTAL SCALE 2 SKIP 2

          GO
          EXIT
          //
 ...
/KEEP REPORT2
/EXIT

END OF PROGRAM
```

The GO and EXIT commands must be included in the report definition in order for the report to be produced correctly.

To generate the report, the appropriate data is SELECTed followed by the QUIZ command. The command must specify the name of the file containing the report as well as the name of the file in which QUIZ expects to find the information for the report.

```
:RELATE

RELATE/3000 V4.10A MON, MAR 22, 1982  2:57 PM (C) CRI

)OPEN FILE CUST
)OPEN FILE INVOICE
)SELECT CUST.@, INVOICE.INVNO, INVOICE.AMOUNT &
&)  BY NUMBER, INVNO &
&)  WHERE CUST.NUMBER=INVOICE.NUMBER AND ST="CA"
)QUIZ REPORT2; SOURCE=INPUT2
)
```

The resulting report is displayed on the following page.

```
O U T S T A N D I N G   I N V O I C E S   R E P O R T

  100   DEXMACH, INC.
        BOX 877 RD1
        SAN JOSE, CA
        800-2111

                 33                348.70
                105                 86.32
                106                 76.40
        CUSTOMER TOTAL             511.42

  400   CUPERCO
        10802 WILKINSON AVENUE
        CUPERTINO, CA
        257-8667

               2738                948.60
              10044                500.00
              23557                 37.00
        CUSTOMER TOTAL           1,485.60

  500   AMERICAN TIRE CO.
        7052 EL CAMINO REAL
        MOUNTAIN VIEW, CA
        941-0000

                727                  3.14
                747                   .97
               8663                 86.00
        CUSTOMER TOTAL              90.11

 1000   NATIONAL AIRLINES
        SAN FRANCISCO INTL AIRPORT
        BURLINGAME, CA
        UNLISTED

              10221                627.01
              10455                335.00
        CUSTOMER TOTAL             962.01
```

[range] RECOVER [TO filename[;options]]
[FOR condition]

Recovers previously deleted data from RELATE/3000 files.

range	Optional. If used, only records in the specified range are recovered. See the RANGE section.
TO filename	Optional. If specified, all options in the "TO filename" section needed to access the file must be appended whether or not the file is already open. If an output file is specified, all recovered lines will be copied into the file and will exist in an undeleted state. Lines in the current file will remain deleted.
FOR condition	Optional. If used, only records meeting the condition are recovered. See the EXPRESSION EVALUATION section.

The current file must have a current index of 0 (the line number) in order to perform a RECOVER.

Records cannot be recovered if DELETE=PHYSICAL was specified for the file using the MODIFY FILE command.

EXAMPLES:

User realizes that he/she may have deleted more lines than should have been, but doesn't recall what was done. RECOVER command can only re-instate lines that have been deleted but not REORGANIZEd or ERASEd. The file must be indexed by line number and be a RELATE/3000 file in order to perform a RECOVER.

```
)SET PATH INVOICE
)SET INDEX 0
INDEX #0 IS NOW THE CURRENT INDEX.
)PRINT

$LINE  INVNO NAME                        NUMBER ST    AMOUNT    TAX SALES

    1  10221 NATIONAL AIRLINES             1000 CA $ 627.01  88.07    36
    2  10455 NATIONAL AIRLINES             1000 CA $ 335.00  40.20    36
    3     33 DEXMACH, INC.                  100 CA $ 348.70  18.00    99
    4   2738 CUPERCO                        400 CA $ 948.60  32.40    36
    5  23557 CUPERCO                        400 CA $  37.00   3 00    86
    6  10044 CUPERCO                        400 CA $ 500.00  35 99    99
    7    105 DEXMACH, INC.                  100 CA $  86.32   4.81    83
    8    106 DEXMACH, INC                   100 CA $  76.40   1 10    87
   11    727 AMERICAN TIRE CO.              500 CA $   3.14   1 59    87
   13   8663 AMERICAN TIRE CO               500 CA $  86.00   1.32    36
```

```
10 LINES PRINTED.
)RECOVER
3 LINES RECOVERED.
)PRINT

$LINE  INVNO NAME                        NUMBER ST   AMOUNT    TAX SALES

    1  10221 NATIONAL AIRLINES             1000 CA $ 627.01  88.07    36
    2  10455 NATIONAL AIRLINES             1000 CA $ 335.00  40.20    36
    3     33 DEXMACH, INC.                  100 CA $ 348.70  18.00    99
    4   2738 CUPERCO                        400 CA $ 948.60  32.40    36
    5  23557 CUPERCO                        400 CA $  37.00   3.00    86
    6  10044 CUPERCO                        400 CA $ 500.00  35.99    99
    7    105 DEXMACH, INC.                  100 CA $  86.32   4.81    83
    8    106 DEXMACH, INC.                  100 CA $  76.40   1.10    87
    9  10002 ALEXANDER HALE & CO.           700 NJ $ 999.99   9.99    45
   10     33 PERFECT SOUND                  800 VA $ 677.77   3.33    83
   11    727 AMERICAN TIRE CO.              500 CA $   3.14   1.59    87
   12    747 AMERICAN TIRE CO.              500 CA $   0.97   0.00    83
   13   8663 AMERICAN TIRE CO.              500 CA $  86.00   1.32    36

13 LINES PRINTED.
)
```

```
RECOVER DATA FROM logfile [;INFORMATION]
                [FILE fileset]
```

Performs data and structural recovery operations on a data base from a log file.

FROM logfile	Required. Indicates the name of the log file containing the data. RELATE must be able to open the file with exclusive read access.
INFORMATION	Optional. When specified, the data files are not recovered. The information report generated by the recovery system is printed.
FILE fileset	Optional. If specified, the fileset must contain the names of the files that should be recovered. If not specified, all files are recovered. The fileset may contain actual file names or '@.group' or '@.@' which will recover all files in the indicated group or in the account. The file names must be separated by commas. The recovery system must be able to obtain exclusive access to all files.

For more information on recovery see the TRANSACTION PROCESSING section.

REDO [commandnumber]

Allows the editing of previous command lines.

commandnumber	Optional. If not specified, the first line of the previous command is made available for modifications. If specified, the item must be composed of a command number optionally including a decimal indicating the line number to be edited.

The following edit operations may be performed on the command line:

D	Deletes the character it is placed under. Multiple D's may be used at the same time.
I	Inserts the text following the I into the command line before the character the I is under.
R	Replaces the text in the command line with the text following the R.
S	Splits the command line into two lines. The character over the S will be the first character of the next line. The S command must be the last command on the line. Any other edits made to the line prior to the S are saved permanently and the subsequently displayed line will be returned to if a // is entered.
//	When entered after edits have been made on the command line, the original command line is returned. If entered when no edits have been made to the command line, the command in the edit buffer is not executed and a new command is requested.
#	Allows editing of the indicated line number of the command. A line may be added to the end of a command by referencing the next unused number. If a new line is added, the system appends a blank to the end of the previous line.

A carriage return causes the command line to be executed. The new command is assigned the next command number and is then displayed on the user's output device before execution. REDO commands and null commands are not saved.

EXAMPLES:

If an error is made in a prior command the command can be edited and then resubmitted. When the command is resubmitted, RELATE assigns a new number to the command which allows the newly created command to be edited as well.

```
)PRINT:S NAME, NUMBER, CITY &
&) FOR ST="CA"

NAME                    NUMBE CITY

HASLETREX INC.            200 SUNNYVALE
DEXMACH, INC.             100 SAN JOSE
CUPERCO                   400 CUPERTINO
AMERICAN TIRE CO.         500 MOUNTAIN VIEW
FINCH, FINCH, & OTTO      600 LOS ANGELES
NATIONAL AIRLINES        1000 BURLINGAME

6 LINES PRINTED.
)REDO
PRINT:S NAME, NUMBER, CITY
        RNUMBER, NAME
PRINT S NUMBER, NAME, CITY
 .1
 FOR ST="CA"
I"CUSTOMERS IN CALIFORNIA"
"CUSTOMERS IN CALIFORNIA" FOR ST="CA"

)PRINT:S NUMBER, NAME, CITY &
&)"CUSTOMERS IN CALIFORNIA" FOR ST="CA"

CUSTOMERS IN CALIFORNIA

NUMBE NAME                 CITY

  200 HASLETREX INC.       SUNNYVALE
  100 DEXMACH, INC         SAN JOSE
  400 CUPERCO              CUPERTINO
  500 AMERICAN TIRE CO.    MOUNTAIN VIEW
  600 FINCH, FINCH, & OTTO LOS ANGELES
 1000 NATIONAL AIRLINES    BURLINGAME

6 LINES PRINTED.
)
```

REORGANIZE FILE filename
[;RESERVE=records]

Removes deleted records and rewrites the index structure of the given RELATE/3000 file.

filename	Required. The name of the RELATE/3000 file that should be reorganized. The file must not currently be open. The user must have been the creator of the file and the file must exist in the log on account. If the file contains a lockword, the lockword should be placed on the filename. If the lockword is not placed on the file name the new file will not contain the lockword.
RESERVE	Optional. If used, this keyword must be followed by a value which indicates the number of records above the count of records currently existing in the file which should be allowed in the new file. If not specified, the new limit will be 20% beyond the current record count, or a total of 100, whichever is greater. Sectors are allocated only as the addition of data requires them, so the file can be given a large reserve without wasting disc space.

The command functions by creating an open temporary file and copying the source file into it. After the file is copied, the index structure is created. The original file is then purged and the new file is renamed and saved in the same domain. If the new file cannot be saved in the permanent domain it will be saved in the temporary domain. Care should be exercised if this command is used in a job as loss of data may result if the file cannot be correctly saved.

The REORGANIZE command will not maintain the original record number ($LINE) sequence.

EXAMPLES:

If a RELATE file's records are logically deleted (made invisible to the user) instead of physically deleted (enabled with the MODIFY FILE command), they remain physically in the file, which enables the RECOVER command to recover them.

When the user wishes to physically remove these deleted lines, renumber the file, adjust the number of records available in the file, or rewrite the indexes, the REORGANIZE command should be used.

```
)SHOW CURRENT

FILE NAME                 =INVOICE.DOC79.RDB

   FILE (OR SET) NAME     =INVOICE
   CURRENT RECORDS        =13 (EOF=13)
   MAXIMUM RECORDS        =100
```

```
    FILE TYPE                   =RELATE/3000
    DISPOSITION,RETENTION       =PERMANENT,PERMANENT
    ACCESS MODE                 =EXCLUSIVE,NOLOCK
    CLUSTERING INDEX            =0
    DATA COMPRESSION            =YES
    CRASHPROOF ACCESS           =YES
    DELETE                      =LOGICAL
    SCAN                        =10
    RECORDS CAN BE              :DELETED,UPDATED,ADDED

)PRINT

$LINE  INVNO NAME                       NUMBER ST    AMOUNT     TAX SALES

    1  10221 NATIONAL AIRLINES            1000 CA $ 627.01   40.76    36
    2  10455 NATIONAL AIRLINES            1000 CA $ 335.00   21.77    36
    3     33 DEXMACH, INC.                 100 CA $ 348.70   22.67    99
    4   2738 CUPERCO                       400 CA $ 948.60   61.66    36
    5  23557 CUPERCO                       400 CA $  37.00    2.40    86
    6  10044 CUPERCO                       400 CA $ 500.00   32.50    99
    7    105 DEXMACH, INC.                 100 CA $  86.32    5.61    83
    8    106 DEXMACH, INC.                 100 CA $  76.40    4.97    87
    9  10002 ALEXANDER HALE & CO.          700 NJ $ 999.99   90.00    45
   10     33 PERFECT SOUND                 800 VA $ 677.77    0.00    83
   11    727 AMERICAN TIRE CO              500 CA $   3.14    0.20    87
   12    747 AMERICAN TIRE CO.             500 CA $   0.97    0.06    83
   13   8663 AMERICAN TIRE CO.             500 CA $  86.00    5.59    36

13 LINES PRINTED.
)DELETE FOR NUMBER=400 OR NUMBER=800
4 LINES DELETED.
```

Even though 4 records have been deleted, a look at the file's structure shows that there are still 13 records physically in the file. A REORGANIZE command is the only way to physically remove those records.

In the first reorganize, we want the file to take up as little space as possible when we are finished, so we give it a RESERVE=0 keyword. The "MAXIMUM RECORDS" shows that we now have room for only 9 records, which was the number of non-deleted records in the file before the REORGANIZE was performed.

If we later change our minds and wish to increase the available file area, we can REORGANIZE the file again. We can either specify the number of additional records to allow or let RELATE use its default sizes.

```
)SHOW CURRENT

FILE NAME                =INVOICE.DOC79.RDB

    FILE (OR SET) NAME       =INVOICE
    CURRENT RECORDS          =9 (EOF=13)
    MAXIMUM RECORDS          =100
    FILE TYPE                =RELATE/3000
    DISPOSITION,RETENTION    =PERMANENT,PERMANENT
    ACCESS MODE              =EXCLUSIVE,NOLOCK
    CLUSTERING INDEX         =0
    DATA COMPRESSION         =YES
    CRASHPROOF ACCESS        =YES
    DELETE                   =LOGICAL
    SCAN                     =10
    RECORDS CAN BE           :DELETED,UPDATED,ADDED

)CLOSE FILE INVOICE
)REORGANIZE FILE INVOICE;RESERVE=0
9 RECORDS REMAIN.
)OPEN FILE INVOICE
)SHOW CURRENT

FILE NAME                =INVOICE.DOC79.RDB

    FILE (OR SET) NAME       =INVOICE
    CURRENT RECORDS          =9 (EOF=9)
    MAXIMUM RECORDS          =9
    FILE TYPE                =RELATE/3000
    DISPOSITION,RETENTION    =PERMANENT,PERMANENT
    ACCESS MODE              =EXCLUSIVE,NOLOCK
    CLUSTERING INDEX         =0
    DATA COMPRESSION         =YES
    CRASHPROOF ACCESS        =YES
    DELETE                   =LOGICAL
    SCAN                     =10
    RECORDS CAN BE           :DELETED,UPDATED,ADDED

)CLOSE FILE INVOICE
)REORGANIZE FILE INVOICE
9 RECORDS REMAIN.
)OPEN FILE INVOICE
```

```
)SHOW CURRENT

FILE NAME              =INVOICE.DOC79.RDB

   FILE (OR SET) NAME     =INVOICE
   CURRENT RECORDS        =9 (EOF=9)
   MAXIMUM RECORDS        =100
   FILE TYPE              =RELATE/3000
   DISPOSITION,RETENTION  =PERMANENT,PERMANENT
   ACCESS MODE            =EXCLUSIVE,NOLOCK
   CLUSTERING INDEX       =0
   DATA COMPRESSION       =YES
   CRASHPROOF ACCESS      =YES
   DELETE                 =LOGICAL
   SCAN                   =10
   RECORDS CAN BE         :DELETED,UPDATED,ADDED

)
```

SELECT [targetlist [[SORT] [UNIQUE] BY keylist] [WHERE condition]]

Indicates what information will be available to the following command. All portions of the command are optional. If none are used, the current SELECT command is dropped and the data accessed reverts to the most recently set path (if any).

targetlist	The target list must be in the following format: name1[=expression][,name2[=expression]]... The target list indicates what fields should be returned and what values they should assume. Each field name can be qualified with a path name to indicate the source file of the field's value when an expression is not given. If a path name is not used, the field must exist in the file referenced by the current path. If an expression is specified, a new field name must be specified. The field's type is determined by the type of the expression. A fieldname may not be duplicated in the target list.
SORT	Optional. If used, the BY keylist clause must also be included. This removes internal restrictions on how the BY clause is evaluated and allows RELATE to choose the most efficient manner of evaluating the request, which may include ignoring existing indexes and creating a new temporary index. This keyword should not be used to create updatable views.
UNIQUE	Optional. If used, the BY keylist clause must also be included. The system will only return to the user the records that contain unique values in the keylist.
BY	Optional. If used, a keylist must follow the "BY" keyword. The results of the selection will be returned sorted by the keylist. The keylist is a list containing the names of fields (without path names) in the targetlist. The results of the selection will be sorted and returned by these fields. If a ":D" is appended to any fieldname in the keylist, that field will be sorted in descending order. Ranges can be specified on commands following the SELECT command only if a BY clause is specified.
WHERE	Optional. If used, this keyword must be followed by an expression that, when evaluated, determines if the records used to create the condition should be returned to the user. See the EXPRESSION EVALUATION section for further details. An expression may contain fieldnames from any currently open file optionally qualified by a path name. If a path name is not used, the field must exist in the file referenced by the current path. The expression may be composed of virtually any number of alphabetic or algebraic comparisons of fields or constants.

The SELECT command is checked for syntax but is not executed until a subsequent command reads a record from a file.

To return all fields from a file an atsign ("@") may be used after a path name in the targetlist. The atsign will add to the targetlist all fieldnames that exist in the file referenced by the path specified that do not yet exist in the targetlist. For example, SELECT A.@, B.@ would return all fields in A and those fields from B that do not duplicate those in A.

Once a file has been opened, the user may perform operations on or with the contents of the file. The information used by a command is determined by the current path or by a SELECT command.

RELATE/3000 makes many assumptions to simplify user interaction. The current path assumption is of primary importance. The current path indicates to RELATE the file from which information should be returned. In the absence of a SELECT command, only records from the file referenced by the current path can be used.

To combine information from more than one file, the user may issue a SELECT command. This command temporarily combines information into a single file. The file may never physically exist but to the operation of the following command it appears to have been created. Usually, a SELECT command is used to combine information for subsequent read operations. It is also possible in some cases to add, delete, or update the contents of a SELECTion. For more information on this please see the Update Views section.

After the SELECT command has been given, only the fields explicitly requested in the targetlist can be referenced in the following command. For example:

```
    )OPEN FILE INVOICES
    )PRINT
```

This lists the entire contents of the INVOICES file.

```
    )SELECT C_NO, INV_NO, AMOUNT WHERE AMOUNT>100.00
    )PRINT
```

This lists only the C_NO, INV_NO, and AMOUNT fields in records containing an AMOUNT greater than 100.00.

AGGREGATES

An aggregate is a function that returns summary information about a file. The following aggregates are defined:

$AVG	Returns the arithmetic average of the qualified records.
$AVGU	Returns the average of qualified records containing unique keyfield values.
$COUNT	Returns the count of the number of qualified records.
$COUNTU	Returns the count of the number of records containing unique keyfield values.
$MAX	Returns the maximum value of the qualified records.
$MEDIAN	Returns the median of the qualified records.
$MIN	Returns the minimum value of the qualified records.
$STD_DEV	Returns the standard deviation of the qualified records.
$SUM	Returns the sum of the expressions of the qualified records.
$SUMU	Returns the sum of the unique expressions of the qualified records.

The format of an aggregate is:

```
aggname ([expression] [BY keyfields] [WHERE condition])
```

Aggregates may be used in the condition clause of another aggregate, the expression portion of an aggregate, the condition of a SELECT command, or the expression portion of the targetlist. An aggregate may not appear in a keylist or take the place of a fieldname in the targetlist.

The data type of an aggregate is determined by the expression clause. That is, if the expression results in a LONG number, the aggregate will represent a LONG number. The $COUNT and $COUNTU aggregates always return a DOUBLE quantity.

aggname	Required. One of $AVG, $AVGU, $COUNT, $COUNTU, $MAX, $MEDIAN, $MIN, $STD_DEV, $SUM, $SUMU.
expression	Optional. A numeric or alphabetic expression that is used to calculate the value of the aggregate for each record qualified by the WHERE clause. An alphabetic field can only be used in the $COUNT, $COUNTU, $MAX, and $MIN aggregates.
BY	Optional. Indicates that the results of the aggregate should be generated for a specific field or fields. If a BY clause is not included, the aggregate applies to the entire file. (See the examples on the following pages.)
WHERE	Optional. Indicates that the records to be used in the aggregate will meet a specified condition.

EXAMPLES:

Consider the following three files: EMP, DEPT, and MEMBER. These describe employees, departments, and the membership of employees in each department. An entry in the MEMBER file asserts that the employee with employee number E_NO is in the department numbered D_NO.

```
)SET PATH EMP
)PRINT

$LINE E_NO NAME      ADDRESS              STATE SALARY

    1    1 BOB       1 MAIN ST            CA     25000
    2    2 MARY      27 SPRUCE ST         CA     35000
    3    3 SAM       4 LOCKWOOD CT        MA     25000
    4    4 GEORGE    17 FERN DR           MA     35000
    5    5 HORACE    1700 EXECUTIVE PL    NY     75000
    6    6 KIM       30 BOX CT #3         CA     37000

6 LINES PRINTED
)SET PATH DEPT
)PRINT

$LINE D_NO DNAME               MGR_E_NO LOCATION

    1    1 MANUFACTURING              4 BLDG 6
    2    2 ACCOUNTING                 5 NEW YORK
    3    3 RESEARCH                   2 MOUNTAIN VIEW
    4    4 PHANTOM                    0 NOWHERE

4 LINES PRINTED
)SET PATH MEMBER
)PRINT

$LINE E_NO D_NO

    1    1    1
    2    1    2
    3    2    3
    4    3    1
    5    4    1
    6    5    2
    7    6    3

7 LINES PRINTED.
```

The SELECT command can be used to perform a wide variety of functions and answer many specific kinds of questions that are very difficult to answer with conventional database systems. This is a simple selection: print the names and addresses of all employees.

```
)SELECT EMP.NAME, EMP.ADDRESS
)PRINT

NAME        ADDRESS

BOB         1 MAIN ST
MARY        27 SPRUCE ST
SAM         4 LOCKWOOD CT
GEORGE      17 FERN DR
HORACE      1700 EXECUTIVE PL
KIM         30 BOX CT #3

6 LINES PRINTED.
```

Here is a restriction on a relation: print the names of all employees who make more than $35,000 a year.

```
)SELECT EMP.NAME WHERE EMP.SALARY>35000
)PRINT

NAME

HORACE
KIM

2 LINES PRINTED.
```

One can also join two or more files: print the employee name and department names of all employees.

```
)SELECT EMP.NAME, DEPT.DNAME WHERE EMP.E_NO=MEMBER.E_NO AND &
&)MEMBER.D_NO=DEPT.D_NO
)PRINT

NAME        DNAME

BOB         MANUFACTURING
SAM         MANUFACTURING
GEORGE      MANUFACTURING
BOB         ACCOUNTING
HORACE      ACCOUNTING
MARY        RESEARCH
KIM         RESEARCH

7 LINES PRINTED.
```

The output can also be sorted by one or more keys, as in this example. Print the department name and the members of each department in alphabetical order.

```
)SELECT DEPT.DNAME, EMP.NAME BY DNAME, NAME WHERE &
&)EMP.E_NO=MEMBER.E_NO AND MEMBER.D_NO=DEPT.D_NO
WARNING: BY CLAUSE WILL CAUSE OUTPUT TO BE COMPUTED THEN SORTED.
)PRINT DNAME:B:S:1

DNAME               NAME

ACCOUNTING          BOB
                    HORACE

MANUFACTURING       BOB
                    GEORGE
                    SAM

RESEARCH            KIM
                    MARY

7 LINES PRINTED.
```

To choose only one of several records that have duplicate values in a given field, the UNIQUE clause can be used. Print the different salaries of all employees.

```
)SELECT EMP.SALARY UNIQUE BY SALARY
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY BY CLAUSE.
)PRINT

SALARY

 25000
 35000
 37000
 75000

4 LINES PRINTED.
```

Notice that the salaries 25000 and 35000 were not output twice. Now, find all employees who make more than their managers.

Here, we need to open file EMP twice with different path names. This is because we are comparing data in the EMP file to itself. In other words, we are using the EMP file for two different things; first, to find information about employees (path E1); and secondly, to find information about their managers (who also happen to be employees; path E2).

```
)OPEN FILE EMP; PATH=E1
)OPEN FILE EMP; PATH=E2
)OPEN FILE MEMBER; PATH=M
)OPEN FILE DEPT; PATH=D
)SELECT E1.NAME WHERE E1.SALARY>E2.SALARY AND &
&)E1.E_NO=M.E_NO AND M.D_NO=D.D_NO AND D.MGR_E_NO=E2.E_NO
)PRINT

NAME

KIM

1 LINE PRINTED.
```

Statistical information may also be acquired about the database or used in restricting the queries. Aggregates play the role of aggregating or collecting data from several records into a single number. With a BY clause on an aggregate you can separate the records into groups. Print the average salary of all employees.

```
)SELECT AVG_SAL=$AVG(EMP.SALARY)
)PRINT

AVG_SA

 38667

1 LINE PRINTED.
```

Find how may employees there are.

```
)SELECT COUNT=$COUNT(EMP.E_NO)
)PRINT

 COUNT

     6

1 LINE PRINTED.
```

With the $COUNT aggregate any field can be used as the argument. Restrictions can be placed on which records are used in the aggregates. Find the sum of the salaries and the number of employees making less than $36,000.

```
)SELECT SUM=$SUM(EMP.SALARY WHERE EMP.SALARY<36000), &
&)COUNT=$COUNT(WHERE EMP.SALARY<36000)
)PRINT

   SUM   COUNT

120000       4

1 LINE PRINTED.
```

The "unique" aggregates ($COUNTU, $SUMU, $AVGU) only use unique values of the field when computing their results. Find how many different salaries there are.

```
)SELECT COUNT=$COUNTU(EMP.SALARY)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

 COUNT

     4

1 LINE PRINTED.
```

Notice that with the $COUNTU aggregate the field in the argument is important, while in the $COUNT aggregate the field in the aggregate is just needed to specify which file to count. In $COUNTU, the field is also used to determine uniqueness. Aggregates may also be used with restrictions: Print the names of all employees who make less than the average salary.

```
)SELECT EMP.NAME WHERE EMP.SALARY<$AVG(EMP.SALARY)
)PRINT

NAME

BOB
MARY
SAM
GEORGE
KIM

5 LINES PRINTED.
```

Print the names of the employee(s) who have the highest salary.

```
)SELECT EMP.NAME WHERE EMP.SALARY=$MAX(EMP.SALARY)
)PRINT

NAME

HORACE

1 LINE PRINTED.
```

Several conditions can be specified in the WHERE clause using logical operators. For example, find everyone who lives in either New York or California and earns more than the median salary.

```
)SELECT EMP.NAME WHERE (EMP.STATE="NY" OR EMP.STATE="CA") &
&)        AND EMP.SALARY>$MEDIAN(EMP.SALARY)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

NAME

KIM
HORACE

2 LINES PRINTED.
```

All the aggregates so far used are simple aggregates that only return a single value. If a BY clause is placed on the aggregate, the records are grouped and the aggregates are performed successively on each group. For example, print the minimum salary of employees in each state.

```
)SELECT EMP.STATE, SAL=$MIN(EMP.SALARY BY EMP.STATE)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

STATE     SAL

CA      25000
MA      25000
NY      75000

3 LINES PRINTED.
```

Find how many people live in each state. A common approach to this type of problem results in a command of the format:

```
      SELECT CA=$COUNT(WHERE EMP.STATE="CA"), &
             MA=$COUNT(WHERE EMP.STATE="MA"), &
             NY=$COUNT(WHERE EMP.STATE="NY")
```

which will work (provided that it doesn't run out of memory in attempting to evaluate all the different WHERE clauses), but has the disadvantage that the user has to know all the states (or job categories or whatever is being counted) and runs the risk of forgetting something. This solution is much faster.

```
)SELECT EMP.STATE, NUM=$COUNT(BY EMP.STATE)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

STATE     NUM

CA          3
MA          2
NY          1

3 LINES PRINTED
```

The aggregates with BY clauses can also be used with conditions. Print the employees in each state with the minimum salaries.

```
)SELECT EMP.STATE, EMP.NAME, EMP.SALARY WHERE EMP.SALARY= &
&)$MIN(EMP.SALARY BY EMP.STATE)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

STATE NAME         SALARY

CA    BOB           25000
MA    SAM           25000
NY    HORACE        75000

3 LINES PRINTED.
```

Some potentially difficult problems can be easily solved with aggregates. The next two problems are cases of what is sometimes referred to as a "semi-outer-join". For example: print the names of departments with no people in them.

