

DQL Programmer's Guide

Chapter I : Basic Concepts

Copyright and Trademarks	10
Acknowledgements	11
Welcome to the DQL Programmer's Guide	12
How this Book Is Organized	12
The Basic DQL Vocabulary	14
Commands	14
Operators	15
Functions	16
Database Objects	17
Relationships	17
Symbols	18
Variables	18
Values	18
Comments	19
Getting Help on DQL Keywords and Concepts	20
General Steps for Creating a DQL Procedure	21
Starting a New DQL Procedure	22
Creating a DQL Script	23
Using the Script Editor to Create a Script	23
Copying from One Script to Another	23
Saving a DQL Procedure	24
Using the Check DQL Option	25
How Security Works with DQL Procedures	26
Loading a DQL Procedure	27
Executing a DQL Procedure	27
Deleting a DQL Procedure	27
MEMBER LIST: Basic DQL Procedure	28
Parts of a DQL Procedure	29
Script	29
Body of a Procedure	30
Page Header and Page Footer	31
Summary Header and Summary Footer	32
Data-Entry Form	32
Printing the Procedure Definition	34

Chapter 1 : DQL Environment

Using the DQL Script Editor	35
Viewing the DQL Script Editor	35
Parts of the DQL Script Editor	36
Moving the Cursor in the DQL Script Editor Window	37
DQL Script Menu	38
Checking a DQL Script	38
Displaying Pick Lists	39
Loading and Saving a DQL Script as an ASCII Text File	40
Clearing the DQL Script Editor	41
Setting Preferences for the DQL Script Editor	41
Script Preferences Dialog Options	41
Searching for a Text String	42
Search Dialog Options	42
Searching and Replacing a Text String	43
Moving Backward and Forward in a Search	44
DQL View Menu Options	45
DQL Toolbar	45

Chapter 2 : DQL Enhancements

Creating a DQL Data-Entry Form and Layout	46
Creating a Data-Entry Form	47
Displaying the Data-Entry Form	48
Using the Data-Entry Form	49
Using a Relationship to Specify Data-Entry Values	50
Alternatives to Using a Data-Entry Form	51
Saving the Data-Entry Form	52
Modifying a Data-Entry Form	52
Formatting DQL Output	53

Chapter 3 : List Records

Using DQL to Create a Report	55
The Purpose of the Query: A Report	56
Telling DataEase Which Records to Select	57
Creating the Script	58
The for Command	58
Specifying Selection Criteria in a Query	58
Using the list records Command	60
Sorting and Grouping Data in a Query	60
Sorting and Grouping Operators	60
Displaying Data from a Related Form	62
Relational Operators	62
Checking a Script for Errors	64
Creating a Procedure Layout	65
Saving and Running a Procedure	67

Chapter 4 : Control Procedures

Using DQL to Manage Your Application	68
Types of DQL Commands	68
Procedure 1: INPUT RESERVATIONS	70
Script for the INPUT RESERVATIONS Procedure	71
Explanation of the Script	71
Saving the Procedure	77
Procedure 2: CALCULATE DISCOUNTS	78
Script for the CALCULATE DISCOUNTS Procedure	78
Explanation of the Script	79
Procedure 3: RESERVATION INVOICES	85
Script for the RESERVATION INVOICES Procedure	86
Explanation of the Script	87
Procedure 4: PROCESS RESERVATIONS	89
Script for the PROCESS RESERVATIONS	89
Explanation of the Control Procedure	89
Summary	90

Chapter 5 : Transaction Processing

Using DQL in a Client-Server Environment	91
Using SQL Commands in a DQL Procedure	92
Using Explicit Transactions to Enhance a DQL	93
Modified INPUT RESERVATIONS Script	94
Explanation of the Modified Script	95
SQL User Permissions	98

Chapter 6 : DQL Tech Tips

DQL Tech Tips: How to	99
Organize the Commands in a Simple Script	100
Choose the Primary Table in a Multi-Table Procedure ...	103
Combine Multiple Selection Criteria using and and or ...	104
Add Group Totals and Grand Totals to a Report	105
List Individual Field Values from a Related Table	107
Use Indexes to Improve DQL Processing Speed	109
Post Totals to Another Table	113
Calculate a Percentage	114
Count the Number of Groups in a Report	116
Count the Number of Records in a Group	118
Produce Multiple Printouts of Records in a Report	120
Create an Accounts Receivable Ageing Report	122
Sort the Values in a Choice List Field	125
List the First Several Records in a Table	126
Find and List Duplicate Records	127

Chapter 7 : SQL Tech Tips

SQL Tech Tips: How to	
Control Transactions in a DQL Procedure	130
Join Tables in a DQL Script	131
Specify an Explicit Inner Join in a DQL Procedure	132
Join Tables Stored on Different Engines	133
Join Two Independent One-to-Many Relationships	134
Avoid Redundant Processing in a DQL Procedure	135
Improve Processing Speed by Creating a Views	136
Use Stored Procedures in a DQL Script	138
Replace Explicit DQL Locks with Semaphores	139

Deadlocks: How to	141
Avoid Deadlocks	142
Detect Deadlocks	143
Prevent Deadlocks	146
Improve Performance	148

Chapter 8 : DQL Lexicon

DQL Lexicon: Introduction	149
DQL Lexicon Typographical Conventions	150
Symbols	151
+ (addition)	152
- (subtraction)	152
/ (division)	153
*(multiplication)	153
*(asterisk)	154
? (question mark)	155
~ (tilde)	156
: (colon)	157
() (parentheses)	158
. (period)	159
; semicolon	160
" " (quotation marks)	161
:= (assignment operator)	162
< (less than)	163
= (same as).....	163
<= (less than or equal to)	163
> (greater than)	164
>= (greater than or equal to)	164
-- (comment)	165
abs (absolute value)	166
acos (arccosine)	167
ad hoc relationship	168
all	170
ampm	172
and	173
any	174
application status	175
asin (arcsine)	176
assign	177
atan (arctangent)	178

atan2 (arctangent 2)	179
backup db (backup database)	180
begin transaction	181
between	183
blank	184
break	185
call menu	186
call program	187
case	188
ceil	190
comment	191
commit	192
Comparison Operators	194
Conditional Statistical Operators	195
Constant Value	197
Control Procedure	198
copy all from	199
cos (cosine)	200
cosh (hyperbolic cosine)	200
count	201
count of	203
count of (example)	204
current	205
data-entry	207
date	208
day	209
db status (database status)	210
define	211
delete records	213
do	214
else	215
end	216
enter a record	218
error messages off	219
error messages on	220
exec SQL	221
exit	225
exp	226
export	227
firstc	229
firstlast	230
firstw	231

floor	232
for	233
Functions	236
futurevalue	237
global	238
Grouping	240
highest of	242
hours	243
if Command	244
if Function	246
import	249
in	250
in groups	251
in order	253
in reverse	255
input using	256
Using the input using Command with Multiforms	258
install application	260
installment	261
item (Statistical Operator)	262
item (Conditional Statistical Operator)	263
jointext	264
julian	265
lastc	266
lastfirst	267
lastw	268
length	269
list records	270
lock	272
lock db (lock database)	274
log	275
log10	275
lower	276
lowest of	277
max	278
mean	279
mean of	280
message	281
midc	285
midw	286
min	287
minutes	288

mod (modulus)	289
modify records	290
month	291
named	292
Nested Actions	294
not	295
Operators	296
or	297
others	298
output	299
percent	300
periods	301
power	302
presentvalue	303
Primary Table	304
Procedural Commands	305
Processing Procedure	306
proper	307
query selection	308
random	310
rate	311
record entry	312
Relational Statistical Operators	313
Relationships	314
reorganize	315
restore db (restore database)	316
rollback	317
run procedure	319
Secondary Table	320
seconds	321
Selection Criteria	322
sin	323
sinh	323
Sorting	324
spellcurrency	325
spelldate	326
spellmonth	327
spellnumber	328
spellweekday	329
sqrt (square root)	330
Statistical Operators	331
std.dev. (standard deviation)	332

std.err. (standard error)	333
sum	334
sum of	335
tan (tangent)	336
tanh (hyperbolic tangent)	336
temp	337
textpos	338
timeampm	339
tran off	340
tran on	341
Transaction Processing	342
Unlock	343
unlock db (unlock database)	345
upper	346
value	347
variable	348
variance	349
weekday	350
while	351
with	353
year	354
yearday	355
yearweek	356

Copyright and Trademarks

This manual supports DataEase version 6.5 and higher.

Publication Date: December 2003.

© DataEase International Ltd. All rights reserved.

DataEase is a registered trademark, and the symbol and slogan are trademarks of DataEase International. All other trademarks and registered trademarks belong to their respective holders.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means: electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Acknowledgments

DataEase 6.5 is the product of:

Peter J. Tabord

Engineering

Eugene V. Bragin

Paul Davis

Andrey Drozdov

Alexander P Dymov

Duncan Ferguson

Kevin Glossop

Alexander N. Glushakov

Andrew Y. Puzfrew

Quality Assurance

Tim Griffith

Mark Povey

Wayne Roberts

Catriona Tabord

Anne Williams

Documentation

Dave Morgan

Marketing

Katia Robertson

Paul Temple

Training

Simon Irwin

DataEase wishes to thank all our customers, especially those who served as beta testers and those who contributed suggestions, ideas, and their valuable time.

Chapter 1 : DQL Basics

Welcome to the DQL Programmer's Guide

The DataEase Query Language (DQL) extends the data processing and reporting capabilities of DataEase far beyond those of a standard form or report. In addition to letting you list, modify, and delete selected records, DQL lets you:

- Create batch processing procedures to enter, modify, or delete sets of records instead of processing each record individually as changes are made in User View.
- Use a special Data-entry form L to collect selection criteria, processing instructions, and other user input each time a procedure is run.
- Monitor the entry of new records and determine how each record is processed depending on its field value.
- Use global variables to pass values from one DQL Procedure to another.
- Call and chain together menuscalls, menus, forms, reports, other DQL Procedures - even other programs - and link these actions together into a single automated Control procedure.

Who Should Read this Book

This book is written for professional application designers, system administrators, and experienced users who want to create DQL programs to extend the power of their applications.

If you are not familiar with DataEase, we recommend that you begin by performing the hands-on lessons in the **DataEase Quick Start Guide (QSG)**.

If you are responsible for designing custom applications, you'll find reference information on creating tables, forms, reports, and menus, defining relationships, and other database design guidelines in the **DataEase Designer's Guide (DG)**.

If you use an application designed by someone else, you'll find all the information you need in the companion volume, **DataEase User's Guide (UG)**.

Before you begin working with DQL, we recommend that you read the **DataEase User's Guide (UG)** and **Designer's Guide (DG)** and become thoroughly familiar with DataEase record entry and document creation operations.

How this Book Is Organized

This book is organized into four sections, as follows:

Section I, **Orientation**, introduces the main features of the DataEase Query Language and the DQL editing environment, and explains the basic methods used to create a DQL Procedure.

Section I contains Chapters 1, 1, and 2.

Section II, **Scripts**, guides you through the creation of several simple and complex DQL Procedures. Section II explains all major DQL concepts and methods, including Control and Processing Procedures, Data-entry forms, and accessing data stored in an SQL database.

Section II contains Chapters 3, 4, and 5.

Section III, **Tech Tips**, answers your most frequently asked DQL questions, solves the most common programming problems, and offers creative examples and short-cuts that show how you can get the most from DQL. Section III contains Chapters 6 and 7.

Section IV, **Keywords**, contains the **DQL Lexicon**, a dictionary-style reference that explains every command, function, and feature of DQL. Most entries in the DQL Lexicon section include a DQL script sample to illustrate how a particular DQL term is used in a script. Section IV contains Chapter 8.

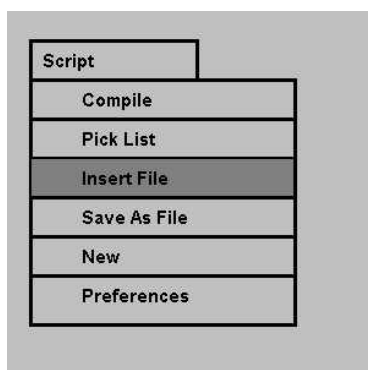
Typographical Conventions

Throughout the DataEase manuals, we've used the following typographical conventions to make the documentation easier to understand.

Term	Typeface Used	Example
Glossary Term	Italic	<i>Data Model</i>
Menu Option	Bold	File>>Open
Function Key	Uppercase	F7 DELETE RECORD
Application Name	bold	Club ParaDEASE
Document/Table Name	Uppercase	MEMBERS
Field Name	Small Capitals	FIRSTNAME
DQL Term	Bold Lowercase	highest of
SQL Command	Uppercase Courier	CREATE TABLE

Glossary terms with special meaning in DataEase are printed in italics the first time they occur in the text and are defined in the on-line glossary available in Help.

When a menu option is explained, a pull-down menu appears in the upper right-hand corner of the page. The menu option under discussion is highlighted, as in the illustration below:



DataEase function keys are assigned names (F2 SAVE AS NEW RECORD). However, you need only press the F2 key to execute the command; you do not need to type SAVE AS NEW RECORD. The key name is included in the documentation to help you recall the purpose of each key when you are learning the product.

The Basic DQL Vocabulary

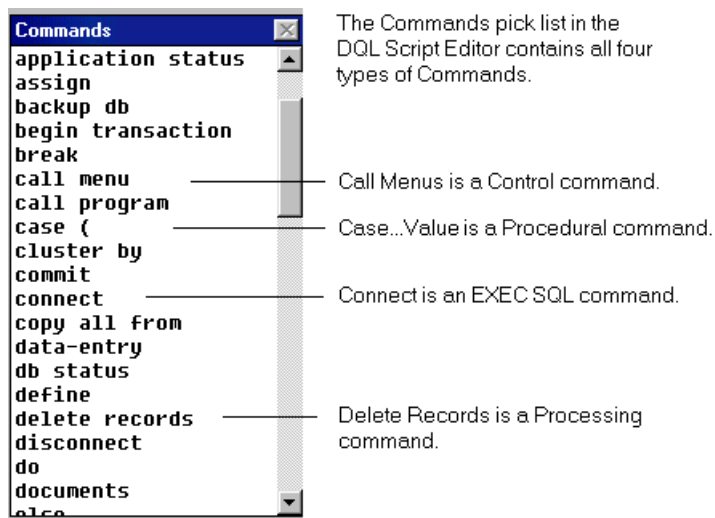
DQL keywords and symbols are organized into nine categories:

- **Commands**
- **Operators**
- **Functions**
- **Database Objects**
- **Relationships**
- **Symbols**
- **Values**
- **Variables and Constants**
- **Comments**

Commands

Commands let you **list**, **enter**, **modify**, or **delete records**, execute another **procedure** or **external program**, and direct output to the screen, disk, or printer. DQL contains four types of commands:

- **Processing Commands** let you directly manipulate data. Examples include list records, modify records, and delete records.
- **Procedural Commands** let you control the flow of actions within a DQL Procedure. Examples include **if...then...else**, **case...value...others**, **while...do**, **define**, and **message**.
- **Control Commands** let you link any number of DQL Procedures together, making it possible to perform several actions on multiple data sources within a single procedure. Examples include run procedure, call menu, call program, import, and backup db.

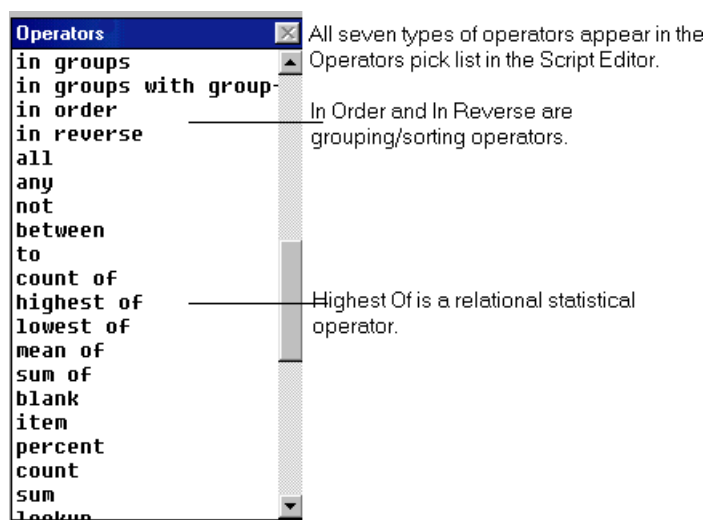


- **SQL Environment Commands** let you view and process data stored in a remote SQL database. Examples include exec SQL, commit, rollback, tran on, and tran off.

Operators

Operators are used to manipulate variables and tell DataEase how to carry out certain Processing and Control Commands (such as the order in which to process records, what statistics to generate, etc.). DQL includes seven types of Operators:

- **Comparison Operators** are used to compare two values. Examples include <= (less than or equal to), > (greater than), and = (equal to).
- **Grouping/Sorting Operators** tell DataEase how to group and order records when displaying or printing data. Examples include in groups, in order, and in reverse.
- **Relational Operators** all and any are used to list records related to the record currently being processed.
- **Statistical Operators** summarize the values of a field for all records processed. Examples include sum, mean, max, and min.
- **Conditional Statistical Operators** generate summary information about a set of records that meet a specified condition. Examples include count, percent, and item.
- **Relational Statistical Operators** summarize information about fields in a set of records related to the record currently being processed. Examples include count of, highest of, lowest of, mean of, and sum of.



- **General Operators**, include the arithmetic operators (* (asterisk) , / , + (addition) , and - (subtraction)), the assignment operator :=, and the logical operators (**and** and **or**).