```
)SELECT DEPT.DNAME WHERE $COUNT(BY MEMBER.D_NO)=0 AND &
&)DEPT.D_NO=MEMBER.D_NO
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

DNAME

PHANTOM

1 LINE PRINTED.
```

A similar problem is to list all the departments and, if there is a manager, the manager's name, otherwise print blanks.

```
)SELECT DEPT.DNAME, MGRNAME=$MAXIMUM($MAX(EMP.NAME &
&)            BY DEPT.MGR_E_NO WHERE EMP.E_NO=DEPT.MGR_E_NO)," ")
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

DNAME              MGRNAME

MANUFACTURING      GEORGE
ACCOUNTING         HORACE
RESEARCH           MARY
PHANTOM

4 LINES PRINTED
```

Aggregates can also be used with joins, as in: Print the sum of the salaries for each department.

```
)SELECT DEPT.DNAME, SAL=$SUM(EMP.SALARY BY DEPT.D_NO WHERE &
&)DEPT.D_NO=MEMBER.D_NO AND MEMBER.E_NO=EMP.E_NO)
WARNING: TEMPORARY INDEX WILL BE CREATED TO SATISFY AGGREGATE BY
CLAUSE.
)PRINT

DNAME                SAL

MANUFACTURING      85000
ACCOUNTING        100000
RESEARCH           72000
PHANTOM                0

4 LINES PRINTED.
)
```

SET INDEX **number/fieldlist**

Makes the indicated index the current index.

number	Optional. The index number of the index desired to become the current index.
fieldlist	Optional. If specified, the first index that contains at least the fields specified is selected. If any of the fields in the fieldlist are descending in the index, a local "D" switch must be appended to the fieldname.

Either an index number or a fieldlist must be specified.

To set the index back to the line number index, a "SET INDEX 0" command should be used.

EXAMPLES:

List all the existing indexes for the file. Set an index by the number of the index and print the file. Then set an index by the first key element and print again.

```
)SET PATH INVOICE
)SHOW INDEX

FILE NAME              =INVOICE.DOC79.RDB

INDEX 1 BY NAME,NUMBER
  WORDS IN KEY (+NODE) : 13 (+1)
  DISTRIBUTION         : 2,2
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES . 1

INDEX 3 BY NUMBER,INVNO
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 2,1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1

INDEX 2 BY AMOUNT:D (CURRENT INDEX)
  WORDS IN KEY (+NODE) : 4 (+1)
  DISTRIBUTION         : 1
  LEVELS IN TREE       : 1
  NUMBER OF USED NODES : 1
```

```
)SET INDEX 3
INDEX #3 IS NOW THE CURRENT INDEX.
)PRINT

NUMBER  INVNO NAME                     ST    AMOUNT     TAX SALES

   100     33 DEXMACH, INC.            CA $ 348.70   18.00     99
   100    105 DEXMACH, INC.            CA $  86.32    4.81     83
   100    106 DEXMACH, INC.            CA $  76.40    1.10     87
   400   2738 CUPERCO                  CA $ 948.60   32.40     36
   400  10044 CUPERCO                  CA $ 500.00   35.99     99
   400  23557 CUPERCO                  CA $  37.00    3.00     86
   500    727 AMERICAN TIRE CO.        CA $   3.14    1.59     87
   500    747 AMERICAN TIRE CO.        CA $   0.97    0.00     83
   500   8663 AMERICAN TIRE CO.        CA $  86.00    1.32     36
   700  10002 ALEXANDER HALE & CO.     NJ $ 999.99    9.99     45
   800     33 PERFECT SOUND            VA $ 677.77    3.33     83
  1000  10221 NATIONAL AIRLINES        CA $ 627.01   88.07     36
  1000  10455 NATIONAL AIRLINES        CA $ 335.00   40.20     36

13 LINES PRINTED.
)SET INDEX AMOUNT:D
INDEX #2 IS NOW THE CURRENT INDEX.
)PRINT

   AMOUNT   INVNO NAME                     NUMBER ST     TAX SALES

$ 999.99    10002 ALEXANDER HALE & CO.       700 NJ    9.99     45
$ 948.60     2738 CUPERCO                    400 CA   32.40     36
$ 677.77       33 PERFECT SOUND              800 VA    3.33     83
$ 627.01    10221 NATIONAL AIRLINES         1000 CA   88.07     36
$ 500.00    10044 CUPERCO                    400 CA   35.99     99
$ 348.70       33 DEXMACH, INC.              100 CA   18.00     99
$ 335.00    10455 NATIONAL AIRLINES         1000 CA   40.20     36
$  86.32      105 DEXMACH, INC.              100 CA    4.81     83
$  86.00     8663 AMERICAN TIRE CO.          500 CA    1.32     36
$  76.40      106 DEXMACH, INC.              100 CA    1.10     87
$  37.00    23557 CUPERCO                    400 CA    3.00     86
$   3.14      727 AMERICAN TIRE CO.          500 CA    1.59     87
$   0.97      747 AMERICAN TIRE CO.          500 CA    0.00     83

13 LINES PRINTED.
```

Index number zero is always by line number. The user can return to line number indexing by using "SET INDEX 0".

```
)SET INDEX 0
INDEX #0 IS NOW THE CURRENT INDEX.
)PRINT

$LINE  INVNO NAME                     NUMBER ST   AMOUNT     TAX SALES

    1  10221 NATIONAL AIRLINES          1000 CA $ 627.01   88.07    36
    2  10455 NATIONAL AIRLINES          1000 CA $ 335.00   40.20    36
    3     33 DEXMACH, INC.               100 CA $ 348.70   18.00    99
    4   2738 CUPERCO                     400 CA $ 948.60   32.40    36
    5  23557 CUPERCO                     400 CA $  37.00    3.00    86
    6  10044 CUPERCO                     400 CA $ 500.00   35.99    99
    7    105 DEXMACH, INC.               100 CA $  86.32    4.81    83
    8    106 DEXMACH, INC.               100 CA $  76.40    1.10    87
    9  10002 ALEXANDER HALE & CO.        700 NJ $ 999.99    9.99    45
   10     33 PERFECT SOUND               800 VA $ 677.77    3.33    83
   11    727 AMERICAN TIRE CO.           500 CA $   3.14    1.59    87
   12    747 AMERICAN TIRE CO.           500 CA $   0.97    0.00    83
   13   8663 AMERICAN TIRE CO.           500 CA $  86.00    1.32    36

13 LINES PRINTED.
)
```

SET PATH pathname

Makes current a file that has already been opened but may not be current due to some other OPEN, CLOSE, or SET PATH command having been issued. The current path indicates to RELATE the file from which information should be returned.

pathname	Required. This is the pathname of a file that has previously been opened but might no longer be the current file.

EXAMPLES:

Once a file has been opened, RELATE maintains speedy access to the file so that it need not be reopened each time access is desired for the user. This is done by maintaining a PATH name for each open file. To access one of these paths and make the file "current", use the SET PATH command.

```
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

INVOICE            INVOICE            INVOICE.DOC79.RDB (CURRENT PATH)
CUST1              CUST1              CUST1.DOC79.RDB
MPECUST            MPECUST            MPECUST.DOC79.RDB
CUST               CUST               CUST.DOC79.RDB

)SET PATH CUST
)SHOW PATH

PATH NAME          FILE NAME          DATABASE NAME

INVOICE            INVOICE            INVOICE.DOC79.RDB
CUST1              CUST1              CUST1.DOC79.RDB
MPECUST            MPECUST            MPECUST.DOC79.RDB
CUST               CUST               CUST.DOC79.RDB (CURRENT PATH)

)
```

```
SHOW
[,ALL]
[,BOUND]
[,CURRENT]
[,FILES]
[,FORMAT]
[,INDEX]
[,KEY]
[,LEVEL]
[,PATHS]
[,RECORD]
[,SELECT]
[,SETS]
[,STRUCTURE]
```

Displays information about open files.

:P	Optional global switch. If included, the requested information is directed to the file RDBLIST. If RDBLIST cannot be opened, the output is directed to the device class "LP".
ALL	If specified, enables BOUND, CURRENT, INDEX, and STRUCTURE.
BOUND	If specified, displays any variables bound to the current cursor. This option will only display information when used from the Host Language Interface routines.
CURRENT	If specified, displays status information on the current file.
FILES	If specified, shows all open files and the locking status on each.
FORMAT	If specified, displays any special print formats along with the basic structure of the current file.
INDEX	If specified, the current file's indexes are displayed. Descending fields are indicated by local "D" switches. The current index is labeled as such.
KEY	If specified, displays a key to the information requested.
LEVEL	If specified, displays the data entry level along with the basic structure of the current file.
PATHS	If specified, displays a list of paths open in the cursor and the files with which these paths are associated.
RECORD	If specified, displays positions of the fields in the record and internal field numbers along with the basic structure of the current file.

SELECT	If specified, displays the order in which files are searched to retrieve the results from the current SELECT command.
SETS	If specified, the current path must reference a set in an IMAGE database. The names of the other accessible sets are displayed.
STRUCTURE	If specified, enables LEVEL, FORMAT, and RECORD.

If no specific requests are made, SHOW displays only the basic structure (RECORD) of the current file.

If no requests are specified, or if any request other than FILES or PATHS is made, a current file must exist.

EXAMPLES:

The SHOW command displays information about open files.

```
)SHOW PATHS

PATH NAME           FILE NAME           DATABASE NAME

INVOICE             INVOICE             INVOICE.DOC79.RDB (CURRENT PATH)
CUST1               CUST1               CUST1.DOC79.RDB
MPECUST             MPECUST             MPECUST.DOC79.RDB
CUST                CUST                CUST.DOC79.RDB

)SHOW CURRENT

FILE NAME                     =INVOICE.DOC79.RDB

    FILE (OR SET) NAME        =INVOICE
    CURRENT RECORDS           =13 (EOF=13)
    MAXIMUM RECORDS           =100
    FILE TYPE                 =RELATE/3000
    DISPOSITION,RETENTION     =PERMANENT,PERMANENT
    ACCESS MODE               =EXCLUSIVE,NOLOCK
    CLUSTERING INDEX          =0
    DATA COMPRESSION          =YES
    CRASHPROOF ACCESS         =YES
    DELETE                    =LOGICAL
    SCAN                      =10
    RECORDS CAN BE            :DELETED,UPDATED,ADDED
```

```
)SHOW

FILE NAME               =INVOICE.DOC79.RDB

             T
             Y PRINT  INT
NAME         P  LEN  SIZE

INVNO        I   6     1W
NAME         A  20    20B
NUMBER       I   6     1W
ST           A   2     2B
AMOUNT       R   8.2   2W
TAX          R   6.2   2W
SALES_MAN    I   5     1W

PRINT LINE WIDTH = 65 CHARACTERS.
)SHOW FORMAT, LEVEL, KEY

FILE NAME               =INVOICE.DOC79.RDB

             T            C            L
             Y PRINT      O            E  INT
NAME         P  LEN  $    M SPECIAL    V SIZE

INVNO        I   6                     4   1W
NAME         A  20                     3  20B
NUMBER       I   6                     3   1W
ST           A   2                     3   2B
AMOUNT       R   8.2 FX                4   2W
TAX          R   6.2                   4   2W
SALES_MAN    I   5                     4   1W

PRINT LINE WIDTH = 65 CHARACTERS.
************************ SHOW KEY ***********************************
*  TYP        = data type                                            *
*     A = alphabetic    D = double integer P = packed decimal        *
*     Z = zoned decimal R = real number    U = unsigned              *
*     I = integer       L = long number                              *
*  PRINT LEN = print length of field [.decimal places]               *
*  INT SIZE   = internal size of field in bytes(B) or words(W)       *
*  $          = dollar sign print    FX=fixed, FL=float              *
*  COM        = comma print                                          *
*  SPECIAL    = special print format.  Either DATE or numeric FORM   *
*     1 = forced leading sign  4 = () around negative                *
*     2 = trailing sign        5 = CR for negative                   *
*     3 = forced trailing sign 6 = CR for negative, DB for positive*
*  LEV        = data entry level                                     *
**********************************************************************
)
```

[range] SORT BY keylist TO filename[;options]
[FOR condition]

Sorts a data file by a keylist.

range	Optional. If used, only records in the specified range are sorted. See the RANGE section.
keylist	Required. A list of fields by which the file will be sorted. All fields listed must exist in the current file but need not exist in the output file.
LOCAL	Optional switches which, if used, are appended to items in the keylist.
	:A Sorts the field in ascending order (default).
	:D Sorts the field in descending order.
TO filename	Required. File to which data is output in sorted order. All options listed in the "TO filename" section needed to open the file must be appended to the filename, whether or not the file is already open.
FOR condition	Optional. If used, only records meeting the condition are sorted. See the EXPRESSION EVALUATION section.

The SORT command is a shorthand method of entering a SELECT for the current file with a BY clause followed by a COPY command.

EXAMPLES:

Sort by AMOUNT all records where the AMOUNT is greater than or equal to $500.00 and write them to the AMT file. Since the file doesn't exist, it is created by RELATE/3000 as a permanent RELATE file.

```
)SET PATH INVOICE
)PRINT

NUMBER   INVNO  NAME                     ST    AMOUNT      TAX  SALES

   100      33  DEXMACH, INC.            CA  $ 348.70    18.00     99
   100     105  DEXMACH, INC.            CA  $  86.32     4.81     83
   100     106  DEXMACH, INC.            CA  $  76.40     1.10     87
   400    2738  CUPERCO                  CA  $ 948.60    32.40     36
   400   10044  CUPERCO                  CA  $ 500.00    35.99     99
   400   23557  CUPERCO                  CA  $  37.00     3.00     86
   500     727  AMERICAN TIRE CO.        CA  $   3.14     1.59     87
   500     747  AMERICAN TIRE CO.        CA  $   0.97     0.00     53
   500    8663  AMERICAN TIRE CO.        CA  $  86.00     1.32     35
   700   10002  ALEXANDER HALE & CO.     NJ  $ 999.99     9.99     45
   800      33 PERFECT SOUND          VA $ 677.77   3.33    83
  1000   10221 NATIONAL AIRLINES      CA $ 627.01  88.07    36
  1000   10455 NATIONAL AIRLINES      CA $ 335.00  40.20    36

13 LINES PRINTED.
)SORT BY AMOUNT TO AMT FOR AMOUNT>=500
THE "AMT" FILE HAS BEEN CREATED AS A PERMANENT RELATE/3000 FILE.
5 LINES SORTED.
)OPEN FILE AMT
)PRINT

$LINE  INVNO NAME                     NUMBER ST   AMOUNT    TAX SALES

     1 10044 CUPERCO                     400 CA $ 500.00  35.99    99
     2 10221 NATIONAL AIRLINES          1000 CA $ 627.01  88.07    36
     3    33 PERFECT SOUND               800 VA $ 677.77   3.33    83
     4  2738 CUPERCO                     400 CA $ 948.60  32.40    36
     5 10002 ALEXANDER HALE & CO.        700 NJ $ 999.99   9.99    45

5 LINES PRINTED.
)
```

[range] SUM [fieldlist] [FOR condition]

Obtains the sum of one or more fields.

range	Optional. If used, only records in the specified range will be summed. See the RANGE section.
:A	Optional global switch. Prints the averages as well as the sums of the indicated fields.
fieldlist	Optional list of fields to be summed. If specified, it must contain only fields with a numeric data type. If not specified, all numeric fields not having a date format will be summed. Groups of fields may be specified using the pattern-matching feature.
LOCALS	Optional switch appearing on any fields in the fieldlist.
	:A Prints the average of the field as well as the sum.
FOR condition	Optional. If used, only records meeting the condition will be summed. See the EXPRESSION EVALUATION section.

If an average is printed, it will contain two more decimal places than the field that was summed.

The sum and average are calculated using long arithmetic.

EXAMPLES:

Obtain a total for AMOUNT and TAX fields in the INVOICE file for NUMBERs of 500. Also compute the average tax.

```
)SET PATH INVOICE
)SET INDEX 3
INDEX #3 IS NOW THE CURRENT INDEX
)500 PRINT

NUMBER  INVNO NAME                  ST      AMOUNT     TAX  SALES

   500    727 AMERICAN TIRE CO.     CA $      3.14    1.59     87
   500    747 AMERICAN TIRE CO.     CA $      0.97    0.00     83
   500   8663 AMERICAN TIRE CO.     CA $     86.00    1.32     36

3 LINES PRINTED
```

```
)500 SUM AMOUNT, TAX:A

   FIELD              SUM                  AVERAGE

    AMOUNT$                  90.11
       TAX                    2.91                  0.9700

3 LINES SUMMED.
```

This can also be done by using the SELECT command.

```
)SELECT TOT_AMOUNT=$SUM(AMOUNT WHERE NUMBER=500), &
&)TOT_TAX=$SUM(TAX WHERE NUMBER=500), &
&)AVG_TAX=$AVG(TAX WHERE NUMBER=500)
)PRINT

TOT_AMOU TOT_TA AVG_TA

$  90.11    2.91   0.97

1 LINE PRINTED.
)
```

```
SYSTEM
[;$CANCEL]
[;$COMMENT]
[;$CPU]
[;$DEMO]
[;$LANGUAGE="language"]
[;$TIME]
```

Assigns or displays system-wide options.

:S	Optional global switch. If included, shows the current status of the system options. If additional parameters are also included, they are processed before the status is displayed.
$CANCEL	This parameter can have either a local "Y" or "N" switch. It controls the control-Y trap when RELATE is being executed from the Host Language Interface routines. If no switch is specified, a "Y" is assumed. If an "N" is specified, the control-Y is disabled. **WARNING: RELATE makes no distinction between output-oriented commands and other operations; therefore, any RDBREAD call after control-Y has been pressed and $CANCEL enabled may return an EOF. This could result in incomplete processing of any command which reads data from a file.**
$COMMENT	This parameter may contain a local "Y", "N", or "S" switch and indicates the action to be taken with comments contained in procedure files. If a switch is not included, a "Y" is assumed. If a "Y" is assumed or given, the user may include a comment at the end of each line in a procedure file. The comment is delimited by a "/*". Any text may appear after the start of the comment. The comment runs until the end of the line. These comments are stripped from the command line before it is displayed when a SHOW option has been specified unless the local "S" switch is also used. These comments may also be used interactively, but they will not be saved by RELATE.
$CPU	This parameter can have either a local "Y" or "N" switch and indicates whether the number of CPU seconds used in the execution of a command should be printed when the command completes. If a switch is not specified, a "Y" is assumed.
$DEMO	This parameter can have either a local "Y" or "N" switch. It indicates if a procedure is being run as a demonstration. If the procedure is a demonstration, the $DEMO_S and $DEMO_E sequences defined in the TERMINAL command are printed around what would be user input to the system. If a switch is not specified, a "Y" is assumed.
$LANGUAGE	This parameter is used to change the language in which RELATE/3000 operates. When the language is changed, all prompts, command names, keywords, messages and errors will appear in the indicated language. The language name must be enclosed in quotes.

$TIME	This parameter accepts either a local "Y" or "N" switch and is used to indicate if the amount of time used in the execution of a command should be printed when the command completes. If a switch is not specified, a "Y" is assumed.

If an error occurs when any of the parameters are assigned, no changes are made to the current system status.

EXAMPLES:

Print the amount of time used to execute commands after the command completes. Changes made to the system parameters do not take effect until the following command.

```
)SYSTEM $TIME:YES
```

Print the number of CPU seconds that it takes for a command to execute. If no switch is specified, "Y" is assumed. Show the current status of the SYSTEM parameters (evaluated AFTER other parameters are processed).

```
)SYSTEM:S $CPU

SYSTEM CONTROL PARAMETERS:

$CPU:YES
$LANGUAGE="ENGLISH"
$TIME:YES
$DEMO:NO
$COMMENT:NO

0.01 MINUTES.
)SYSTEM $TIME:N; $CPU:N
0.0 CPU SECONDS, 0.00 MINUTES.
)SYSTEM:S

SYSTEM CONTROL PARAMETERS:

$CPU:NO
$LANGUAGE="ENGLISH"
$TIME:NO
$DEMO:NO
$COMMENT:NO

)
```

```
TERMINAL
[;$CCTL]
[;$CLEAR="clear sequence"]
[;$CRT]
[;$DEMO_S="demonstration start sequence"]
[;$DEMO_E="demonstration end sequence"]
[;DEVICE=devicerange]
[;$LINES=# of lines per page]
[;$SPACE_B=# of lines to space at the bottom of page]
[;$SPACE_T=# of lines to space at the top of page]
[;$TYPE="terminal name"]
[;$WIDTH=# of characters]
```

Assigns or displays the values of terminal parameters.

:S	Optional global switch. If used, the current status of the terminal options are displayed. No keywords (other than DEVICE) may be specified on the same command.
:T	Optional global switch. If used, the terminal types in the RELATE terminal table are displayed. No keywords or other switches may be specified on the same command.
:U	Optional global switch allowed only for users with System Manager capability. Updates the message catalog with new terminal parameters when used in conjuction with the DEVICE keyword. See the "System Manager" paragraph.
$CCTL	This parameter can have either a local "Y" or "N" switch and is used to indicate whether or not the terminal understands carriage control codes. If a switch is not specified, "Y" is ix 'carriage control' assumed. $CCTL:N should be specified if a form-feed is not recognized by the terminal or if paper of an unusual size is being used and a form-feed would not position the paper to the top of the page. $CCTL has no effect if the output device is a CRT ($CRT:Y).
$CLEAR	This specifies the sequence of characters that must be transmitted to position to the top of a page, or, if the output device is a CRT, to clear the screen. The sequence may be a maximum of six characters long.
$CRT	This parameter accepts either a local "Y" or "N" switch and is used to indicate if the output device is a CRT. If a switch is not specified, a "Y" is assumed. When the output device is a CRT, RELATE/3000 will output twenty-two lines of information and then request that a key be hit to continue the output. IF $CLEAR is properly set, the screen will be erased before each page is printed.

$DEMO_S="demonstration start sequence" $DEMO_E="demonstration end sequence"	These parameters define the sequence of characters that should be printed at the beginning and end of each line of user-entered information when a procedure file is executing in a DEMO mode (See the SYSTEM command). Each sequence may have no more than 6 characters.
DEVICE	Allowed only for users with System Manager capability, and only in conjunction with a :S or :U switch. The devicerange is a device number or range of device numbers, including zero (0) and from 20 to 399. See the "System Manager" paragraph.
$LINES=# of lines	This indicates the number of lines of information that exist on a page.
$SPACE_B=# of lines $SPACE_T=# of lines	These parameters set the number of lines that the system will skip at the top and bottom of each new page. The number of lines skipped may not exceed 63 and the total must be at least 10 less than the number of lines per page. These parameters have no effect if the output device is a CRT. The default for hardcopy devices is three of each.
$TYPE="terminal type"	This parameter is used to indicate the type of terminal in use. When $TYPE is assigned and the requested type can be located in the terminal type table maintained by RELATE/3000, the $CCTL, $CLEAR, $CRT, $LINES, $SPACE_B, $SPACE_T, and $WIDTH parameters are set.
$WIDTH=# of characters	This parameter is used to indicate the width in characters of the output device. The width may not be less than 40 or more than 250.

If an error occurs when any of the parameters are processed, no changes are made to the current terminal configuration.

System Manager

The System Manager has the ability, with the :U switch, to modify RELATE's terminal table in two ways. First, the terminals recognized by the $TYPE parameter (device numbers in the 300's) can be altered. This allows you to change a TYPE from one that your installation doesn't use to one that is used, and alter its associated parameters.

Secondly, the system manager can specify the parameters for terminals located on each of the system's logical devices (0 for the printer and 20-299 for terminals). Then, when a user logs on, RELATE will automatically set up their terminal to match the specified configuration.

EXAMPLES:

Show the user's current terminal type. The device number is zero because the examples are generated from a job.

```
)TERMINAL:SHOW

DEV                                                   SPACE
 #    TYPE        CCTL   CLEAR      CRT  LINES    TOP  BOTTOM  WIDTH
  0  P300         NO     '12        NO      60      0      0      84
                                            60                    84
```

Show the available terminal types.

```
)TERMINAL:T

DEV                                                   SPACE
 #    TYPE        CCTL   CLEAR      CRT  LINES    TOP  BOTTOM  WIDTH
300               NO     '12        NO      66      3      3      80
301  ADM3         NO     '26        YES     24      0      0      80
302  AJ832        YES    '12        NO      66      3      3     132
303  ASR43        NO                NO      51      3      3     132
304  DIABLO       YES    '12        NO      66      3      3     132
305  DEC          YES    '12        NO      66      3      3     132
306  HP2601A      YES    '12        NO      66      3      3     132
307  HP2635A      YES    '12        NO      66      3      3     250
308  HP2640B      NO     '27U       YES     24      0      0      80
309  HP2623A      NO                YES     24      3      3     132
310  TVI920C      NO     '12        YES     66      3      3      80
311  QUME         YES    '12        NO      66      3      3     132
312  QMS1200      YES    '12        NO      66      3      3     132
313  D1650        YES    '12        NO      66      3      3     132
314  D630         YES    '12        NO      66      3      3     132
```

Change the current terminal type and display the results.

```
)TERMINAL $TYPE="ADM3"; $WIDTH=72
)TERMINAL:S

DEV                                                   SPACE
 #    TYPE        CCTL   CLEAR      CRT  LINES    TOP  BOTTOM  WIDTH
  0  ADM3         NO     '26        YES     24      0      0      80
                                            24                    72
```

The System Manager can alter the standard table and set up terminals for specific logical devices.

```
)TERMINAL:U DEVICE=309;$TYPE="HP2623A";$CRT;$LINES=24
)TERMINAL:U DEVICE=25;$TYPE="HP2623A"
)TERMINAL:S DEVICE=20/30

DEV                                             SPACE
 #    TYPE       CCTL   CLEAR     CRT  LINES   TOP  BOTTOM  WIDTH
 20              NO     '12       NO      66     3     3      80
 21              NO     '12       NO      66     3     3      80
 22              NO     '12       NO      66     3     3      80
 23              NO     '12       NO      66     3     3      80
 24              NO     '12       NO      66     3     3      80
 25  HP2623A     NO               YES     24     3     3     132
 26              NO     '12       NO      66     3     3      80
 27              NO     '12       NO      66     3     3      80
 28              NO     '12       NO      66     3     3      80
 29              NO     '12       NO      66     3     3      80
 30              NO     '12       NO      66     3     3      80

)
```

UNLOCK

Releases all locks held by RELATE.

The UNLOCK command cannot be issued when a transaction is in progress.

EXAMPLES:

Once locks have been manually obtained with the LOCK command they must be manually released.

```
)SHOW FILES

L
O
C
K FILE NAME         DATABASE NAME

Y CUST              CUST.DOC79.RDB

  INVOICE           INVOICE.DOC79.RDB

)UNLOCK
)SHOW FILES

L
O
C
K FILE NAME         DATABASE NAME

  CUST              CUST.DOC79.RDB

  INVOICE           INVOICE.DOC79.RDB

)
```

**UPDATE [assignment[,...]] [TO filename1[;options]]
USING[:D] filename2[;options] [BY keylist]**

Updates and/or copies records with duplicate keys from a secondary file.

:D	Optional global switch which, if used, is appended to the UPDATE command. Deletes each record from the current file as it is updated.
:F	Optional global switch. Updates only the first entry in a key.
assignment	Optional. May contain fields from either the current file, the output ("TO") file, if specified, and the input ("USING") file. Fields in the output file and in the input file must be qualified by the file's path name.
TO filename1	Optional. If included, each record updated in the current file will be copied to filename1. Any data fields that exist in filename1 but do not exist in the current file will be obtained from filename2. All fields that do not match names in either file are set to zeroes or blanks depending on type. Any options listed in the "TO filename" section needed to access the file must be appended to the filename, whether or not the file is already open.
USING filename	Required. The "USING" file contains records that are matched with records in the current file by the fields in the keylist. When a matching set of records is found, the records are translated into the format of the "TO" file (if it was specified) and then the assignments are executed.
LOCALS	Optional switch which, if used, is appended to the USING keyword.
	:D Deletes each record from the "USING" file as it is updated.
BY keylist	Optional. If not included, the current index will be used for the update. All fields in the current index must also exist in filename2 and be of the same type and length. If included, the fields in the keylist must be the first fields in the current key. The fields must also exist in filename2 and be of the same type and length.

The UPDATE command reads records serially from the USING file and then locates all records in the current file that match the fields given in the keylist (hence, the smaller file should be the "USING" file). When a match is found and a "TO" file exists, a new record is created in memory for the "TO" file. This record is created by copying the record from the current file into the format of the "TO" file and then updating any fields that are in both the "TO" and "USING" files, but not in the current file, from the "USING" file. Any assignments requested are now executed. If any of the assignments change the value of fields in the current file or the "USING" file, these records are rewritten. If any assignments are for fields in the "TO" file, the new record will be updated. Finally, any :D switches are acted on and the new "TO" record is actually placed in the "TO" file.

If the current index is by line number each record in the "USING" file will be matched with each record in the current file.

EXAMPLES:

Update NAME and ADDRESS using the information on those fields from the CUST file. NUMBER field must exist in both the CUST and CUST1 files. NUMBER must be the first field in the current index.

```
)SET PATH CUST
)PRINT NAME, NUMBER, ADDRESS

NAME                     NUMBE ADDRESS

ALEXANDER HALE & CO.       700 83A SAN PEDRO
AMERICAN TIRE CO.          500 7052 EL CAMINO REAL
CUPERCO                    400 10802 WILKINSON AVENUE
DEXMACH, INC.              100 BOX 877 RD1
FINCH, FINCH, & OTTO       600 87 NORTH FIRST, SUITE 243C
HASLETREX INC.             200 89 BEST WAY
NATIONAL AIRLINES         1000 SAN FRANCISCO INTL AIRPORT
PERFECT SOUND              800 415 FAIR OAKS AVENUE

8 LINES PRINTED.
)SET PATH CUST1
)PRINT NAME, NUMBER, ADDRESS

NAME                     NUMBER ADDRESS

ALEXANDER HALE & CO.        700 83A SAN PEDRO
AMERICAN TIRE CO.           500 7052 EL CAMINO REAL
CUPERCO                     400 10802 WILKERSON AVENUE
DEXMACH, INC.               100 PO BOX 1568
HASLETREX, INC.             200 89 BEST WAY
NATIONAL AIRLINES          1000 SAN FRANCISCO INTL AIRPORT

6 LINES PRINTED.
)CREATE INDEX BY NUMBER
INDEX #2 HAS BEEN CREATED.
6 LINES INDEXED.
INDEX #2 IS NOW THE CURRENT INDEX.
)UPDATE NAME=CUST.NAME, ADDRESS=CUST.ADDRESS USING CUST BY NUMBER
6 LINES READ FROM CURRENT FILE.
8 LINES READ FROM USING FILE.
6 LINES UPDATED IN CURRENT FILE.
)SET INDEX NAME
INDEX #1 IS NOW THE CURRENT INDEX.
```

```
)PRINT NAME, NUMBER, ADDRESS

NAME                  NUMBER ADDRESS

ALEXANDER HALE & CO.     700 83A SAN PEDRO
AMERICAN TIRE CO.        500 7052 EL CAMINO REAL
CUPERCO                  400 10802 WILKINSON AVENUE
DEXMACH, INC.            100 BOX 877 RD1
HASLETREX INC.           200 89 BEST WAY
NATIONAL AIRLINES       1000 SAN FRANCISCO INTL AIRPORT

6 LINES PRINTED.
)
```

SECTION 3

H O S T   L A N G U A G E   I N T E R F A C E

HOST LANGUAGE INTERFACE

RELATE/3000 is designed to be used as a stand-alone language for interactive users and as a data sublanguage embedded in a host programming language. Most RELATE query, data manipulation, data definition, and data control functions can be executed directly from a user's program. These commands provide a procedural interface to a database. Additionally, a non-procedural interface mechanism is supported that standardizes the interface to RELATE, IMAGE, KSAM, and MPE files.

RELATE interfaces to FORTRAN, COBOL, SPL, and PASCAL with subroutines. Special routines are supported for BASIC that pack and unpack data.

A program accesses RELATE through a cursor. A cursor is a fifty word integer array that maintains status information and links the user's application with the RELATE process. For procedural access the cursor is generally associated with a RELATE command. For non-procedural access the cursor generally references a single file. If an error occurs due to the execution of a subroutine, the first element of the cursor will be returned as a non-zero value. The program may then obtain the English language message associated with the error number by calling RDBERROR. The CURSOR FORMAT section should be consulted for other types of information returned to the user's program.

A program may have as many cursors initialized as desired, subject to the restrictions of MPE and the hardware in regards to the number of open files and the memory space available. The Host Language Interface routines create the RELATE Command Interpreter as a son process. Information is passed back and forth through a shared extra data segment in response to calls made by the application program.

All user programs must be prepared (PREPed) with DS (data segment) and PH (process handling) capability. The application program will be aborted by MPE if it has not been prepared with the required capabilities.

To access files, a program must issue OPEN commands in the same manner as a user in the Command Interpreter. The path that is created when the file is opened is associated with the passed cursor. Each cursor represents a group of zero or more access paths, any of which can be used as the current path. Path names on different cursors do not conflict with each other. Any file that is opened becomes accessible to any other cursors, and can be accessed by more than one cursor at a time by creating a path to the file by opening the file again on the desired cursor.

CALL SUMMARY

CALL NAME	DESCRIPTION
RELATE	Passes a command to the RELATE/3000 data base management system.
RDBADD	Adds a new record to the file associated with the passed cursor.
RDBBIND	Binds a memory location for a return value.
RDBCLOSE	Closes a cursor.
RDBDELETE	Deletes the current record from the file associated with the passed cursor.
RDBERROR	Returns information on an error condition that exists in a cursor.
RDBINFO	Returns information on the current file or status of the system.
RDBINIT	Initializes a cursor.
RDBPOINT	Positions to a specific record on the file associated with the passed cursor for reading.
RDBREAD	Reads the next record from the file associated with the passed cursor.
RDBREPOINT	Repositions to a specific record on the file opened in shared mode to verify that it has not been altered by another user before an UPDATE.
RDBUPDATE	Updates the current record on the file associated with the passed cursor.

The current record is set by the RDBADD, RDBREAD, and RDBPOINT calls and may be changed by a RELATE call. The current record is neutralized by the RDBDELETE call.

The table below illustrates the format of the call statements used in each of the supported languages.

COBOL	CALL "name" USING parameter,...
FORTRAN	CALL name (parameter,...)
SPL	name (parameter,...)
BASIC	linenumber CALL name (parameter,...)
or	linenumber *name (parameter,...)
PASCAL	name (parameter,...)

All procedures may be called directly from any of the five host languages since they are not TYPE procedures, do not use the SPL OPTION VARIABLE capability, and all parameters are passed by reference as word addresses. A special set of calls is provided to interface with the BASIC language. The format for these calls is included at the end of this section.

RELATE (cursor,command,commandlen)

The RELATE call passes a command to the RELATE/3000 Command Interpreter process. The command may be any command except REDO. If a CREATE FILE command is executed, at least one of the keywords STRUCTURE or FIELDS must be included (see CREATE FILE for further information).

cursor	A cursor to which the command will be associated. If the command is either a CREATE or an OPEN of a file, the cursor will be associated with the path formed. If a SELECT command is given, the cursor will be associated with the results of the SELECTion.
command	An integer array containing the command to be executed. The command may have up to 1500 characters.
commandlen	An integer variable that indicates the length in bytes of the command. If -1 is passed, the command is assumed to be terminated with a null or a backslash ("\").

After the command is executed the current file is rewound. A subsequent read on the file will return the first record in the file.

RDBADD (cursor,eof,list,listlen,source)

The ADD call is used to place a new record into a file.

cursor	A cursor that is associated with an open path. The path must reference a file to which the user has ADD access.
eof	A logical variable that returns TRUE if the record could not be written because the file is filled. Any other error during the add will return TRUE as well as an indication of the cause of the error in the cursor.
list	An integer array that contains the names or numbers of the fields to be added. If field names are passed, they must be separated by commas. If field numbers are passed, a zero element terminates the list. An atsign ("@") may be used to indicate that the values for all fields are supplied. If the user does not possess the ADD ability for all fields, the atsign represents those fields to which the user has access.
listlen	An integer variable that indicates the length in bytes of the list. If -1 is passed, the list is assumed to be terminated with a null, a backslash ("\"), or contains field numbers. If zero is passed, a zeroed record is written to the file. If a -2 is passed, the list is ignored and any substitution variable names which match a name in the file are used to compose the new record.
source	An integer array that contains the data to be added to the file, if listlen is not 0 or -2. The order, type, and size of the data must correspond exactly to the format of the fields in the file. If listlen is 0 or -2, this array must be passed but it is ignored.

The add call will reset any break value that had been given during an RDBPOINT.

RDBBIND (cursor,bind,relatevar,relatevarlen,var)

The BIND call is used to associate program variables with return variables used in an ADD, POINT, UPDATE, or READ call.

cursor	A cursor to which the return variable will be associated.
bind	An integer variable that should be set to 1 to bind the field or 0 to remove the field from the bind table.
relatevar	An integer array containing the name of the return variable to be used. Return variables must be a valid fieldname and must appear on the left side of the assignment operator in a SELECT command or duplicate the names of fields in the file. After the fieldname, the user may specify the TYPE and SIZE of the field as well as any special print formatting. If TYPE and SIZE are not specified, they will be obtained from the field which is being bound. If the variable being bound is alphabetic, the FORMAT will be set to zero.
relatevarlen	An integer variable that indicates the length in bytes of the text in relatevar. If -1 is used, the system assumes that relatevar is terminated with a null or a backslash ("\").
var	An integer variable or array that is used as the address of the return variable.

The BIND call for return variables may be made at any time. This allows a RELATE command to be given followed by calls to obtain information on the data that will be returned.

Bound variables are lost any time the current path in a cursor is changed. The path can be changed by a SELECT, CREATE FILE, OPEN FILE, CLOSE FILE, PURGE FILE, CLOSE DATABASE, or SET PATH command.

RDBCLOSE (cursor)

A CLOSE call releases the resources allocated to the cursor in the database process. After a cursor is closed, an INIT call must be made with the cursor prior to any other RDB calls. Closing a cursor logically disconnects the passed cursor from the database process and may close files or databases if they are not referenced in any other cursor.

cursor	An initialized cursor.

RDBCLOSEX (cursor)

A CLOSEX call releases the resources allocated to any cursors in this segment and closes down the RELATE process associated with the cursor. The process is created by the first INIT or INITX call.

cursor	An initialized cursor.

RDBDELETE (cursor)

The DELETE call deletes the current record.

cursor	A cursor that is associated with an open path. The path must reference a file to which the user has DELETE access.

RDBDELETE will delete the last record referenced by an RDBPOINT, an RDBREAD, an RDBADD, or an RDBUPDATE. To delete several adjacent records, RDBREAD or RDBPOINT should be called prior to each RDBDELETE call.

RDBERROR (cursor,mode,error,errorlen)

The ERROR call returns the text corresponding to an error condition in the cursor.

cursor	A cursor that contains a non-zero in the first element.
mode	An integer variable that indicates the operation to be performed on the error message: 1 Return the error message to the user in the error array. 3 Display the error message on RDBOUT (usually $STDLIST). The error array and errorlen parameters are ignored but must be passed.
error	An integer array into which the error mesage will be placed. The message is terminated with a null.
errorlen	An integer variable that must be set to the number of bytes available in the error array prior to the call. The actual byte count of the message is returned. If the length is returned as a negative number, it indicates that the message did not fit into error. If 0 is returned, no error existed.

RDBINFO (cursor,mode,qualifier,information,words)

The INFO call returns information about the file referenced by the cursor passed or about the current state of the RELATE system.

cursor	A cursor that is associated with an open access path.
mode	An integer variable that indicates the type of information that should be returned.
qualifier	An integer variable that qualifies the type of data requested. For some modes this parameter is ignored.
information	An integer array of at least 30 elements into which the information will be placed.
words	An integer variable that specifies the size in words of the information array.

Certain classes of information may be returned. These classes are indicated by the hundreds digit of the mode. The type of information from a particular class is indicated by the ones and tens digits. Information may be returned on the following:

100	Indexes
200	Fields
300	Bound Fields
400	Transaction Status
500	Databases
600	Files
700	Paths

Mode	Qualifier Information Returned
100	Returns the number of indexes defined in the first word and the number of each index defined in the following words.
101 Index Number	Returns the number of fields in the index in the first word and the field numbers that comprise the index in the following words. If the field is in descending order in the index, the field number is negative.
102 Index Number	Returns information about the index number:

WORD	CONTENTS
1	Contains 1 if the index is a unary index.
2	The type of the index: 0 B-Tree. 1 Hashed (Image Master). 2 Hashed-linked (Image Detail). 3 Sort Order. 4 Record number.
3	The number of levels in the tree for RELATE B-Tree indexes.
4	Not used.
5-6	The number of nodes allocated for RELATE B-Tree indexes.
7	The number of words in the key.
8-29	Reserved

Mode	*Qualifier Information Returned*
200	Returns the number of fields in the file associated to the passed cursor.
201 Field Number (Negative for Internal Field Number)	Returns information about the field:

WORD	CONTENTS
1-5	Fieldname.
6	The number of characters in the fieldname.
7	Type (see the Data Type Codes section).
8	Words.
9	Print width.
10	Decimals.
11	The field can be used when a record is added.
12	The field can be used when a record is changed.
13	The word offset to the field in the current record.
14	The internal (IMAGE) field number.
15	The print format of the field as set by the CREATE FILE and MODIFY FIELD commands. See the Print Formats section for details.
16	Data entry level.
17-29	Reserved.

202	Returns information about type conversion problems for bound variables. One element is returned for each field. The causes and error numbers are described in the Type Conversion Errors section.

Mode	Qualifier Information Returned
300	Returns the number of fields bound to the passed cursor.
301 Field Number	Returns information about a bound field. The information is returned in the same format as in mode 201 except that word 13 contains the address in the user's program of the field, and word 14 is zero.

Mode	Qualifier Information Returned
400	Returns information about the current transaction state:

WORD	CONTENTS
1	The transaction level. A zero indicates the absence of a transaction.
2	The locking status: 0 No locks pending. 1 Locks are pending. 2 Locks applied.
3	The number of locks pending or applied.

Mode	Qualifier Information Returned
500	Returns the number of currently opened databases in the first word and the number of each database in the following words.
501 Database Number	Returns information about the database:

WORD	CONTENTS
1-14	The normalized (contains the group and account) name of the database terminated with a null.
15	The type of the database: 1 RELATE/3000 2 MPE 3 KSAM 4 IMAGE/3000 DATABASE
16	The number of files open in the database.
17	The current disposition of the database. 1 Permanent 2 Temporary 4 None
18	The retention of the database. 1 Permanent 2 Temporary 4 None
19-29	Reserved.

Mode	Qualifier Information Returned
600	Returns the number of currently open files in the first word and the number of each file in the following words.
601 File Number	Returns information about the file:

WORD	CONTENTS
1-9	The local name of the file, or dataset terminated with a null.
10	The number of the database to which this file is associated. If the file is a view, this is zero.
11-12	A doubleword value that indicates the current number of records in the file.
13	The type of the file: 1 RELATE/3000 2 MPE 3 KSAM 4 IMAGE/3000 DATASET 6 A SELECTION or view
14	The number of fields in the file.
15	The number of words in the data record.
16	Reserved.
17-18	A doubleword value that indicates the EOF position on the current file. This is identical to elements 11 and 12 for an IMAGE data set.
19-20	A doubleword value that indicates the limit of the file.
21-29	Reserved.

602

Returns information about the current file:

WORD	CONTENTS
1	The current path number.
2	The current index number.
3	The current file number.
4	The current database number if the current file is not a SELECTion.

Mode	Qualifier Information Returned
700	Returns the number of currently open paths in the passed cursor in the first word and the number of each path in the following words.
701 Path Number	Returns information about the path:

WORD	CONTENTS
1-9	The path name terminated with a null.
10	The file number to which the path is associated.
11	The index number to which the path is associated.
12-29	Reserved.

RDBINIT (cursor)

The INIT call creates a cursor in the database processes and initializes the cursor passed. An INIT call must be the first RDB call made with the cursor and must be made for each cursor that will be used.

cursor	An integer array at least 50 words in length.

RDBINITX (cursor, loadlist, listlen)

The INITX call creates a cursor in the database processes and initializes the cursor passed. An INIT or INITX call must be the first RDB call made with the cursor and must be made for each cursor that will be used. The INITX call is a superset of the INIT call and need only be used if special loading instructions are required.

cursor	An integer array at least 50 words in length.	
loadlist	An integer array which can contain special loading or operational instructions. The options should be separated with semicolons (";"). The list can not contain any blanks and all keywords must be spelled out completely. The options are as follows:	
	SEGMENT=num	Instructs RELATE to use 'num' as the identity of the extra data segment which will be used for communications between the user's process and RELATE. The segment number must be between 1 and 10000. A new RELATE process is created for each new segment number used. Use of this option allows a single user process to use several RELATE processes and allows several user processes to concurrently execute with their own RELATE process. Any one segment number should not be used by more than one user process.
	SIZE=num	Instructs RELATE to initially allocate, and maintain at least this allocation, of DL area (which is used by tables created within RELATE). The size must be either -1, which says to use the default size, or between 0 and 20 (representing 20 * 1024 words). Larger numbers will reduce the number of times that RELATE must expand the stack during file opens and SELECT commands, easing the burden on MPE's memory manager. Smaller numbers will improve the performance of sorting by allowing the system sort routine more stack space.
listlen	An integer variable that indicates the length in bytes of loadlist. If -1 is passed, the list is assumed to be terminated with a null or a backslash ("\").	

This process terminates and the data segment is released when an RDBCLOSEX is called.

RDBPOINT **(cursor,key,words,break,found)**

The POINT call locates a record by a key value.

cursor	A cursor that is associated with an open path which contains an index.
key	An integer array that contains the key value to be searched for. If words is 0 or -2, this array must be passed but it is ignored.
words	An integer variable that specifies the number of words in the key array passed by the user. This value may be less than the actual number of words in the index. If 0 is used, the file is rewound. If a -2 is used, the values of the substitution variables are used to compose the key. This is done by searching the current key for bound fields. The POINT key is composed of current values from the bound program variables in order of the current key until a key field is found which is not bound. Words must be 0 if a view without a BY clause is being searched.
break	An integer variable that indicates the number of fields in the current index that should be checked for a control break. When a control break is encountered, an EOF is returned from the read and the contents of variables in the user's program are not changed. The twelfth (12) cursor location returns the number of the field that caused the break to occur. Field one is the least significant field in the key. When the actual EOF is encountered on the file, cursor (12) will return the value initially specified for break. An additional read will return an EOF and a zero in cursor (12).
found	A logical variable that returns TRUE if a record exactly matching the key was found. If words is zero, found is returned as TRUE if any records exist in the file.

A POINT call can be made for a file of any type. In RELATE, KSAM, and MPE files, the point positions to a location on the file (or in an index). Subsequent reads made on the file will return records starting with the key requested or the next largest key if an exact match could not be found. In IMAGE master sets, only the record pointed to can be read. In IMAGE detail sets, only records containing the same search item can be read. For IMAGE sets, EOF is returned as TRUE when an attempt is made to read additional records.

If an RDBPOINT is not done before an RDBREAD, the first record in the file will be returned.

RDBREAD (cursor,eof,list,listlen,destination)

The READ call returns the next record from a file.

cursor	A cursor that is associated with an open access path.
eof	A logical variable that returns TRUE when an attempt is made to read a record past the end of the file. If a POINT had previously been made with break greater than zero, TRUE is returned when the next key does not equal the current key. In this case, a record is not returned. If the file is not an IMAGE file, another read can then be executed to obtain the first record in the next key.
list	An integer array that contains the names or numbers of the fields to be read. If field names are passed, they must be separated by commas. If field numbers are passed, a zero element terminates the list. An atsign ("@") may be used to indicate that all field values should be read.
listlen	An integer variable that indicates the length in bytes of the list. If -1 is passed, the list is assumed to be terminated with a null or a backslash ("\") or contains field numbers. If zero is passed, a record from the input file is read but not returned. If a -2 is passed, the list is ignored and any substitution variable names which match a field name in the file are changed to reflect the data read.
destination	An integer array into which the data is read if listlen is not 0 or -2. The order, type, and size of the data is unchanged when read. If listlen is 0 or -2, this array must be passed but it is ignored.

If a read is done before an RDBPOINT is executed, the first record of the file is returned.

If a serial read of an IMAGE master set is used and records are deleted during the read, records may be skipped. This problem, which is described in the IMAGE reference manual, is caused by migrating secondaries.

RDBREPOINT (cursor,key,words,break,found)

The REPOINT call relocates a record that has previously been read to preserve the multi-user checksum.

cursor	A cursor that is associated with an open path which contains an index.
key	An integer array that contains the key value to be searched for. If words is 0 or -2, this array must be passed but it is ignored.
words	An integer variable that specifies the number of words in the key array passed by the user. This value may be less than the actual number of words in the index. If 0 is used, the file is rewound. If a -2 is used, the values of the substitution variables are used to compose the key. This is done by searching the current key for bound fields. The key is composed of current values from the bound program variables in order of the current key until a key field is found which is not bound. Words must be 0 if a view without a BY clause is being searched.
break	An integer variable that indicates the number of fields in the current index that should be checked for a control break. When a control break is encountered, an EOF is returned from the read and the contents of variables in the user's program are not changed. The twelfth (12) cursor location returns the number of the field that caused the break to occur. Field one is the least significant field in the key. When the actual EOF is encountered on the file, cursor (12) will return the value initially specified for break. An additional read will return an EOF and a zero in cursor (12).
found	A logical variable that returns TRUE if a record exactly matching the key was found. If words is zero, found is returned as TRUE if any records exist in the file.

A REPOINT call can be made for a file of any type. In RELATE, KSAM, and MPE files, the point positions to a location on the file (or in an index). Subsequent reads made on the file will return records starting with the key requested or the next largest key if an exact match could not be found. In IMAGE master sets, only the record pointed to can be read. In IMAGE detail sets, only records containing the same search item can be read. For IMAGE sets, EOF is returned as TRUE when an attempt is made to read additional records.

This command is only useful for access to an updatable path open in SHARED mode. It allows RELATE find a record that has previously been read and verifies that no other user has altered the record before an UPDATE or a DELETE is done. The checksums that indicate the status of a record are maintained on a segment basis, so if the file is opened in different cursors within the same segment, the checksums will be properly maintained.

RDBUPDATE (cursor,list,listlen,source)

The UPDATE call changes the values of fields in the current record.

cursor	A cursor that is associated with an open access path. The path must reference a file to which the user has update access.
list	An integer array that contains the names or numbers of the fields to be updated. If fieldnames are passed, they must be separated by a comma. If field numbers are passed, a zero element terminates the list. An atsign ("@") may be used to indicate that all updatable fields will be changed.
listlen	An integer variable that indicates the length in bytes of the list. If -1 is passed, the list is assumed to be terminated with a null or a backslash ("\") or contains field numbers. If -2 is passed, the list is ignored and any return variable names which match a field name in the file are used to update the record.
source	An integer array that contains the new data if listlen is not -2. The order, type, and size of the data must correspond exactly to the fields specified in the list. If listlen is -2 this array must be passed but it is ignored.

BASIC INTERFACE

To simplify access to RELATE/3000 from BASIC programs, special interface routines are provided. The BASIC language interface routines perform the following actions:

1) Converts all byte addresses (string variables) to word addresses.

2) Converts the data types on entries that normally require integer variables or expressions.

3) Packs and unpacks information in the read and writelists for reads, adds, points, and updates.

4) Updates the logical length of string variables into which data is placed if string variables are used in a read list. This is not done on variables that are bound.

STRING VARIABLES

The physical length (DIM) of a string variable determines the number of characters (bytes) read and the logical length of a string variable determines the number of characters written. Thus, the physical length of a string variable specified in a DIM or COM statement should exactly match the size of the fields read.

On the other hand, the same string variable can be used to write items of varying sizes. Substring designators should be used to ensure that the actual string passed to the field fills the item to be written. For example, if the field is 8 characters long, and substring S$(3) is 2 characters long, S$(3,10) or S$(3;8) fills the item with the S$(3) substring and appends 6 blanks.

If the string variable is an array, the length of each string element or of the concatenated string elements should correspond to the length of the field to be written. This can be ensured by specifying substring designators when assigning the value of elements in string arrays.

BRELATE (cursor(*), commandstring)

The RELATE call passes a command to the RELATE/3000 Command Interpreter process. The command may be any command except REDO. If a CREATE FILE command is executed, at least one of the keywords STRUCTURE or FIELDS must be included (see CREATE FILE for further information).

cursor	Required. An integer array containing no less than 50 elements.
commandstring	Required. A string constant, variable or expression that contains the command to be executed.

See the RELATE call for further information.

BDBADD (cursor(*), eof [,list ,putlist])

The ADD call is used to place a new record into a file.

cursor	Required. An integer array containing no less than 50 elements.
eof	Required. A single integer, real, or long variable that returns 1 if an end of file is encountered. A 0 is normally returned.
list	Optional. If not included, any bound variables are written to the file. If included, this must be a string constant, variable, or expression containing the names of the fields for which information is supplied in the putlist. The list may consist of an atsign ("@") which represents all fields to which the user has ADD access. The list may also be an integer array containing the field numbers of the fields to be used. The array must be terminated with a zeroed element.
putlist	Optional. One or more integer, real, long, or complex constants or variables or string arrays, constants, or variables from which the new record will be composed.

See the RDBADD call for further information.

BDBBIND (cursor(*), name [,variable])

The BIND call is used to associate program variables with return variables used in a RELATE command or ADD, POINT, UPDATE, or READ calls.

cursor	Required. An integer array containing no less than 50 elements.
name	Required. A string constant, expression, or variable containing a fieldname optionally followed by its type, size, and print format.
variable	Optional. A single integer, real, long, or string variable or array that is bound to the name given above. If not specified, the field contained within name is unbound.

If a string variable is bound the string should not be subscripted in an attempt to bind a field to a portion of the string. Any attempt to do this will cause BASIC to create a temporary variable and pass its address to RELATE. This cannot be detected by RELATE and will cause incorrect results. The length of bound string variables is not adjusted by a BDBREAD call. Thus, the length must be set in the user's program prior to the execution of the BDBREAD. The simplest way to do this is to execute an assignment of the form 'LET X$ (1;Y)=" "' where Y is the number of characters in the field.

See the RDBBIND call for further information.

BDBCLOSE (cursor(*))

A CLOSE call releases the resources allocated to the cursor in the database process. After a cursor is closed, an INIT call must be made with the cursor prior to any other BDB calls. Closing a cursor logically disconnects the passed cursor from the database process and may close files or databases if they are not referenced in any other cursor.

cursor	Required. An integer array containing no less than 50 elements.

See the RDBCLOSE call for further information.

BDBCLOSEX (cursor(*))

A CLOSEX call releases the resources allocated to any cursors in this segment and closes down the RELATE process associated with this cursor. The process is created by the first INIT or INITX call.

cursor	An initialized cursor.

BDBDELETE (cursor(*))

The DELETE call deletes the current record.

cursor	Required. An integer array containing no less than 50 elements.

See the RDBDELETE call for further information.

BDBERROR (cursor(*)[, mode, error])

The ERROR call returns the text corresponding to an error condition in the cursor.

cursor	Required. An integer array containing no less than 50 elements.
mode	Optional. An integer, real, or long variable indicating a mode of operation. If not specified, mode 3 is assumed.
error	Required if mode is used. A string variable into which the current error message is returned.

See the RDBERROR call for further information.

BDBINFO (cursor(*), mode[, qualifier], information)

The INFO call returns information about the file referenced by the cursor passed or about the current state of the RELATE system.

cursor	Required. An integer array containing no less than 50 elements.
mode	Required. An integer, real, or long variable indicating the information to be returned.
qualifier	Optional. An integer, real, or long variable that qualifies the mode. If not passed, the qualifier is assumed to be zero.
information	Required. An integer, real, long, or string variable or array into which the requested information is returned.

See the RDBINFO call for further information.

BDBINIT (cursor(*)[, loadlist])

The INIT call creates a cursor in the database processes and initializes the cursor passed. An INIT call must be the first BDB call made with the cursor and must be made for each cursor that will be used.

cursor	Required. An integer array containing no less than 50 elements.
loadlist	Optional. A string constant, variable, or expression that contains special loading or operational instructions as described under RDBINITX. This need only be used if special loading instructions are required.

If a loadlist is specified, BDBCLOSEX must be used to terminate the process.

See the RDBINIT call for further information.

BDBPACK(cursor(*), buffer(*), offset, packlist)

The PACK call moves information from the variables in the packlist into the buffer starting at the byte indicated by offset.

cursor	Required. An integer array containing no less than 50 elements. The array need not represent an initialized cursor.
buffer	Required. A variable (usually an array) into which the contents of the packlist will be placed.
offset	Required. An integer, real, or long variable which represents the starting byte location within buffer. Zero represents the first byte in buffer. When the call completes, offset is increased to represent the number of bytes moved into the buffer.
packlist	One or more integers, real, long, complex or string variables or arrays which contain the data to be packed into the buffer.

BDBPOINT (cursor(*)[, break, found[, pointlist]])

The POINT call locates a record by a key value.

cursor	Required. An integer array containing no less than 50 elements.
break	Optional. An integer, real, or long expression, constant or variable that indicates the number of fields in the current key that should be checked for a control break when a read is performed.
found	Optional. An integer, real, or long variable that returns 1 if a record containing the given key is found. A 0 is returned otherwise. If the break parameter is not passed, found and pointlist must not be passed.
pointlist	Optional. One or more integer, real, long, complex, or string constants or variables which are used to create the key to be located in the file. If not included, any bound variables are used to compose the key.

PARAMETERS	ACTION
cursor	The file is rewound. No breaks are set.
cursor, break	The file is rewound. The break fields are set.
cursor, break, found	A point is performed with the bound variables.
cursor, break, found, pointlist	A point is performed with the passed variables.

See the RDBPOINT call for further information.

BDBREAD (cursor(*), eof[, list[, readlist]])

The READ call returns the next record from a file.

cursor	Required. An integer array containing no less than 50 elements.
eof	Required. A single integer, real, or long variable that returns 1 when an end of file is encountered. A 0 is normally returned.
list	Optional. If not included, any bound variables are read from the file. If included, this must be a string constant, variable, or expression containing the names of the fields for which information is supplied in the putlist. The list may consist of an atsign ("@") which represents all fields to which the user has READ access. The list may also be an integer array containing the field numbers of the fields to be used. The array must be terminated with a zeroed element.
readlist	Optional. One or more integer, real, long, complex, or string variables or arrays into which the next record from the file is returned.

See the RDBREAD call for further information.

BDBREPOINT (cursor(*)[, break, found[, pointlist]])

The REPOINT call relocates a record that has previously been read to preserve the multi-user checksum.

cursor	Required. An integer array containing no less than 50 elements.
break	Optional. An integer, real, or long expression, constant or variable that indicates the number of fields in the current key that should be checked for a control break when a read is performed.
found	Optional. An integer, real, or long variable that returns 1 if a record containing the given key if found. A 0 is returned otherwise. If the break parameter is not passed, found and pointlist must not be passed.
pointlist	Optional. One or more integer, real, long, complex, or string constants or variables which are used to create the key to be located in the file. If not included, any bound variables are used to compose the key.

PARAMETERS	ACTION
cursor	The file is rewound. No breaks are set.
cursor, break	The file is rewound. The break fields are set.
cursor, break, found	A point is performed with the bound variables.
cursor, break, found, pointlist	A point is performed with the passed variables.

See the RDBREPOINT call for further information.

BDBUNPACK(cursor(*), buffer(*), offset, unpacklist)

The UNPACK call moves information from the buffer into the variables in the unpacklist starting at the byte indicated by offset.

cursor	Required. An integer array containing no less than 50 elements. The array need not represent an initialized cursor.
buffer	Required. A variable (usually an array) which contains data from a BDBREAD or which has been previously packed.
offset	Required. An integer, real, or long variable which represents the starting byte location within buffer to obtain the data required to fill the variables in the unpacklist. Zero represents the first byte in buffer. When the call completes, offset is increased to represent the number of bytes moved.
unpacklist	One or more integers, real, long, complex or string variables or arrays into which the data from buffer will be unpacked.

BDBUPDATE (cursor(*) [, list[, updatelist]])

The UPDATE call changes the values of fields in the current record.

cursor	Required. An integer array containing no less than 50 elements.
list	Optional. If not included, any bound variables are written to the file. If included, this must be a string constant, variable, or expression containing the names of the fields for which information is supplied in the putlist. The list may consist of an atsign ("@") which represents all fields to which the user has CHANGE access. The list may also be an integer array containing the field numbers of the fields to be used. The array must be terminated with a zeroed element.
updatelist	Optional. One or more integer, real, long, complex, or string constants, expressions, or variables from which the current record will be updated.

See the RDBUPDATE call for further information.

PRINT FORMATS

When information on fields is returned from the RDBINFO or BDBINFO subroutines, the print format word has the following meaning:

If right 3 bits (13-15)=0 (a numeric format):

bits (10-12)	=1, if commas are to be printed.
bits (4-9)	=0, for leading minus. =1, for leading minus or plus. =2, for trailing minus. =3, for trailing minus or plus. =4, for parentheses around negatives. =5, CR for negatives. =6, CR for negative and DR for plus.
bits (1-3)	=0, for no $. =1, for floating $. =2, for fixed $.
bit (0)	=not used.

If right 3 bits (13-15)=1 (a date format):

bits (10-12)	=0, if no separator. =1, if slash ("/") is separator. =2, if blank (" ") is separator. =3, if dash ("-") is separator. =4, if period (".") is separator.
bits (7-9)	=0, if first position is ignored. =1, if first position is MM. =2, if first position is DD. =3, if first position is JJJ. =4, if first position is YY. =5, if first position is CCCC. =6, if first position is NNN.
bits (4-6)	=0, if second position is ignored. =#, represents same data as bits 7-9.
bits (1-3)	=0, if third position is ignored. =#, represents same data as bits 7-9.
bit (0)	=not used.

DATA TYPE CODES

When information on fields is returned from the RDBINFO or BDBINFO subroutines, the field types are encoded according to the table below:

TYPE	FIELD TYPE
1	Alphabetic (2 characters per word).
2	Zoned decimal (2 digits per word).
3	Integer (1 word).
4	Double integer (2 words).
5	Real (2 words).
6	Long (4 words).
7	Packed decimal (4 digits per word).
8	Unsigned (1 word).

See the FIELD SIZE LIMITATIONS section under the CREATE FILE command for detailed information about the types.

DATA TYPES INTERFACE

RELATE	IMAGE	SPL	FORTRAN	COBOL	BASIC	RPG
I	I,J	INTEGER	INTEGER	COMP S9 TO S9(4)	INTEGER	BINARY
D	I2,J2	DOUBLE INTEGER	INTEGER*4	COMP S9(5) TO S9(9)	-	BINARY
R	R2	REAL	REAL	-	REAL	-
L	R4	LONG	DOUBLE PRECISION	-	LONG	-
P	P	-	-	COMP 3	-	NUMERIC
Z	Z	-	-	DISPLAY PICTURE 9	-	CHARACTER
A	U,X	BYTE	CHARACTER	DISPLAY PICTURE A, PICTURE X	STRING	CHARACTER
U	K1	LOGICAL	LOGICAL	DISPLAY PICTURE A	SIGNED INTEGER	-
A	I4,J4	-	-	COMP S9(10) TO S9(18)	-	BINARY
A	OTHERS	-	-	-	-	-

TYPE CONVERSION ERRORS

When a listlen of -2 is used in an ADD, READ, or UPDATE call or a word count of -2 is given in a POINT call, the Host Language Interface routines will automatically perform any type conversions required by the return variables defined by the user. This mechanism allows programs to be written that are completely independent of the format of the data in the database.

During the conversion operation RELATE may encounter errors because of the reasons indicated below. If all of the error numbers are negative or zero no error is reported. If any positive values are found an error indication is returned in the cursor. The actual field(s) that caused the error can be found by using mode 202 of RDBINFO. If an error is found in a point, add, or update, the operation is not performed. If the error is caused by a read, as much of the requested data as possible is returned to the user's application.

ERROR	DESCRIPTION
-3	The source contained only blanks. Zero is output.
-2	Too many decimals existed in the source. The source was truncated to the allowed number of decimals.
-1	The source contained more digits than could correctly be printed or more characters than would fit into the field. This error can be generated for integer, double, real, long, or unsigned values and indicates that a conversion from alphabetic, zoned, or packed format contained more digits or characters than could correctly be printed. For numeric fields, the number has been converted and accepted without change. For alphabetic fields, trailing characters have been truncated.
0	No error.
1	The source contained more digits than could be accepted or the value of the number exceeded the maximum for the data type. The conversion did not take place.
2	Not returned.
3	The source contained an invalid character. The conversion did not take place. This error is returned when an invalid digit is encountered in an alphabetic to numeric conversion. It can also be caused by an invalid digit or sign in a packed or zoned number.
4	Attempted conversion of negative number to logical. The conversion did not take place.
5	The source contained only a sign. The conversion did not take place.
6	Invalid date. The conversion did not take place.

7 The address of the variable is no longer within the user's stack (only returned when bound variables are used).

CURSOR FORMAT

The cursor is the means of communication between a user's application and the RELATE/3000 data base management system. A cursor must be included in every call made to the system. The contents of the cursor are updated by RELATE and should not be modified by the user.

WORD	CONTENTS
1	RELATE/3000 error number.
2	MPE or KSAM file system error number.
3	IMAGE/3000 error number.
4	If the error was caused by, or can be associated with, a position in a passed command line, this contains the character offset from the beginning of the line.
5-6	Reserved.
7-8	The record number of the last record added, read, or updated.
9-10	Reserved.
11	The number of the Host Language Interface routine last called.
12	Contains the number of the field in the key that caused a control break.
13-16	Reserved.
17	File Type: 1 RELATE/3000 2 MPE 3 KSAM/3000 4 IMAGE/3000 6 Selection or View
18	The number of the last executed RELATE/3000 command. This location will contain a zero if the last Host Language Interface intrinsic call was not to RELATE. For a list of the command numbers see Appendix B.
19-20	Reserved.
21	Contains a 1 or a 0. A 1 indicates that records can be added to the file referenced by this cursor.
22	Contains a 1 or a 0. A 1 indicates that records can be updated on the file referenced by this cursor.

23	Contains a 1 or a 0. A 1 indicates that records can be deleted from the file referenced by this cursor.
24	Contains a 1, 2 or 3. A 1 indicates that the cursor references a file to which the user has exclusive access. A 2 indicates semi-exclusive access and a 3 indicates share access.
25	Contains a 1 or a 0. A value of 1 indicates that the file must be locked before operations are performed on the file.
26	Contains a 1 or a 0. A 1 indicates that the user is the creator of the file.
27-28	The count of the number of records not added, updated, or deleted in the last command or since the last RDBPOINT call, because of a security violation.
29-30	The count of the number of records returned from the current SELECT command or the number read since the last RDBPOINT call.
31-32	The count of the records not read because of a security violation in the last command or since the last RDBPOINT.
33	Contains a 1 or a 0. A 1 indicates that the last record could not be added, changed, or deleted because of a security violation.
34	Reserved.
35-36	Indicates the maximum number of records that could be returned from the current file or SELECTion.
37-38	A doubleword quantity indicating the number of milliseconds of real time the last command took to execute. This value is only available if enabled with the $TIME parameter on the SYSTEM command.
39-40	A doubleword quantity indicating the number of milliseconds of CPU time the last command took to execute. This value is only available if enabled with the $CPU parameter on the SYSTEM command.
41-45	Reserved.
46.(0:9)	Reserved.
46.(10:1)	Contains a 1 or a 0. A value of 1 indicates that the last RELATE command was terminated with a Control-Y.
46.(11:1)	Contains a 1 or a 0. A value of 1 indicates that the last RELATE command caused the current index to be changed.
46.(12:1)	Contains a 1 or a 0. A value of 1 indicates that the last RELATE command caused the current path to be changed.

46.(13:1)	Contains a 1 or a 0. A value of 1 indicates that the last output from RELATE was 'PRESS ANY KEY TO CONTINUE'.
46.(14:1)	Contains a 1 or a 0. A value of 1 indicates that RELATE will not execute commands passed to it because an IF command was executed which returned a negative result.
46.(15:1)	Contains a 1 or a 0. A value of 1 indicates that the last RELATE command caused output to the terminal.
47	The number of this cursor.
48	The PIN of the database process assigned to this cursor.
49	The extra data segment number (XDS) of the data segment used for communication between the user's program and the database process.
50	The extra data segment indentifier used for the communications segment.

PROGRAMMATIC CALLS EXAMPLES

On the following pages is a sample program written in BASIC, COBOL, FORTRAN, and SPL, demonstrating usage of all of the programmatic calls to RELATE/3000. These examples are not guaranteed to use excellent programming methodology or to be the best use of the language or calls. They are only attempts to demonstrate how the calls can be utilized in the available languages. They all, however, compile and run correctly.

The program:

1. Uses RDBINIT to initialize cursors for two files.
2. Uses the "OPEN" command with RELATE to open an already existing RELATE/3000 file called VENDORS containing the fields:

   ```
   NAME, A, 20
   VENDNUM, I, 5
   PARTNO, I, 5
   STATE, A, 2
   ```

   and containing an index by VENDNUM.
3. Uses the "CREATE" command with the RELATE call to create a new RELATE/3000 file called PARTS containing the fields:

   ```
   PARTNO, I, 10
   DESCR, A, 20
   COLOR, A, 4
   QTY, I, 5
   ```
4. Uses RDBINFO to ascertain information concerning the VENDORS file.
5. Uses RDBERROR to evaluate errors returned from previous calls.
6. Uses RDBREAD to read through the VENDORS file.
7. Uses RDBADD to add records to the PARTS file.
8. Uses RBCLOSE to close the PARTS file and cursor.
9. Uses RDBBIND to bind fields in the VENDORS file to variables in the program.
10. Uses the "SET INDEX" command with the RELATE call to set the index in VENDORS.
11. Uses RDBPOINT to find a specific record in VENDORS.
12. Uses RDBDELETE to delete a record from VENDORS.
13. Uses RDBREAD to demonstrate use of the variables bound by RDBBIND.

14. Uses RDBUPDATE in conjunction with bound variables to modify records in VENDORS.

15. Uses RDBCLOSE to close the VENDORS file and cursor.

BASIC LANGUAGE HLI EXAMPLE