Functions

A function is a routine that performs a particular calculation, text manipulation, or other data processing task. DQL provides 58 functions grouped into nine categories: Date, **Time**, **Spell**, **Text**, **if**, **Math**, **Financial**, **Scientific**, and **Trigonometric**.

A function is usually followed by one or more **parameters** (also called arguments) enclosed within parentheses. A parameter is information you supply to specify the operation of the function.

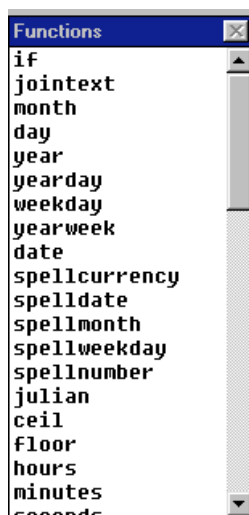
For example, the `firstc` function lets you extract a number of characters from the beginning of a text value. To use the `firstc` function, you must supply information telling DataEase: (1) from what text value to extract characters, and (2) how many characters to extract. The statement below shows the `firstc` function and the two parameters needed to extract the first four characters in a LAST NAME field:

```
firstc (LAST NAME,4)
```

When processing a MEMBERS record for a member named Williams, the above statement returns the value, "Will".

DataEase lets you use functions in field Derivation Formulas as well as in a DQL Procedure.

The Functions pick list in the Script Editor is illustrated below:



Database Objects

Database objects are the entities you create in an application to store and access data. Tables, columns, fields, relationships, and documents are all examples of database objects that can be specified in a DQL script.

For example, the script below specifies two table names and three column names that tell DataEase what data to print in a report:

```
for MEMBERS ;
  list records
    FIRST NAME ;
    LAST NAME ;
  all FAMILY MEMBERS FIRST NAME .
end
```

Relationships

Relationships link records stored in different database tables. In a DQL Procedure, you can access data in a related table using predefined relationships (stored in the Relationships form) and **adhoc relationships** that are defined within the procedure.

The figure below shows the Relationships form.

Relationships

CHOOSE THE TABLES TO RELATE...

Records in are related to records in

based on the following fields being equal: (define at least one set of fields)

=

=

=

OPTIONS..

Unique Relationship Names - records in each Table can be referenced as:

(Note: if these tables are used in another relationship, Unique Names are required.)

Referential Integrity - the effect a change to Main Form key field has on SubForm key

(use Toolbar for other record actions)

Symbols

See **DQL 8** for more information on specific DQL language elements.

A **symbol** is a character used to punctuate a DQL script. Examples include a period, comma, semicolon, and parenthesis.

Additionally, special symbols called **wild card symbols** are used to represent unknown characters. Wild card symbols include the asterisk (*), tilde (#), and question mark (?). The example below shows a question mark used to take the place of an unknown character:

```
for MEMBERS with LAST NAME = "Anders?n " ;
list records
    LAST NAME;
    FIRST NAME.
end
```

Variables

A variable is a value that can change while a script is being processed. A variable may contain a number, numeric string, text string, time, or date value.

When you define a variable, you specify a word or letter to represent the variable in the script. You also specify what type of data is to be stored in the variable. The following line defines a variable named DISCOUNT, used to store a number value:

```
define temp "DISCOUNT" Number.
```

Once a variable is defined, you can use the assign command to set its value. The following line sets the value of the DISCOUNT variable to fifteen percent of a member's TOTAL DUE:

```
assign temp DISCOUNT := 0.15 * TOTAL DUE.
```

Although a variable can hold only one value at a time, that value can change any number of times during processing. In the example above, the value of DISCOUNT might change with each MEMBERS record processed.

DataEase lets you define a temporary variable to store a value during a single DQL Procedure, or a global variable used to pass a value from one DQL Procedure to another.

Values

Values are the data processed by a procedure. A value can be any number or text string stored as a constant or variable, or the contents of a data field (the current value in the CLUB NAME field, for example).

Comments

Comments are notes and explanations you can include in a DQL script to make it easy for you or another programmer to understand. Indicate the beginning of a comment with a double hyphen (-). When executing the script, DataEase ignores all text between the double hyphen and the end of the line.

```
-- This script lists MEMBERS from Texas.

for MEMBERS with STATE =  "TX" ;    -- select Texas members.
list records                        -- Print the members'
  FIRST NAME;                      -- first and last names.
  LAST NAME.
end                                -- end of script.
```

Getting Help on DQL Keywords and Concepts

The DataEase online Help system lets you get help on any DQL keyword or concept. Most DQL topics in the Help system also contain a script sample that you can copy and paste directly into your script.

To view a list of DQL-related Help topics, choose **Help>>Contents**. When DataEase displays the Help Contents screen, click on the Procedure icon. DataEase displays a list of all DQL terms and concepts.

Click on a DQL term to jump directly to a description of that term.

How to Access DQL Help

1. Choose **Help>>Contents**. DataEase opens the Help window and displays the Help Contents screen.
2. Click on the Procedure icon in the Contents screen. DataEase displays a list of topics (jump terms), one for each DQL keyword and concept.
3. Click on a DQL topic. DataEase jumps to an explanation of the selected topic.
4. Choose **File>>Exit** to exit the online Help system.

General Steps for Creating a DQL Procedure

A DQL Procedure can perform a simple and direct action (such as listing how many members live in Connecticut) or a series of complex actions. For example, you can execute a procedure at the end of the day that prints several summary reports, exports data via modem to each branch office, and finally backs up the database. Regardless of the level of complexity, all procedures have one thing in common, a **script**.

A DQL script is a series of instructions that tell DataEase what actions to perform. The type of script determines what other elements the DQL procedure contains. For example, a script that produces a printed report needs a layout. On the other hand, a script that deletes a group of records from a table does not require a layout since this script doesn't generate any visible output.

The steps required to create a typical DQL Procedure are:

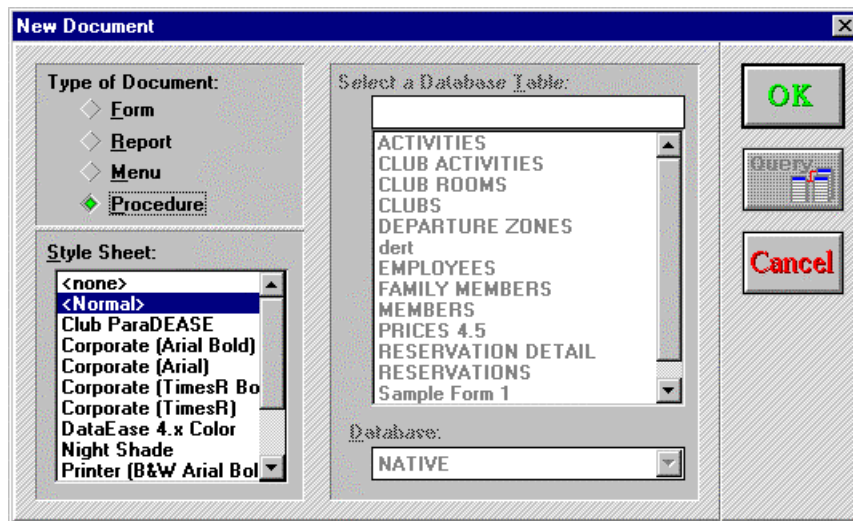
- Start a new procedure document (required).
- Create a Data-entry form (optional).
- Create a scriptSCRIPT (required).
- Save the procedure (required if you want to access the procedure again in the future)

Although you normally perform the steps in the order listed above, DataEase lets you vary the sequence. For example, if you know ahead of time that you want to include a Data-entry form for the procedure, you can create the form before you write the script.

Starting a New DQL Procedure

When you start a new DQL Procedure, DataEase displays the New Document Dialog shown below. If the procedure produces visual output, you can select a Style Sheet. Other options in this dialog are disabled.

When you finish making your selections in the New Document dialog, click OK to display the Script Editor.



How to Start a New DQL Procedure

1. Choose **File>>New>>Procedure**. DataEase displays the New Document dialog.
2. Double-click on the style sheet you want the layout to use.
3. Click OK. DataEase automatically opens the Script Editor Window.

Creating a DQL Script

See **DQL 3, 4** for examples of the for, if..then..else, case, and while..do statements used in the context of a complete script.

Every DQL Procedure must have a script (a series of DQL commands that tell DataEase what actions you want to perform). DataEase gives you two ways to create a script. You can:

- Create a new script using the Script Editor.
- Use the Script Editor to modify an existing script or copy portions of one script to another.

Using the Script Editor to Create a Script

In the Script Editor you can enter a script by typing your selections directly into the editor, by making selections from the interactive pick lists displayed at the bottom of the Script Editor Window, or by copying text from another source (e.g., another script, external text file, or a sample script from the online help facility).

The advantage to making selections from the interactive pick lists is that you don't need to remember the exact names of different objects in the application since they are displayed for you. If you type your selections directly into the editor, you must correct any spelling errors before you can execute the procedure.

Copying from One Script to Another

DataEase provides three ways to copy all or part of a script so you can use it in another procedure:

- Copy all or part of an existing script to the Windows Clipboard, then paste the copy into another script and use the copied script as a starting point for a new script or as an addition to another existing script. With both procedures open, use the Copy option to copy all or part of the first script to the Clipboard, then use the Paste option to paste the text from the first script into the second script.
- Select **File>>Save As** and save the existing script under a new name. This option lets you keep the original procedure intact and use the newly named copy as the starting point for defining a new procedure.
- Copy all or part of a sample script from the online Help facility.

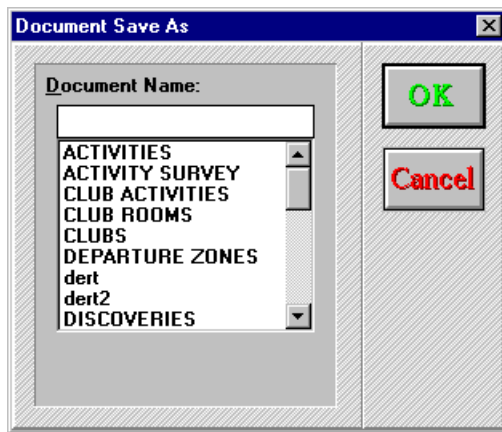
How to Create a DQL Script

There are four possible ways to create a script:

- Type the script directly into the Script Editor window.
- Double-click each of the required commands in the interactive pick lists displayed at the bottom of the Script Editor window.
- Choose **Script>>Insert File**. Then double-click to select the directory where the text file is stored and double-click to select the filename to load.
- Copy a code sample from the online Help system to the Clipboard and then to the Script Editor.

Saving a DQL Procedure

You can save a DQL Procedure any time during the creation of the procedure. If you are saving the procedure for the first time, DataEase displays the Document Save As dialog shown below.



Enter a procedure name up to 20 characters long. It can be any name you like, but each document in the same database must have a unique name. After you specify the name, click OK. DataEase saves the procedure on disk.

To save an existing procedure under a new name, choose **File>>Save As**. DataEase displays the Document Save As dialog, where you can enter a new name for the procedure. After you enter a new procedure name, the modified procedure is saved under the new name. DataEase also retains the original (unmodified) procedure under the original name.

Using the Check DQL Option

Although you can save a script at any time, you cannot execute a procedure whose script contains spelling and/or syntax errors. You must correct all syntax and spelling errors in your script before DataEase can execute the procedure. DataEase automatically checks both spelling and syntax when you choose **Script>>Check DQL**.

DataEase prevents you from running a procedure until you use the Check DQL option to verify the script. DataEase performs two functions when you run Check DQL. DataEase checks the script for spelling and syntax errors and prepares the script for execution. If any errors are detected, DataEase displays the error message shown below, indicating both the location and nature of the error.

{bmct dqli-18.bmp}

After you have successfully run the Check DQL option, you can execute the Procedure.

How to Use the Check DQL Option

- Choose **Script>>Check DQL**. If there is an error in the script, DataEase indicates the location of the error with an error message.
- Correct the error and choose **Script>>Check DQL** again. Repeat the process until the script is error-free. When there are no more errors, DataEase compiles the script.

How Security Works with DQL Procedures

Every user, regardless of their security level, can create a DQL Procedure. The settings selected in the Document Securities dialog control each user's ability to view, modify, delete, and execute a procedure. Since a procedure is a type of document, these settings can vary from one procedure to another. The default settings are shown in the table below.

How Security Affects DQL Procedures

User Security Required to ...	Document Security Option That Controls This Function	Default Setting
View an existing Procedure	View Layout	These options are automatically set to the level of the user who created the document.
Modify an existing Procedure	Modify Layout	
Delete an existing Procedure	Delete Document	
Open an existing Procedure	Open Document	Low 3

The default settings for each of these security controls is established when the document is created. For example, if a Medium3 level user creates a document, the Document Security options for View Layout, Modify Layout, and Delete Document are automatically set to Medium3. Any user with an adequate security level (e.g., Medium3 or higher in this example) can change the default settings at a later date.

DataEase prohibits you from listing, modifying, or deleting data to which you have inadequate view or write access. You can include the names of any columns in a script, but when the script is compiled, if you have inadequate security access to any of the columns listed, DataEase displays a warning message.

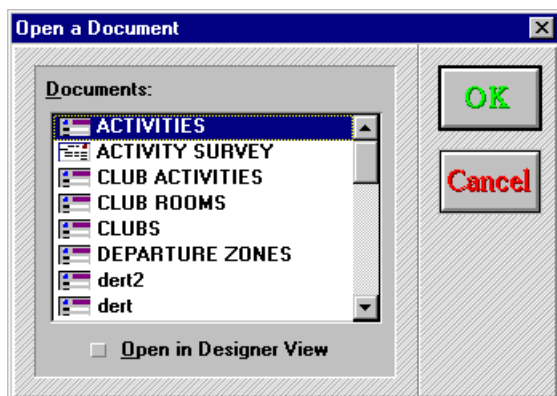
DataEase displays a similar warning message when you attempt to execute such a procedure. The table below shows the action DataEase takes when you execute a procedure that accesses data to which you have inadequate view or write security.

Data Access Restrictions

DQL Command	Result
list records	DataEase lists only those data values to which you have adequate view security.
modify records	DataEase modifies only those columns to which you have adequate write security.
delete records	If the table contains any columns to which you have inadequate view or write security, Data Ease will not delete any of the records in the table.
enter a record	DataEase enters only those columns to which you have adequate write security.

Loading a DQL Procedure

You can load a previously created procedure into memory to view it, modify it, or run it. Choose **File>>Open** to display a list of all the existing documents in the application. Specify the procedure you want to load by clicking its name. Check Open in Designer View, then click OK to view or modify the procedure.

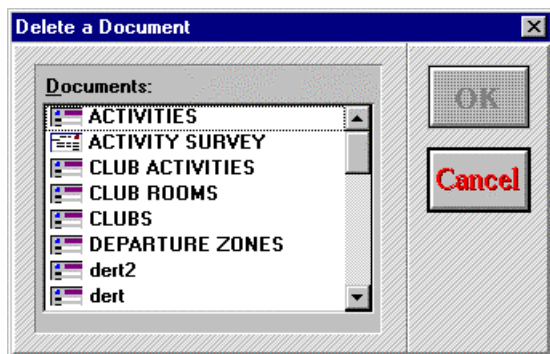


Executing a DQL Procedure

If you define a procedure and then switch to User View, DataEase processes the procedure loaded in the computer's memory. If you choose **File>>Open** in User View, you can run a procedure by selecting it from the dialog shown above. You can also run a procedure by highlighting it in the Application Catalog and clicking the User View icon.

Deleting a DQL Procedure

DataEase lets you permanently delete a procedure that is no longer necessary. When you choose **File>>Delete**, DataEase displays the Delete a Document dialog, which lets you specify the procedure you want to delete by highlighting its name then clicking OK. When you click OK, DataEase displays an alert dialog where you must verify that you want to delete the procedure before it is removed from the disk.



MEMBER LIST: Basic DQL Procedure

The **Club ParaDEASE** MEMBER LIST procedure appears throughout the rest of this chapter to illustrate basic DQL concepts. MEMBER LIST is a simple DQL Procedure that lists club members living in California and their family members.

As you read the following pages, you may want to view the DQL Script Editor and the components of the MEMBER LIST procedure on screen.

To open the procedure, start the **Club ParaDEASE** sample application if you have not already done so. When DataEase displays the Application Catalog, double-click on the Procedures item in the Catalog to display the list of **Club ParaDEASE** procedures.

Parts of a DQL Procedure

Like a form, report, or menu, a DQL Procedure is a type of DataEase document. A procedure contains many of the same elements as other documents, including the Body, Page Header and Footer, and Summary Header and Footer, that control the appearance of a procedure's printed output. However, a DQL Procedure contains two additional elements that do not appear in a form or report: a script and a Data-entry form. The script appears in the DQL Script Editor window.


This section briefly describes how the various parts of a DQL Procedure fit together.

To open the procedure in Designer View, click once on the MEMBER LIST procedure name in the Catalog, then click on the Designer View button on the Toolbar. DataEase displays the procedure.

Script

A script is a set of DQL instructions that DataEase executes when you open a procedure document. A script can ask a question about data in one or more tables, list or manipulate that data, or chain together any number of separate data management operations.

The figure below shows the script for the MEMBER LIST procedure. If you are viewing the



```

DataEase: Club_ParaDEASE - [EDIT PROCEDURE: MEMBER LIST]
File Edit View Script Document Application Window Help
For MEMBERS
  with STATE = "CA" ;
  list records
    MEMBER ID ;
    LAST NAME ;
    FIRST NAME ;
    all FAMILY MEMBERS FIRST NAME ;
    all FAMILY MEMBERS DATE OF BIRTH .
  
```

Every DQL Procedure contains a script.

A script is a set of DQL instructions that tells DataEase how to process data and perform other actions.

MEMBER LIST procedure and the Script Editor is not currently displayed in the active window, choose View - Procedure to activate the Script Editor.

The script above tells DataEase:

- Find all records in the MEMBERS table for members who live in California.
- List the MEMBER ID, LAST NAME, and FIRST NAME for each selected MEMBERS record.
- For each MEMBERS record, find all related FAMILY MEMBERS records and list the FIRST NAME and DATE OF BIRTH values from each of those records.

The output generated by this script appears below.

DataEase: Club_ParaDEASE - [PROCEDURE : MEMBER LIS]

File Edit View Goto Query Application Window Help

MEMBER ID LAST NAME FIRST NAME

00013	Sullivan	Hope
FIRST NAME	DATE OF BIRTH	
Hope	03/14/66	
Warren	06/16/63	
00016	Young	John
FIRST NAME	DATE OF BIRTH	
John	04/21/42	
Aubrey	11/09/68	
Beth	05/01/44	
00037	Kowalski	Ross
FIRST NAME	DATE OF BIRTH	
Ross	04/13/60	
Lydia	03/28/63	
00058	Steinberg	Julia
FIRST NAME	DATE OF BIRTH	
Julia	12/06/54	
Ray	03/19/79	
Lois	08/30/82	
Ralph	07/17/52	

Body of a Procedure

When you define a script that generates output, the output is printed in the body of the document. You can create a layout for the body to determine how that output appears when printed.

Creating a layout for a DQL Procedure is just like creating a report layout: you can use DataEase's automatic layout options or you can define a custom layout.

To view a layout on screen, open the procedure in Designer View, then choose **View>>Body**. The figure below shows the layout for the MEMBER LIST procedure.

DataEase: Club_ParaDEASE - [DESIGN PROCEDURE : MEMBER LIST]

File Edit View Objects Document Application Window Help

Table Grid Form

MEMBER ID	LAST NAME	FIRST NAME
MEMBER ID	LAST NAME	FIRST NAME
FIRST NAME	DATE OF BIRTH	
FIRST NAME	DATE OF BIRTH	

Create a procedure layout just as you do for a form or report. This layout displays data from the primary table (MEMBERS) on the Main form.

Records from the related table (FAMILY MEMBERS) appear in the Subform.

DataEase also lets you create a DQL Procedure that does not generate any output. For example, you can create a procedure that simply deletes outdated records, performs an import, backs up the application, or transfers data from one table to another. When you run that procedure, DataEase displays it as a blank screen until the procedure is executed.

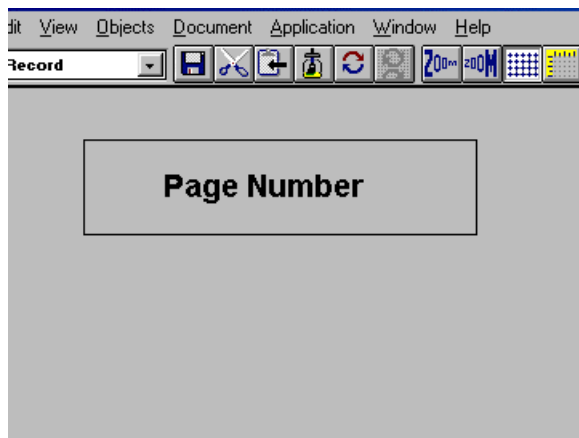
Page Header and Page Footer

You can enhance the printed output of a DQL Procedure by creating Page Header or Page Footer. Page Headers and Footers appear on each printed page and can contain text, graphics, a page number, the current date, and other useful information.

To view a document Page Header or Page Footer on screen, choose **View>>Page Header** or **View>>Page Footer**.



A report Title is added to the Page Header area.



Report output is further enhanced by adding Page Numbers to the Report Footer area.

To create a Page Header or Page Footer for a DQL Procedure, follow the same steps used to create Page Header or Page Footer for a form or report: choose **View>>Page Header**, then create the text objects, images, lines, variables, and any other objects you want to include.

Summary Header and Summary Footer

See **DG 6** for information on creating Summary Variables.

You can enhance the printed output of a DQL Procedure by creating a Summary Header or Summary Footer. Summary Headers and Footers appear on each printed page between the Page Header and Footer and the body of the document, and can contain text, graphics, a page number, the current date, and other useful information.

To view a document Summary Header or Summary Footer on screen, choose **View>>Summary Header** or **View>>Summary Footer**.

To create a Summary Header or Summary Footer for a DQL Procedure, follow the same steps used to create Page Header or Page Footer for a form or report: choose **View>>Summary Header**, (then create the text, images, lines, variables, and any other objects you want to include).

The Difference Between Summary and Page Headers and Footers

- If the output of a procedure or report is too wide to fit onto a single page, it is tiled across the width of several pages.
- Pages Headers and Footers appear at the top and bottom of every tiled page.
- Summary Headers and Footers appear once for every 'vertical' page, that is, they do not repeat on every tiled page.

Data-Entry Form

A Data-entry form is a special form that appears at the start of a DQL Procedure. A Data-entry form lets you specify selection criteria and other information used by the procedure before processing begins. For example, a Data-entry form can appear at the beginning of a mailing label printing procedure to let you specify a particular group of customers for whom to print labels.

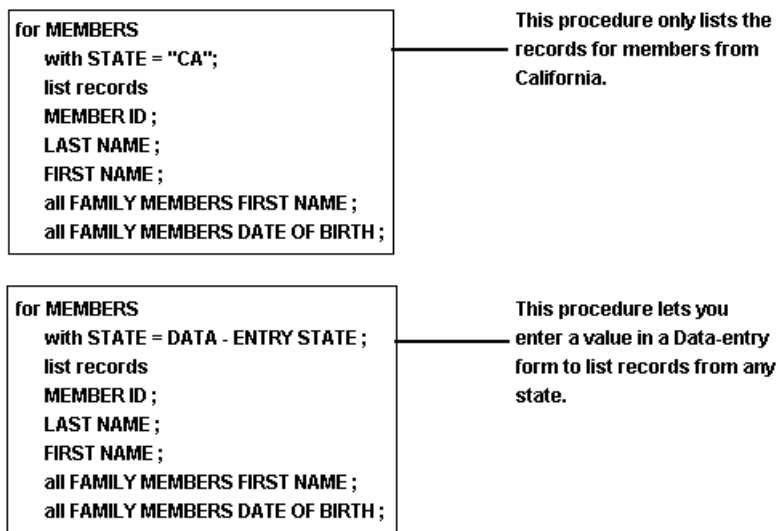
As originally designed, the MEMBER LIST procedure can list records from only one state, California. By adding a Data-entry form like the one shown in the figure below, you can make the procedure flexible enough to list records for any state.

The screenshot shows a window titled "DataEase: Club_ParaDEASE - [PROCEDURE : MEMBER LIST]". The window has a menu bar with "File", "Edit", "View", "Goto", "Query", "Application", "Window", and "Help". Below the menu bar is a toolbar with various icons. The main area of the window is a data-entry form with a grey background. The form has the title "Filter members by State" in large, bold, black text. Below the title, it says "List families from which state?". Then, it says "Enter a two-letter code in the field, then press F2 to continue" followed by a small rectangular input field.

When you use a Data-entry form, you must usually modify the DQL script so DataEase recognizes the input from the Data-entry form. The illustration on the next page shows how the MEMBER LIST script can be modified to work with a Data-entry form.

See DQL 2 for information on creating a Data-entry form.

The figure below shows the original MEMBER LIST script (above) and the modified version for use with a Data-entry form (below). The modified version tells DataEase to use the selection criterion specified in the Data-entry form instead of a "hard coded" value (e.g., STATE = "CA").



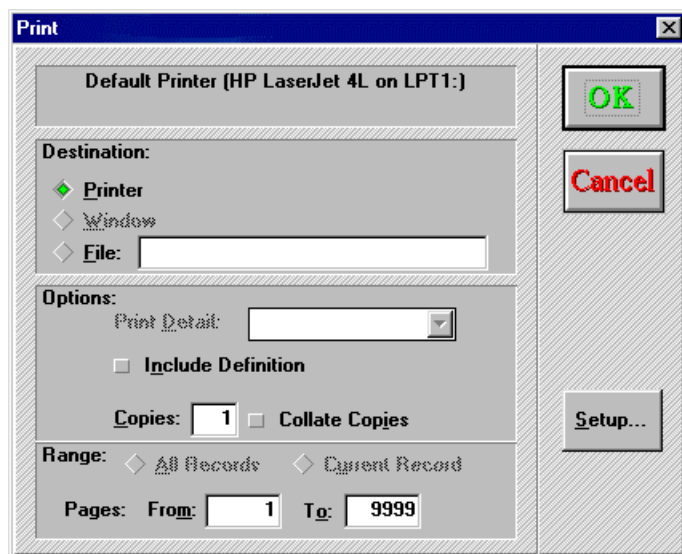
DataEase can automatically redisplay a Data-entry form **after** you run a DQL Procedure, so you can run that procedure repeatedly, processing a different set of records each time the procedure is executed. The data-entry form appears before the Print dialog, so you could also use it to first print to the screen, then to a printer.

Printing the Procedure Definition

The Procedure Definition is a printed document summary of the procedure. The definition is composed of four items:

- A copy of the script.
- A copy of the Data-entry form if one is defined for the procedure.
- A copy of the layout.
- A description of each object included in the layout. Each field description includes the Field Name, Type, Length, and a Yes or No indicating whether leading and trailing spaces in the field are removed when the output is displayed.

To print the definition of the procedure currently in memory, select **File>>Print**. DataEase displays the Print dialog. Check the Include Definition box in the Options section of the Print dialog, then click OK. DataEase automatically sends the output to the default printer.



You cannot print the Procedure Definition on the screen. If you select Include Definition and there is no correctly configured printer connected to the computer, DataEase displays an error message and cancels the print command.

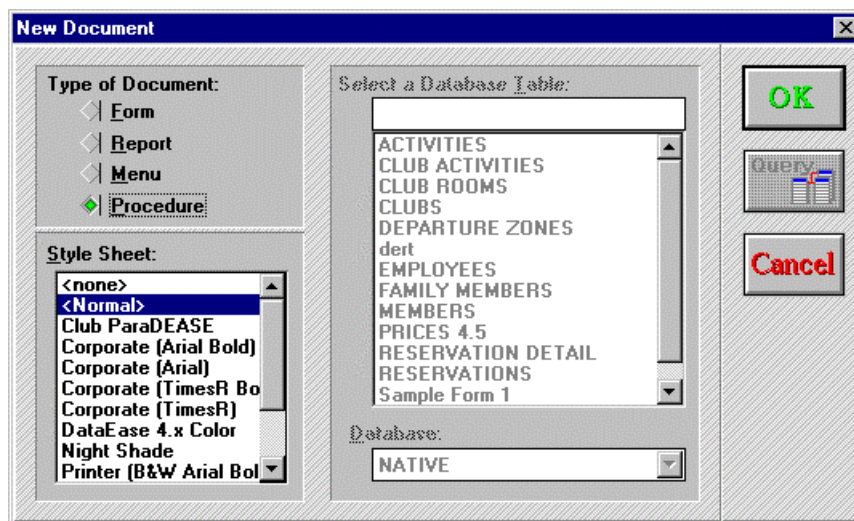
Chapter 1 : DQL Environment

Using the DQL Script Editor

This chapter explains how to create a DQL script and how to use each of the features available in the DQL programming environment, including the script editor box, interactive pick lists, and the special menu options that are only available when you are working in the DQL script editing environment.

Viewing the DQL Script Editor

To display the DQL Script Editor, choose **File>>New>>Procedure**. When DataEase displays the New Document dialog, click OK.



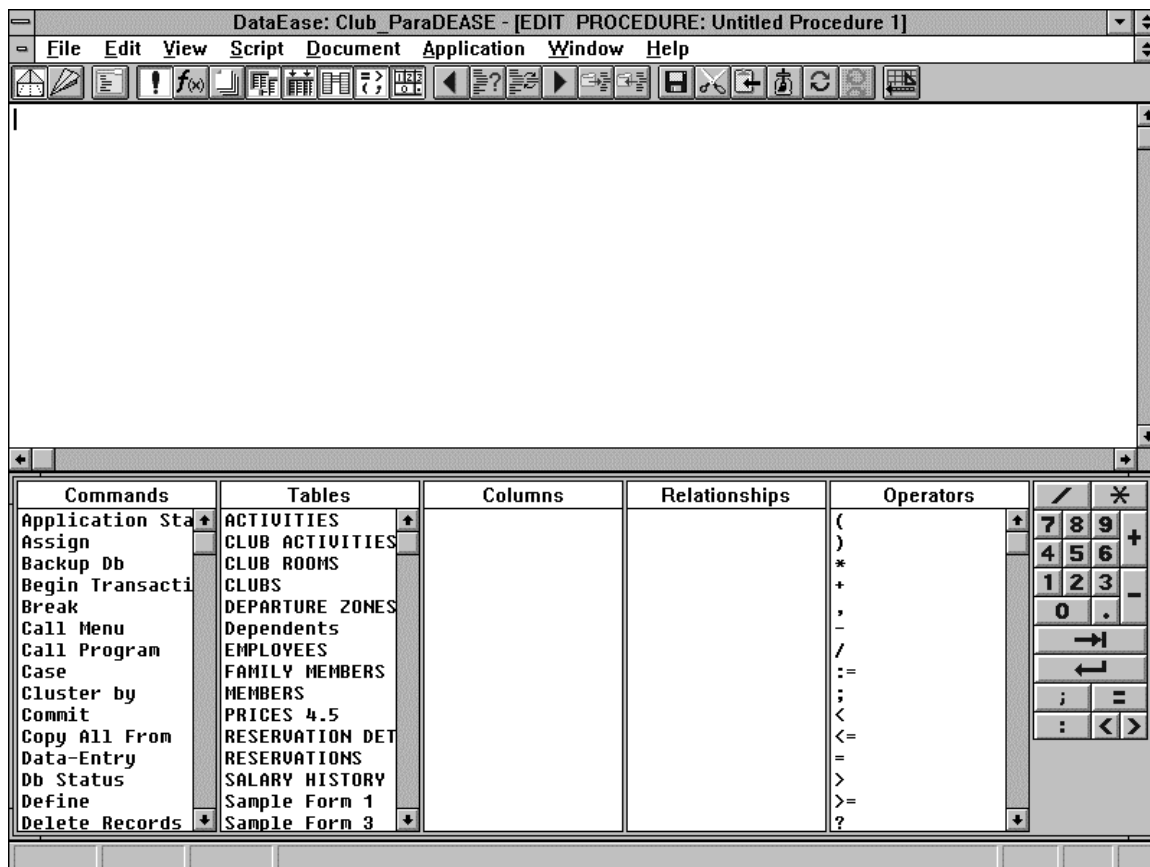
Choose a style sheet if your report generates print output, then click OK to close the New Document Dialog and open the Script Editor.

DataEase opens two windows: one displays the layout of the procedure; the other displays the Script Editor.

Parts of the DQL Script Editor

The DQL Script Editor, illustrated below, contains several features that help you write DQL scripts.

The DQL Script Editor contains the script editor box where you enter your DQL script...



...and interactive pick lists, from which you can select DQL commands, operators, and functions, and the names of the database tables, columns, and relationships in the application.

Interactive Pick Lists

The interactive pick lists contain all the DQL keywords, and all the application's document, table, column, and relationship names, so you don't have to memorize them in order to write a script. Just double-click the terms you want. DataEase enters them into the script automatically at the location of the text cursor.

If you prefer, you can simply type a script into the script editor box instead of using the pick lists.

Resizable Panels

You can adjust the height of the interactive pick list panel by dragging its top border up or down. If you're thoroughly familiar with DQL and the elements in your application, you may want to hide the pick lists to increase the text editing area in the Script Editor. You can hide individual pick lists by clicking on the appropriate pick list icons, or hide the entire pick list panel by choosing **Script>>Pick Lists>>Show Pick Lists**.

Moving the Cursor in the DQL Script Editor Window

The table below summarizes the cursor movement keys used in the DQL Script Editor. You can also use the mouse and scrollbars to navigate around the script editor box.

Cursor Movement in the DQL Script Editor

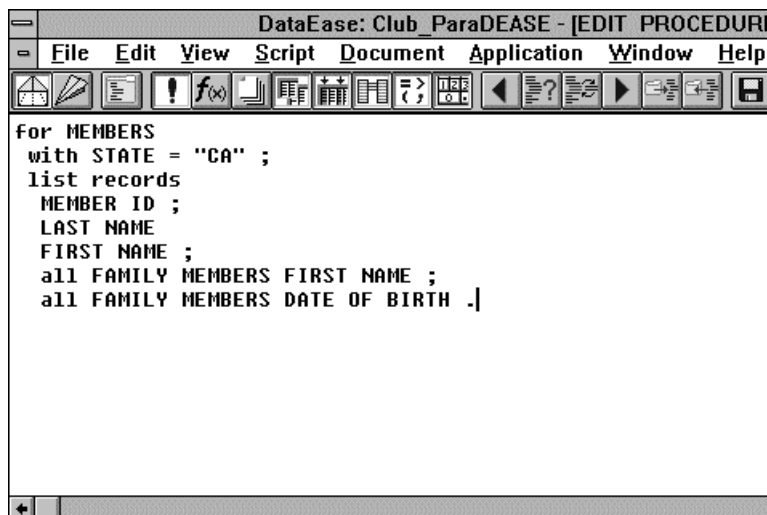
Keystroke	Function
← or —	Move the cursor one character to the left or right.
↑ or ↓	Move the cursor one line up or down.
Ctrl + ←	Move the cursor one word to the left.
Ctrl + —	Move the cursor one word to the right.
Shift + —	Select one character to the right.
Shift + ←	Select one character to the left.
Ctrl + Shift + —	Select one word to the right.
Ctrl + Shift + ←	Select one word to the left.
Del	Delete the character at the cursor location.
Backspace	Delete the character to the left of the cursor location.
Home	Move the cursor to the beginning of a line.
End	Move the cursor to the end of a line.
PgUp or PgDn	Display the next or previous screen.
Ctrl + Home	Move the cursor to the beginning of the script.
Ctrl + End	Move the cursor to the end of the script.