```
BASEX
  10 REM ............V(*) and P(*) will act as cursors.
  20 INTEGER V1[40],P1[40],V2[20]
  30 INTEGER V[50],P[50]
  40 DIM C$[100],F$[80],E$[120],V$[5],S$[2]
  50 DIM X$[20],Y$[4]
  60 INTEGER C,F,M,V1,F1,N,P,B
  70 INTEGER I,Q,Q1
  80 REM ............integer variables with "2" are used as logicals.
  90 INTEGER V2,P2,F2
 100 REM ...........
 110 REM ............First operation must always be to initialize cursor
 120 REM ............BASIC calls may be made either with
 130 REM ............          CALL functionname      or
 140 REM ...........            *   functionname.
 150 REM ...........
 160 CALL BDBINIT(V[*])
 170 *BDBINIT(P[*])
 180 REM .............Initialize end-of-file logicals.
 190 P2=0
 200 V2=0
 210 REM ............
 220 REM ..............We wish to open the already existing file called
 230 REM ..............VENDORS.  After execution of the command, the
 240 REM .............. cursor V(*) will be associated with the VENDORS
 250 REM .............. file.
 260 REM .............
 270 C$="OPEN FILE VENDORS"
 280 CALL BRELATE(V[*],C$)
 290 REM ..............We wish to create a new file called PARTS
 300 REM .............. containing four fields.  After the command is
 310 REM .............. executed, the cursor P(*) will be associated
 320 REM .............. with the PARTS file.
 330 C$=&
    "CREATE FILE PARTS;FIELDS=(PARTNO,I,10),(DESCR,A,20),(COLOR,A,4),(&
QTY,I,5)"
 340 CALL BRELATE(P[*],C$)
 350 REM ..............We will be looking at information in the VENDORS
 360 REM .............. file concerning its fields NAME and PARTNO.
 370 F$="NAME,PARTNO"
 380 REM ..............We will obtain information about the VENDORS
 390 REM .............. file.  The info that we want is about fields, so
 400 REM .............. we will call RDBINFOR with a mode of 201.  We
 410 REM .............. need to pass it field numbers, so we use F1 as
 420 REM .............. the field number counter.
 430 M=201
 440 F1=0
 450 REM ...............If no error is found (error number in V[1]) and
 460 REM ...............we have not yet found the NAME field (data about
 470 REM ...............field is in V2(*)), then we continue to obtain
 480 REM ...............info.
```

```
490 IF V[1]=0 AND NOT FNE(V2[1],"NA") AND NOT FNE(V2[2],"ME") THEN DO
500   F1=F1+1
510   CALL BDBINFO(V[*],M,F1,V2[*])
520   GOTO 490
530 DOEND
540 REM ...............If an error has been found, ascertain the messag
550 REM ...............and return.
560 IF V[1]<>0 THEN DO
570   M=1
580   CALL BDBERROR(V[*],M,E$)
590   PRINT E$
600   END
610 DOEND
620 REM .............If no error was found, then the length of the NAME
630 REM .............field is in V2[8].
640 REM .............Proceed to read the first record from the VENDORS
650 REM ............. file.  NAME and PARTNO will be placed into V1(*)
660 N=V2[8]
670 CALL BDBREAD(V[*],V2,F$,V1[*])
680 REM .............If no end-of-file found, then compare the part
690 REM ............. number.  If found the proper one, then read data
700 REM ............. for the PARTS file into P1(*) and add that data
710 REM ............. to the PARTS file.
720 IF NOT P2 AND NOT V2 THEN DO
730   REM .............Data for the PARTNO field begins right after
740   REM ............. the end of the NAME field.
750   P=V1[N+1]
760   IF 1230<=P AND P<=1239 THEN DO
770     INPUT "PARTNO?",Q
780     INPUT "DESCR?",X$
790     INPUT "COLOR?",Y$
800     INPUT "QTY?",Q1
810     F$="@"
820     CALL BDBADD(P[*],P2,F$,Q,X$,Y$,Q1)
830   DOEND
840   F$="NAME,PARTNO"
850   CALL BDBREAD(V[*],V2,F$,V1[*])
860   GOTO 720
870 DOEND
880 IF P2 THEN GOTO 1310
890 REM ............RDBCLOSE closes access to a cursor, but leaves
900 REM ............ its associated file open.  Close the file
910 REM ............ first, then the cursor.
920 C$="CLOSE FILE PARTS"
930 CALL BRELATE(P[*],C$)
940 CALL BDBCLOSE(P[*])
950 REM ............Whatever the definition of the field VENDNUM was
960 REM ............ in the file, it wil now be read into the program
970 REM ............ as an ASCII string of 5 chars.  It will be read
980 REM ............ into the variable V$.  V$[1;5] must be set to
990 REM ............ something in order to set its length as RELATE
1000 REM ............ calls will not set the length for bound variables
1010 F$="VENDNUM;SIZE=5;TYPE=ALPHA"
1020 CALL BDBBIND(V[*],F$,V$)
```

```
1030 V$[1;5]=" "
1040 REM .............Whatever printlength and type are defined for
1050 REM ............. STATE in VENDORS file will be used in the program
1060 REM ............. Data will be associated with the variable S$.
1070 REM ............. See note on V$ above.
1080 F$="STATE"
1090 *BDBBIND(V[*],F$,S$)
1100 S$[1;2]=" "
1110 REM ............. Set index by VENDNUM.
1120 C$="SET INDEX VENDNUM"
1130 CALL BRELATE(V[*],C$)
1140 FOR I=1000 TO 5000 STEP 100
1150   CALL BDBPOINT(V[*],0,F2,I)
1160   IF F2 THEN DO
1170     CALL BDBDELETE(V[*])
1180     CALL BDBPOINT(V[*],0,F2,I)
1190     IF F2 THEN DO
1200       CALL BDBREAD(V[*],V2)
1210       IF S$[1;2]="CA" THEN DO
1220         V$[1;5]="00000"
1230         CALL BDBUPDATE(V[*])
1240       DOEND
1250     DOEND
1260   DOEND
1270 NEXT I
1280 F$="CLOSE FILE VENDORS"
1290 CALL BRELATE(V[*],C$)
1300 CALL BDBCLOSE(V[*])
1310 END
1320 REM ..............Function compares the integer value N with
1330 REM .............. the 2 characters in the string.  Returns 1
1340 REM ............   if equal.
1350 DEF INTEGER FNE(N,X$)
1360   INTEGER M
1370   M=256*NUM(X$[1;1])
1380   M=M+NUM(X$[2;1])
1390   IF N=M THEN RETURN 1
1400   ELSE RETURN 0
1410 FNEND
1420 REM ...........Function places passed 2 characters into
1430 REM ........... integer starting at Pth character of X$.
1440 DEF INTEGER FNC(X$,P)
1450   INTEGER N
1460   N=256*NUM(X$[P;1])
1470   N=N+NUM(X$[P+1;1])
1480   RETURN N
1490 FNEND
```

COBOL LANGUAGE HLI EXAMPLE

```
$CONTROL USLINIT,MAP
  IDENTIFICATION DIVISION.
     PROGRAM-ID.     RELATE-SAMPLE.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
    WORKING-STORAGE SECTION.
    77  COM-LEN                 PIC S9999 USAGE IS COMPUTATIONAL.
    77  F-LEN                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  MODE-NUM                  PIC S9999 USAGE IS COMPUTATIONAL.
    77  V-INFO-SIZE  PIC S9999 USAGE IS COMPUTATIONAL VALUE 20.
    77  FIELD-NO                  PIC S9999 USAGE IS COMPUTATIONAL.
    77  B-LEN                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  NAME-LEN                  PIC S9999 USAGE IS COMPUTATIONAL.
    77  PART-NO                   PIC S9999 USAGE IS COMPUTATIONAL.
    77  BIND                      PIC S9999 USAGE IS COMPUTATIONAL.
    77  WORDS-IN-KEY              PIC S9999 USAGE IS COMPUTATIONAL.
    77  I                         PIC S9999 USAGE IS COMPUTATIONAL.
    77  V-EOF                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  P-EOF                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  FOUND                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  DUMMY                     PIC S9999 USAGE IS COMPUTATIONAL.
    77  QTY                       PIC S9999 USAGE IS COMPUTATIONAL.
    01  TEMP-IO               PIC XXXX.
    01  I-O-AREA.
        04  BUFFER                PIC X(120).
    01  NUM-AREA REDEFINES I-O-AREA.
        04 BUFF-NUM OCCURS 60 TIMES PIC S9999 USAGE IS COMP.
*.........V-CUR and P-CUR are the cursors.
    01  V-CUR
        04 VEND-CUR OCCURS 50 TIMES PIC S9999 USAGE IS COMP.
    01  P-CUR.
        04 PART-CUR OCCURS 50 TIMES PIC S9999 USAGE IS COMP.
    01  V-DATA.
        04 VEND-DATA OCCURS 40 TIMES PIC S9999 USAGE IS COMP.
    01  V-DATA-C REDEFINES V-DATA PIC X(80).
    01  P-DATA.
        04 PART-DATA OCCURS 40 TIMES PIC S9999 USAGE IS COMP.
    01  P-DATA-C REDEFINES P-DATA.
        04   P-DATA-CA            PIC XX.
        04   P-DATA-CB            PIC X(20).
        04   P-DATA-CC            PIC X(4).
        04   P-DATA-CD            PIC X(54).
    01  V-INFO
        04 VEND-INFO OCCURS 20 TIMES PIC S9999 USAGE IS COMP.
    01  V-INFO-C REDEFINES V-INFO.
        04 V-INFO-NAME          PIC X(4).
        04 FILLER               PIC X(36).
    01  COMMAND.
        04 COMMAND-LINE OCCURS 100 TIMES PIC S9999 USAGE IS COMP.
    01  COMM-C REDEFINES COMMAND PIC X(200).
    01  FIELDS.
```

```
           04 FIELD-LIST OCCURS 40 TIMES PIC S9999 USAGE IS COMP.
       01  FIELDS-C REDEFINES FIELDS PIC X(80).
       01  V-NUM.
           04 VEND-NUM OCCURS 3 TIMES PIC S9999 USAGE IS COMP.
       01  V-NUM-C REDEFINES V-NUM PIC X(6).
       01  STATE.
           04 STATE-FIELD PIC S9999 USAGE IS COMP.
       01  STATE-C REDEFINES STATE PIC X(2).
   PROCEDURE DIVISION.
    MAIN-ROUTINE.
 *.........Cursors must be initialized before using.
          CALL "RDBINIT" USING V-CUR.
          CALL "RDBINIT" USING P-CUR.
 *.......... initialize end-of-file logicals.
          MOVE ZERO TO P-EOF.
          MOVE ZERO TO V-EOF.
 *.........Now open the already existing file called VENDORS.
 *.........The OPEN command has a length of 17 characters.
          MOVE "OPEN FILE VENDORS" TO COMM-C.
          MOVE 17 TO COM-LEN.
 *...........After execution of the OPEN command, V-CUR will be
 *...........associated with the VENDORS file.
          CALL "RELATE" USING V-CUR, COMMAND, COM-LEN.
 *...........Create a new file called PARTS with four fields.  If
 *...........a length of -1 is given, the command ends with a null
 *...........or a backslash.  After the command is executed, PCUR
 *...........will be associated with the PARTS file.
          MOVE "CREATE FILE PARTS;FIELDS=(PARTNO,I,10),(DESCR,A,20),
 -          "(COLOR,A,4),(QTY,I,5)\" TO COMM-C.
          MOVE -1 TO COM-LEN.
          CALL "RELATE" USING P-CUR, COMMAND, COM-LEN.
 *...........Set up partial field list for VENDORS file with
 *...........11 characters.
          MOVE "NAME,PARTNO" TO FIELDS-C.
          MOVE 11 TO F-LEN.
 *...........Find information about fields.
          MOVE 201 TO MODE-NUM.
          MOVE ZERO TO FIELD-NO.
 *...........While no errors encountered, look for NAME field.
 *...........Obtain info about the FIELDNOs and put it in V-INFO.
   LOOK-FOR-NAME.
          IF VEND-CUR(1) IS = ZERO AND V-INFO-NAME IS NOT = "NAME"
              AND VEND-INFO(5) IS NOT = 4
              ADD 1 TO FIELD-NO,
           CALL "RDBINFO" USING V-CUR, MODE-NUM, FIELD-NO, V-INFO
 -               , V-INFO-SIZE,
              GO TO LOOK-FOR-NAME.
 *.............Error found. Return error message
          IF VEND-CUR(1) IS NOT = ZERO
              MOVE 120 TO B-LEN,
              MOVE 1 TO MODE-NUM,
              CALL "RDBERROR" USING V-CUR, MODE-NUM, NUM-AREA, B-LEN,
              DISPLAY I-O-AREA,
              GO TO END-IT.
```