DQL Script Menu

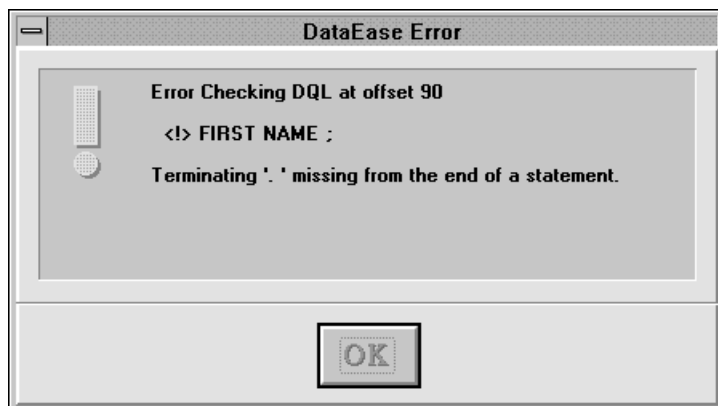
When you display the DQL Script Editor, DataEase displays the Script menu on the Menu Bar in place of the Objects menu. Use the options on the Script menu to check the syntax of a script, customize the Script Editor, load or save a script as a text file, and clear all text from the Script Editor.

Checking a DQL Script

When you choose **Script>>Check DQL**, DataEase displays a message to help you identify and fix the problem.



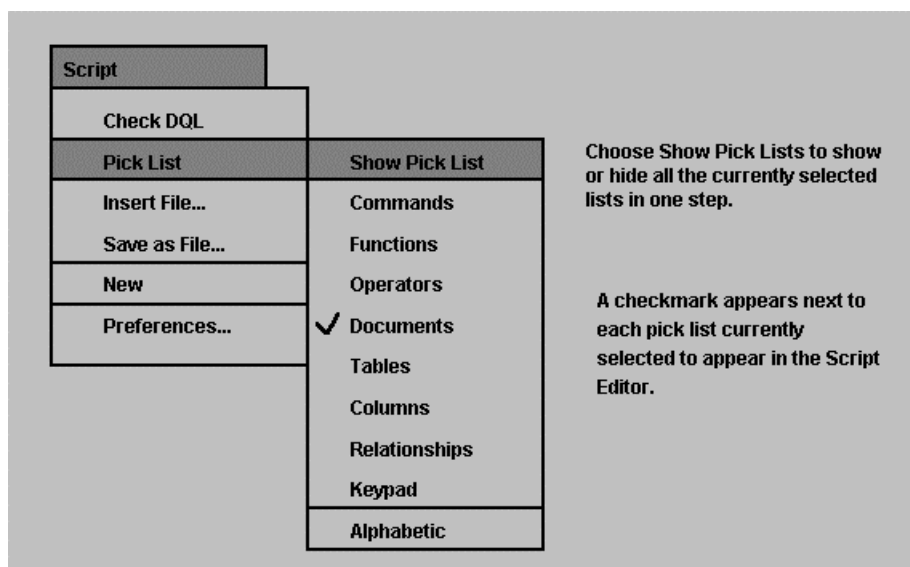
A syntax error occurs when you omit or mistype a required keyword or punctuation mark. The figure below shows an example of a common syntax error and the message DataEase displays upon finding it.



Although DataEase lets you save a procedure that contains errors, you cannot run the procedure until you check the DQL syntax and correct any errors.

Displaying Pick Lists

When you choose **Script>>Pick Lists**, DataEase displays a cascade menu that lets you hide or show the entire pick list panel or show individual lists (Commands, Tables, Columns, etc.) in the window. The last option on the cascade menu, Alphabetic, lets you display the pick lists in their conventional order or alphabetic order.



Loading and Saving a DQL Script as an ASCII Text File

Because DataEase lets you load and save a DQL script as an ASCII text file, you can create a script in a text editor other than the DQL Script Editor. Then you can build a library of frequently used DQL scripts and routines saved as text files.

When you choose **Script>>Insert File**, DataEase lets you load the contents of an ASCII text file into a script. Place the cursor at the location where you want to insert the text, then choose **Script>>Insert File**. DataEase displays the Load File dialog, which lets you specify the name and directory of the source file.

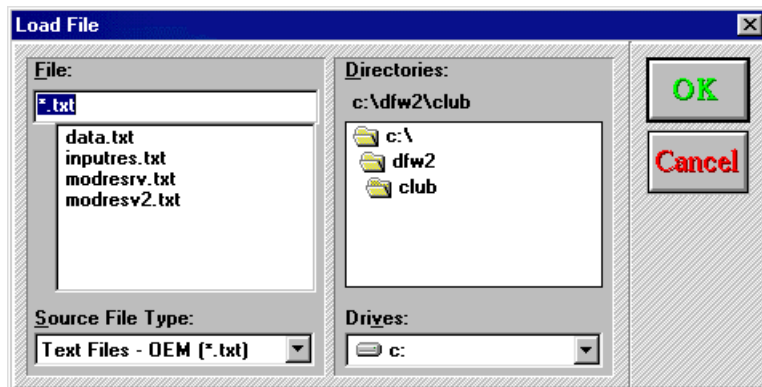
```
{bmct dqlOne-6.bmp}
```

When you click OK, DataEase closes the dialog and inserts the contents of the text file into the script.

How to Insert an ASCII Text File into a DQL Script

- Place the cursor in the script at the location where you want to insert the contents of the text file.
- Choose **Script>>Insert File**. DataEase displays the Load File dialog.
- Specify the drive, directory, and name of the source text file, then click OK. DataEase inserts the text at the cursor location.

When you choose **Script>>Save As File**, which lets you save a script as an ASCII text file.



DataEase displays the Save As File dialog.

When you click OK, DataEase closes the dialog and saves the script.

How to Save a DQL Script as an ASCII Text File

1. Choose **Script>>Save As File**. DataEase displays the Save As File dialog.
2. Specify a target drive and directory in which to save the file.
3. Specify a name for the text file, then click OK. DataEase saves the file.

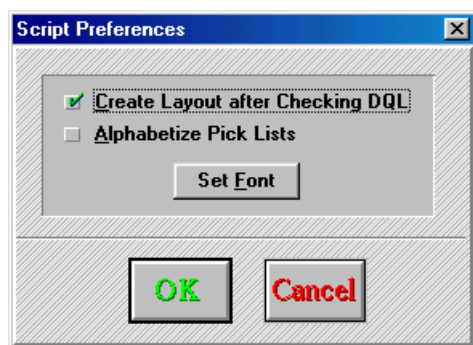
Clearing the DQL Script Editor

When you choose **Script>>New**, DataEase lets you quickly clear all text from the Script Editor. If you use the Script Editor to create multiple scripts or routines in a single session, you can use the New option to clear the Script Editor between scripts.

Setting Preferences for the DQL Script Editor

See **DG 6** for information on setting font attributes.

When you choose **Script>>Preferences**, DataEase displays the Script Preferences dialog shown below.



Script Preferences Dialog Options

Create layout after Checking DQL tells DataEase to display the body of the procedure layout immediately after the script is successfully checked. Deselect this option if you don't want to define a layout for the procedures.

Alphabetize Pick Lists tells DataEase to display all pick lists in alphabetic order. If you uncheck this option, DataEase displays the DQL keywords in their conventional order (as originally established in the character-based versions of DataEase).

Set Font lets you change the font, font size, color, and special effects of the text displayed in the Script Editor. When you change the Script Editor font, DataEase displays the Script the pick lists in the new font.

Searching for a Text String

When you choose **Edit>>Search**, DataEase displays the Search dialog, which lets you search a script for a text string. For example, you can use Search to locate each occurrence of a field name, table name, or variable name in a script.



Search Dialog Options

Search for lets you specify the text string.

Options let you specify additional search criteria:

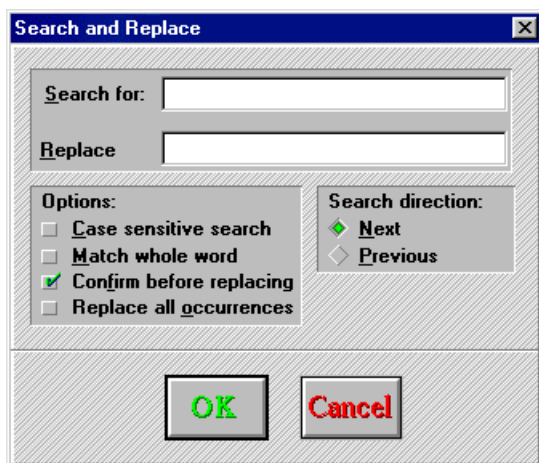
- **Case sensitive search** tells DataEase to find text strings that match your specified string exactly, including the capitalization of the letters. For example, if you select this option and search for the string, "MemberID", DataEase ignores "MEMBERID", "memberid", and "memberID".
- **Match whole word** tells DataEase to ignore a text string if it occurs as part of another word. For example, if you select this option and search for the string, "date", DataEase ignores "date1", "date_of_birth", "postdated", "update", and "validate".

Search Direction tells DataEase to search forward or backward through the script.

- **Next** tells DataEase to search forward, finding each instance of the target string that occurs between the current cursor location and the end of the script.
- **Previous** tells DataEase to search backward, finding each instance of the text string that occurs between the current cursor location and the beginning of the script.

Searching and Replacing a Text String

When you choose **Edit>>Search and Replace**, DataEase displays the Search and Replace dialog, which lets you search for a text string and replace each instance of that string with another text string. For example, you can use Search and Replace to replace each occurrence of a field name in a script with a different field name.



DataEase lets you decide whether to replace each individual occurrence of the specified text string, or replace all occurrences in one step.

Search and Replace Dialog Options

Search for lets you specify the text string.

Replace lets you specify the text you want DataEase to use to replace the target text string.

Options let you specify additional search criteria for the target text string, ask DataEase to prompt you before replacing each occurrence of the text string, and specify whether you want to replace one occurrence or all occurrences of the target string.

- **Case sensitive search** tells DataEase to find text strings that match your specified string exactly, including the capitalization of the letters. For example, if you select this option when searching for the string, "MemberID", DataEase ignores "MEMBERID", "memberid", and "Memberid".
- **Match whole word** tells DataEase to ignore a text string if it occurs as part of another word. For example, if you select this option and search for the string, "date", DataEase ignores "date1", "date_of_birth", "postdated", "update", and "validate".
- **Confirm before replacing** tells DataEase to highlight the target text string and prompt you to click Yes to replace the string or click No to leave the string unchanged.
- **Replace all occurrences** tells DataEase to replace all instances of the target text string that occur after the current cursor position. If you use this option while the Confirm before Replacing option is activated, DataEase asks for confirmation before replacing each instance of the target text string.

Search Direction lets you specify whether DataEase searches forward or backward through the script.

- **Next** tells DataEase to search forward, finding each instance of the target text string that occurs between the current cursor location and the end of the script.
- **Previous** tells DataEase to search backward, finding each instance of the target text string that occurs between the current cursor location and the beginning of the script.

How to Search for and Replace Text in a DQL Script

1. Choose **Edit>>Search and Replace**. DataEase displays the Search and Replace dialog.
2. Enter the text string you want to search for in the Search for text box.
3. Enter the replacement text in the Replace text box.
4. Specify the appropriate search Options.
5. Specify a Search direction. Click Next if you want DataEase to search forward in the script starting at the current cursor location. Click Previous to search backward through the script.
6. Click OK. DataEase finds the specified text string.

Moving Backward and Forward in a Search

When you choose **Edit>>Next**, DataEase searches forward to find each instance of the specified text string that occurs between the current cursor location and the end of the script.

When you choose **Edit>>Previous**, DataEase searches backward to find each instance of the specified text string that occurs between the current location and the beginning of the script.

DQL View Menu Options

See **DG 5** for information on viewing different parts of a document.

In the DQL environment, the first seven options on the View menu let you choose which part of a DQL Procedure is displayed in the active window. The View menu options let you display the body of a DQL Procedure, Summary Header, Page Header, Page Footer, Summary Footer, or Data-entry form in the active document window. The DQL Script Editor appears in its own window, so you can display the script, pick lists, and any of the other components of a procedure on screen at the same time.

DQL Toolbar

When the DQL Script Editor is displayed in the active window, DataEase replaces the Designer Toolbar with the DQL Toolbar. The DQL Toolbar lets you execute the most frequently used script editing actions with a single mouse click.

The DataEase for Windows DQL Toolbar



Full information on DataEase Toolbars can be found in the Help file.

Chapter 2 : DQL Enhancements

Creating the Data-Entry Form and Layout of a DQL Procedure

See **DQL I** for information on starting a new procedure, writing a script, and saving a procedure.

Creating a typical DQL Procedure includes five steps.

1. Start a new procedure document (required).
2. Create a Data-entry form (optional).
3. Create a script (required).
4. Create a procedure layout (optional) .
5. Save the procedure (required if you want to access the procedure again in the future).

Although you normally perform the steps in the order listed above, DataEase lets you vary the sequence. For example, if you know ahead of time that you want to include a Data-entry form for the procedure, you should create it before you write the script.

This chapter explains how to perform two optional steps, creating a Data-entry form and designing a layout.

Creating a Data-Entry Form

See **DG 3** and **6** for information on how to create a record entry form.

The Data-entry form is a special type of form that lets you enter varying information each time you run a procedure. The Data-entry form is a part of the procedure, just like the script and the layout. DataEase automatically creates a Data-entry form for each procedure. However, if you don't define any objects in the Data-entry form, DataEase treats the procedure as if it has no Data-entry form. If you have already created a script, you can easily add a Data-entry form by choosing **View>>Data-entry Form**. DataEase displays a blank document editor window on which you define the Data-entry form.

In general, you define the Data-entry form the same way you do a record entry form. However, because there is no underlying table associated with a Data-entry form, any data entered on the Data-entry form is not stored in the database. Therefore, the Data-entry form cannot contain:

- Indexed fields.
- Unique fields.
- Sequenced ID fields.
- Virtual fields.
- Subforms.

In all other aspects, defining the Data-entry form is exactly the same as defining a record entry form. You create Text, Field, Button, and other Document objects anywhere on the form. You can also use the Windows Clipboard to copy objects from an existing record entry form. You can create only one Data-entry form for each procedure. However, you can use the DQL input using or record entry commands to allow multiple input forms for a procedure.

How to Create a Data-entry Form

1. Choose **View>>Data-entry Form**. DataEase displays a blank form object.
2. Place and define Field, Text, Button, or Document objects on the form.
3. Choose **File>>Save**. If you haven't previously saved the procedure, DataEase displays the Document Save As dialog. Type a name for the procedure up to 20 characters in length.

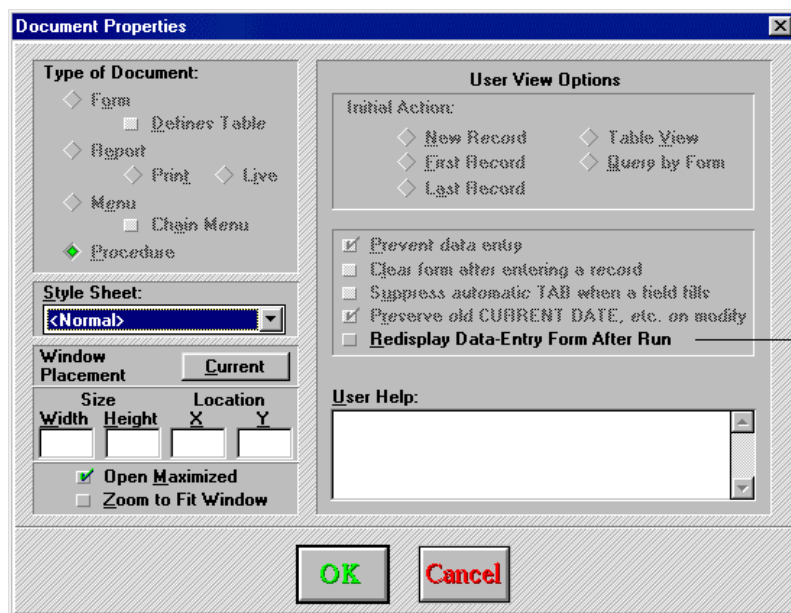
Displaying the Data-Entry Form

See **DQL 4** and **DQL Lexicon** for information on the input using command.

By default, the Data-entry form is displayed once when the procedure is first run.

The Document Properties dialog contains an option called "Redisplay Data-Entry Form After Run".

- If you check the Redisplay Data-Entry Form After Run box, each time the procedure is completed, the Data-entry form reappears and you can enter different data to be used in processing the script as many times as you want. When you finish running the procedure, choose **File>>Close** to exit the Data-entry form.
- Deselect Redisplay Data-Entry Form After Run if you only want to run the procedure once. The Data-entry form appears on the screen once at the start of the procedure. When DataEase finishes processing the procedure, it is automatically closed.



To eliminate the Data-Entry form entirely, deselect the Redisplay Data-Entry Form After Run option.

Select the Redisplay Data-Entry Form After Run option to display the Data-Entry form repeatedly. Deselect this option to display the Data-Entry form once only.

Note: The DQL input using command can be used to input data at any point during a procedure.

Using the Data-Entry Form

See **DG 8** for information on how to create a relationship

When you run a procedure that has a Data-entry form, the values you enter on the Data-entry form are used according to the instructions in the script. Each time the procedure is run, DataEase displays the Data-entry form, lets you type in data, and then uses the data to process the script.

In addition to supplying selection criteria for the script, a Data-entry form can be used to specify:

- Data entered or modified in one or more tables.
- Variables used in computations.
- Data used in conditional processing.
- Data printed in the report output.
- Important reminders to the user who runs the report (e.g., instructions on what type of paper to place in the printer).

Using a Relationship to Specify Data-Entry Values

If you define a relationship between a Data-entry form and a database table, you can use a lookup formula to automatically display information from the table in the Data-entry form.

When you create the relationship, the Data-entry form is identified by the name of its corresponding DQL Procedure. For example, suppose you want to enter a MEMBER ID number on the Data-entry form associated with your MEMBER_STATUS report and have DataEase lookup the corresponding LAST NAME. You must first create a relationship between MEMBERS and the Data-entry form, as illustrated in the figure below.

Relationships

CHOOSE THE TABLES TO RELATE...

Records in are related to records in

based on the following fields being equal: (define at least one set of fields)

<input type="text"/>	=	<input type="text"/>
<input type="text"/>	=	<input type="text"/>
<input type="text"/>	=	<input type="text"/>

OPTIONS..

Unique Relationship Names - records in each Table can be referenced as:

(Note: if these tables are used in another relationship, Unique Names are required.)

Referential Integrity - the effect a change to Main Form key field has on SubForm key

(use Toolbar for other record actions)

Save **Done**

Note: the procedure name will NOT appear in the table names picklist. You must type this in yourself.

Alternatives to Using a Data-Entry Form

See **DQL 8** for more information on the record entry and input using commands.

In a script, you tell DataEase to use the field values on the Data-entry form by placing the keyword `data-entry` before the field name. For example, the script statement

```
for MEMBERS with STATE = data-entry STATE
,list records
```

...tells DataEase to list only the members who live in the state specified in the `STATE` field on the Data-entry form.

The Data-entry form can only be used to input data at the start of a procedure. DataEase offers two other data input methods in addition to the Data-entry form.

The DQL record entry command allows records to be entered into a record entry form at any point during a procedure. The records are directly entered into the database; they are not processed by the script.

The DQL input using command also lets you use a record entry form to input information. The command may be used at any point during a procedure. The input using command lets the script process the records before they are entered into the database.

The differences between the three data input methods are summarized in the table below.

Data Entry Methods

Data Input Method	Use in Script	Full Record Entry Facility	DQL Control
data-entry	At Start Only	No	Yes
record entry	Anywhere	Yes	No
input using	Anywhere	Yes	Yes

Saving the Data-Entry Form

The Data-entry form is saved when you save the corresponding DQL Procedure. If you are saving a procedure for the first time, DataEase displays the Document Save As dialog. Enter a name for the procedure up to 20 characters long.

The data entered into a Data-entry form, however, is never saved on disk. The data is only saved in the computer's temporary memory and is erased when the procedure finishes processing.

Modifying a Data-Entry Form

When you display the Data-entry form in Designer View, you can modify it just the way you modify a record entry form. Remember, however, that if you add or delete fields on the Data-entry form, you should also modify the script to reflect those changes.

Formatting DQL Output

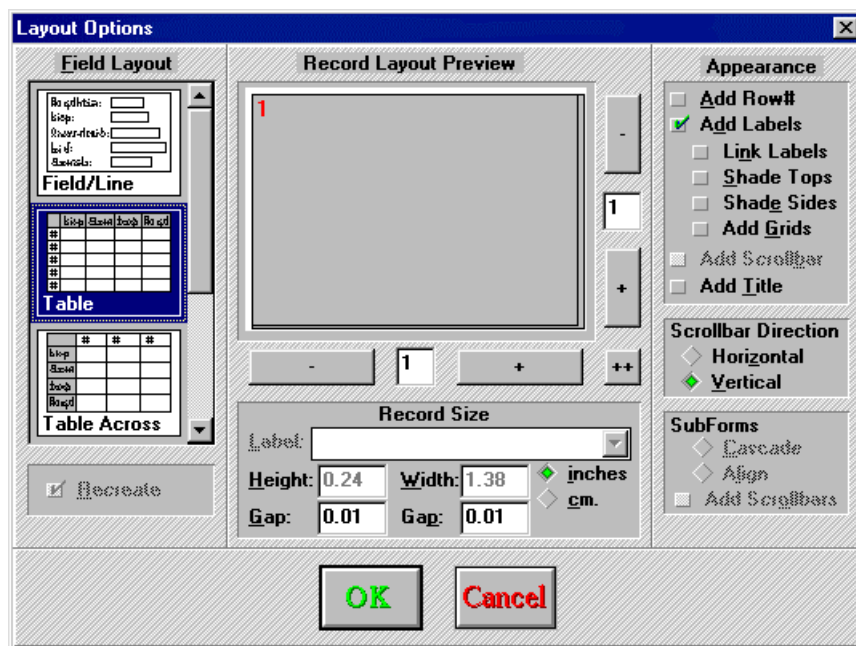
See **DG 3** and **6** for information on how to create a document layout.

Many DQL procedures generate output. For example, when you run a procedure that contains a list records command, DataEase processes the script and generates output, a list of records selected by the script. Output can be displayed on the screen, printed, or saved as a file. A visible form of output, particularly when it is printed on paper, is commonly called a report.

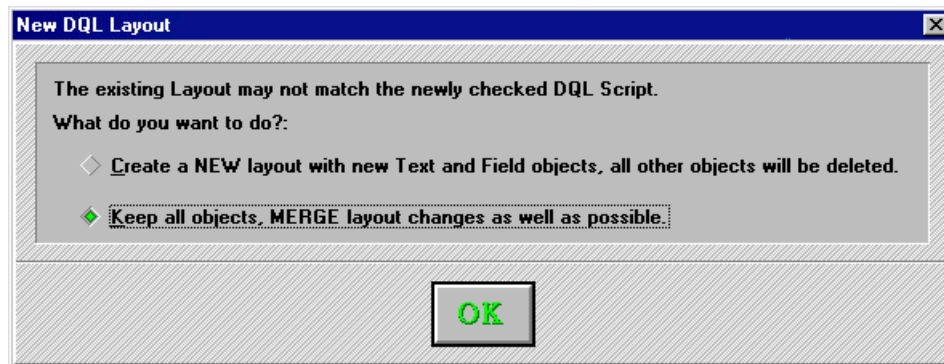
Some procedures create output that is not visible. A script that modifies records in the database, for example, does not display the modified result unless the script incorporates a list records command after the records are modified.

There are also procedures that do not generate any output. For example, a script that calls a user menu does not generate output.

If the script contains a list records command, DataEase generates output that you can display in a default procedure layout. Choose **Script>>Preferences**, DataEase displays the Script Preferences dialog. Deselect the Create Layout after Checking DQL option.



You can also create your own layout. When you choose **Script>>Check DQL** and successfully compile the script, DataEase automatically closes the Script Editor Window and displays the Layout Options dialog. The Layout Options dialog lets you control the appearance of your procedure output. You can select one of several predefined layouts, modify an existing layout, or create a custom layout on the screen. By default, DataEase automatically creates a columnar layout using field names as column headings. You can also choose **View>>Body** to view an existing layout at any time.



If you have already created a layout for the procedure, when you successfully compile the script, DataEase displays the New DQL Layout dialog shown below.

Chapter 3 : List Records

Using DQL to Create a Report

A **query** is a DQL script that asks a question about data in one or more tables. For example, you may want to find out which **Club ParaDEASE** vacation spots are the most popular among club members living in New England, or generate a list of all club members with large families.

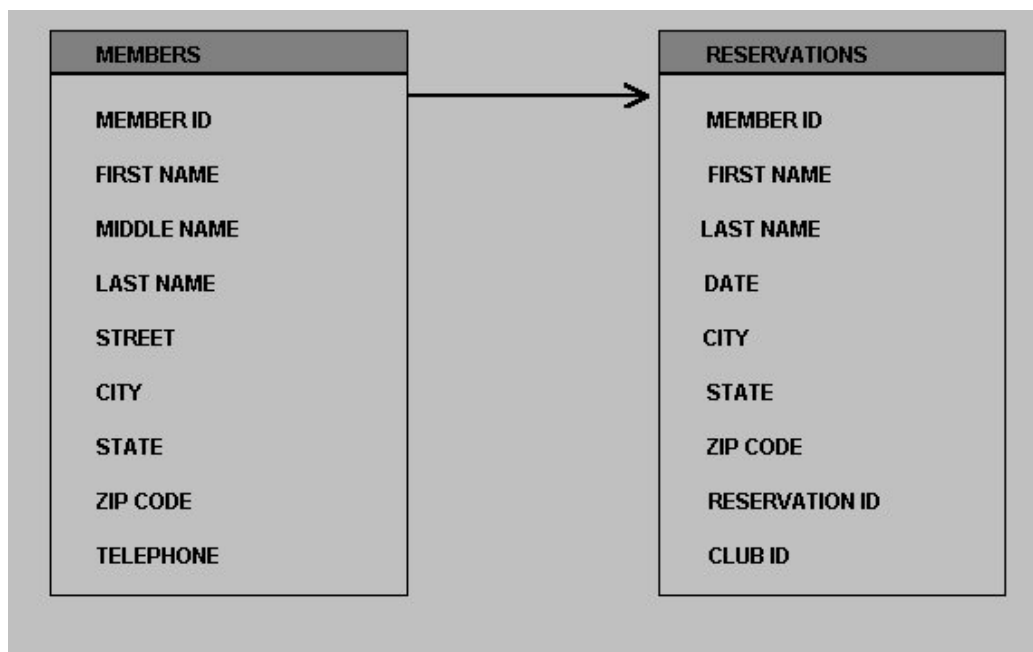
This chapter shows how to create a simple query that lists data from the MEMBERS and RESERVATIONS tables in the **Club ParaDEASE** sample application.

As you read through the example in this chapter, you may want to perform the actions described in the text. If so, open the **Club ParaDEASE** sample application, then create a new DQL Procedure following the steps beginning on the next page. If you find a DQL term you don't understand, refer to the DQL Lexicon (Chapter 8 in this book) for an explanation.

The MEMBERS and RESERVATIONS tables share a predefined relationship based on the linking MEMBER ID field. Both tables store the same information in this field: a unique number assigned to each club member.

The MEMBERS table has a **one-to-many** relationship to the RESERVATIONS table. This means that for any one MEMBERS record, there may be **many** related records in the RESERVATIONS table (i.e., all reservations records that have the matching MEMBER ID).

In a similar manner, the RESERVATIONS table has a **many-to-one** relationship to the MEMBERS table. There may be many records in the RESERVATIONS table related to any **one** record in the MEMBERS table.



The Purpose of the Query: A Report

In this example, we want our query to generate a report that displays:

- The members' last names listed in alphabetical order.
- The date of each member's most recent reservation.
- The total amount of each member's annual membership fees.

The members' names and annual fee data are stored in the MEMBERS table; the reservation date information is stored in the RESERVATIONS table. To generate the report, we have to create a query that tells DataEase to combine information from these two tables and display the output as a report. The illustration below shows a sample of the report output we want (although it hasn't yet been alphabetized).

LAST NAME	highest of RESERVATION DATE	TOTAL DUE
Perrault	10/11/94	215.00
Christano	07/13/94	280.00
Williams	02/15/94	105.00
Riggs	11/03/94	100.00
Shannon	12/26/94	115.00
Johannsson	11/05/94	105.00
LaCoste	08/17/94	140.00
Jones	12/22/94	175.00
Bennington	11/14/94	135.00
Edwards	11/04/94	100.00
Young	02/14/95	105.00
Forrest	06/24/95	175.00

Telling DataEase Which Records to Select

As we create the query, we're going to add selection criteria to limit the report to only those members who pay more than \$90 per year in annual dues. In a query, the selection criteria tell DataEase which records to select for processing. We want DataEase to select from the MEMBERS table only those records with a value greater than 90 in the TOTAL DUE field (the field that stores annual fee data).

In creating any query, we must tell DataEase at least three things:

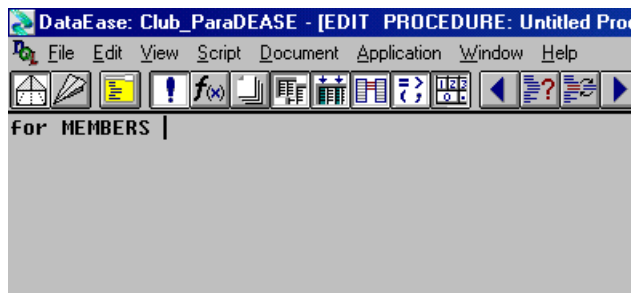
- What table (or tables) contain the required data.
- Which records to process in each table.
- What action to perform on the selected records (in this example, we want to list the information from those records in our report).

MEMBER	LAST NAME	FIRST NAME	MIDI	TOTAL DUE
00001	Birnbaum	Roger	I	65.00
00002	Perrault	George	R	215.00
00003	Christino	Laura	M	280.00
00004	Williams	Laura	R	105.00
00005	Riggs	Cynthia	T	100.00
00006	Shannon	Patrick	M	115.00
00007	Johannsson	David	S	105.00
00008	LaCoste	Fred	M	140.00
00009	Hagerty	Sara	L	65.00
00010	Cohen	Samuel	T	70.00
00011	Jones	Anita	L	175.00
00012	Bennington	William	L	135.00
00013	Sullivan	Hope	L	70.00
00014	Davis	Jim		35.00
00015	Edwards	Kenneth	J	100.00

Creating the Script

See **DQL I** for information on creating a new procedure document.

Before you can write a script, you must create a DQL Procedure document. Choose **File>>New>>Procedure**. When DataEase displays the New Document dialog, choose a Style Sheet if desired, then click OK to close the dialog and display the DQL Script Editor.



The for Command

The **for** command is frequently the first word in a query. **for** is used in conjunction with a table name to tell DataEase from which table to gather information.

The **for** command tells DataEase to select records in a particular table and perform a group of actions on each of those records. After all the actions are performed for the first record, the next record is read, and all the actions are performed again. DataEase repeats this "loop" until all selected records in the table have been processed.

The **Primary table** is the first table specified in a query. Usually, it is the table that holds the key data you want to view or manipulate. MEMBERS is the Primary table in this query because this table contains the members' names and annual fee data. You can also access data in a Secondary table (a table related to the Primary table) in a query, as you'll see later in this example.

Specifying Selection Criteria in a Query

DataEase needs to know if you want it to use all the records in the Primary table, or to select only some of them based on certain selection criteria. We only want to see some of the records-the records for members who have an annual membership fee greater than \$90.00.

You tell DataEase that you want to include selection criteria by inserting the keyword **with** in the query. Press the Enter key to begin a new line, then enter **with** into the query either by double-clicking the command in the Commands pick list or by typing it manually. The query appears as shown below, indicating that you want to select only some records from the MEMBERS table.

```
for MEMBERS
with
```

Specifically, we want DataEase to process records for only those members whose annual fees (stored in the TOTAL DUE field) are greater than \$90. The following DQL statement specifies this criterion:

```
TOTALDUE  > 90
```

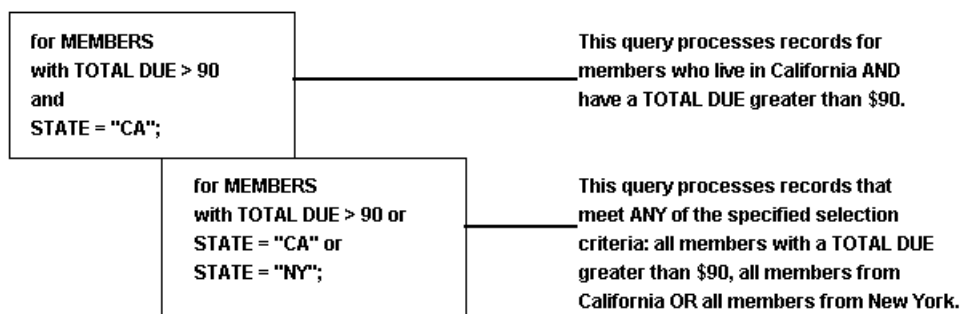
To insert this selection criterion into the query, you can type it manually, or double-click TOTAL DUE in the Columns pick list, select the Comparison Operator, >, from the Operators picklist, and the value, 90, via the keypad

See **DQL 8** for more information on using the and and or operators.

The query reads:

```
for MEMBERS
with TOTALDUE  > 90
```

Although this query contains only one selection criterion, DataEase also lets you select records based on multiple selection criteria. For example, if you want to process only the records of members from certain states with a TOTAL DUE of more than \$90, you can use the and and or operators to join more than one selection criterion, as shown below:



However, for this query we want to select records only on the basis of the value in the TOTAL DUE field. To tell DataEase that there are no additional selection criteria, end the statement with a semicolon. The semicolon is required to mark the end of the selection criteria used to select records from the primary table.

The query reads:

```
for MEMBERS
with TOTALDUE  > 90 ;
```

Note: When you enter a currency value into a script, do not include a dollar sign or commas. It is not necessary to enter a decimal point unless you are specifying a decimal value.

Using the list records Command

See **DQL 8** for more information on comparison operators.