```
*...............Number of words of data in the NAME field.
      MOVE VEND-INFO(8) TO NAME-LEN.
*...............Read first record from VENDORS file.  Place
*...............NAME ans PARTNO data into V-DATA.
      CALL "RDBREAD" USING V-CUR, V-EOF, FIELDS, F-LEN, V-DATA.
*...............While no end-of-file is found, read info from
*...............terminal into P-DATA and add records to PARTS
*...............file using info in P-DATA.
      PERFORM ADD-A-RECORD UNTIL V-EOF IS NOT = ZERO OR
        P-EOF IS NOT = ZERO.
 ADD-A-RECORD.
             MOVE NAME-LEN TO I.
             ADD 1 TO I.
         MOVE VEND-DATA(I) TO PART-NO.
         IF PART-NO IS > 1229 AND PART-NO IS < 1240
           MOVE "PARTNO?" TO BUFFER,
           DISPLAY I-O-AREA,
           MOVE ZEROS TO TEMP-IO,
           ACCEPT TEMP-IO,
           MOVE TEMP-IO TO PART-DATA(1),
           MOVE "DESCR?" TO BUFFER,
           DISPLAY I-O-AREA,
           MOVE SPACES TO BUFFER,
           ACCEPT I-O-AREA,
           MOVE BUFFER TO P-DATA-CB,
           MOVE "COLOR?" TO BUFFER,
           DISPLAY I-O-AREA,
           MOVE SPACES TO BUFFER,
           ACCEPT I-O-AREA,
           MOVE BUFFER TO P-DATA-CC,
           MOVE "QTY?" TO BUFFER,
           DISPLAY I-O-AREA,
           MOVE ZEROS TO TEMP-IO,
           ACCEPT TEMP-IO,
           MOVE TEMP-IO TO PART-DATA(14),
           MOVE "@" TO FIELDS-C,
           MOVE 1 TO F-LEN,
           CALL "RDBADD" USING P-CUR, P-EOF, FIELDS, F-LEN, P-DATA.
        MOVE "NAME,PARTNO" TO FIELDS-C.
        MOVE 11 TO F-LEN.
        CALL "RDBREAD" USING V-CUR, V-EOF, FIELDS, F-LEN, V-DATA.
 END-OF-ADD.
     IF P-EOF IS NOT = ZERO THEN GO TO END-IT.
*...............RDBCLOSE closes access to cursor but leaves the
*...............file open.  Close the file first.
     MOVE "CLOSE FILE PARTS\" TO COMM-C.
     MOVE -1 TO COM-LEN.
     CALL "RELATE" USING P-CUR, COMMAND, COM-LEN.
     CALL "RDBCLOSE" USING P-CUR.
*...............Binding variables.  Whatever the definition of the
*...............field VENDNUM in the file was, it will be read as
*...............an ASCII string of 5 characters into the variable
*...............V-NUM.  An FLEN of -1 assumes line ends with a null
     MOVE "VENDNUM;SIZE=5;TYPE=ALPHA" TO FIELDS-C.
```

```
        MOVE -1 TO F-LEN.
        MOVE 1 TO BIND.
        CALL "RDBBIND" USING V-CUR, BIND, FIELDS, F-LEN, V-NUM.
*.............Bind the STATE field to the STATE variable.
*.............Whatever the printlength and type are in the
*.............VENDOR file will be used by the program.
        MOVE "STATE" TO FIELDS-C.
        MOVE 5 TO F-LEN.
        CALL "RDBBIND" USING V-CUR, BIND, FIELDS, F-LEN, STATE.
*.............Set index in VENDORS file by VENDNUM.
        MOVE "SET INDEX VENDNUM\" TO COMM-C.
        MOVE -1 TO COM-LEN.
        CALL "RELATE" USING V-CUR,  COMMAND,  COM-LEN.
*..........Assumption made that VENDNUM is an integer in the
*..........VENDORS file.  Search for vendor numbers by 100's.
        PERFORM SEARCH-FOR-RECORDS THROUGH SEARCH-EX
            VARYING I FROM 1000 BY 100 UNTIL I IS > 5000.
            MOVE "CLOSE FILE VENDORS" TO COMM-C.
            MOVE -1 TO COM-LEN.
            CALL "RELATE" USING V-CUR, COMMAND, COM-LEN.
            CALL "RDBCLOSE" USING V-CUR.
 END-IT.
            STOP RUN.
 SEARCH-FOR-RECORDS.
             MOVE 1 TO WORDS-IN-KEY.
             MOVE ZERO TO DUMMY.
          CALL "RDBPOINT" USING V-CUR, I, WORDS-IN-KEY, DUMMY, FOUND.
             IF FOUND IS NOT = ZERO THEN PERFORM FOUND-ONE
          THROUGH FOUND-EX.
  SEARCH-EX.    EXIT.
 FOUND-ONE.
                   CALL "RDBDELETE" USING V-CUR.
                   CALL "RDBPOINT" USING V-CUR, I, WORDS-IN-KEY, DUMMY
-                          , FOUND.
                   IF FOUND IS NOT = ZERO THEN PERFORM FOUND-ANOTHER.
  FOUND-EX.        EXIT.
 FOUND-ANOTHER.
                     MOVE -2 TO F-LEN.
                     CALL "RDBREAD" USING V-CUR, V-EOF, FIELDS, F-LEN
-                          , V-DATA.
                     IF STATE-C IS = "CA"
                         MOVE ZEROES TO V-NUM-C,
                         CALL "RDBUPDATE" USING V-CUR,  FIELDS
-                           , F-LEN, V-DATA.
```

FORTRAN LANGUAGE HLI EXAMPLE

```
$CONTROL USLINIT,FILE=5-6,LOCATION
C.........VCUR and PCUR are the cursors.
          INTEGER VCUR(50),PCUR(50)
          INTEGER VDATA(40),PDATA(40),VINFO(20)
          INTEGER COMM(100),FIELDS(40),VNUM(3),STATE(2)
          INTEGER IBUFF(40)
          INTEGER COMLEN,FLEN,MODE,VINFOSIZE,FIELDNO,WORDSINKEY
          INTEGER BLEN,NAMELEN,PARTNO,BIND,OUTDEV,INVALUE
          LOGICAL VEOF,PEOF,FOUND,DUMMY
          CHARACTER CCOMM*200,CFIELDS*80,CVINFO*40,BUFFER*80
          CHARACTER CSTATE*4,CVNUM*6,CPDATA(80),INVAL*20,CINCHAR*80
          CHARACTER INCHAR(80)
          EQUIVALENCE (COMM,CCOMM),(FIELDS,CFIELDS),(VINFO,CVINFO)
          EQUIVALENCE (IBUFF,BUFFER),(PDATA,CPDATA),(VNUM,CVNUM)
          EQUIVALENCE (STATE,CSTATE),(INCHAR,CINCHAR)
C........Housekeeping
          VEOF=.FALSE.
          PEOF=.FALSE.
          VINFOSIZE=20
          INDEV=5
          OUTDEV=6
C.........Must initialize cursors before using
          CALL RDBINIT(VCUR)
          CALL RDBINIT(PCUR)
C..........Now open the already existing file called VENDORS.
C..........The OPEN command has a length of 17 characters.
          CCOMM="OPEN FILE VENDORS"
          COMLEN=17
C...........After execution of the OPEN command, VCUR will be
C...........associated with the VENDORS file.
          CALL RELATE(VCUR,COMM,COMLEN)
C...........Create a new file called PARTS with four fields.
C...........If a length of -1 is given, the command ends with
C...........a null or a backslash.  After the command is executed,
C...........PCUR will be associated with the PARTS file.
          CCOMM="CREATE FILE PARTS;FIELDS=(PARTNO,I,10),(DESCR,A,20),
     +(COLOR,A,4),(QTY,I,5)\"
          COMLEN=-1
          CALL RELATE(PCUR,COMM,COMLEN)
C...........Set up partial field list for VENDORS file with
C...........11 characters.
          CFIELDS="NAME,PARTNO"
          FLEN=11
C...........Find information about fields.
          MODE=201
          FIELDNO=0
C...........While no errors encountered, look for NAME field.
C...........Obtain info about the FIELDNOs and put it in VINFO.
100       IF (VCUR(1).EQ.0) GOTO 200
C.............Error found. Return error message
              BLEN=80
```

```
            MODE=1
            CALL RDBERROR(VCUR,MODE,IBUFF,BLEN)
         IF (BLEN.LT.0) THEN BLEN=80
            WRITE(OUTDEV,300)BUFFER
300         FORMAT("0",A80)
            RETURN
200      IF (CVINFO[1:4].EQ."NAME".AND.VINFO(6).EQ.4) GOTO 400
            FIELDNO=FIELDNO+1
            CALL RDBINFO(VCUR,MODE,FIELDNO,VINFO,VINFOSIZE)
            GOTO 100
C...............Number of words of data in the NAME field.
400      NAMELEN=VINFO(8)
C.............. Read first record from VENDORS file.  Place
C.............. NAME and PARTNO data into VDATA.
         CALL RDBREAD(VCUR,VEOF,FIELDS,FLEN,VDATA)
C...............While no end-of-file encountered, read info
C...............from terminal into PDATA and add records to
C...............PARTS file using info in PDATA.
500      IF (VEOF.OR.PEOF) GOTO 700
C...............The first NAMELEN words of VDATA are taken up
C...............with the field NAME.  Assuming that PARTNO is
C...............an integer field, the following word will be
C...............the field PARTNO.
            PARTNO=VDATA(NAMELEN+1)
            IF(1230.GT.PARTNO OR.PARTNO.GT.1239) GOTO 600
                 DISPLAY "PARTNO"
                 ACCEPT PDATA(1)
                 DISPLAY "DESCR"
                 READ(INDEV,520)CINCHAR
520              FORMAT(A20)
                 DO 522 I=1,20
522              CPDATA(2+I)=INCHAR(I)
                 DISPLAY "COLOR"
                 READ(INDEV,530)CINCHAR
530              FORMAT(A4)
                 DO 532 I=1,4
532              CPDATA(22+I)=INCHAR(I)
                 DISPLAY "QTY"
                 ACCEPT PDATA(14)
C.................Set up for adding data to PARTS file - add all
C.................fields.
                 CFIELDS="@"
                 FLEN=1
                 CALL RDBADD(PCUR,PEOF,FIELDS,FLEN,PDATA)
C.................Read next record from VENDORS file.
600          CFIELDS="NAME,PARTNO"
             FLEN=11
             CALL RDBREAD(VCUR,VEOF,FIELDS,FLEN,VDATA)
             GOTO 500
700      IF (PEOF) RETURN
C............RDBCLOSE closes access to cursor but leaves the
C............file open. Close the file first.
         CCOMM="CLOSE FILE PARTS\"
         COMLEN=-1
```

```
        CALL RELATE(PCUR,COMM,COMLEN)
        CALL RDBCLOSE(PCUR)
C...
C............Binding variables.  Whatever the definition of
C............the field VENDNUM in the file was, it will be read
c............as an ASCII string of 5 characters into the variable
C............VNUM.  An FLEN of -1 assumes line ends with a backslash.
        CFIELDS="VENDNUM;SIZE=5;TYPE=ALPHA\"
        FLEN=-1
        BIND=1
        CALL RDBBIND(VCUR,BIND,FIELDS,FLEN,VNUM)
       IF (VCUR(1).NE.0) DISPLAY VCUR(1)
        CFIELDS="STATE"
C............Bind the STATE field to the STATE variable.
C............Whatever the printlength and type are in the VENDOR
C............file will be used by the program.
        FLEN=5
C.........initialize end-of-file logicals.
         VEOF=.FALSE.
         PEOF=.FALSE.
        CALL RDBBIND(VCUR,BIND,FIELDS,FLEN,STATE)
C............Set index in VENDORS file by VENDNUM
        CCOMM="SET INDEX VENDNUM\"
        COMLEN=-1
        CALL RELATE(VCUR,COMM,COMLEN)
        DO 800 I=1000,5000,100
C..............Assumption made that VENDNUM is an integer in the
C..............VENDORS file.  Search for vendor numbers by 100's.
             WORDSINKEY=1
             DUMMY=.FALSE.
         FOUND=.FALSE.
             CALL RDBPOINT(VCUR,I,WORDSINKEY,DUMMY,FOUND)
             IF(.NOT.FOUND) GOTO 800
C...............A record was found with VENDNUM exactly matching
C...............the value of I.  Delete the record.  Search for
C...............another match.  If found, read it(RDBPOINT does
C...............not perform a read).  FIELDS does not need to be
C...............initialized as FLEN=-2 which means to read bound
C...............variables.  VDATA will not be filled in this case.
               CALL RDBDELETE(VCUR)
               CALL RDBPOINT(VCUR,I,WORDSINKEY,DUMMY,FOUND)
               IF(.NOT.FOUND)GOTO 800
                 FLEN=-2
                 CALL RDBREAD(VCUR,VEOF,FIELDS,FLEN,VDATA)
                 IF(CSTATE[1:2].NE."CA")GOTO 800
                   CVNUM="00000"
C..........FLEN is still -2, so binds are still in effect.
C..........Therefore FIELDS is ignored and all bound fields are
C..........updated.  VDATA is ignored and current values of bound
C..........variables are used to update the files.
                   CALL RDBUPDATE(VCUR,FIELDS,FLEN,VDATA)
800       CONTINUE
         CCOMM="CLOSE FILE VENDORS\"
         COMLEN=-1
```

```
      CALL RELATE(VCUR,COMM,COMLEN)
      CALL RDBCLOSE(VCUR)
      STOP
      END
```

```
$CONTROL USLINIT
  BEGIN
    INTRINSIC READ,PRINT,BINARY;
    INTEGER ARRAY   V'CUR(0:49),P'CUR(0:49);
    INTEGER ARRAY   V'DATA(0:39),P'DATA(0:39),V'INFO(0:19);
    INTEGER ARRAY   COMM(0:99),FIELDS(0:39),V'NUM(0:2),STATE(0:1);
    INTEGER         COM'LEN,F'LEN,MODE,V'INFO'SIZE:=20,FIELD'NO;
    INTEGER         B'LEN,NAME'LEN,PART'NO,BIND,WORDS'IN'KEY,I;
    LOGICAL         V'EOF:=FALSE,P'EOF:=FALSE,FOUND,DUMMY;
    LOGICAL ARRAY   BUFFER(0:59);
    INTEGER POINTER IBUFF=BUFFER;
    BYTE    ARRAY   CBUFF(*)=BUFFER;
    BYTE    POINTER CV'INFO=V'INFO;
    BYTE    ARRAY   CFIELDS(*)=FIELDS;
    BYTE    ARRAY   CP'DATA(*)=P'DATA;

    PROCEDURE RELATE(CUR,COMMAND,LEN);
    INTEGER ARRAY CUR,COMMAND;
    INTEGER       LEN;
    OPTION EXTERNAL;

    PROCEDURE RDBADD(CUR,EOF,LIST,LISTLEN,SOURCE);
    INTEGER ARRAY CUR,LIST,SOURCE;
    INTEGER       LISTLEN;
    LOGICAL       EOF;
    OPTION EXTERNAL;

    PROCEDURE RDBBIND(CUR,BIND,RELATEVAR,VARLEN,VAR);
    INTEGER ARRAY CUR,RELATEVAR,VAR;
    INTEGER       BIND,VARLEN;
    OPTION EXTERNAL;

    PROCEDURE RDBCLOSE(CUR);
    INTEGER ARRAY CUR;
    OPTION EXTERNAL;

    PROCEDURE RDBDELETE(CUR);
    INTEGER ARRAY CUR;
    OPTION EXTERNAL;

    PROCEDURE RDBERROR(CUR,MODE,ERROR,ERRLEN);
    INTEGER ARRAY CUR,ERROR;
    INTEGER       MODE,ERRLEN;
    OPTION EXTERNAL;

    PROCEDURE RDBINFO(CUR,MODE,QUAL,INFO,WORDS);
    INTEGER ARRAY CUR,INFO;
    INTEGER       MODE,QUAL,WORDS;
    OPTION EXTERNAL;

    PROCEDURE RDBINIT(CUR);
```

```
INTEGER ARRAY CUR;
OPTION EXTERNAL;

PROCEDURE RDBPOINT(CUR,KEY,WORDS,DUMMY,FOUND);
INTEGER ARRAY CUR,KEY;
INTEGER       WORDS,DUMMY;
LOGICAL       FOUND;
OPTION EXTERNAL;

PROCEDURE RDBREAD(CUR,EOF,LIST,LISTLEN,DEST);
INTEGER ARRAY CUR,LIST,DEST;
INTEGER       LISTLEN;
LOGICAL       EOF;
OPTION EXTERNAL;

PROCEDURE RDBUPDATE(CUR,LIST,LISTLEN,SOURCE);
INTEGER ARRAY CUR,LIST,SOURCE;
INTEGER       LISTLEN;
OPTION EXTERNAL;


RDBINIT(V'CUR);                  <<must initialize cursors before using>>
RDBINIT(P'CUR);

                                       <<initialize end-of-file logicals>>
P'EOF:=FALSE;
V'EOF:=FALSE;

MOVE COMM:="OPEN FILE VENDORS";    <<we want to open the already       >
                                     << existing file called VENDORS>>
COM'LEN:=17;           <<there are 17 characters in the OPEN command>>
RELATE(V'CUR,COMM,COM'LEN);       <<execute the OPEN command.  V'CUR>>
                                  << will now be associated with the>>
                                  << VENDORS file.                  >>

MOVE COMM:="CREATE FILE PARTS;FIELDS=(PARTNO,I,10),(DESCR,A,20),
 (COLOR,A,4),(QTY,I,5)\";         <<create a new file called PARTS   >>
                                  << containing four fields.         >>
COM'LEN:=-1;           << a length of -1 indicates that the command>>
                        << is terminated with a null or a backslash>>
RELATE(P'CUR,COMM,COM'LEN);   <<execute the CREATE command.  P'CUR>>
                       <<will now be associated with the PARTS file>>
MOVE FIELDS:="NAME,PARTNO";   <<partial field list for VENDORS file>>
F'LEN:=11;                       <<11 characters in the field list>>

MODE:=201;                                     <<want info about fields>>
FIELD'NO:=0;                           <<field # counter for RDBINFO>>
WHILE V'CUR=0 AND CV'INFO<>"NAME" AND V'INFO(5)<>4
                                               << while no errors found>>
                                               <<find field called NAME>>
 DO BEGIN
  FIELD'NO:=FIELD'NO+1;
  RDBINFO(V'CUR,MODE,FIELD'NO,V'INFO,V'INFO'SIZE);    <<obtain info>
             <<about field number FIELD'NO and place it in V'INFO>.
  END;
```

```
    IF V'CUR<>0 THEN BEGIN
                                                               <<error encountered>>
      B'LEN:=120;                        <<number of characters in buffer area>>
      MODE:=1;                                               <<return error message>>
      RDBERROR(V'CUR,MODE,IBUFF,B'LEN);
      PRINT(BUFFER,60,%40);
      RETURN;
      END;

    NAME'LEN:=V'INFO(7);    <<number of words of data in the NAME field>>

    RDBREAD(V'CUR,V'EOF,FIELDS,F'LEN,V'DATA); <<read first record from>>
                                              <<VENDORS file and place the  >>
                                              <<NAME and PARTNO fields' data>>
                                                              << into V'DATA>>

    WHILE NOT V'EOF AND NOT P'EOF DO BEGIN      <<while neither file has>>
                                                       <<reached end-of-file>>
      PART'NO:=V'DATA(NAME'LEN); <<the first NAME'LEN words are taken >>
                              <<up with the field NAME.  Assuming that >>
                            <<PARTNO is an integer field, the following>>
                                        <<word will be the field PARTNO>>
      IF 1230<=PART'NO<=1239 THEN BEGIN
        P'DATA:="  ";
        MOVE P'DATA(1):=P'DATA,(39);
        MOVE BUFFER:="PARTNO?";
        PRINT(BUFFER,-7,%40);
        B'LEN:=READ(BUFFER,-120);
        P'DATA:=BINARY(CBUFF,B'LEN);
        MOVE BUFFER:="DESCR?";
        PRINT(BUFFER,-6,%40);
        B'LEN:=READ(BUFFER,-120);
        MOVE CP'DATA(2):=CBUFF,(B'LEN);
        MOVE BUFFER:="COLOR?";
        PRINT(BUFFER,-6,%40);
        B'LEN:=READ(BUFFER,-120);
        MOVE CP'DATA(22):=CBUFF,(B'LEN);
        MOVE BUFFER:="QTY?";
        PRINT(BUFFER,-4,%40);
        B'LEN:=READ(BUFFER,-120);
        P'DATA(13):=BINARY(CBUFF,B'LEN);
        MOVE CFIELDS:="@";     <<set up for ADDing data to PARTS file - >>
                                           << we want to add all fields>>
        F'LEN:=1;
        RDBADD(P'CUR,P'EOF,FIELDS,F'LEN,P'DATA); <<add a record to the>>
                                                <<PARTS file using the  >>
                                               <<information in P'DATA  >>
        END;
    MOVE CFIELDS:="NAME,PARTNO";
    F'LEN:=11;
    RDBREAD(V'CUR,V'EOF,FIELDS,F'LEN,V'DATA);<<read next record from>>
                                                        <<VENDORS file>>
    END;
  IF P'EOF THEN RETURN;
  MOVE COMM:="CLOSE FILE PARTS\";
```

```
COM'LEN:=-1;
RELATE(P'CUR,COMM,COM'LEN);<<RDBCLOSE closes the cursor but leaves>>
                           <<the file open.  Close the file first>>

RDBCLOSE(P'CUR);

MOVE FIELDS:=("VENDNUM;SIZE=5;TYPE=ALPHA",0);         <<whatever the >>
     <<definition of the field VENDNUM in the file was, it will be>>
                                  << read into the program as an>>
                                  <<ASCII string of 5 characters>>
F'LEN:=-1;                 <<assumes string terminates with a null>>
BIND:=1;
RDBBIND(V'CUR,BIND,FIELDS,F'LEN,V'NUM);          <<V'NUM will contain >>
                                  <<the ASCII string from VENDNUM >>
                                  <<whenever a read is performed>>
MOVE FIELDS:="STATE";             <<will use whatever printlength>>
                                  <<and type are defined for STATE>>
                                        << in the VENDOR file>>

F'LEN:=5;
RDBBIND(V'CUR,BIND,FIELDS,F'LEN,STATE);

MOVE COMM:=("SET INDEX VENDNUM",0);             <<set index by VENDNUM>>
COM'LEN:=-1;                <<terminates with a backslash or null>>
RELATE(V'CUR,COMM,COM'LEN);

FOR I:=1000 STEP 100 UNTIL 5000 DO BEGIN
  WORDS'IN'KEY:=1;
  DUMMY:=FALSE;
  RDBPOINT(V'CUR,I,WORDS'IN'KEY,DUMMY,FOUND);<<assumptionmade that>>
                                     <<VENDNUM is an integer in>>
                                           <<the VENDORS file>>
  IF FOUND THEN BEGIN             <<a record was found with VENDNUM>>
                                  <<exactly matching the value of I>>
                                             <<delete the record>>
    RDBDELETE(V'CUR);                     <<look for another match>>

    RDBPOINT(V'CUR,I,WORDS'IN'KEY,DUMMY,FOUND);
    IF FOUND THEN BEGIN
      F'LEN:=-2;
      RDBREAD(V'CUR,V'EOF,FIELDS,F'LEN,V'DATA);       <<Read the next>>
                    <<record. POINT does not read a record.  A read>>
                            << must be performed to actually obtain>>
                          <<data.   FIELDS has not been initialized>>
                        <<because a BIND has been performed on the>>
                           <<file and F'LEN has been set to -2 to >>
                        <<indicate that the bound variables should>>
                           <<be read.  V'DATA will not be filled in>>
                                 <<this case.                      >>
      IF STATE="CA" THEN BEGIN
        MOVE V'NUM:="00000";
        RDBUPDATE(V'CUR,FIELDS,F'LEN,V'DATA);     <<FLEN is still-2.>>
<<Therefore,BINDs are still in effect. Therefore FEILDS is ignored and>>
           <<all bound variables are updated, and V'DATA is ignored>>
               <<and the current values of the bound variables are >>
                                          <<used to update the file.>>

        END;
```

```
            END;
          END;
        END;
      MOVE COMM:="CLOSE FILE VENDORS\";
      COM'LEN:=-1;
      RELATE(V'CUR,COMM,COM'LEN);
      RDBCLOSE(V'CUR);
      END.
```

SECTION 4

F I L E S Y S T E M D E S C R I P T I O N S

FILE SYSTEM DESCRIPTIONS

RELATE/3000 can manipulate IMAGE, KSAM, and MPE files in addition to RELATE files. This allows the user great freedom in defining his application system because the file system that most closely meets his requirements for speed, efficient multi-user access, and flexibility can be used.

This section describes the methods of accessing IMAGE databases, RELATE/3000 formatted files, KSAM/3000 files, and MPE files. The types of operations that are allowed on files of each type are also presented.

ACCESS TO IMAGE DATABASES

IMAGE is a data base management system supported by HP and is designed for an interactive, transaction oriented environment. IMAGE allows two file (dataset) levels: master sets and detail sets.

Master sets are used to store information on uniquely identifiable entities. Each master set contains a single field key. Data is stored according to a hashing algorithm performed on the key. The key values must be unique. This key is referred to as a search item.

Detail sets are used to store information concerning related events or items. In an Accounts Receivable system a particular customer's outstanding invoices would most likely be maintained in a detail data set. Each detail set is normally linked to one or more master sets through a single field. These linkages are accomplished by physical pointers in the database and are referred to as paths. A record may not be added to a detail data set unless a record exists in each master set that the detail set is associated with.

A third type of set (an automatic master) can also exist in an IMAGE database. This set is maintained automatically by IMAGE and must be related to at least one detail set. The set is only allowed to contain a single data item. The set is normally used to maintain paths on items that are extremely numerous or do not need to be verified against a master set.

Creating An Image Database

RELATE cannot create or initialize an IMAGE database. This can be done with programs provided by HP. After the database has been created, RELATE can be used on the sets within the database as if these sets existed as individual files. Thus, the user can copy information from one set to another, from or to KSAM, MPE, or RELATE files and even to another IMAGE database.

RELATE is considered a user program by the IMAGE system and cannot violate any constraints enforced by IMAGE. This means, for example, that records added to detail sets must contain valid master set search items. When an IMAGE error is encountered, a command will terminate. A user should not request an operation that could result in an error. For example, if master records for all customers in Denver should be deleted, the user should first verify that no detail records depend on the existence of these masters.

Accessing An Image Database

Accessing an IMAGE database requires the use of two OPEN commands. First, the database must be opened; second, the particular set(s) that will be used must be opened.

To open an IMAGE database, the OPEN DATABASE command is used with a TYPE of IMAGE. This command must also include any required password and may include the access mode desired. This OPEN physically opens the database.

A second OPEN command must be issued for the particular dataset desired. This command must include the set name desired and the DATABASE keyword followed by the name of the database opened in the first step. The database name must be specified in order for RELATE to know which database the set is in since several databases (possibly using the same schema) can be open at one time.

RELATE makes no distinction between master sets and detail sets. It is possible to open several sets within a single database.

When a set is opened, RELATE uses standard IMAGE calls to create the structure for the set. During this process, fieldnames may be adjusted by RELATE. This adjustment consists of removing all non-alphabetic, non-numeric characters and then limiting the length of the name to ten characters. This operation may cause duplicate names which can be adjusted by using the MODIFY FIELD command. RELATE uses the field numbers supplied by IMAGE to access data so the change of name does not cause file access errors. RELATE can correctly handle all IMAGE data types except sub-items which are not aligned on a word boundary. When an IMAGE dataset containing a compound item is accessed RELATE creates a field definition for each field. These expanded fields count against the 126 fields per file limit in RELATE.

The print lengths of fields are assigned the following default values which may be changed with the MODIFY FIELD command:

TYPE	PRINT LENGTH
A	Two times the number of words in the field.
I	6
D	10
R	8.2
L	16.2
P	Four times the number of words in the field. Two decimal positions are assumed.
Z	Two times the number of words in the field plus 1. Two decimal positions are assumed unless the field is only 1 word long.

After the field structure is created, the index information is generated. Each search item in a set represents an index through which the data may be accessed. When a master set is accessed, the only index available is the search item. When a detail set is opened, all paths into the set through master sets can be used as indexes. RELATE will make use of a search item if such use will speed access. An index is also created to reference the information by record number.

In the RELATE Command Interpreter, many commands allow a range of key values to be specified. For B-tree type indexes this causes records to be returned in order by key. In a hashed indexing system as used by IMAGE, records cannot be returned in order. This changes the meaning and operation of the range parameter.

The range parameter should only contain single key values as exists in a current index. That is, the range normally indicates that directed reads should be done by the system. Any other sequence of values will cause a sequential search of the dataset.

Image Security

IMAGE maintains security information on each database. This security is based on a password that the user must supply when the database is opened. The password determines to what fields and datasets the user can read and write. RELATE operates within the security set up by IMAGE. Views may be created using IMAGE datasets. This allows a DBA to place record level security on an IMAGE database. This has no effect on QUERY or the IMAGE interface mechanisms.

EXAMPLE:

```
    )OPEN DATABASE INV;TYPE=IMAGE;PASSWORD=STOCKBOY
    )OPEN FILE INVENTORY;DATABASE=INV
```

The first command opens the database and creates the IMAGE control block. If an access mode is not specified, a mode of 3 (exclusive read/write) is assumed. The second OPEN makes the information in the INVENTORY dataset available.

ACCESS TO KSAM FILES

KSAM is a file access method supported by HP that is similar to the access methods for RELATE data. RELATE places key information and data in the same file. In KSAM a second file exists which contains only the key information. The table below compares and contrasts RELATE files with KSAM files.

RELATE	KSAM
Multiple keys are allowed.	Multiple keys are allowed.
Indexes can be created or purged after a file has been created.	All indexes must be defined when the file is created.
Duplicate key values can be allowed or disallowed.	Duplicate key values can be allowed or disallowed.
Procedural access can be accomplished through RDB calls.	Procedural access can be accomplished through file system intrinsics.
Up to 30 indexes may be defined per file.	Up to 16 indexes can be defined per file.
Generic, approximate, and partial keys can be used in procedural access.	Generic and approximate keys can be used in procedural access.
Indexes may be ascending or descending.	Indexes must be ascending.
Multiple fields of various types can be used to compose keys, and they need not be contiguous in a record.	A key must contain a single data type and must be contiguous within a record.
Does not support odd byte length keys or variable length records.	Supports variable length records and keys with an odd number of bytes.
The index structure uses a B-Tree.	The index structure uses a B-Tree.

Access to KSAM files through the RELATE Command Interpreter or the Host Language Interface routines is almost identical to RELATE files except that KSAM files do not contain structure information. This information must be given in the OPEN command for the file.

RELATE will allow several paths to be placed on the same KSAM file. However, multiple positions can not be maintained correctly if data in the file is changed. This is also true of a scan through an index in which the key values are changed. This situation cannot be corrected or prevented by RELATE due to an inability to obtain sufficient status information from MPE.

Creating a KSAM File

RELATE/3000, the utility program KSAMUTIL, or the file system intrinsic FOPEN can be used to create KSAM files. If the file is not created from RELATE the following rules must be followed in order to access the file from RELATE:

1) The file must contain fixed length records.

2) The record size must be an even number of bytes.

3) All fields must begin on a word boundary. This requirement forces alphabetic and decimal fields to contain an even number of characters.

4) Record numbering must start at zero (the default).

KSAM imposes several other constraints on the index structure. Indexes must:

1) Consist of contiguous fields.

2) Not start at the same field.

3) If of more than one field, be composed solely of A or U fields.

Additional information is in the KSAM/3000 Reference Manual (HP part number 30000-90079).

Accessing a KSAM File

To access a KSAM file, the structure must be included in the OPEN command or a file of any type must previously have been opened. This file can then be used as a template for the structure of the KSAM file. The structure file may be of any type. When the KSAM file is opened, the following operations occur:

1) The file is checked to ensure that it contains fixed length records.

2) The width of the structure is compared to the record width of the file.

3) The KSAM key information is obtained from the operating sytem and compared to the structure. RELATE ensures that:

    a) Each key begins on a field boundary.

    b) The key contains exactly one or more fields and that it does not end in the middle of a field.

    c) If the key contains multiple fields, the data type can support this situation. Alphabetic and logical fields can be composed in this manner.

If the structure meets all of these conditions, the file is loaded and the index structure is created. The index arrangement is not obtained from the structure file because it is contained in the KSAM file.

EXAMPLE:

```
)OPEN FILE FILE1
)OPEN FILE KFILE; TYPE=KSAM; STRUCTURE=FILE1
```

In this example, FILE1 is a RELATE file and is used to provide the format for the KFILE. After the KFILE is opened, it is initially accessed in record number sequence. If access is desired in the primary key sequence, the command SET INDEX 1 can be executed. RELATE allows all 16 KSAM keys to be accessed and will make full use of them when responding to a query.

EXAMPLE:

```
)OPEN FILE KCUST; TYPE=KSAM; FIELDS=(CUST, I, 4), (AMOUNT, R, 7.2)
```

In this example, the KCUST file is opened and the structure of the file is supplied in the command line.

KSAM File Security

The KSAM access mechanisms provide no security above that available from MPE. It is possible to use RELATE security on KSAM files in the same way as RELATE files.

ACCESS TO MPE FILES

RELATE will access MPE files containing fixed length records composed of fields containing an even number of bytes. Access is supported for files on direct access devices only. Access to MPE files is primarily serial in nature since indexes do not exist.

Creating an MPE File

MPE files can be created by virtually any subsystem (FCOPY, EDITOR, etc.) or language. It is also possible to create an MPE file directly from the MPE command language or the RELATE Command Interpreter.

EXAMPLE:

```
)OPEN FILE MASTER
)CREATE FILE MPEFILE;&
&)STRUCTURE=MASTER; TYPE=MPE; RECORDS=20000
```

In this example, the MASTER file is a RELATE file. The MPEFILE is created as an MPE file that is compatible with the structure of the MASTER file and can contain up to 20000 records. No information about the structure is maintained in the MPE file. This makes it possible to access the MPEFILE with a structure that contains the same record width as the MASTER file but not the same format.

EXAMPLE:

```
)OPEN FILE MASTER
)COPY TO MPEFILE; TYPE=MPE FOR TERRITORY=5
```

In this example, an MPE file is created as the result of a query. All records from the MASTER file that contain territory 5 will be placed into the file. In this example, the MPEFILE is created from the format of the MASTER file (the STRUCTURE keyword could have been used to specify a different file).

Accessing an MPE File

To access an MPE file, the structure must be included in the OPEN command or a file of any type must previously have been opened and specified as the structure. This file can then be used as a template for the structure of the MPE file.

MPE files may only be accessed by record numbers. No method presently exists to specify the sort sequence of the file. The index structure of the template file is ignored.

EXAMPLE:

```
      )OPEN FILE FILE1
      )OPEN FILE MFILE; TYPE=MPE; STRUCTURE=FILE1
```

In this example, FILE1 is a RELATE file and is used to provide the structure for the MFILE.

MPE File Security

The MPE file system provides lockword protection on MPE files. This protection mechanism is supported by RELATE. It is also possible to use RELATE security on MPE files in the same way as RELATE files. If the security mode in a group is privileged, MPE files will be created as privileged files.

FUNCTIONAL RESTRICTIONS

Constraints imposed by the underlying file systems used by RELATE limit the operations available by file type. This section summarizes what operations can and cannot be performed on or with the file types supported by RELATE.

It is assumed that the user of the file is also the creator of the file and thus can normally perform all functions on it. If the user is not the creator, security restrictions imposed by the DBA can further limit the allowable operations on a function or record basis.

File Creation

IMAGE/3000 databases cannot be created by RELATE/3000.

Adding Records

Records can normally be added to all file types until a limit (specified when the file was created) is reached. No space is wasted in the main file for overflow areas, etc. Other problems may prevent this limit from being reached:

1) In RELATE and KSAM files, the index file may fill because its size was incorrectly specified or because of poor packing of the keys (the second problem is more prevalent within KSAM). In addition, records may not be added in violation of a unary index.

2) In IMAGE master datasets, key values may not be duplicated. In IMAGE detail sets, records must exist in all related master sets (except automatic masters) that contain the supplied key values. Records cannot be added to automatic master datasets.

Changing Values

The values of all fields can be changed in MPE files.

All values can be changed in RELATE and KSAM files unless the new value causes a duplicate key to be created on a unary index. In rare cases, the index file may fill during a change operation. RELATE will recover from this situation and prevent the record from being changed.

Search or sort items may not be changed in either IMAGE master datasets or IMAGE detail sets. Values cannot be changed by the user in automatic master datasets.

Deleting Records

Records may not be deleted from MPE files.

Records may always be deleted from RELATE or KSAM files.

Records may always be deleted from an IMAGE detail data set. Records can only be deleted from an IMAGE master set if no detail set entries are associated with it. Records cannot be deleted from automatic masters by the user.

SECTION 5

T R A N S A C T I O N   P R O C E S S I N G

TRANSACTION PROCESSING

Introduction

RELATE/3000 contains a transaction processing facility to ease the problems of updating files based on complex user interaction and programmatic applications. RELATE also contains a logging facility which can be used to recover from data base corruption caused by hardware or operating system failures. These facilities are a feature of RELATE II.

Transaction Processing

A transaction is a modification to a database ("transformation of state") which has the following properties:

1) If the database was accurate before the transaction, it must be accurate afterwards (a "legal" operation). A sample of a legal transformation is a General Ledger entry where several records must be added or modified to maintain the correct balance.

2) Either all actions must happen all at once or none must happen (an "atomic" operation). That is, no other users can view or adjust any of the information taking part in the transaction while the transaction is taking place.

3) Once the transaction is complete, the modifications have been posted to the database (a "durable" operation).

A transaction has nothing to do with screen processing (although a screen processing system can generate transactions).

A complete transaction processing facility is composed of four separate functional modules defined as follows:

1) A mechanism to hold changes made to files in some form of pre-log file until a transaction completes. This phase isolates the data base from application program aborts and allows the transaction to appear to be atomic.

2) A mechanism that takes the changes saved in the pre-log file and applies them to the data base. This module is often referred to as a DO/REDO module.

3) A module to UNDO transactions that cannot be completed. This module backs out changes that have been made by the DO/REDO module. A transaction could be aborted by the data base management system because of resource problems, by an invalid action (such as a unary key violation), or by an application program request.

4) A logging mechanism to save the changes made by the DO/REDO module. The logs can subsequently be used to bring the system up to date after a failure.

Controlling Transactions

The actions performed by the transaction processing module of RELATE are controlled with the BEGIN TRANSACTION, ABORT TRANSACTION, and COMMIT TRANSACTION commands. The resources required to execute the transaction are controlled by the LOCK and UNLOCK commands.

To start a transaction a BEGIN TRANSACTION command is issued. After the command is given all changes made to files through RELATE commands or the Host Language Interface routines are saved in a pre-log file for subsequent posting. When a COMMIT TRANSACTION command is issued the information in the pre-log file is posted to the data base. If an ABORT TRANSACTION command is issued the pending changes are ignored.

If a BEGIN TRANSACTION has been issued and a system failure occurs the data base will be in a consistent state because no changes have actually been made. If, however, a failure occurs after the COMMIT TRANSACTION and before RELATE requests a new command (or control is returned to the user's application) the data base may be in an inconsistent state.

This window of vulnerability is generally very small compared to the time required to generate the complete transaction.

The pre-log file is a temporary file called RDBTPLOG created by RELATE with a default size based on the number of records in the largest file open at the beginning of the transaction. In general, proper use of transaction processing should not use up the space in the log file.

The system allows transactions to be nested. This enables the user to "seal off" logically complete portions of a transaction before the entire transaction has been completed. This nesting is accomplished by executing several BEGIN TRANSACTION commands. Each command increases the nesting level by one and must be followed by a like number of COMMIT TRANSACTION commands. For example:

```
A                          LOCK
--------------------------BEGIN
| 1
|     B____________BEGIN
|     | 2
|     |
|     |____________COMMIT
| 3
|     C____________BEGIN
|     | 4
|     |____________COMMIT
|
| 5________________COMMIT
                   UNLOCK
```

This example is composed of the transaction "A" (at level 1) which is composed of transactions "B" and "C", both of which are at level 2. The numbers indicate places at which file modifications could be performed.

Transaction Posting

Once a transaction has been COMMITted at level 1 RELATE will post the changes to the appropriate files. The posting operation may not proceed in the order in which the transaction was generated. Because of this, a transaction should not operate on the same record twice. RELATE will guarantee that all records will be added to IMAGE master sets before records are added to any detail set.

Certain special classes of operations are recognized by the system and function as follows:

1) An attempt to delete a record more than once is ignored.

2) An attempt to add a record in violation of a unary index is ignored.

3) An attempt to update the same record more than once will generate an error unless the resulting records are identical.

Multi-User Access

During multi-user (shared) update access to files which have not been locked (IMAGE, KSAM, and MPE as well as RELATE), record checksums are created as each record is read. A subsequent request to update, delete, or locate the record (through an RDBREPOINT) will verify prior to the completion of the operation that the record has not been modified by some other user. If the record has been modified, the operation will not complete and an error will result.

A maximum of 50 record checksums are maintained at any one time due to memory limitations. An attempt to update a record which does not have a checksum calculated will not produce an error. If records are updated through a transaction, the 50 record limit does not apply and all records are marked by checksums.

Locking Modes

For RELATE to update a shared file (particularly RELATE files) a lock must be obtained. RELATE internally deals with two types of locks: an update lock and a read lock. Update locks prevent concurrent reading of a file by another user. Read locks prevent another user from updating a file as it is being read. For non-RELATE files, read locks are ignored, for performance reasons.

Locks can be applied by the user or an application with the LOCK FILE command. This generates an update lock. Locks are automatically generated when the Host Language Interface routines are used. RDBADD, RDBDELETE and RDBUPDATE generate update locks. RDBREAD, RDBPOINT and RDBREPOINT generate read locks. These locks are only held for the duration of the call.

The Command Interpreter generates a long-term update lock (as if LOCK FILE had been executed) for all files in any cursor in which records will be added, changed or deleted. Due to locking restrictions imposed by MPE, this may result in an error and prevent a command from executing. Update locks are obtained and maintained for the duration of a command for performance reasons.

Locking Restrictions

Under the MPE operating system each file is considered to be a separate resource for the purpose of locking. The operating system restricts each user to one resource lock at a time. This means that only a single file can be locked during a transaction. IMAGE is a special case and allows several sets to be locked if the request is made by a single call to the IMAGE system. These restrictions impose constraints on the types of transactions that can be correctly processed by RELATE. Essentially, only one MPE, KSAM or RELATE file can be locked at any one time. Multiple IMAGE sets in the same database are considered to be one lock. KSAM index files are managed automatically and do not need to be counted. If a file is opened exclusively or semi-exclusively it is assumed to be locked but does not count against the total.

Logging

Transaction Logging is a mechanism which allows data recovery from system failures. This mechanism is built into RELATE II and is controlled by the Data Base Administrator. The facility operates by saving information about changes to both the contents of the data base and the structure of the data base. When a system failure occurs and it is suspected that the data base has been corrupted the Data Base Administrator can correct the problem by restoring a copy of the data base which is known to be consistent and apply the subsequent log files to it.

Implementation

The RELATE logging facility has been implemented using the IPC (Interprocess Communications) capability of the MPE file system. A brief description of the necessary operations within MPE to enable logging follows. For detailed information on IPC the MPE Intrinsics Reference manual should be consulted.

RELATE allows a data log and an event log to be maintained. The data log is required to allow recovery. The event log contains information about actions made by the user or on behalf of the user by an application program or the RELATE Command Interpreter. The event log can be used to analyze response times and access patterns in order to improve performance. The data log is updated as file changes are made. If the change is made within a command the entry is bracketed with begin and end transaction markers. This enables the recovery operation to suppress data changes due to the incomplete execution of a command. To achieve this capability in a program using the record level Host Language Interface routines the BEGIN TRANSACTION and COMMIT TRANSACTION commands should be used. If transactions are not used from the HLI, each RDB call which adjusts the database is considered to be a transaction. The event log is written as each action occurs to the data base.

Enabling MPE Logging

RELATE logs to standard message files. A message file can be created with an MPE build command similar to the following:

```
BUILD LOG;MSG;REC=,20
```

RELATE requires that the file have a 128 word record size but the blocking factor can be varied for each log file. A small blocking factor provides a better recovery potential since less information would be lost because it has not yet been deposited on the disc. A larger blocking factor will decrease logging overhead (and increase application performance).

It is recommended that log files be placed into a separate group within each account (or possibly a separate account). The group security should allow write access to all users in the account. Read access should be disallowed to prevent accidental reading (and loss) of the log data. When a failure occurs which requires recovery the Data Base Administrator should disable write access and enable read access. This not only allows RELATE to read the logs for recovery but will prohibit users from accessing files which must be recovered (since the log files cannot be opened). Once the recovery is complete, the original security should be put back.

Care should be taken to insure that log files are of sufficient size to prevent the end of file from being reached. If a log file fills, subsequent RELATE actions which require log records to be written will fail. If this event occurs in the middle of a transaction, the data base will be left in an inconsistent state. It will then be necessary to restore a backup copy of the data base and perform the recovery procedure. Reaching the end of a log file is similar in effect to a system failure and thus should be avoided. Consequently, log files should be built with a total capacity far exceeding their required size and consisting of many extents (up to 32) of which only enough to satisfy the expected capacity are initially allocated.

Each Data Base Administrator should determine a log maintenance cycle for the data base. For example, suppose the data base is maintained on a daily cycle. This means that at the beginning of each day, a log file is created by the Data Base Administrator with the MPE BUILD command. At the end of the day, the Data Base Administrator stores a copy of the data base to tape and purges the log file. The determination of the duration of this maintenance cycle depends on at least two considerations: the amount of time needed to store the data base periodically, and the amount of time required to recover the data base from the log file if the system fails. The more often the data base is stored, the smaller the log file (and hence, the shorter the recovery time) will be.

Enabling RELATE Logging

In order to enable RELATE logging the dictionary (RDBDD) for the account must have previously been created (with the CREATE DICTIONARY command), and the log file created with the MPE BUILD command. After this, the Data Base Administrator can enable data logging with the ENABLE DATA LOGGING command or event logging with the ENABLE EVENT LOGGING command.

Data logging is controlled on a group by group basis. That is, all the changes made to RELATE files that reside in a single group are logged to a single log. When data logging is enabled RELATE will save sufficient information to allow files to be recovered to a consistent state after a system failure. Changes to open temporary files (RETENTION=NONE) or session temporary files (RETENTION=TEMP) are not logged. The format of the records in both logs is given in Appendix D.

Event logging is controlled on a user basis. Each user can log to a different file and different events can be logged for each user. Normally, however, all users will log to the same event file which in many cases is also the data log. The event log provides information to the Data Base Administrator and is not used by RELATE.

Data Logging Considerations

The RELATE logging capability logs structural changes as well as data changes. This allows for the recovery of not only the contents of files but the actual organization of the files. This capability is limited somewhat by restrictions imposed by MPE.

The operating system does not allow the creation of files across account boundaries. Thus, RELATE cannot create files outside of the logon account. This prevents RELATE from correcting structurally any files in other accounts. Because of this, RELATE has been designed to work effectively within the confines of an account. RELATE will not honor security restrictions or logging requests which have been placed on files in other accounts.

When data logging is performed (with recovery capabilities in mind) the system implementor should not create procedures which directly update files in one account from files in another account. This should be done in a two step process. First, the adjustments from the source account should be copied into transaction files in the destination account. Then, a procedure should be executed in the destination account to perform the actual updates.

When a recovery operation is likely to encounter structural changes a second problem should be considered. Specifically, the creator name on the files being recovered may change. MPE will assign the current user name as the creator name to each file as it is built. Thus, any recovery procedure that creates new files may alter the security matrix. It is recommended that all files created in secured groups be created by the user (usually the account librarian or account manager) which will be performing the recovery. If only data changes are made, this problem can be ignored. Users should not purge files from MPE, instead, the file should be purged with PURGE FILE command.

Recovery System

The RELATE Recovery System may be executed in the event of a system failure, provided that a data base backup copy has been stored and all subsequent data base modifications have been logged to one or more log files. Recovery entails restoring the backup data base and the use of the RECOVER DATA command to re-apply the data base modifications from the log files.

Although the logging and recovery system is designed to successfully re-execute all transactions that completed before the system failure, there is a possibility that some transactions will not be recovered. The reasons for this occurrence are:

1) One or more records could be lost in the system buffers if the system fails before they are written to the log file.

2) A transaction may have originally failed to complete due to the system failure, and is therefore suppressed.

3) The wrong backup data base was restored. Recovery will yield erroneous results if this occurs.

If any transaction fails to be recovered, all subsequent transactions of the same process are suppressed as well.

Several groups of files can log to the same log file unambiguously. If all files that logged to the same log file are recovered simultaneously, then all backup copies must be restored prior to running the recovery system.

WARNING: In the event of a system failure, do not allow people to use the system before running the recovery system. Log records may have been lost due to the system failure. If logging is resumed without a recovery, the resulting discontinuous log file would cause invalid results in the event that the log files are subsequently needed.

Recovery Procedures

Before recovery can begin, the Data Base Administrator must restore the data base to the state at which logging was initiated using the MPE :RESTORE facility. If the data base has logged to more than one log file the earliest one must also exist.

The actual recovery is performed with the RECOVER DATA command.

To perform the recovery the user must not have any files open within RELATE. The recovery is performed by reading data from the log file, grouping the data into transactions, and possibly performing the transaction. A transaction is defined as a call to RDBADD, RDBDELETE, or RDBUPDATE, or a RELATE command which is not made between BEGIN and COMMIT TRANSACTION commands. File structural changes are considered to be a transaction. Any file manipulation performed between BEGIN and COMMIT TRANSACTION commands are treated as a single transaction by the recovery system.

In normal circumstances the system will recover any incomplete transaction which is not terminated by a system failure marker. That is, if an application aborts (or is aborted) and logging is not immediately discontinued the data changes made by the process will be made during recovery. This ensures that any subsequent transaction which operated on the same data manipulated by the transaction that aborted is consistent (to the extent that it duplicates the original situation). The assumption is therefore made that subsequently logged data will correct the problem caused by the application.

In order for RELATE to recover a file it must open it in exclusive mode. Also, if a fileset parameter is not given on the RECOVER DATA command, RELATE will attempt to recover all files in the logon account at once. This may exhaust the amount of memory available to RELATE. The problem can be resolved by recovering files in individual groups or individual files. Additionally, if the INFORMATION parameter is given, RELATE will not recover data but will provide information on which files would be recovered. Part of this information is a list of files open at the time of the system failure. If this number is small, it may be appropriate to only recover those files (since under most situations closed files cannot be damaged).

During the recovery, RELATE makes use of three files to consolidate the data from various aspects of a transaction. The first file (called RDBLOG) is used to compensate for an error in the operating system. The contents of the log file are copied into this file prior to the start of the recovery operation. If insufficient disc space exists for the copy a file equation may be used to place this file on a tape drive or serial disc. Another file is used to group together records that comprise a single log entry. The third file (called RDBSTAGE) is used to group an entire transaction. This file may fill. If this happens, RELATE will attempt to open a larger file and copy the staged data into the new file and purge the original file. If an error occurs during this process RELATE will terminate and the recovery procedure must be started over when sufficient disc space is available. The RDBSTAGE file is originally opened with a record limit of 1023. Each larger file will be opened with at least 33% more space than the existing size. A file equation can be used to alter the original size of the file.

Post-Recovery Procedures

After a recovery has been completed, the Data Base Administrator and system manager have three procedural options. Whatever option is chosen determines the recovery procedure in the event of a second system failure. Together, the Data Base Administrator and system manager or console operator should agree upon the best post-recovery procedure in order to avoid confusion at recovery time. The options are:

1) The Data Base Administrator stores a new backup data base copy, and the system manager or operator starts a new log file from the console. In the event of a subsequent system failure, the new backup data base is restored and recovery is performed from the new log file.

   This option allows for a straightforward recovery procedure but delays users from accessing the data base until the new backup copy has been generated.

2) A new backup tape is not generated. The system manager or operator resumes transaction logging to the same log file. In the event of a subsequent system failure, the original data base copy is restored and recovery is performed from the log file.

   This procedure is the same as the original recovery, but takes longer due to the additional log file records. Users can access the data immediately after the recovery without waiting for the data base to be stored.

A log file should not be restarted before the data base has been recovered since some log records could have been lost in the system failure. Thus, the log file may not be consistent with the actual state of the data base. A recovery is necessary to bring the data base and log file into agreement before restarting the log process.

3) A new backup data base is not generated; the system manager or operator initiates logging to a new log file. In the event of a system failure, the old data base copy is restored and two recoveries are executed: the first against the old log file, the second against the new log file.

Until a new data base backup copy is generated, if the system manager or operator consistently starts logging to a new log file after a system failure, a total recovery preceded by n failures requires n executions of the recovery system.

SECTION 6

U P D A T I N G   V I E W S

Introduction

A view is a method of looking at a collection of data elements (records) that are organized into a file. The view may be a representation of an actual file, or it may be a "virtual" file that is actually made up of portions of several files or databases.

Views provide the ability to look at information pulled from several files or databases and manipulate or report on this data as if it were a single physical file. Through the view, the files being used can be updated and changed.

Defining a View

A view is a file, the contents of which are defined by a SELECT command. Thus, when a SELECT command is issued to RELATE, the current path becomes a view.

Views can also be saved in a file with the CREATE VIEW command and opened with the OPEN FILE command. When a view is saved in a file, only the information of how to construct the view is saved, not the records that comprise the view. Associated with every view may be:

1) One or more base files that actually contain data.
2) Restrictive conditions that eliminate records from the base files so they will not appear in the view.
3) Joining conditions that join the base files.
4) A target list that defines the fields in the view.
5) A BY clause that defines the sort order of the view.
6) A UNIQUE clause that causes duplicate records to be eliminated from the view.

For example, the following are view definitions.

```
)SELECT EMP.NAME, EMP.SALARY WHERE EMP.STATE="CA"
```

This view has two fields and one restrictive condition. There is only one base file in this view: EMP. This view contains the names and salaries of all employees in the EMP file that live in California.