Now that you've specified which records to process, DataEase needs to know what to do with those records. Because we want to list data from the selected records, enter the list records command into the query. Press Enter to begin a new line, then double-click list records in the Commands pick list. The query reads as follows:

```
for MEMBERS
with TOTALDUE > 90 ;
    list records
```

For each selected record, the list records command tells DataEase to list the items you're about to specify in the next part of the query. The most common type of list item is the name of a field (data column).

Note: Although DataEase does not require it, you may want to make your scripts easy to read by indenting the list records command and the list items that follow the command. It is not required to start each new statement on a separate line other than for the sake of readability.

Sorting and Grouping Data in a Query

Because we want to list members alphabetically by name, the first item we want in the procedure output is the data from the LASTNAME column. Press Enter to start a new line, then double-click LASTNAME in the Columns pick list. DataEase inserts the column name into the script, as shown:

```
for MEMBERS
with TOTALDUE > 90;
    list records
        LASTNAME
```

Sorting and Grouping Operators

If you run the procedure using the script as it appears above, DataEase lists the members' last names in the order in which they were entered into the database. DQL offers four operators that let you specify a more useful order for your list of records. Each option appears in the Operators pick list and is briefly explained below:

- in order sorts the records in ascending alphabetical order (putting Adams ahead of Beecher).
- in reverse sorts in descending alphabetical order (putting Zimmerman before Young).
- in groups sorts the records in ascending order into groups that have the same value in the specified column (e.g., all members who live in Alabama could be listed together as a group with their names arranged in alphabetical order, followed by all members from Alaska, and

so on). You can generate statistical totals for each group, such as the total number of members living in each state.

- in groups with group-totals is included for compatibility with previous versions of DataEase. Both this option and "in groups" can be used to generate group statistics.

For our alphabetized report, select in order by double-clicking it in the Operators pick list or by typing it into the query after LAST NAME. End the line with a semicolon, as shown:

```
LASTNAME in order ;
```

Note: The semicolon, which is used to end a statement, is a DQL syntax requirement. A semicolon must appear in every for statement (after the table name, or if selection criteria are used, after the last selection criterion) unless the for statement is nested inside another for statement. You must also insert a semicolon after each data item specified in the list records section of the query, except the last item.

Displaying Data from a Related Form

See **DQL 8** for more information on the sorting and grouping operators.

The query so far reads:

```
for MEMBERS
with TOTALDUE    > 90 ;
    list records
        LASTNAME in order ;
```

Relational Operators

We want DataEase to list the date of each member's most recent reservation. However, this information does not appear in the Columns list because reservation dates are not stored in the MEMBERS table. Instead, the DATE field is stored in the RESERVATIONS table.

Because a pre-defined relationship exists between the RESERVATIONS table and the Primary table (MEMBERS), we can use a relational operator to access any information we need from the RESERVATIONS table. DQL provides seven relational operators (any, all, count of, highest of, lowest of, mean of, and sum of) that let you retrieve information from a table related to the Primary table.

To tell DataEase to find the most recently dated reservation for each member, enter the highest of relational statistical operator into the script before specifying the column name. Press Enter to start a new line, then double-click highest of in the Operators pick list or type it directly into the script:

```
for MEMBERS
with TOTALDUE    > 90 ;
    list records
        LASTNAME in order ;
        highest of
```

This tells DataEase to read all the records (related to the current record being processed) in the related table and report the highest value in the column among those records. After the relational operator highest of, we must specify the name of the relationship followed by the name of the column whose value we want to include.

Specify the RESERVATIONS table name by double-clicking it in the Tables list. DataEase enters the table name into the script and updates the Columns list to display the names of columns in the RESERVATIONS table.

Tables		Columns	
ACTIVITIES		MEMBERS ID	
CLUB ACTIVITIES		FIRST NAME	
CLUB ROOMS		MIDDLE INITIAL	
CLUBS		LAST NAME	
DEPARTURE ZONES		STREET	
EMPLOYEES		CITY	
MEMBERS		STATE	When you highlight in the Tables list.....
PRICES 4.5		ZIP CODE	
RESERVATION DETAIL		TELEPHONE	
RESERVATIONS		PAYMENT METHOD	
Sample Form 1		CARD NO.DataEase updates the Columns list to display the names of that table's columns
Sample Form 3		EXPIRATION DATE	
Task_Activities		ADULT FEES	
Temp Form 70		CHILD FEES	
		TOTAL DUE	
		FAVORITE ACTIVITY	

Specify the DATE column name by double-clicking it in the Columns list. DataEase enters it into the script. End the statement with a semicolon. The script now reads as follows:

```
for MEMBERS
with TOTALDUE > 90 ;
list records
LAST NAME in order ;
highest of RESERVATIONS DATE ;
```

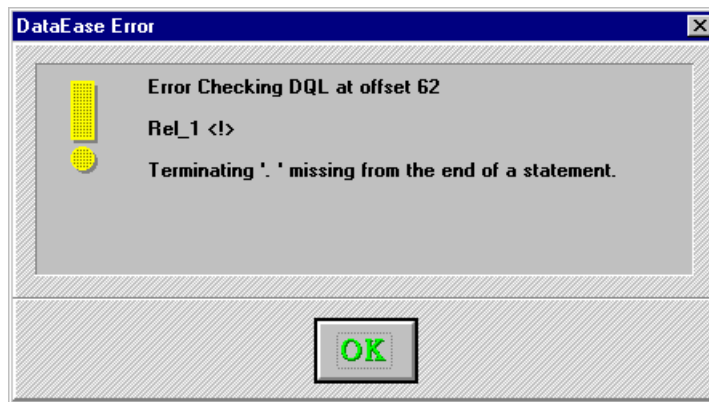
We want this procedure to list one more item: the membership fee paid by each member. This information is stored in the TOTAL DUE column in the MEMBERS table, so highlight MEMBERS in the Tables pick list. DataEase displays that table's column names in the Columns list.

Double-click TOTAL DUE in the Columns list to enter this item into the script, as shown:

```
for MEMBERS
with TOTALDUE > 90 ;
list records
LAST NAME in order ;
highest of RESERVATIONS DATE ;
TOTAL DUE
```

Checking a Script for Errors

The final step in creating a script is to check it for missing punctuation, incorrect table or column names, and other syntax errors. Choose **Script>>Check DQL**. DataEase scans the script and finds that it is not quite complete. A message similar to the one shown below appears:



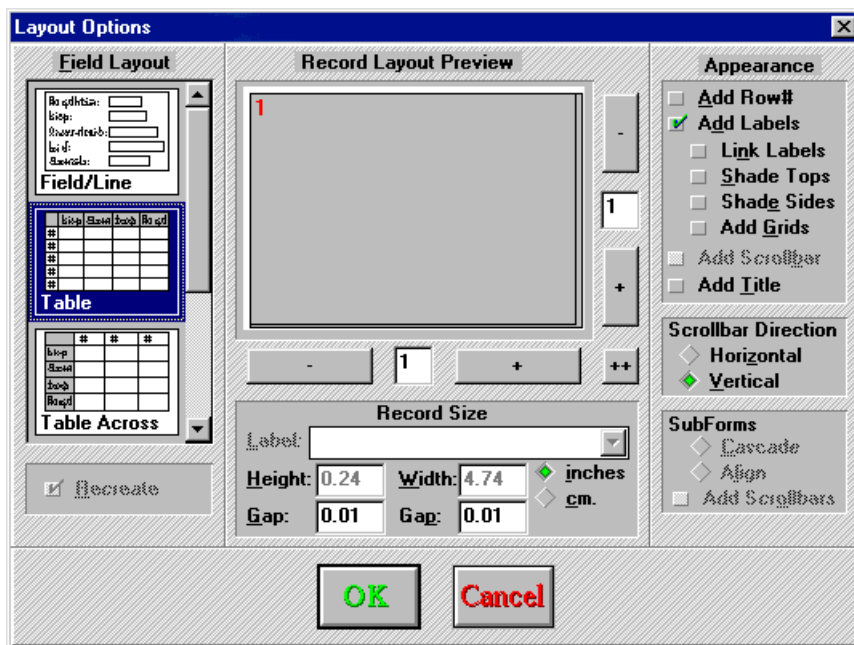
Click OK to close the message box. DataEase places the cursor at the location of the error and lets you add the missing period. The completed script reads:

```
for MEMBERS
with TOTALDUE  > 90 ;
    list records
        LASTNAME in order ;
        highest of RESERVATIONS DATE ;
        TOTALDUE .
```

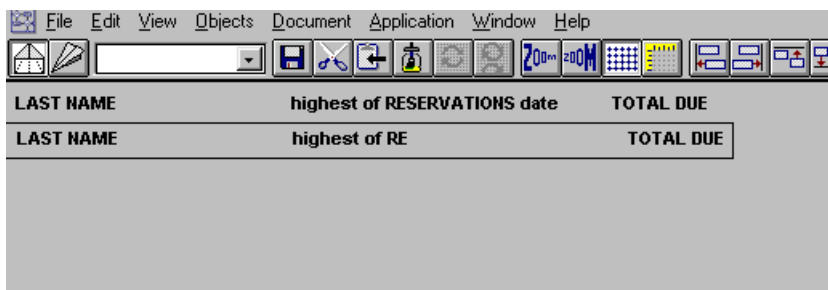
You cannot run a procedure until the script passes a final syntax check, so choose **Script>>Check DQL** once again.

Creating a Procedure Layout

After you successfully complete the Check DQL step, DataEase normally displays the Layout Options dialog, which lets you specify an initial procedure layout and select other options.



Click OK to accept the default row and column (Table) format. DataEase closes the dialog and generates the layout shown below.



See **DQL 8** for more information on relational and relational statistical operators.

Modifying the layout for the output generated by a DQL Procedure is done just like modifying a form or report layout. You can create new objects, including images and buttons, modify the size and appearance of existing objects, and apply Styles to give your outputs a consistent look.

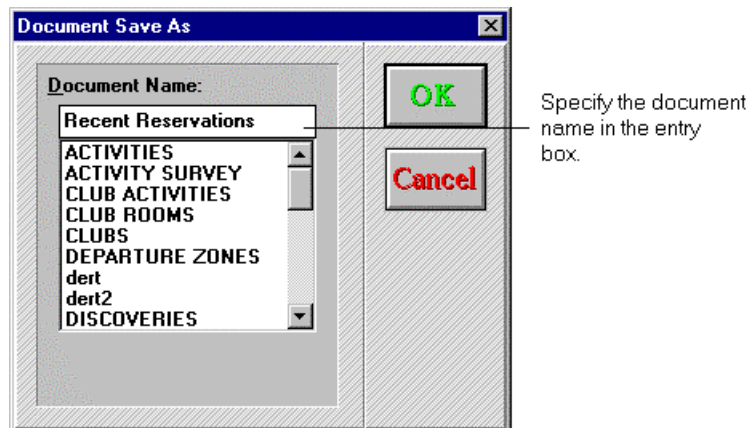
Before running the procedure, replace the automatically generated column headings with those illustrated in the figure below. To replace a column heading, delete an existing column heading, click on the Text tool on the Object Palette, then click on the layout where you want to create the new column heading. Enter the desired text, then set its style by choosing a style from the Styles list box on the Toolbar. (Or simply edit the column heading text object).

The screenshot shows the DataEase software interface with the title bar 'DataEase: Club_ParaDEASE - [DESIGN PROCEDURE : Untitled Procedure 1]'. The menu bar includes File, Edit, View, Objects, Document, Application, Window, and Help. The toolbar contains various icons for design and editing, including a 'Table Grid Form' dropdown. The main workspace displays a table design for 'Recent Reservations'. The table has three columns: 'Member Name', 'Latest Reservation', and 'Membership Fee'. The data row contains 'LAST NAME', 'highest of RE', and 'TOTAL DUE'.

Member Name	Latest Reservation	Membership Fee
LAST NAME	highest of RE	TOTAL DUE

Saving and Running a Procedure

To save the procedure, choose **File>>Save**. When DataEase displays the Document Save As dialog, type RECENT RESERVATIONS in the entry box, then click OK. DataEase saves the document.



To run the procedure, click on the User View button. The figure below shows how the procedure output might look when DataEase displays the output on the screen.

RECENT RESERVATIONS		
Member name	Latest	Membership
	Reservation	Fee
Albert	11/27/94	120.00
Anders	02/27/95	120.00
Anderson	02/14/95	115.00
Archer	08/18/95	155.00
Baldwin	10/11/95	100.00
Bennington	11/04/94	135.00
Bickford	03/14/95	135.00
Boruslewicz	03/25/95	100.00
Bouchard	01/14/95	100.00

As you see, the selected members (those whose annual fee is greater than \$90) are listed in alphabetical order along with their annual membership fees and the date of their last reservations.

Chapter 4 : Control Procedures

Using DQL to Manage Your Application

In addition to generating simple reports like the list records example described in the preceding chapter, the DataEase Query Language can also be used to link documents and database processing procedures together to manage an application. By combining DQL control structures and processing commands, you can easily navigate from one document to another and automatically add, process, manipulate, and delete data from multiple database tables without manually keying the information into each and every record.

Processing Commands	Procedural Commands	Control Commands
list records modify records delete records enter a record for lock unlock query selection input using	output message if else end while break exit case value others define assign ; (semicolon)	run procedure call menu call program record entry import reorganize db status backup db restore db lock db unlock db install application application status lock

Types of DQL Commands

To control all aspects of an application, the DQL is divided into three groups of commands (see list above):

Processing Commands are used to add, modify, delete, or display the data in the application.

Control Commands are used to manage access to the documents in the application.

Procedural Commands are used to organize and control the flow of actions in a DQL Procedure. These commands can be used to conditionally invoke Processing and Control commands.

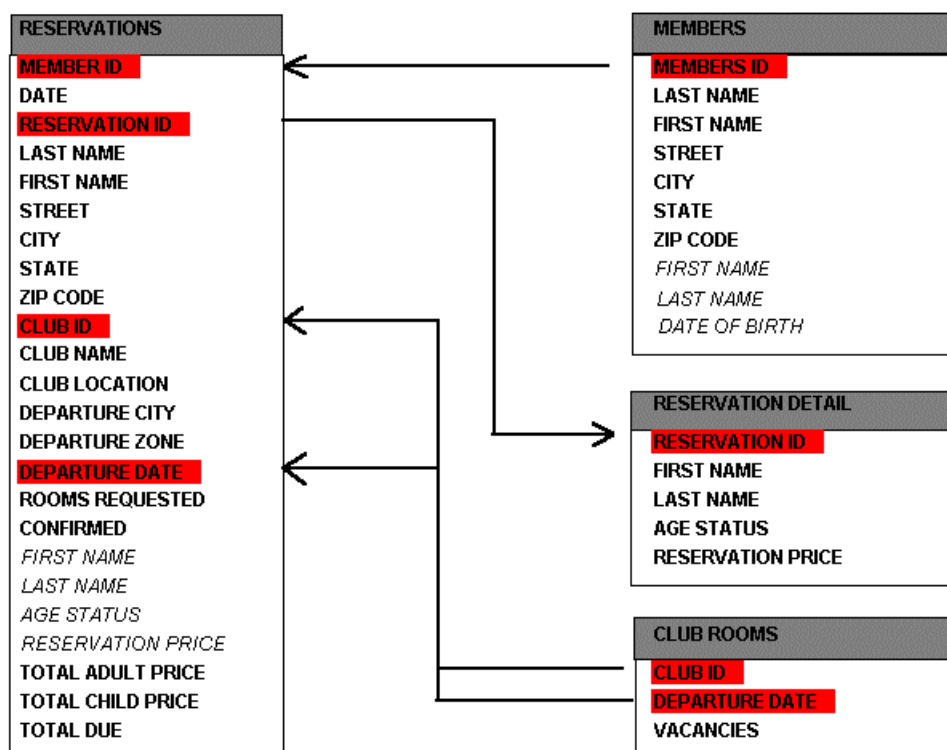
Note: In the character-based version of DataEase, DQL processing is divided into two different types of procedures: Control (program control) and Processing (database manipulation) procedures. Although DataEase does not require you to divide your scripts into separate Control and Processing procedures, doing this makes your scripts easier to understand and enhances concurrent access to data.

The automated **Club ParaDEASE** reservations system discussed in this chapter has been divided into three Processing procedures and a Control procedure to facilitate understanding, but all the commands could be combined in a single script. Other benefits gained by dividing the **Club ParaDEASE** reservations system into several DQL Procedures are discussed at appropriate places throughout the chapter.

This chapter is designed to demonstrate every element in the DataEase Query Language in a realistic context. The chapter shows you how to use many basic programming techniques such as processing loops, variables, and data manipulation functions to manage a complex business application.

The examples in this chapter show how to combine DQL Processing, Control, and Procedural commands, into a powerful procedure that is used to control the entire **Club ParaDEASE** reservation system, including illustrations on how to maintain an inventory of available rooms at each club, calculate discounts, and generate invoices. Although your application may be very different, the manner in which you use individual commands is similar regardless of the type of business you are planning to automate.

The examples in this chapter use the set of tables shown below. The RESERVATION DETAIL and CLUB ROOMS forms are standard, single-level forms. The RESERVATIONS form is a Multiform; the fields shown in italic are actually a view of the RESERVATION DETAIL Subform. The MEMBERS table is also a Multiform; the fields shown in italic are a view of the FAMILY MEMBERS Subform. Tables that have a predefined relationship are indicated by the connecting lines. Relationship Match fields are shown in bold.



Procedure 1: INPUT RESERVATIONS

The first of three procedures that we'll combine into a Control procedure manages how travel reservations are entered into the **Club ParaDEASE** application. Before we actually save a reservation record, however, we need to ensure that there are adequate vacancies at the requested club on the specified date. The INPUT RESERVATIONS procedure uses a Data-entry form to let us check on room availability. Based on the results of our Data-entry inquiry, we can tell DataEase to enter the reservation record or display a warning if there aren't enough accommodations at the destination club.

As shown below, the INPUT RESERVATIONS Data-entry form contains four fields: CLUB ID, CLUB NAME, DEPARTURE DATE, and ROOMS REQUIRED. The CLUB ID and DEPARTURE DATE fields are used to specify the date and destination for the reservation. The ROOMS REQUIRED field is used to indicate how many rooms the caller wants to reserve. The CLUB NAME field contains a Derivation formula that looks up the name of the club.

The Data-entry form also contains two buttons. The OK button invokes the Save Record action, which causes DataEase to begin processing the script. The Cancel button invokes the Close Document action, which tells DataEase to close the INPUT RESERVATIONS procedure.

The reservation clerk enters a club ID, Departure Date, and the number of rooms needed in the Data-entry form.
...then clicks OK to continue, or Cancel to abandon the procedure.

When the reservation clerk enters the CLUB ID and DEPARTURE DATE in the Data-entry form, DataEase uses an ad hoc relationship to find the matching record in the CLUB ROOMS table. The matching record tells DataEase how many rooms are available at the destination club on the requested date.

In addition to checking room availability at the destination club, the INPUT RESERVATIONS procedure posts new member information from the MEMBERS table. This information is entered into two related tables: the appropriate member information is posted into the RESERVATIONS table (using the data entered into the Data-entry form) and the number of vacancies is updated in the CLUB ROOMS table.

Script for the INPUT RESERVATIONS Procedure

The complete script for the INPUT RESERVATIONS procedure is shown below:

```

if data-entry ROOMS REQUIRED > any CLUB ROOMS with
(CLUB ID = data-entry CLUB ID and DEPARTURE DATE =
data-entry DEPARTURE DATE) VACANCIES then

    message "Not enough rooms are available. |
Please choose another club or date." window.
else
    while current status not = 1 do
    input using MEMBERS into "TEMPMEM" .
    case (current status)
        value 1 :
            exit
        value 2 :
            modify records in CLUB ROOMS with
            (CLUB ID = data-entry CLUB ID and
            DEPARTURE DATE = data-entry DEPARTURE DATE)
            VACANCIES:= VACANCIES- data-entry ROOMSREQUIRED.

            enter a record in RESERVATIONS
            copy all from TEMPMEM ;
            CLUB ID:= data-entry CLUB ID;
            DEPARTURE DATE := data-entry DEPARTURE DATE ;
            ROOMSREQUIRED := data-entry ROOMSREQUIRED .

            record entry RESERVATIONS .
        others
            message "You are not authorized to |
            modify or delete records " window .
    end
    end
end
end

```

Explanation of the Script

Now, let's examine this script one statement at a time:

```

if data-entry ROOMS REQUIRED > any CLUB ROOMS with
(CLUB ID = data-entry CLUB ID and DEPARTURE DATE

```

```
= data-entry DEPARTURE DATE) VACANCIES then
```

The script begins with an if Command. The if Command tells DataEase to execute one action (the action specified after the keyword then) if the statement that follows the if command is true, and a different action (the action specified after the keyword else) if the statement is false.

In our example, the if command tells DataEase to check the number of rooms requested in the Data-entry form against the number in the VACANCIES field in the matching record in the CLUB ROOMS table. If the number in the VACANCIES field in the CLUB ROOMS table is greater than the number in the ROOMS REQUESTED field on the Data-entry form, DataEase processes the reservation. Otherwise, DataEase displays the message:

```
Not enough rooms are available.
Please choose another club or date.
```

In order to check the VACANCIES field in the matching record in the CLUB ROOMS table, DataEase needs to know how the record in the Data-entry form is related to the record(s) in the CLUB ROOMS table. The script tells DataEase how to find the matching record in the CLUB ROOMS table by specifying an ad hoc relationship:

```
(CLUB ID = data-entry CLUB ID and DEPARTURE DATE = data-entry
DEPARTURE DATE)
```

See **DG 8** for more information on DataEase relationships.

An ad hoc relationship is a DQL expression that tells DataEase how the records in two different forms are related when no permanent relationship has been defined in the DataEase Relationships form. The ad hoc relationship expressed above tells DataEase to find the record in the CLUB ROOMS table that has the same CLUB ID and DEPARTURE DATE as the values entered in the Data-entry form.

For all purposes, an ad hoc relationship is similar to a relationship stored in the Relationships form, except an ad hoc relationship is only valid while the procedure is in progress (the relationship is discarded when the procedure completes execution).

The next section of the INPUT RESERVATIONS script (below) uses two Procedural commands, while and case, and the input using Processing command to tell DataEase what to do if there are sufficient vacancies at the destination club on the date requested in the Data-entry form.

```
while current status not = 1 do
input using MEMBERS into "TEMPMEM" .
    case ( current status )
    value 1 :
        exit .
    value 2 :
```

The next section of the script tells DataEase to modify the value stored in the VACANCIES field in the matching record in the CLUB ROOMS table by subtracting the number of rooms requested on the Data-entry form.


```

modify records in CLUB ROOMS with
(CLUB ID    = data-entry CLUB ID and
DEPARTURE DATE = data-entry DEPARTURE DATE)
VACANCIES:   = VACANCIES-data-entry ROOMSREQUIRED.

```

The last section of the script tells DataEase to enter a new record in the RESERVATIONS form detailing all the required personal and family information, destination club, reservation date, etc.

```

enter a record in RESERVATIONS
copy all from TEMPMEM ;
CLUB ID:= data-entry CLUB ID;
DEPARTURE DATE := data-entry DEPARTURE DATE ;
ROOMSREQUIRED := data-entry ROOMSREQUIRED .
record entry RESERVATIONS .

```

Finally, if the caller decides to cancel at any point, we want DataEase to discard all the data entered by the reservations clerk. Therefore, we need a way to gather information from the caller and then tell DataEase how to process it.

The DQL case and input using commands can be used together to gather information during a DQL Procedure and then process the information in several different ways. Similar to the if command, the DQL case command tells DataEase to perform one of a number of possible actions based on which value is stored in the current status variable. The current status variable is a special DQL system variable that tells DataEase which menu command or keystroke the user executes after entering data as shown below.

Current Status Values

When the user finishes entering data in the temporary input form and makes this menu selection...	...DataEase stores this value in the current status variable
File>>Close	1
File>>Save As New Record	2
File>>Save	3
File>>Delete	4

Since much of the information required in the MEMBERS form must also be entered into the RESERVATIONS form, we only want to input this information once and then let DataEase automatically copy it into the appropriate database tables. The DQL input using command tells DataEase to display a database form as a temporary form that is used to gather the information which will later be posted into the appropriate database tables.

The while command sets up a processing loop. In our example, the while command tells DataEase to continue processing each new reservation in the manner described above as long as the value in the current status variable does not equal one. When current status equals one, DataEase exits the while loop and resumes processing the remainder of the script.

The diagram below shows the flow of information from the TEMPMEM form, depending on which processing key is used.

Club ParaDEASE
The Antidote to Documentation

MEMBERS Previous Next

Member ID: 00202

New
Save
Delete
Close
Clubs
Reservations

Last Name: First Name: Initial:
 Street:
 City: Zip Code:
 Expiration Date: Telephone: () -

Favorite Activity: Method of Payment:
 Card No.

Family Members

First Name	Last Name	Date of Birth

Membership Fees
 Adult Annual Fee \$35.00
 Child Annual Fee \$15.00
 Adult Fees: 0
 Child Fees: 0
 Total Due: 0.00

From TEMPMEM when you press....

F2, DataEase enters a new record in RESERVATIONS and updates the number of VACANCIES in CLUB ROOMS.

F7 or F8, DataEase displays a message that deletions and updates are not allowed.

Now that you have an overview, let's examine the remainder of the INPUT RESERVATION script introduced earlier in a little more detail.

The first line of the script reads:

```
while current status not = 1 do
```

This line tells DataEase to continually loop through and process each of the statements that follow until the current status variable indicates that the user chose **File>>Close** to close the input form.

The next line in the script uses the input using command. The input using command provides all the facilities of record entry under the control of the DataEase Query Language. As each record is entered into the TEMPMEM form, the record is held in memory until a processing action is invoked. When an action is initiated, the script takes control and processes the current record according to the instructions in the script:

```
input using MEMBERS into "TEMPMEM" .
```

This line tells DataEase to display the TEMPMEM form. In our example, we'll copy member information to the RESERVATIONS form and update CLUB ROOMS data.

The next line tells DataEase to check the value in the current status variable:

```
case ( current status )
```

The case command tells DataEase to compare an expression (e.g., current status) to a series of values and execute a different action based on which comparison is true.

The next two lines tell DataEase what to do if the value of the current status variable is "1":

```
value 1 :
    exit .
```

Remember that the value in the current status variable is set to one when the user invokes a close document action. The exit command tells DataEase to stop processing the script. When the user chooses **File>>Close**, DataEase terminates all script processing.

The next line tells DataEase to check the current status variable for a value of "2" (the action invoked when you click a Save button or icon or choose **Edit>>Save As New Record**). If the value of the current status variable is "2", the following section of the script is executed:

```
value 2 :
  modify records in CLUB ROOMS with (CLUB ID =
  data-entry CLUB ID and DEPARTURE DATE =
  data-entry DEPARTURE DATE)
  VACANCIES:= VACANCIES-data-entry ROOMSREQUIRED .
  enter a record in RESERVATIONS
  copy all from TEMPMEM ;
  CLUB ID:= data-entry CLUB ID;
  DEPARTURE DATE := data-entry DEPARTURE DATE ;
  ROOMSREQUIRED := data-entry ROOMSREQUIRED .
  record entry RESERVATIONS .
```

This section of the script locates the matching record in CLUB ROOMS and decreases the number of available rooms.

This section enters all the information from the temporary form as a new record in RESERVATIONS.

The first line tells DataEase to save a new record in the RESERVATIONS table. The subsequent lines tell DataEase which information to enter as the new record. The enter a record command can be used with copy all from or to specify individual fields. In our example, we do a combination of both. Copy all from TEMPMEM tells DataEase to copy all field values in TEMPMEM to the corresponding fields (e.g., those that have the identical names) in RESERVATIONS. Any fields whose name differs must be dealt with on an individual basis as demonstrated in the last three lines of the above example. The enter a record command is terminated by a period.

Since all the information needed to enter a new reservation isn't included in the RESERVATIONS form, the next line of the script provides direct record entry access to RESERVATIONS.

```
record entry reservations .
```

The record entry command lets you perform traditional record entry operations from within a procedure. In the RESERVATIONS form we can enter additional information (e.g., names of family members included in the reservation) not included in the MEMBERS form. The record entry command is terminated by a period. This is the last command executed when the user invokes a save action.

The next section tells DataEase what to do if the user presses any of the remaining processing keys:

```
others
    message    "You are not authorized to modify or
delete records.  "  window .
end
```

The others command encompasses any value that is not specifically tested with a value command. In our example, that means any value other than 1 or 2. If the user invokes a modify or delete action the value in the current status variable is set to 3 (modify) or 4 (delete). For the sake of simplicity, we are disallowing those actions. If the user tries to modify or delete a record, DataEase displays the message shown above. The period terminates the message command and the end terminates the case command since there are no other values to test.

Note: If we hadn't included the others command or additional value commands to handle the current status values of 3 and 4, DataEase would simply ignore any attempt the user made to modify or delete a record.

The only commands remaining are two end commands:

```
end
end
```

To satisfy the DQL syntax requirements, a script must contain one end command for each for, if, while and/or case command.

The complete script for the INPUT RESERVATIONS procedure reads:

```
if data-entry ROOMS REQUIRED > any CLUB ROOMS with
(CLUB ID = data-entry CLUB ID and DEPARTURE DATE =
data-entry DEPARTURE DATE) VACANCIES then
    message    "Not enough rooms are available. |
Please choose another club or date."  window.
else
    while current status not = 1 do
input using MEMBERS into    "TEMPMEM" .
    case ( current status )
        value 1 :
            exit
```

```

value 2 :
    modify records in CLUB ROOMS with
        (CLUB ID = data-entry CLUB ID and
        DEPARTURE DATE = data-entry DEPARTURE DATE)
    VACANCIES:= VACANCIES- data-entry ROOMSREQUIRED.
    enter a record in RESERVATIONS
    copy all from TEMPMEM ;
    CLUB ID:= data-entry CLUB ID;
    DEPARTURE DATE := data-entry DEPARTURE DATE ;
    ROOMSREQUIRED := data-entry ROOMSREQUIRED .
    record entry RESERVATIONS .
others
    message "You are not authorized to |
    modify or delete records " window .
end
end
end
end

```

This script checks the availability of rooms at the club and date requested. If there aren't adequate accommodations, it displays an error message, then redisplay the Data-entry form so the data entry clerk can try another club/date combination. Otherwise, DataEase saves a new record in the RESERVATIONS form and decrements the number of available rooms at the requested club on the specified date.

Saving the Procedure

If you are creating this procedure as you read, before proceeding to the next section, choose **File>>Save As**. DataEase displays the Document Save As dialog.

Enter INPUT RESERVATIONS as the document name and click OK. DataEase saves the procedure on disk and returns you to the script. After the procedure is saved on disk, you can later execute the procedure by any of the following methods:

- Choose **File>>Open**, select the procedure name from the list of documents DataEase displays, then click OK. Open the procedure in User View to execute it or in Designer View to view or modify it.
- If the procedure is already open in Designer View, choose **File>>User View**.

Specify the name of the procedure in a run procedure command within a script or on a custom menu.

Next, we'll create a procedure that determines the discount given to each member based on the number of children and adults included in a reservation.

Procedure 2: CALCULATE DISCOUNTS

The second procedure is based on the fact that **Club ParaDEASE** discounts trip reservations based on the number of children and adults included in each reservation. The object of the procedure is to calculate the following totals:

- Number of children going on the trip.
- Number of adults going on the trip.
- Dollar amount of the trip.
- Dollar amount of the discount applied to the cost of the trip.

After each of these totals is determined, we use a variable (an expression that represents a varying value) to store the calculated result. Variables let us use the values as often as we want within the procedure without repeating the formulas used to derive them.

The script for the CALCULATE DISCOUNTS procedure must tell DataEase to do all of the following:

- Create **five** separate variables to store: each member's reservation total, the number of children, the number of adults, each member's total discount amount, and the RESERVATION ID.
- Set up discounts based on the number of children and adults with the discounts expressed as a dollar amount to be subtracted from the total reservation price.
- Process the related RESERVATION DETAIL records, calculating the sum of the RESERVATION PRICE for all family members included in the reservation.
- Modify the RESERVATIONS records, entering each reservation's final price in the TOTAL DUE field.

Script for the CALCULATE DISCOUNTS Procedure

The full script for the CALCULATE DISCOUNTS procedure reads as follows:

```
define temp    "KIDS"    Number .
define temp    "ADULTS"  Number .
define global  "DISCOUNT" Number .
define global  "RESTOTAL" Number .
define global  "RESERVATION#" Number .
for RESERVATIONS
with CONFIRMED = NO ;
    assign temp KIDS := count of RESERVATION DETAIL
    named "KIDCOUNT" with (AGE STATUS = "child" ).
    assign temp ADULTS := count of RESERVATION
    DETAIL named "ADULTCOUNT" with (AGE STATUS =
    "adult" ).
    assign global RESTOTAL := sum of RESERVATION
    DETAIL RESERVATION PRICE   
    assign global RESERVATION # := RESERVATION ID .
    case (temp KIDS)
```

```

        value = 1 :
        assign global DISCOUNT := global RESTOTAL * 0.05 .
        value = 2 :
        assign global DISCOUNT := global RESTOTAL * 0.10 .
        value >= 3 :
        assign global DISCOUNT := global RESTOTAL * 0.15 .
    end

    if temp ADULTS > 2 then
        assign global DISCOUNT := global DISCOUNT + ((
global
        RESTOTAL - global
        DISCOUNT) * 0.05) .
    end
    modify records
        TOTAL DUE := global RESTOTAL- (global RESTOTAL *
        global DISCOUNT ).
    run procedure RESERVATION INVOICE .
end

```

Explanation of the Script

Notice first that this script does not start with a for command. When using variables in a script, you should define the variables at the beginning. Defining variables at the start of a procedure makes it easy to find (and change) the variables if necessary, and it also makes the script easier to follow.

Now, let's look at this script one statement at a time.

The first line in the script creates and names a temporary variable to hold the total number of children taking the trip:

```
define temp "KIDS" Number ; .
```

This first line has five elements:

- The DQL define command. This tells DataEase to create a variable.
- The keyword temp. This tells DataEase the variable is a temporary variable, not a global variable.
- The name of the variable (KIDS). The name of a variable must be enclosed in quotes when it is first specified in a script.
- The type of the variable (Number). A variable can be a number, a text expression, or any other DataEase field type except Choice, Dollar, Sequenced ID, or Yes/No.
- A period. The define command must always end with a period.

The first five lines of the script are similar in purpose:

```
define temp "KIDS" Number .
define temp "ADULTS" Number .
```

```

define global  "DISCOUNT"  Number .
define global  "RESTOTAL"    Number .
define global  "RESERVATION#" Number .

```

These statements create two temporary variables and three global variables. The temporary variables include one to store the number of children and one to store the number of adults. Global variables are used to store the total reservation price, the discount amount applied to each reservation total, and the RESERVATION ID. A global variable can be passed from one procedure to another, avoiding the need to recalculate the value each time you use it. Since the RESERVATIONS form doesn't store the discount amount or the pre-discount total, we'll store both values in global variables so they can be passed along and used in the next procedure. The RESERVATION ID is also stored in a global variable so the ID can be passed to the next procedure to identify which record to process.