```
)SELECT DEPT.DNAME, EMP.NAME WHERE DEPT.D_NO=MEMBER.D_NO &
&) AND MEMBER.E_NO=EMP.E_NO
```

There are three base files in this view: EMP, DEPT, and MEMBER. This view contains department names and the names of the employees in the departments. The names are taken from the EMP file, the departments are taken from the DEPT file, and the relationship (which indicates which employees are in which departments) is found in the MEMBER file.

Using Views

There are eight basic operations that may be performed on a view. These operations are the same ones that may be performed on physical files. They are:

- CREATE a view.
- OPEN a view.
- READ a record from a view.
- CHANGE a record in a view.
- ADD a record to a view.
- DELETE a record from a view.
- CLOSE a view.
- PURGE a view.

Opening a View

All of the RELATE commands that manipulate files may also be used to manipulate views. The result will be to modify the base relations in the view in a way such that, after the command, the view will be in the same state it would be if the same operations were performed on a normal file. Since views can potentially be very complicated SELECT commands there is not always a unique way to modify the base relations to cause the correct change to occur to the view. For this reason, not all operations can legally be applied to every view. Later in the section we will discuss the requirements that a view must adhere to in order to be updatable and exactly how the base files are modified. Usually it will be obvious which operations can and cannot be performed. Consider the view:

```
    )SELECT AVG=$AVG(EMP.SALARY)
```

By examining what this view contains it becomes obvious that it cannot be updated. This view selects the average salary of all employees. If the average salary is changed it is impossible to determine which employee salaries to adjust to effect this change. This view can not be changed, only read.

When a BY clause appears on a selection, the view may be used as if an index existed containing the fields in the clause. This allows a range on RELATE commands and allows the index to be used for various purposes from the Host Language Interface routines.

Reading from a View

All views can be read unless restricted by security requirements. Any security restrictions that may apply to the base files do not apply to the view. This arrangement allows the DBA to create views for users which contain information from files normally not accessable to the user. Security restrictions can be applied to the view itself if the view has been saved in a file with the CREATE VIEW command.

When a view is read, the base files are read to compute the data for the view. It cannot be over-emphasized that a view does not contain any data; a view only contains directions to obtain data. Therefore, if a base file is changed the changes are instantly reflected in the view. A view cannot become out of syncronization with the base files that define the view.

In RELATE/3000 views are implemented so that records are returned to the user (or his application) as soon as possible. This means that there is usually very little time delay between when a PRINT command is executed and when data starts appearing on the terminal. This is possible because the entire view is not constructed before the first record is returned. As the records that comprise the view are computed they are immediately made available. There are two exceptions to this rule. If a sort condition is applied to the view and the sort condition cannot be satisfied by using a sorted index on one of the base relations, the entire view must be computed, and sorted, before a record can be returned. This will result in a delay between the PRINT command and when the first record is returned. Second, if several base relations are joined in a view and the joining fields are not indexed, RELATE may decide to create a temporary index on the base file to prevent a combinatorial explosion of records to be searched. Creating an index is a task that must be performed before the first record can be returned. This will also cause a delay between the PRINT command and when the first record is returned. In most cases, however, views are computed quickly and the results are promptly returned.

Join Conditions

Often when constructing a view, data is needed from fields in more than one file. These files are normally connected with join conditions. For example, in the following database,

```
EMP(E_NO, NAME)
DEPT(D_NO, DNAME, MGR_NO)
```

A view may be required that contains the department name from the DEPT file and the manager name found in the EMP file. The manager name is found in the EMP file by using the manager number from the department file to look through the employee numbers in the EMP file. Thus, the view would be constructed as:

```
)SELECT DEPT.DNAME, EMP.NAME WHERE EMP.E_NO=DEPT.MGR_NO
```

The condition in this case is called a join condition. The join condition does not eliminate any records from the view, but is used to join two files together. In most cases requiring information from two files a join condition is necessary.

Join conditions always have the form "<field1>=<field2>" where <field1> and <field2> are fields from different base files. Files may be joined on fields of different data types. For instance, in the above example the E_NO field from the EMP file could have been an integer field and the MGR_NO field in the DEPT file could have been a double integer field. RELATE will automatically perform the required data conversions.

When adding conditions to a view they are normally ANDed together. For example, the database:

```
PARTS(P_NO, PNAME)
SUPPLIER(S_NO, SNAME)
SHIPMENT(P_NO, S_NO, QTY)
```

defines parts, suppliers, and the quantity of each part shipped by each supplier. A view that defines part names, supplier names and quantity supplied requires data from three files. Since three files are used, two join conditions are necessary. The parts file must be joined to the shipment file and the supplier file must be joined to the shipment file. The view would be defined as:

```
)SELECT PARTS.PNAME, SUPPLIER.SNAME, SHIPMENT.QTY &
&)  WHERE PARTS.P_NO=SHIPMENT.P_NO &
&)  AND SUPPLIER.S_NO=SHIPMENT.S_NO
```

Notice that the two join conditions are ANDed together. Other conditions that might be applied to the view would also be ANDed. For example, if we wanted to look at only those shipments with a quantity greater than 5000, we would define the view:

```
)SELECT PARTS.PNAME, SUPPLIER.SNAME, SHIPMENT.QTY &
&)  WHERE PARTS.P_NO=SHIPMENT.P_NO &
&)  AND SUPPLIER.S_NO=SHIPMENT.S_NO &
&)  AND SHIPMENT.QTY>5000
```

This example has two join conditions and one restrictive condition.

Update Requirements

All the RELATE commands that can modify files can also be used to modify views. When a view is modified the change is made by modifying the base relations in the view. There are certain requirements about which views can be modified and restrictions to which operations can be performed on the modifiable views.

The following requirements must be met to allow a view to be updated:

1) The view must contain only one file or all the files in the view must contain at least one unary index. (See CREATE INDEX ;UNARY).

2) The view definition does not contain a UNIQUE clause.

3) The view definition does not contain a BY clause or, if a BY clause is given, it must be satisfied by an existing sorted index on any of the base files.

Requirement 1 is neccessary for RELATE to correctly decide which of the base files to modify when several files are joined together.

Adding to a View

When a record is added to a view, a record can be added to every base file in the view definition. The data for each field in the base file is obtained from the corresponding field from the view. If a field in the base file has no corresponding field in the view then zero or blank will be used depending on the type. Any type conversion that was done to define the field in the view will be undone when a record is added to the view. For example:

```
)SELECT EMP.NAME, SALARY=$REAL(EMP.SALARY)
```

If the SALARY field from the EMP file is a packed field, when a record is added to this view the SALARY field is entered as a real number but is converted to a packed number when the record is added to the EMP file.

If a view has any restrictive conditions, records added to the view must satisfy these conditions. For example:

```
)SELECT EMP.NAME, EMP.STATE WHERE EMP.STATE="CA"
```

This restricts the view to those records of employees in California. An error will occur if an attempt is made to add a record with STATE="NY" because this record does not satisfy the restrictions of the view. Any legal modifications to a view can be undone by another modification to the view. Thus if the system allowed the above record to be added to the view it could not be deleted, because it would not be a member of the view.

If a view contains (more than one file and) a join condition and one of the fields in the condition is a field in the view, the value of the field from the view will be used for the value of both fields from each file in the join condition. For example,

```
EMP(NAME, D_NO)
DEPT(D_NO, DNAME)
```

)SELECT EMP.NAME, DEPT.D_NO, DEPT.DNAME WHERE EMP.D_NO=DEPT.D_NO

If the record (NAME="FRED", D_NO=3, DNAME="R&D") is added to this view then the record (NAME="FRED", D_NO=3) would be added to the EMP file and the record (D_NO=3, DNAME="R&D") would be added to the DEPT file. The EMP.D_NO field is not explicitly mentioned in the target list of the view, but the data is obtained though the DEPT.D_NO field and the join condition.

In the above example, if what is intended is to add Fred to department 3, it makes sense to add the record to the EMP file, but presumably department 3 already exists in the DEPT file. Since it is not desirable to add department 3 again, somehow this must be detected. This is the reason for requirement 1. Since this view contains more than one file in its view definition both files must contain a unary index. The unary index for the EMP file would be NAME since that is the field that uniquely determines records in the EMP file. The unary key in the DEPT file would be D_NO. When records are added to a view, any unary index violation on the base files is ignored. In this case, if department 3 previously existed, a unary index violation would result when the record was added. If department 3 did not exist, then it would be added. In either case, the desired result is obtained.

It is a good idea when creating views to have all unary keys in the base files as fields in the view. If this is not done, it becomes impossible to add records to the base files. One record may be added since zero will be used as the value of the missing field, but after that, unary index violations will result and additional records could not be added to the view.

Deleting from a View

A record in a view consists of one record from each base file in the view definition. When a record in a view is deleted, one or more of the records in the base files are deleted. The decision of which base files to delete records from is made by observing the unary indexes in the base files, the view definition and the command used to delete the records. When there is only one file in the view definition, no decision is necessary and the record is deleted from the base file.

If more than one file is used in the view definition then records are deleted from the base files that are restricted by <u>all</u> fields in the conditions from the delete command. A field can be used to restrict file A if:

a) the field is a field in file A, or

b) the field restricts records from a file B that is joined to file A and the join condition uses the unary index key of file B.

For example,

```
EMP(E_NO, NAME, D_NO) 132767          Unary key: E_NO
DEPT(D_NO, DNAME) 132767              Unary key: D_NO
```

```
)SELECT EMP.E_NO, EMP.NAME, DEPT.D_NO, DEPT.DNAME &
&) WHERE EMP.D_NO=DEPT.D_NO
)DELETE FOR NAME="FRED"
```

In this example, records would be deleted from the EMP base file, but not the DEPT base file. The NAME field is the only field used in a restrictive condition in the DELETE command, and the NAME field restricts the EMP file because it is a field in the EMP file. The NAME field cannot restrict records from the DEPT file so no records will be deleted from the DEPT file.

Now, consider the same view and the following command:

```
)DELETE FOR DNAME="ACCOUNTING"
```

This time records will be deleted from both the DEPT file and the EMP file. The "ACCOUNTING" department and all employees in the department will be deleted. The DNAME field is the only field used in the restrictive condition in the DELETE command, and the DNAME field restricts the DEPT file because it is a field within that file. The DEPT file is joined to the EMP file and the join condition (EMP.D_NO=DEPT.D_NO) uses the unary key of the DEPT file; so by definition (b) above, the DNAME field can restrict records in the EMP file; therefore, records will also be deleted from the EMP file.

A field is part of the restrictive condition if:

a) The field is used in the FOR clause of the command.

b) The field is part of the current index of the view.

c) The field is used in a <u>restrictive</u> condition in the view definition.

Updating in a View

The fields of a view are defined by the target list of the SELECT command that defines the view. Any of the fields of a view may be changed subject to the following rules and restrictions:

1) If a field that is part of a join condition is changed, then the change is made to the fields from the base files on both sides of the join condition.

2) If a view contains more than one base file, then changes are only made to those base files that are dependent upon <u>all</u> the fields being changed and <u>all</u> fields used to restrict records from the view in the command. If a field in the view is defined by a field from a base file that cannot be changed then that field of the view cannot be changed. If a field in the view is used in a join condition, the field cannot be changed if both fields from the join condition cannot be changed.

   This rule is similar to the rule used in deleting records from a view. A base file A is said to be dependent upon a field if:

   a) the field is a field in file A, or

b) a file B which is joined to file A is dependent on the field and the join condition uses the unary key(s) of FILE B.

Restrictions:

1) Only fields that are defined in the SELECT command target list with a simple field name, an assignment, or a data type conversion may be changed. Fields that are defined as expressions or aggregates may not be changed.

2) Fields may not be changed if the change causes the conditions of the view to be unsatisfied.

For example,

```
PARTS(P_NO, PNAME, COLOR) 132767        Unary key:P_NO
SUPPLY(S_NO, SNAME, CITY)132767         Unary key:S_NO
SHIPMENT(P_NO, S_NO, QTY)132767         Unary key:P_NO, S_NO
```

```
)OPEN FILE PARTS; PATH=P
)OPEN FILE SUPPLY; PATH=S
)OPEN FILE SHIPMENT; PATH=M
)SELECT P.@, S.@, M.@ WHERE P.P_NO=M.P_NO AND S.S_NO=M.S_NO
)LET COLOR="RED" FOR PNAME="HERRINGS"
```

In this case the PARTS file is the only base file that is dependent upon both COLOR and PNAME so only the PARTS base file is changed.

```
)LET QTY=500 FOR PNAME="DERRIGIBLES" AND SNAME="BRITAIN"
```

In this case the SHIPMENT file is dependent upon QTY and also PNAME and SNAME since the PARTS file and the SUPPLY file are both joined to the SHIPMENT file using a unary key. The command changes the appropriate QTY field in the SHIPMENT file.

```
)LET PNAME="GOLD" FOR QTY>1000 AND PNAME="LEAD"
```

In this case only the SHIPMENT file is dependent upon both the PNAME field and the QTY field. An attempt has been made to change the PARTS file in the LET command. This will cause an error because there may be other records in the SHIPMENT file that reference the P_NO for lead. If the PNAME of lead is changed to gold it would have been changed for records other than those restricted by QTY>1000.

```
)LET PNAME="GOLD" FOR PNAME="LEAD"
```

This command does not cause an error since all PNAMES of lead are to be changed to gold.

```
)LET P_NO=3 FOR P_NO=2
```

In this case the P_NO field is part of a join condition and both the PARTS file and the SHIPMENT file are dependent on P_NO so records are changed in both files.

```
)LET P_NO=3 FOR P_NO=2 AND SNAME="BRITAIN"
```

In this case only the shipment file is dependent on both P_NO and SNAME so the shipment with P_NO=2 and SNAME="BRITAIN" is changed to P_NO=3, but no change is made to the PARTS file.

SECTION 7

S E C U R I T Y

SECURITY

RELATE/3000 has several levels of security that allow complete control of data within a database.

RELATE provides security at the file (or view), field, record, and operation level. These security provisions, coupled with those features available from MPE, provide an unobtrusive yet effective means of controlling information in even the most complex environment.

When a user executes RELATE the system automatically checks for files in the PUB group of the user's account. The absence of files used by the system to verify the user's access capabilities indicates to RELATE that a non-secure environment exists. When used in this manner, RELATE operates with only the security provisions provided by MPE. These provisions include: password protection at the user, group, and account level; lockword protection at the file level; and access restrictions on files, groups, and accounts. In many installations these security provisions are sufficient. At other installations, additional security provisions may be required to protect individual files, records, and possibly fields. This section of the manual describes MPE security and the effects of MPE security on RELATE. It also describes the security features supported by RELATE and the interaction of the RELATE and IMAGE security mechanisms.

Security Systems Overview

Security on the HP3000 is provided at several levels by the MPE operating system. These security measures include:

1) A valid user name, account name, and sometimes a group name must be known in order to log-on to the computer. A user who is not logged-on cannot obtain any information from the system.

2) A password (or passwords) must be known to validate the user identification.

3) Access is generally prohibited across accounts and groups. This secures your files against other people who may have logged-on.

4) Lockwords may be required to access individual files. This secures files within a group from users authorized to log-on to the group but who may not be authorized to access all of the information in the group.

If the IMAGE data base management system is used, the user must also know a valid password to access the database. This password determines what fields and datasets the user may read and update.

When RELATE is used in an unsecured environment only the MPE security (and possibly IMAGE security) mechanisms come into play. When a secured environment is created by an account librarian (AL), RELATE security is added to whatever restrictions are imposed by MPE and IMAGE. In a secure environment, the account librarian becomes the database administrator (DBA). It is the DBA's responsibility to control access to the information in the account. Seven commands for controlling a secure environment are available to the DBA. The commands work in pairs to grant or withdraw various forms of authorization:

ENABLE/DISABLE	These commands control the RELATE secure/unsecure modes of operation. A secure environment can be created in a single group, many groups, or all groups of an account.
ALLOW/DISALLOW	These commands are used to restrict the types of functions that particular users can perform in particular groups. The functions include the ability to use the RELATE Command Interpreter, batch access, and the creation of new files.
PERMIT/DENY	These commands are used to restrict the types of operations that particular users can perform on particular files. The operations that can be performed on a file include adding, deleting, and changing records.
CREATE DICTIONARY	This command creates the RELATE security dictionary, RDBDD. Security can't take effect until this file exists.

MPE Security Overview

The security provisions provided by MPE are the least useful yet most effective built into the system. These restrictions are enforced at the file level and essentially either permit or deny access to a file. Group and account restrictions may additionally inhibit access to a file. RELATE operates within the MPE environment and does not circumvent any of its security provisions. In many instances, it may be necessary to eliminate (through MPE commands) some of these MPE restrictions in order to give RELATE better control over a database.

The security provisions for files can be likened to a jigsaw puzzle. The puzzle is treated as a single item by MPE to the extent that a user has access to all of it or none of it. The security provisions within RELATE attempt to break the puzzle up into smaller pieces (not necessarily individual parts). Another distinction is that RELATE security can vary the information that a user has access to after a file is opened; MPE security only affects <u>which</u> files can be accessed. As the puzzle is divided, better control is obtained for each portion. Notice that the division also makes things more complex and that at some point the overhead of putting the puzzle back together again becomes a great burden. This is the case with any system that allows security to be placed on items of various sizes (granularities). Indeed, when security is applied at various levels and to individual items it is nearly impossible to see the entire picture.

This section attempts to proceed from the largest granule of security (the account level) to the smallest (restrictions on files, records, and fields). The section will also differentiate between security provided by MPE, RELATE/3000, and IMAGE.

MPE Security - Logon

The first level of security that must be passed occurs in the logon procedure. At this point, the user must know a valid USER, ACCOUNT, and possibly a GROUP name as well as any passwords assigned to these names. If a user cannot logon, none of the additional security restrictions come into play. Once a user has logged-on, additional restrictions are imposed first by MPE and then possibly by IMAGE (if an IMAGE database is in use) and RELATE. These restrictions are determined primarily by attributes assigned to the user and his USER name by the DBA.

MPE Security - Account Level

When a new account is created by a user with system manager (SM) capability, the group PUB and a user with account manager (AM) capability are created. If security provisions (ACCESS provisions) are not provided when the account is created, the default provisions of (R, A, W, L, X:ACC) are used. These provisions allow an account member (AC) to read, access, write, lock, and execute any file within the account if appropriate group and file restrictions also allow these operations. More importantly, these provisions prevent any other user on the system from accessing files within the account (unless the file is RELEASED; see MPE SECURITY - FILES). The security at this level prevents most interactions between users in separate accounts.

MPE Security - Group Level

In order for group level security to come into play, the user must first pass through the account level security. If the account level security provisions deny access to the file, the request is immediately terminated (unless the file is RELEASEd; see MPE SECURITY - FILE LEVEL).

Many groups can be created within an account. The first group of an account (PUB) is created when the account is created. It is normally considered the library group for the account and is assigned provisions such that any person who logs onto the account can read or execute files from the PUB group. Additionally, the account librarian (AL) and group users (GU—any user who has PUB as his home group or is logged on to the PUB group) can write, append to, lock, or save files in the group. RELATE security mechanisms use the PUB group for data storage.

When additional groups are created in the account, only the group user (GU) is given the ability to read, append to, write, lock, execute, or save files within the group. These restrictions prevent any other user (except the account manager) from performing any functions on any of the files in the group.

MPE Security - File Level

In order for the file level security to come into play the user must first pass through the account and group levels of security. If any of the account or group security provisions prevent access to the file, the request is immediately terminated unless the file is RELEASEd.

When a file is created, any user who can get through the account and group levels of security restrictions can read, append to, write, lock, or execute the file. Normally, only the group user (GU) and account manager (AM) can perform these functions.

It is possible to temporarily inhibit all of the security provisions on a file. This can be accomplished with the MPE RELEASE command. After a file has been RELEASEd, any user on the system can perform any operation on the file. Obviously, files should not be RELEASEd if security is a consideration. After a file has been RELEASEd the normal security provisions of the file can be reimposed by issuing the SECURE command for the file.

Another method of altering security provisions on a file in a less drastic way is the ALTSEC command. When a file is created, default security provisions are used. These provisions allow a group user (GU) to read, write, lock, execute, and append to the file. If the file is in the PUB group, any user can read or execute the file but only the account librarian (AL) can write, lock, or append to the file.

MPE Security - Summary

In general, MPE security provisions operate from an outside-in manner; that is, the ACCOUNT security provisions prevent significant access across accounts and the GROUP security provisions prevent significant access across groups. Each level of security eliminates certain operations and users until finally, at the file level, the particular functions available to a particular user are known.

MPE security provisions are an effective method of preventing access but fall far short of an effective method of allowing access. For example, if a user develops a program in a group within the DEV account and wishes to transfer that program to a group in the ONLINE account, several alternatives are available:

1) The program can be RELEASEd and copied from the DEV account while in the ONLINE account. This method poses the least risk if the user then remembers to SECURE the original file. If the file is not SECUREd, any user on the system can get a copy of the program (or even purge it).

2) A user who has system manager (SM) capability can copy the file into PUB.SYS (or a group and an account with similar security provisions). This is possible because a system manager can read any file on the system. The developer can then copy the file into the ONLINE account and the system manager can then purge the file in PUB.SYS. This method is more complicated than the first and suffers from the same problem (except that another user could not alter the program, only copy it).

3) Alter the security provisions of the DEV account and the group in which the program resides so that direct access to the program is possible from the ONLINE account. This method is at least as complex as the second method and not only allows the desired access but also allows anyone else on the system to access any file within the group.

Clearly, each of these methods compromises the security of one or more files. What is needed is a method of granting only the user in the ONLINE account read access to only the single file in the DEV account. MPE does not allow this mode of data sharing.

RELATE allows this kind of data sharing on the group level (MPE restrictions make it impossible at the account level). RELATE allows the DBA (in a secured environment) to share individual files with individual users. In addition, the type of access (read, write, etc.) allowed to a file can be restricted on a user by user basis.

RELATE operates in two security modes. The first (and the default) uses only the security provisions within MPE to determine what type of access, if any, is allowed to any particular user. Users can create, purge, and manipulate files at will. This mode of operation is most convenient for first-time users and for any environment where data security is not of much importance.

RELATE also operates in a secured mode in which users are allowed particular functions on particular files. This mode is ENABLEd by an account librarian who then becomes the Data Base Administrator (DBA) for the account. It is the responsibility of the DBA to ALLOW the appropriate capabilities and to PERMIT the appropriate operations to each user in the account.

Security Screening

When RELATE/3000 begins operation it checks for the RDBDD file in the PUB group of the user's account. If this file exists, RELATE will operate in a secured mode and screen any requests against the appropriate capability matrices. If this file does not exist, no additional checks are made for authorization by RELATE. If a secure environment is created after RELATE has begun execution, it will not be enforced until the next execution. Likewise, if the DBA alters the status of the user's log-on group either to or from a secure environment, it will not take effect until the next execution of RELATE. RELATE maintains the information on the log-on group in memory on the assumption that most operations will be performed there.

If the user is operating in a secure environment, the RDBDD.PUB file is checked to determine if the user has the ability to perform the particular function. If information allowing the function cannot be found, the request is denied. If a record is found, the function is checked and may be denied. If the user has not been given the ability to access RELATE interactively, RELATE would terminate at this point.

For some functions, a record that references a group name other than the user's log-on group may be used. This frequently occurs when a user attempts to create or use a file in a group other than his log-on group. This method of searching allows the DBA to determine what a user can do in each individual group. For example, the DBA may allow a user to create permanent files in one group but only temporary files in others.

Operation, Record, and Field Level Security

Once a group is secured, the only user that may access files within the group is the creator of the file and the DBA (account librarian). In a secured environment, it is the responsibility of the DBA to create the database and authorize users to access it. This authorization is given by using the PERMIT command.

As each permit is issued, one or more permit type entries are placed into the RDBDD file. To revoke permission, the DENY command must be used specifying which entries to delete.

Privileged Files

Upon authorization by the system manager, files created in secured groups by an Account Librarian will be privileged files. These files will block attempts to access them through system utilities or programs.

SECTION 8

R E L A T E / 3 0 0 0   I N T E R N A L S

FORMAL FILE DESIGNATORS

RELATE/3000 uses three formal file designators for command input and printed output. These files may be redirected by the user for various purposes. The names of these files and the purpose for each is as follows:

RDBIN	This file must contain the commands which RELATE will execute. The file is usually opened as $STDINX. If a file with this name exists when RELATE is invoked interactively, the commands contained in the file will be executed before the user is prompted for information. The file is ignored if RELATE is begun from the Host Language Interface routines.
RDBOUT	RELATE outputs all messages (errors, warnings, and printed text) to this file. The file is usually opened as $STDLIST. After RDBIN and RDBOUT are opened, the system verifies that RDBIN is $STDINX and if so then determines if the two files are duplicative. If the files are not duplicative, all information read from RDBIN will be placed into RDBOUT.
RDBLIST	This file is accessed when a global "P" switch appears on a command name. The "P" indicates "printer" although this file may be equated to any device type. When the switch is recognized the file is opened. If a file cannot be opened, the system attempts to open the device class "LP". When the file is closed and the output device is spooled, the system will display the spool file number used. If the output is not spooled, the actual filename is displayed.

RELATE also accesses the following files:

RDBECAT.PUB.CRI	This file contains the error messages, prompts and other control information used by RELATE. This file is opened as soon as RELATE begins execution. If the file cannot be opened, RELATE will terminate. If RDBECAT is not in PUB.CRI, a file equation can be used to redirect RELATE to the correct location.
RDBHELP.PUB.CRI	This file contains the HELP information. The file is opened the first time the HELP command is used. If RDBHELP is not in PUB.CRI, a file equation can be used to redirect RELATE to the correct location.
RDBTPLOG	This file is used as the holding file for data prior to the completion of a transaction. It is automatically created by RELATE when the first BEGIN TRANSACTION command is executed. Its size is determined at that time. The limit is calculated to allow each record in the largest currently open file to participate in the transaction. A file equation can be used to specify its size or to force the file to be created on a specific device.
RDBSTAGE	This file is used during recovery operations. Its functions and specifications are described in the Transaction Processing section.

When the Host Language Interface (HLI) routines are used they attempt to create RELATE.PUB.CRI as a son process. If RELATE.PUB.CRI does not exist, an attempt is made to load the program referenced by the formal file designator RELATE.

A P P E N D I C E S

APPENDIX A

COMMAND FORMATS

:mpecommand

PURPOSE: Executes an MPE command from RELATE.

ABORT [ENTIRE] TRANSACTION

PURPOSE: Aborts a transaction in progress.

ADD [SEPARATOR="character"]

PURPOSE: Adds data to the current file.

GLOBALS:	I	Obtains data from the procedure file.
	L	Displays line number of added line.

NOTE: Functions on RELATE, KSAM, MPE, and IMAGE files.

ADD FIELD fieldname, type, printlength [.decimals] [;options]

PURPOSE: Adds a new field to the structure of the current file.

ALLOW functionlist [IN grouplist] [BY userlist] [;DEFAULT]

PURPOSE: Creates or adds to a capability matrix that determines what operations a user can perform in a particular group.

NOTE: This command can only be executed by the account librarian in the PUB group.

BEGIN TRANSACTION

PURPOSE: Begins a new transaction.

[range] CHANGE [fieldspecs] [FOR condition]

PURPOSE: Selectively modifies data in a file.

GLOBALS:	I	Obtains data from a procedure file.
	L	Prints line number.
	S	Suppresses current key.
LOCALS:	P	Prints value of field and does not request new value.

NOTE: Functions on RELATE, KSAM, MPE, and IMAGE files.

Field specs may be either:

```
          field
              or
          (field [;PROMPT="text"] [;DEFAULT=YES/NO])
```

CLOSE [DATABASE databasename] | [FILE filename[;DATABASE=databasename] | [PATH pathname]

PURPOSE: Closes paths, files, or databases.

NOTE: If no parameters are specified, everything referenced in the current cursor is closed. Functions on RELATE, KSAM, MPE, and IMAGE files.

CLOSE RDBLIST

PURPOSE: Terminates spooling for the file RDBLIST.

COMMIT TRANSACTION

PURPOSE: Commits the current transaction.

COMPARE fieldlist WITH filename1[;options] [MATCHES [TO] filename2[;options]]
[ERRORS [TO] filename3[;options]] [BY keylist]

PURPOSE: Compares two files by key.

LOCALS: (on items in the fieldlist)

E	Contains error message or # of error.
F	Contains file name or file # in which error occurred.
I	Contains index # in which error occurred.
R	Contains record # in which error occurred.

NOTE: Functions on RELATE, MPE, and KSAM files. Matching records go to "MATCHES" file. Unmatched records go to "ERRORS" file.

COMPILE CATALOG source INTO destination [;WARN] [;OLD=oldmaster]

PURPOSE: Allows the user to create a message catalog in his own language.

[range] CONSOLIDATE [fieldlist] TO filename[;options] [BY keylist] [FOR condition]

PURPOSE: Creates a summary of the current file.

GLOBALS: D Deletes each record used in the consolidation.

LOCALS: (on items in fieldlist)

A	Averages the field.
C	Counts number of records used.
F	Takes the first value.
G	Takes the greatest value.
L	Takes the last value.
S	Takes the smallest value.
T	Totals the field.

NOTE: Functions on RELATE, KSAM, and MPE files.

[range] COPY [assignment[,...]] TO filename[;options] [FOR condition]

PURPOSE: Copies information from the current file to the "TO" file.

GLOBALS: D Deletes each record from current file as it is copied.

NOTE: Functions on RELATE, IMAGE, KSAM, and MPE files.

CREATE DICTIONARY

PURPOSE: Creates the data dictionary used to store the security and view information used in a secure enivronment by RELATE/3000

NOTE: This command can only be executed by the account librarian in the PUB group.

```
CREATE FILE filename [,keyfilename]
  [;TYPE=RELATE|KSAM|MPE] [;STRUCTURE=pathname]
  [;RECORDS=recordcount] [;RETENTION=PERMANENT|TEMPORARY|NONE]
  [;CODE=filecode] [;PATH=pathname] [;FIELDS=fieldnamelist]
  [;INDEXES=indexlist] [;PRIVILEGED]
```

PURPOSE: Explicitly creates a new file.

GLOBALS: I Obtains field descriptions from a procedure file.

NOTE: Creates only RELATE, KSAM or MPE files.

After the message "Enter Fieldname, Type, Size", the user will enter field descriptions of the format:

fieldname, type, printlength [;FORMAT=number|"datestring"|UPPERCASE|LOWERCASE] [;INTERNAL=internal#] [;LEVEL=entrylevel] [;DOLLAR=FIXED|FLOAT|NONE] [;COMMA=YES|NO]

CREATE INDEX [number] BY fieldlist [;UNARY]

PURPOSE: Creates a new index for the current file.

LOCALS: (on items in the fieldlist)
A Ascending field.
B Descending field.

NOTE: Functions only for RELATE files.

CREATE VIEW viewname
viewcommands

PURPOSE: Creates a view and stores it permanently.

GLOBALS: I Obtains view commands from a procedure file.
D Adds view to the RDBDD file.

NOTE: Viewcommands are composed of one or more OPEN commands followed by a SELECT command.

[range] DELETE [FOR condition]

PURPOSE: Deletes records from the current file.

NOTE: Either range or condition must be specified. Functions on RELATE, KSAM and IMAGE files.

DENY READ/DELETE/ADD/CHANGE/ALL ON file [BY userlist]

PURPOSE: Allows the DBA to revoke previously allowed file operations.

NOTE: This command can only be executed by an account librarian in the PUB group.

DISABLE DATA LOGGING [IN grouplist]

PURPOSE: Instructs RELATE to discontinue logging changes to RELATE files in the indicated groups.

DISABLE EVENT LOGGING [BY userlist]

PURPOSE: Instructs RELATE to discontinue logging event data for the indicated users.

DISABLE SECURITY [IN grouplist]

PURPOSE: Reinstates a non-secure environment in the groups specified.

NOTE: This command can only be executed by the account librarian in the PUB group.

DISALLOW functions [IN grouplist] [BY userlist]

PURPOSE: Removes capabilities from a matrix that determines what operations a user can perform in a particular group.

NOTE: This command can only be executed by the account librarian in the PUB group.

ENABLE DATA LOGGING [IN grouplist] TO logfile

PURPOSE: Instructs RELATE to begin logging changes to permanent RELATE files in the indicated groups.

ENABLE EVENT LOGGING [OF events] [BY userlist] TO logfile

PURPOSE: Instructs RELATE to begin logging the events specified for the users specified.

ENABLE SECURITY [IN grouplist]

PURPOSE: Creates a secure RELATE/3000 environment in the groups specified.

NOTE: This command can only be executed by the account librarian in the PUB group.

END, EXIT or //

PURPOSE: Terminates access to RELATE/3000.

GLOBALS:

C	Prints total CPU time used in run.
T	Prints connect time used in run.

ERASE FILE filename [;DATABASE=databasename]

PURPOSE: Erases the indicated file.

EXECUTE filename [;SHOW=YES|NO|SAME]

PURPOSE: Executes RELATE/3000 commands from a file.

FIX FILE filename|fileset [;CREATOR]

PURPOSE: Converts files from the RELATE 4.4 format to the RELATE 4.5 format.

FIX FORMAT MAP=mapfile; FROM=inputfile [;EBCDIC] TO outputfile

PURPOSE: Reformats information from the inputfile to the outputfile.

HELP [commandname] [requests]
HELP ERROR errorrange

PURPOSE: Displays information concerning errors or command formats and functions.

GLOBALS: F Form-feeds the output.
P Directs output to the printer.

NOTE: Requests may be any of:

ALL, FUNCTIONS, COMMANDS, SYNTAX, PURPOSE, KEYWORDS, DESCRIPTION, or EXAMPLES.

IF [condition]...ELSE...ENDIF

PURPOSE: Allows conditional execution of commands.

IGNORE [ALL] ERROR[S] [errornumber]

PURPOSE: Allows the user to ignore errors on the next command.

[range] LABEL [modifiers] USING formatfile[;options] [FOR condition]

PURPOSE: Displays or prints output in a user-specified format.

GLOBALS: I Suppresses forms alignment request.
P Directs output to printer.

NOTE: Modifiers may be any combination of:

ACROSS=number, DOWN=number, LINES=number, REPEAT=number, WIDTH=number, SUPPRESS, FORMATTED.

[range] LET assignment [,...] [FOR condition]

PURPOSE: Makes arithmetic or alphabetic assignments.

NOTE: Functions on RELATE, MPE, KSAM, and IMAGE files.

LIST COMMANDS [rangelist] [TO filename[;RECORDS=records][;WIDTH=width]]

PURPOSE: Lists the indicated RELATE commands.

GLOBALS: P Directs output to the printer.

LIST FILE filename

PURPOSE: Lists the contents of the indicated text file.

GLOBALS: P Directs the output to the printer.

LOCK DATABASE databasename
LOCK FILE filename [;DATABASE=databasename]

PURPOSE: Informs RELATE that the indicated database or file should be locked for the next transaction.

MODIFY FIELD fieldlist; formatlist

PURPOSE: Changes existing field descriptions in a file.

GLOBALS: K Keeps changes permanently. This switch can only be used on RELATE files.

NOTE: Items in the formatlist may be any of the following:
[FORMAT=number|"datestring"|UPPERCASE|LOWERCASE]
[;INTERNAL=internal#] [;LEVEL=entrylevel] [;DOLLAR=FIXED|FLOAT|NONE]
[;COMMA=YES|NO] [;NAME=fieldname] [;SIZE=printlength]

```
MODIFY FILE filename [;CLUSTER=index] [;COMPRESS=YES|NO]
  [;CRASHPROOF=YES|NO] [;DELETE=LOGICAL|PHYSICAL] [;SCAN=blocks]
```

PURPOSE: Alters the manner in which data is handled in a particular file.

```
NOTE [any text]
```

PURPOSE: Makes a comment.

GLOBALS: D Displays the text.

```
OPEN DATABASE databasename
  ;TYPE=IMAGE [;INFORMATION]
  [;PASSWORD=password]
  ;MODE=accessmode
```

PURPOSE: Opens an IMAGE database.

```
OPEN FILE|SET filename
  [;TYPE=RELATE|MPE|KSAM|IMAGE] [;INFORMATION] [;ERASE]
  [;STRUCTURE=pathname] [;DOMAIN=PERMANENT|TEMPORARY]
  [;DATABASE=databasename] [;RETENTION=PERMANENT|TEMPORARY|NONE]
  [;PATH=pathname]
  [;FIELDS=fieldlist] [;MODE=accessmodes]
```

PURPOSE: Opens a file, set, or path.

NOTE: Functions for RELATE, IMAGE, KSAM, and MPE files.

```
OPEN RDBLIST
```

PURPOSE: Spools all output for RDBLIST until a CLOSE RDBLIST is encountered.

PAUSE ["comment"]

PURPOSE: Causes RELATE to pause until RETURN is pressed.

PERMIT READ|DELETE|ADD|CHANGE|ALL ON file [;FIELDS=fieldlist]
[BY userlist] [FOR condition]

PURPOSE: Allows the DBA to authorize individual users to perform functions on files, records, or fields.

NOTE: This command can only be executed by an account librarian in the PUB group.

[range] PRINT [fieldlist] [FOR condition]

PURPOSE: Displays information from the current file.

GLOBALS:	number	Skips a line after every number of lines.
	D	Displays filename, index #, fields, time, and page number on output.
	F	Form-feeds the output.
	L	Prints the line number.
	N	Suppresses printing of the fieldnames.
	P	Directs output to line printer.
	S	Suppresses the current key.
	T	Totals all numeric fields.

LOCALS: (on items in the fieldlist)

	number	Skips number of lines on a control break.
	B	Uses field as control break.
	F	Form-feeds on control break.
	H	Uses field or text in heading.
	S	Suppresses field if it hasn't changed.
	T	Totals the field.

NOTE: Functions on RELATE, KSAM, MPE, and IMAGE files.

PURGE FILE filename

PURPOSE: Purges the indicated file.

NOTE: Functions on RELATE, KSAM and MPE files.

PURGE INDEX number

PURPOSE: Purges an existing index from the current file.

NOTE: Only works on RELATE files.

PURGE VIEW filename

PURPOSE: Purges an existing view.

GLOBALS: D Purges the view from RDBDD.

QUIZ[;P] reportname [;SOURCE=datafile] [;PARM=parm] [;MAXDATA=maxdata]

PURPOSE: Invokes the QUIZ reportwriter.

[range] RECOVER [TO filename[;options]] [FOR condition]

PURPOSE: Recovers previously deleted data.

RECOVER DATA FROM logfile [;INFORMATION] [FILES fileset]

PURPOSE: Recovers the content and structure of the data base after a system failure.

REDO [commandnumber]

PURPOSE: Allows editing of the previous command line.

NOTE: Performs the following edit functions:

D	Deletes a character.
R	Replaces a character.
S	Splits a line into two lines.
I	Inserts a character.
.#	Edits the line indicated. .ix 'REDO','syntax'

REORGANIZE FILE filename [;RESERVE=records]

PURPOSE: Removes deleted records and rewrites the index structure of the given filename.

SELECT [targetlist [[SORT] [UNIQUE] BY keylist] [WHERE condition]]

PURPOSE: Indicates what information will be available to the following command.

SET INDEX number|fieldlist

PURPOSE: Makes the indicated index the current index.

SET PATH pathname

PURPOSE: Accesses a file that has already been opened.

SHOW [,ALL] [,BOUND] [CURRENT] [,FILES] [,FORMAT] [,INDEX] [,KEY] [,LEVEL] [,PATHS] [,RECORD] [,SELECT] [,SETS] [,STRUCTURE]

PURPOSE: Displays information about open files.

[range] SORT BY keylist TO filename[;options] [FOR condition]

PURPOSE: Sorts a datafile by a keylist.

LOCALS: (on items in the keylist)

A	Sorts in ascending order.
D	Sorts in descending order.

NOTE: Functions on RELATE, MPE, KSAM, and IMAGE files.

[range] SUM [fieldlist] [FOR condition]

PURPOSE: Obtains the sum of one or more fields.

GLOBALS: A Prints averages as well as sums.

LOCALS: (on items in the fieldlist)
A Prints average of the field as well as the sum.

SYSTEM [$COMMENT] [;$CPU] [;$DEMO] [;$LANGUAGE="language"] [;$TIME] [;$CANCEL]

PURPOSE: Assigns or displays system-wide options.

GLOBALS: S Shows current status of system parameters.

TERMINAL [;$CCTL] [;CLEAR="clear sequence"] [;$CRT] [;$LINES=#lines] [;$SPACE_B=#spaces at bottom] [;$SPACE_T=# spaces at top] [;$TYPE="terminal name"] [;$WIDTH=# characters] [;$DEMO_S="start seq"] [;$DEMO_E="end seq"]

PURPOSE: Assigns or displays the values of terminal parameters.

GLOBALS:

S	Shows current status of the user's terminal.
T	Shows the terminal types known to RELATE.

NOTE: A user with System Manager capability can also use a GLOBAL "U" and a "DEVICE=" parameter.

UNLOCK

PURPOSE: Unlocks all locked files.

UPDATE [assignment [,...]] [TO filename1[;options]] USING[:D] filename2[;options] [BY keylist]

PURPOSE:	Updates and/or copies records with duplicate keys from a secondary file.
GLOBALS:	D Deletes each record from the current file as it is updated. F Updates only the first entry in a key.
LOCALS:	D Deletes each record from the USING file as it is updated.
NOTE:	Functions on RELATE, KSAM, and IMAGE files.

APPENDIX B

COMMAND NUMBERS

When the RELATE/3000 Host Language Interface routines are used, complete commands may be passed to the system. In some applications it may be useful to know what these commands were. Location eighteen (18) of the cursor returns a numeric indication of the command. The table below can be used to obtain the actual command name.

1	CHANGE
2	
3	SELECT
4	DELETE
5	"//"
6	END
7	EXIT
8	ALLOW
9	OPEN
10	PRINT
11	SHOW
12	CREATE
13	REORGANIZE
14	RECOVER
15	SUM
16	CONSOLIDATE
17	UPDATE
18	PURGE
19	SORT
20	LET
21	COPY
22	CLOSE
23	MODIFY
24	LABEL
25	COMPARE
26	NOTE
27	SYSTEM
28	ERASE
29	ADD
30	QUIZ
31	HELP
32	TERMINAL
33	
34	
35	SET
36	EXECUTE
37	DISALLOW
38	ENABLE
39	PERMIT
40	DENY
41	REDO

42	DISABLE
43	FIX
44	
45	
46	MPE Command
47	BEGIN
48	COMMIT
49	PAUSE
50	DRAW
51	PLOT
52	LOCK
53	UNLOCK
54	ABORT
55	IGNORE
56	REPORT
57	
58	IF
59	ENDIF
60	ELSE
61	COMPILE

APPENDIX C

TERMINATION CONDITIONS

During the execution of RELATE/3000 certain serious error conditions may be encountered. These conditions are primarily caused by a lack of resources (usually memory) or an inability of MPE to perform an operation required by RELATE. These conditions will cause RELATE to terminate execution immediately.

In the case of an MPE related difficulty the problem may disappear if the request is made when the system is not as busy. In the case of a lack of memory closing unused files may enable the desired operation to be performed.

The errors that can be generated are summarized below.

99	Illegal execution of instructions in the RELATE/3000 timing system. Indicates a hardware or RELATE failure.
100	Invalid system control block address. The memory location containing the master table location has been destroyed. Indicates a RELATE failure.
101	Unable to open $STDINX. Indicates an MPE failure.
102	Invalid virtual memory management table address. Indicates a RELATE failure.
103	Unable to read from $STDINX. Indicates an MPE failure.
104	The Host Language Interface routines could not correctly locate the shared extra data segment used for communications. Indicates an MPE or RELATE failure.
105	The Host Language Interface routines could not correctly retrieve information from the communications extra data segment. Indicates an MPE or RELATE failure.
106	Illegal Host Language Interface instruction. Indicates a hardware or RELATE failure.
107	Unable to open RDBOUT (usually $STDLIST). Indicates an MPE failure.
464	MPE could not obtain more memory because the stack is frozen. This should never occur. It indicates a serious problem within MPE.

APPENDIX D

LOG RECORD FORMATS

All RELATE log file entries are prefixed with the following information:

WORDS	CONTENTS
0	Writer's ID (Enabled with FCONTROL 46)
1	Data Code (Enabled with FCONTROL 46)
2	A 1 indicates a RELATE data logging record. A 2 indicates a RELATE event logging record. Records containing values other than a 1 are ignored during recovery.
3	Sequence Number for entries which span multiple records
4	Number of data words in the entire entry
5	Data offset

The following entry types are generated from DATA logging:

BEGIN TRANSACTION

WORDS	CONTENTS
0	Code 1
1-2	Not Used
3	Command Number (If interactively executed)

COMMIT TRANSACTION

WORDS	CONTENTS
0	Code 2
1-2	Not Used
3	Command Number (If interactively executed)

RECORD ADD

WORDS	CONTENTS
0	Code 7
1-2	Not Used
3	Not Used
4	Physical File Table Number
5-6	Record Number
7	Offset to Data
	Record Added

RECORD DELETE

WORDS	CONTENTS
0	Code 8
1-2	Not Used
3	Not Used
4	Physical File Table Number
5-6	Record Number
7	Offset to Data
	Record Deleted

RECORD UPDATE

WORDS	CONTENTS
0	Code 9
1-2	Not Used
3	Not Used
4	Physical File Table Number
5-6	Record Number
7	Offset to Data
	Original Record and Updated Record

FILE ACCESS

WORDS	CONTENTS
0	Code 10
1-2	Not Used
3	Operation:
	1= Create
	2= Open
	3= Close
	4= Purge
4	Physical File Table Number
5	Words of data
6	Offset to data
	Data

INDEX CREATE/PURGE

WORDS	CONTENTS
0	Code 11
1-2	Not Used
3	Not Used
4	Physical File Table Number
5	Command Code (See Appendix C)
6	Characters in the Command
7	Offset to Command
	Command

The following entry types are generated from COMMAND event logging:

COMMAND INITIATION

WORDS	CONTENTS
0	Code 4
1-2	Not Used
3	Command Number (If interactively executed)
4	Not Used
5	Command Code (See Appendix C)
6	Characters in the Command
7	Offset to Command Start
	Command

COMMAND INFORMATION

WORDS	CONTENTS
0	Code 5
1-2	Not Used
3	Command Number (If interactively executed)
4	Information Code (None currently defined)
5	Offset to Information
	Information

COMMAND TERMINATION

WORDS	CONTENTS
0	Code 6
1-2	Not Used
3	Command Number (If interactively executed)
4-5	CPU Time (Only available if enabled by the SYSTEM command)
6-7	Wall Clock Time (Only available if enabled by the SYSTEM command)
8	RELATE Error Number
9	MPE, KSAM, or IMAGE Error Number

The following entry types are generated from STARTUP event logging:

RELATE INITIATION

CODE	CONTENTS
0	Code 15
1	User Session or Job Number (Not yet available)
2	Session or Job Input Device Number
3-7	User Name (Null Terminates)
8-12	Group Name (Null Terminates)
13-17	Account Name (Null Terminates)
18	Access Source: 0= Interactively 1= Host Language Interface #= CRI Subsystem
19	PIN of Calling Process (Not yet available)
20-24	Program Name (Not yet available)
25-29	Program Group (Not yet available)
30-34	Program Account (Not yet available)

RELATE TERMINATION

CODE	CONTENTS
0	Code 16
1	User Session or Job Number (Not yet available)
2	Session or Job Input Device Number
3	RELATE Error Number
4	MPE Error Number
5	PIN of Calling Process (Not yet available)

The following entry types are generated from ACCESS event logging:

CURSOR INITIALIZATION

0	Code 20
1	Cursor Number
2	PIN of Calling Process (Not yet available)

CURSOR RELEASE

0	Code 21
1	Cursor Number
2	PIN of Calling Process (Not yet available)

FILE OPEN

0	Code 22
1	Cursor Number
2	File Number
3-7	File Name
8-12	Group Name
13-17	Account Name
18-26	Set Name (If IMAGE)

FILE CLOSE

0	Code 23
1	Cursor Number
2	File Number

APPENDIX E

USER DEFINED FUNCTIONS

The user may define his own functions by writing a procedure called RDBFUNCTIONS and placing it in either the group, account, or system SL. This procedure will contain the definitions of any user defined functions.

The RDBFUNCTIONS procedure is never called directly by the user; it is referenced automatically by RELATE when an unidentified function is encountered. The user should use the new functions in the same manner that standard RELATE functions are used.

A RDBFUNCTIONS procedure may occur in all three SL's. To locate a function, RELATE first looks at the group SL then the account SL, followed by the system SL. If none of the procedures recognize the function, an error results.

The definition of the RDBFUNCTIONS procedure is as follows:

```
RDBFUNCTIONS(NAME,NAME'LEN,MODE,PARMS);
             IA   IV       IV   IA
```

Where:

NAME	Is the name of the function.
NAME'LEN	Is the length of the function name in bytes.
MODE	Process mode: 0 = Validate NAME, set function number 1 = Assign data types to parameters and result 2 = Evaluate function 3 = Reset function
PARMS	Data passed to and from function (depends on MODE).

PARMS ARRAY CONTENTS

OFFSET	(words marked with * are automatically filled by RELATE.)
0	Function number.
1	Maximum number of parameters allowed in function.
2	Minimum number of parameters allowed in function.
3*	Actual number of parameters found in function call.
4	Number of words required for inter-record communication.
5	Word address of inter-record communication space.
6	Data format of result of function: BITS: 0-3 Data type (see Data Type Codes in HLI) 4-7 Number of decimal places 8-15 Print length including decimals
7	Data size of result of function: BITS: 0-7 Word length of data 8-15 Reserved for RELATE
8*	Word address of data result of function.
9*	Number of words per parameter entry (currently 5).
10*	Starting offset in parm table of parameter entries (currently N=11)
N+0*	Word address of parameter data.
N+1*	Actual format of parameter (same format as word 7).
N+2*	Actual size of parameter (same format as word 8).
N+3	Desired format of parameter (same format as word 7).
N+4	Desired size of parameter (same format as word 8).

MODES

MODE 0 (Validate name):

Uses: NAME
NAME'LEN

User Sets: Function number (between 1 and 32767).
Maximium number of parameters allowed.
Minimum number of parameters allowed.

Error: If the function is not found then the function number (first word of PARMS) should be set to zero.

MODE 1 (Assign data types):

Uses: Function number.
Actual number of parameters found in function.
Number of words per parameter entry.
Starting offset in parameter table of parameter entries.
For each parameter:
  Actual data format
  Actual data size

User Sets: Number of words required for inter-record call space if needed.
Data format of result.
Data size of result.
For each parameter:
  Desired data format
  Desired data size

MODE 2 (Evaluate function):

Uses: Function number.
Actual number of parameters found in function.
Number of words per parameter entry.
Starting offset in parameter table of parameter entries.
Word address of inter-record space if used.
Data format of result.
Data size of result.
Word address of data result.
For each parameter:
  Word address of parameter data
  Desired data format
  Desired data size

User Sets: Evaluate function and place result in data result area.

MODE 3 (Reset function):

Uses:	Function number. Actual number of parameters found in function. Number of words per parameter entry. Starting offset in parameter table of parameter entries. Word address of inter-record space if used. For each parameter: Desired data format Desired data size
User Sets:	Initialize inter-record space if used.

EXAMPLES:

```
PROCEDURE RDBFUNCTIONS(NAME,NAME'LEN,MODE,PARMS);
VALUE NAME'LEN,MODE;
INTEGER ARRAY NAME,PARMS;
INTEGER NAME'LEN,MODE;
BEGIN
  BYTE    POINTER RESULT'STR,PARM'STR;
  BYTE    ARRAY   BNAME(*)=NAME;
  INTEGER         I,LEN,N;
  INTEGER POINTER IR'SPACE;
  LONG    POINTER RESULT'LONG,PARM'LONG, IR'LONG;
  EQUATE MAX'MOV'AVG=10;

  CASE MODE OF BEGIN
    BEGIN                                              <<mode 0=VALIDATE>>
      PARMS(0):=0;                                 <<null function number>>
      IF BNAME="$REVERSE" THEN BEGIN           <<reverse a passed string>>
        PARMS(0) =1;                                <<function number 1>>
        PARMS(1):=1;                               <<max # of parameters>>
        PARMS(2):=1;                               <<min # of parameters>>
        END;
      IF BNAME="$MAVG" THEN BEGIN              <<take the moving average>>
        PARMS(0):=3;                                <<function number 3>>
        PARMS(1):=2;                                    <<max # of parms>>
        PARMS(2):=1;                                    <<min # of parms>>
        END;
      END;

    BEGIN                                          <<mode 1=ASSIGN TYPES>>
                              <<NOTE: PARMS(10)=offset to 1st parameter>>
      CASE PARMS(0)-1 OF BEGIN
        BEGIN                                     <<function 1=REVERSE>>
          PARMS(6):=PARMS(PARMS(10)+1);    <<result format=parm format>>
          PARMS(7):=PARMS(PARMS(10)+2);        <<result size=parm size>>
          PARMS(PARMS(10)+3):=
           PARMS(PARMS(10)+1);     <<desired parm format=actual format>>
          PARMS(PARMS(10)+4):=
           PARMS(PARMS(10)+2);             <<desired parm size=actual>>
          END;
        ;                                     <<function 2=NOT DEFINED>>
        BEGIN                                        <<function 3=MAVG>>
          PARMS(6):=PARMS(PARMS(10)+1);    <<result format=parm format>>
          PARMS(6).(0:4):=6;                        <<result type=long>>
          PARMS(7).(0:8):=4;                     <<result size=4 words>>
          PARMS(4):=MAX'MOV'AVG*4+1;               <<inter-record area>>
          PARMS(PARMS(10)+3):=
           PARMS(PARMS(10)+1);             <<desired parm form=actual>>
          PARMS(PARMS(10)+3).(0:4):=6;         <<desired parm type=long>>
          IF PARMS(3)=2 THEN BEGIN                    <<2 parms passed>>
            PARMS(PARMS(10)+8):=%30006;         <<integer, length of 6>>
            PARMS(PARMS(10)+9):=1;                          <<one word>>
            END;
        END;
      END;
    END;

  BEGIN                                                  <<mode 2=EVALUATE>>
    CASE PARMS(0)-1 OF BEGIN
      BEGIN                                         <<function 1=REVERSE>>
        @RESULT'STR:=W'TO'B(PARMS(8));           <<results byte address>>
        @PARM'STR:=W'TO'B(PARMS(PARMS(10)+0));  <<parm byte address>>
        LEN:=PARMS(6).(8:8);                       <<result print len>>
        WHILE LEN>0 AND PARM'STR(LEN-1)=" " DO LEN:=LEN-1;
        FOR I:=0 STEP 1 UNTIL LEN DO
         RESULT'STR(I):=PARM'STR(LEN-I-1);
        FOR I:=LEN STEP 1 UNTIL PARMS(6).(8:8)-1 DO
         RESULT'STR(I):=" ";
        END;
      ;                                         <<function 2=NOT DEFINED>>
      BEGIN                                            <<function 3=MAVG>>
        IF PARMS(3)=2 THEN BEGIN                     <<if 2 parms passed>>
          @IR'SPACE:=PARMS(PARMS(10)+5);
          N:=IR'SPACE;              <<number of records to incl in avg>>
          END
        ELSE N:=5;                                <<default is 5 records>>
        @RESULT'LONG:=PARMS(8);                       <<address of result>>
        @PARM'LONG:=PARMS(PARMS(10));               <<1st parm address>>
        @IR'SPACE:=PARMS(5);                    <<inter-record address>
        MOVE IR'SPACE(MAX'MOV'AVG*4):=IR'SPACE(MAX'MOV'AVG*4-4),
         (-((MAX'MOV'AVG-1)*4));
        @IR'LONG:=@IR'SPACE(1);
        IR'LONG(0):=PARM'LONG;
        IF IR'SPACE(0)<N THEN IR'SPACE(0):=IR'SPACE(0)+1;
        RESULT'LONG:=0L0;
        FOR I:=0 STEP 1 UNTIL IR'SPACE(0)-1 DO
         RESULT'LONG:=RESULT'LONG+IR'LONG(I);
        IF IR'SPACE(0)>0 THEN
         RESULT'LONG:=RESULT'LONG/LONG(DOUBLE(IR'SPACE(0)));
        END;
      END;
    END;

  BEGIN                                                     <<mode 3=RESET>>
    CASE PARMS(0)-1 OF BEGIN
      ;                                                       <<1=REVERSE>>
      ;                                                   <<2=NOT DEFINED>>
      BEGIN                                                      <<3=MAVG>>
        @IR'SPACE:=PARMS(5);
        IR'SPACE(0):=0;
        END;
      END;
    END;
  END;
END;
```

APPENDIX F

RELATE/3000 SERVER PROCESS OPERATION

RELATE/3000 uses a server process to coordinate access to resources which must be shared by many processes on the system. At present, this is primarily the extra data segments used for buffers on RELATE files. The server process is reponsible for creating, assigning and releasing the data segments at the request of a user's RELATE process. The server process must therefore be running whenever RELATE files must be accessed.

The server process communicates with user processes through message files. These files are created by the server process in the SERVER.CRI group.

The RDBWRSRV file is used to pass file open and close messages to the server. When the server receives a file open message it searches its tables to determine if the file is currently open by any RELATE process. If the file is not open, an extra data segment is created and assigned to the file. The newly created or previously assigned extra data segment number is returned to the user's process in the RDBRDSRV file.

The server maintains a list of the open files, the PINS which have a particular file open and the identifier assigned to each PIN's access to the RDBWRSRV file. When a file close message is received by the server the access count on the file is decremented and the PIN number is removed from the list. If the access count goes to zero the extra data segment is released and the file name is removed from the server's table. If a user process aborts, all files associated with the process are closed and their extra data segments may be released.

INDEX

READER COMMENT SHEET

We welcome your evaluation of this manual and its related software product. Your comments and suggestions assist us in improving our publications and software. Please use additional pages if necessary.

1. Does this manual clearly and accurately describe all the features of its associated software?

2. Are the concepts and words in this manual easy to understand?

3. Is the format of this manual convenient in arrangement and readability?

4. Are the index and table of contents complete and useful?

5. Are the examples clear, correct, and informative?

COMMENTS:

Please mail to:

PUBLICATIONS MANAGER
COMPUTER RESOURCES INCORPORATED
5333 BETSY ROSS DRIVE
P.O. Box 58004
SANTA CLARA, CA 95052