The next part of the script tells DataEase the name of the primary form and which records to select:

```

for RESERVATIONS
with CONFIRMED    =    NO ;

```

The group of records we want DataEase to select are the **unconfirmed** RESERVATIONS records. Reservations are not considered **confirmed** until the control procedure described at the end of this chapter is complete. This for command processes each record that stores the value NO in the CONFIRMED field.

The next part of the script tells DataEase how to calculate the number of children taking the trip (the value we want to assign to the "KIDS" variable):

```

assign temp KIDS := count of RESERVATION DETAIL
named  "KIDCOUNT"  with (AGE STATUS  =    "child" ) .

```

The above statement uses the DQL assign command:

```

assign temp KIDS :=

```

This statement tells DataEase to assign KIDS a value. The assignment operator symbol (:=) follows the name of the variable. This operator serves the same purpose with a temporary variable that it did in the previous procedure following a modify records command. The value of the variable is determined by the expression on the right of the symbol. The remainder of this line reads:

```

count of RESERVATION DETAIL named  "KIDCOUNT"
with (AGE STATUS  =    "child" );

```

This part of the assign statement tells DataEase what value to assign to the KIDS variable - the number of related records that store the value child in the AGE STATUS field.

DataEase lets you add criteria to a predefined relationship (e.g., with AGE STATUS = "child"). This is another use of an ad hoc relationship. When you create an ad hoc relationship in this manner, any additional criteria you specify applies to the relationship for the remainder of the

script. But before we stipulate the selection criterion (AGE STATUS = "child"), we use the named operator to rename the relationship.

In addition to creating ad hoc relationships, you can use the named operator to provide a custom relationship name, much the way you do in the Relationships form. When you rename a relationship in a script, the original predefined relationship remains unchanged. This feature lets you use either relationship throughout the remainder of the script: the original predefined relationship or the modified, renamed relationship.

Note: You can't use the same relationship name to specify two different sets of ad hoc criteria. If you want to specify two or more sets of criteria for the same relationship, you must provide a unique name for each new relationship. The unique name lets DataEase distinguish different groups of records selected from the same table.

Prior to this line in the sample script, the relationship between RESERVATIONS and RESERVATION DETAIL is based on the original Match fields specified in the Relationships form (e.g., RESERVATION ID = RESERVATION ID). After this line in the script, the relationship has changed. The new relationship still links records based on the matching RESERVATION ID but now RESERVATIONS records are related only to those matching RESERVATION DETAIL records that contain the value **child** in the AGE STATUS field. Because we established a unique name for the new relationship (KIDCOUNT), we still have access to the original relationship (RESERVATION DETAIL), which we can use in its original form or create an ad hoc relationship again using different criteria.

The next two lines of the script serve a similar purpose:

```
assign temp ADULTS := count of RESERVATION DETAIL
named "ADULTCOUNT" with (AGE STATUS = "adult" ) .
```

This time the assign statement tells DataEase what value to assign to the ADULTS variable: the number of related records that store the value **adult** in the AGE STATUS field. Because we still want to be able to access the original RESERVATION DETAIL relationship to calculate the total cost of the reservation later in the script, we again rename the relationship (ADULTCOUNT) before we specify the ad hoc criteria.

The next line of the script assigns a value to the global variable that stores the total price of the reservation:

```
assign global RESTOTAL := sum of RESERVATION
DETAIL RESERVATION PRICE .
```

This line tells DataEase to process all the related RESERVATION DETAIL records (not just the related adult or child records) and to sum the value in the RESERVATION PRICE field. Notice that this time we use the original relationship rather than adding any ad hoc criteria because we want DataEase to process all the related records.

The next line of the script assigns a value to the last of the global variables:

```
assign global RESERVATION# := RESERVATION ID .
```

This line tells DataEase to assign the value in the RESERVATION ID field in the current record to the RESERVATION # variable. We will use this variable as the selection criteria in the next procedure to identify the record we want DataEase to process.

Because the assign statements are contained within the for loop, each variable is reassigned each time a new record is processed by the for command.

The whole script up to this point reads as follows:

```
define temp "KIDS" Number .
define temp "ADULTS" Number .
define global "DISCOUNT" Number .
define global "RESTOTAL" Number .
define global "RESERVATION" Number .

for RESERVATIONS
with CONFIRMED = NO ;

    assign temp KIDS := count of RESERVATION DETAIL
    named "KIDCOUNT" with ( AGE STATUS = "child" ) .
    assign temp ADULTS := count of RESERVATION
    DETAIL named "ADULTCOUNT" with ( AGE STATUS =
    "adult" ) .
    assign global RESTOTAL := sum of RESERVATION
    DETAIL RESERVATION PRICE .
    assign global RESERVATION := RESERVATION ID .
```

The next section of the script begins with a case command:

```
case ( temp KIDS)
```

The discount applied to each reservation is determined first by the number of children taking the trip. An additional discount is given for reservations that include more than two adults. This line of the script tells DataEase to check the value currently stored in the KIDS variable.

Remember that each time DataEase cycles through the for loop, the value in each variable is rederived.

The next two lines of the script tell DataEase what to do if one child is included in the reservation:

```
value = 1 :
    assign global DISCOUNT := global RESTOTAL * 0.05 .
```

If the value in the KIDS variable is one, DataEase assigns the DISCOUNT variable a value equal to five percent of the total reservation cost.

The next four lines of the script are similar to the last two:

```
value = 2 :
```

```

        assign global DISCOUNT := global RESTOTAL * 0.10 .
value >= 3 :
        assign global DISCOUNT := global RESTOTAL * 0.15 .
end

```

If the value in the KIDS variable is not 1, DataEase checks the next value statement. If the value is 2, DataEase assigns the DISCOUNT variable a value equal to 10 percent of the reservation total. If the value in the KIDS variable is 3 or more, DataEase assigns the DISCOUNT variable a value equal to 15 percent of the reservation total. The next line of the script contains an end command. This end terminates the case command.

The next three lines of the script tell DataEase to increase the discount amount if more than two adults are included in the reservation:

```

if temp ADULTS > 2 then
        assign global DISCOUNT := global DISCOUNT + ( (
global RESTOTAL - global DISCOUNT ) * 0.05 ) .
end

```

Each reservation total is discounted an additional 5 percent for three or more adults. The if command checks the value in the ADULTS variable. If that value exceeds 2, the value in the DISCOUNT variable is reassigned to figure in the additional 5 percent. The parentheses in the assign statement clarify the sequence of mathematical operations. When parentheses are used, DataEase performs math operations from the inside out. In this example, DataEase subtracts the current value in the DISCOUNT variable from the amount held in the RESTOTAL variable. The result is multiplied by 5 percent and that value is added to the original discount dollar amount.

The end command terminates the if command.

The last section of the query modifies the RESERVATIONS record to enter the discounted reservation cost in the TOTAL DUE field:

```

modify records
        TOTAL DUE := global RESTOTAL - global DISCOUNT.

```

Because this portion of the script is still under the control of the for command, DataEase modifies just the record currently being processed. The TOTAL DUE field is set equal to the total cost of the reservation minus the discount amount DataEase calculated earlier in the script. The modify records processing command is terminated by a period.

The final command tells DataEase to run another procedure:

```

        run procedure RESERVATION INVOICE .
end

```

run procedure is a Control command that tells DataEase to run another procedure that prints an invoice for the current RESERVATIONS record. Because this command is within the for loop, the RESERVATION INVOICE procedure is run once for each record processed by the for command. The RESERVATION INVOICE procedure is explained in the next section of this

chapter. The only remaining command is another end command. This end terminates the for loop.

The completed script for the CALCULATE DISCOUNTS procedure reads as follows:

```

define temp    "KIDS"    Number .
define temp    "ADULTS"  Number .
define global   "DISCOUNT"  Number .
define global   "RESTOTAL"  Number .
define global   "RESERVATION #"  Number .
for RESERVATIONS
with CONFIRMED = NO ;
    assign temp KIDS := count of RESERVATION DETAIL
    named "KIDCOUNT" with ( AGE STATUS = "child" ) .
    assign temp ADULTS := count of RESERVATION
    DETAIL named "ADULTCOUNT" with ( AGE STATUS =
    "adult" ) .
    assign global RESTOTAL := sum of RESERVATION
    DETAIL RESERVATION PRICE .
    assign global RESERVATION # := RESERVATION ID .
        case ( temp KIDS )
            value = 1 :
            assign global DISCOUNT := global RESTOTAL * 0.05 .
            value = 2 :
            assign global DISCOUNT := global RESTOTAL * 0.10 .
            value >= 3 :
            assign global DISCOUNT := global RESTOTAL * 0.15 .
        end
    if temp ADULTS > 2 then
        assign global DISCOUNT := global DISCOUNT + ( (
global
        RESTOTAL - global
        DISCOUNT ) * 0.05 ) .
    end
    modify records
        TOTAL DUE := global RESTOTAL - ( global RESTOTAL *
        global DISCOUNT ) .
    run procedure RESERVATION INVOICE .
end

```

If you are creating this script as you read, choose **File>>Save** to save the procedure on disk. When DataEase asks you to name the procedure, enter CALCULATE DISCOUNTS.

Procedure 3: RESERVATION INVOICES

The RESERVATION INVOICES procedure tells DataEase what information to print on each reservation invoice.

<u>Club Paradease Invoice</u>			
Reservation:	00218	Member ID:	00376
			10/09/99
Member:	Donovan		
	283 Lake Plymouth Drive		
	Pantucket	RI:	02061-000
Destination:	Playa Blanca	Mexico	
Departure Date:	03/11/99	Departure From:	Boston
First Name	Last Name	Age Status	Price
Alexander	Donovan	Adult	1,300
David	Donovan	Child	556
Reservation Total			1,856
Discount			371
Total Due			1,485

The information we want to appear on the **Club ParaDEASE** invoice includes the following items:

- RESERVATION ID.
- MEMBER ID.
- DATE
- MEMBER NAME.
- ADDRESS.
- DESTINATION.
- DEPARTURE information.
- FIRST and LAST NAME of each family member taking the trip.
- AGE STATUS of each family member.
- PRICE for each family member.
- DISCOUNT.

- Total Price for all family members.
- Cost of reservations.

Script for the RESERVATION INVOICES Procedure

To generate a report that can be formatted like the invoice shown on the previous page, the script includes the following lines:

```

define global  "RESTOTAL"  Number .
define global  "DISCOUNT"  Number .
define global  "RESERVATION#"  Number .

for RESERVATIONS
with CONFIRMED  =  NO ; ;
    list records
    RESERVATION ID in groups ;
    MEMBER ID ;
    DATE ;
    FIRST NAME ;
    LAST NAME ;
    STREET ;
    CITY ;
    STATE ;
    ZIP ;
    CLUB NAME ;
    CLUB LOCATION ;
    DEPARTURE DATE ;
    DEPARTURE CITY ;
    all RESERVATION DETAIL FIRST NAME ;
    all RESERVATION DETAIL LAST NAME ;
    all RESERVATION DETAIL AGE STATUS ;
    all RESERVATION DETAIL RESERVATION PRICE ;
    global DISCOUNT ;
    global RESTOTAL ;
    global RESTOTAL - global DISCOUNT .

```

In addition to listing information on the invoice, we also want this procedure to do one last important task: change the value in the CONFIRMED field from No to Yes. This is specified by adding three more lines to the script:

```

    modify records
        CONFIRMED := YES .
end

```

Explanation of the Script

See **DG 6** for information on how to create the layout of a report document.

Now, let's review this script line by line.

```
for RESERVATIONS
```

The for command selects the script's Primary form—the RESERVATIONS form.

```
with CONFIRMED = NO ;
  list records
```

The with operator introduces the record selection criteria: process only records with the value No in the CONFIRMED field. The list records command tells DataEase what items we want to appear in the output. These items (called list items) follow the list records command. In this report there are two types of list items: (1) fields from the Primary and/or Secondary forms, and (2) values that were calculated during the processing of the previous script but do not actually appear on any database forms. Those values were passed to this procedure via global variables.

The next line tells DataEase to group RESERVATION DETAIL records with the same RESERVATION ID together and to arrange the groups in ascending order.

```
RESERVATION ID in groups ,
```

Each of the following lines specifies a field on the RESERVATIONS form. Here, we are telling DataEase to include the value in each of these fields in the report output.

```
MEMBER ID ;
DATE ;
FIRST NAME ;
LAST NAME ;
STREET , ;
CITY ;
STATE ;
ZIP ;
CLUB NAME ;
CLUB LOCATION ;
DEPARTURE DATE ;
DEPARTURE CITY ;
```

The next four lines are used to retrieve information stored in the related RESERVATION DETAIL file.

```

all RESERVATION DETAIL FIRST NAME ;
all RESERVATION DETAIL LAST NAME ;
all RESERVATION DETAIL AGE STATUS ;
all RESERVATION DETAIL RESERVATION PRICE ;

```

Each of these four lines specifies a field on the RESERVATION DETAIL form. The relational operator all tells DataEase to find every RESERVATION DETAIL record that has the same RESERVATION ID as the RESERVATIONS record currently being processed.

The next line lists one of the three global variables:

```

global DISCOUNT ;

```

In the previous procedure, we calculated the discount amount to apply to each reservation based on the number of children and adults included in the reservation. The value DataEase calculated was assigned to a global variable called DISCOUNT. Since the amount of the discount was calculated in the previous procedure, we don't have to calculate the discount again in the current procedure. We simply use a global variable to pass the value to this procedure.

The next line lists another global variable whose value was calculated in the previous procedure:

```

global RESTOTAL ;

```

In the last procedure we told DataEase to sum the PRICE field values in all the related RESERVATION DETAIL records for each RESERVATIONS record processed. That calculated value was then assigned to the RESTOTAL variable. This line tells DataEase to list the value in the RESTOTAL variable in the printed invoice. This information is displayed as the overall Reservation Total on the invoice.

The final line under the list records command calculates the **net total** for the invoice:

```

global RESTOTAL - global DISCOUNT .

```

This line tells DataEase to subtract the value in the global DISCOUNT variable from the amount stored in the global RESTOTAL variable. The result is the net total for each order, taking into account each member's discount as calculated in the previous procedure. This information is displayed as the Total Due on the invoice.

The last three lines of the script tell DataEase to modify each record:

```

      modify records
        CONFIRMED := YES .
    end

```

As each RESERVATIONS record is processed, DataEase changes the value in the CONFIRMED field from No to Yes. This ensures that a reservation that has already been processed will not be processed again.

Procedure 4: PROCESS RESERVATIONS

As we mentioned at the start of this section, a Control procedure can be used to manage almost every element of DataEase, performing tasks such as combining menus, input forms, procedures, conditional actions, and output into fully automated business applications.

A Control procedure can be used to:

- Execute any DataEase menu option from within the procedure. In this respect, Control procedures are similar to chain menus. However, Control procedures let you combine DQL conditional logic with the chain menu processing capabilities.
- Link DQL Procedures together and execute them automatically.
- Pass values among procedures.

Now, we'll demonstrate how **Club ParaDEASE** can link the preceding procedures together to automate the entire process for entering and posting reservations, calculating reservation discounts, and printing reservation invoices. To link the procedures, you must create a new script that combines the two Processing procedures into a Control procedure.

The script for this Control procedure is listed immediately below. Following the script, the purpose of each line in the Control procedure is explained.

Script for the PROCESS RESERVATIONS Control Procedure

The Control procedure script contains only five lines:

```
define global  "DISCOUNT"  Number .
define global  "RESTOTAL"   Number .
define global  "RESERVATION#" Number.

      run procedure  "INPUT RESERVATIONS" .
      run procedure  "CALCULATE DISCOUNTS" .
```

Explanation of the Control Procedure

The first three lines of the Control procedure define the global variables that are used in procedures 2 and 3:

```
define global  "DISCOUNT"  Number .
define global  "RESTOTAL"   Number .
define global  "RESERVATION #"  Number .
```

To pass a global variable from one procedure to another, the variable must be defined identically in each procedure that uses it as well as the main Control procedure. A global variable can be assigned a value just once, and used by each individual procedure, or manipulated and reassigned in one or more procedures. The variables defined in this control procedure don't have values assigned to them until the CALCULATE DISCOUNTS procedure begins processing.

The fourth line of the Control procedure specifies:

```
run procedure "INPUT RESERVATIONS" .
```

The run procedure command executes a procedure just as if you chose **File>>Open** to run a procedure. When DataEase reaches the end of the procedure, the next command is automatically executed in the Control procedure.

When the INPUT RESERVATIONS procedure is run, DataEase creates a new record in the RESERVATIONS form for each new reservation, copying the information from the TEMPMEM temporary holding form. Finally, the number of rooms required for the new reservation are subtracted from the VACANCIES field in the CLUB ROOMS form.

The fifth line specifies:

```
run procedure "CALCULATE DISCOUNTS" .
```

This command executes the CALCULATE DISCOUNTS procedure that: (1) calculates the total dollar amount of the reservation, (2) calculates the total number of children and adults included in the reservation, and (3) calculates the dollar amount of the discount to be applied to the cost of the reservation.

The third procedure, RESERVATION INVOICES is called from within the CALCULATE DISCOUNTS procedure so RESERVATION INVOICES is not included in the control procedure. Since the define and run procedure commands do not require any special terminating command other than a period, no further commands are required.

Summary

In this example, when you choose **File>>Open** and select the PROCESS RESERVATIONS Control procedure, DataEase displays the Data-entry form, then the MEMBERS form. You can then enter records in the MEMBERS form using the standard DataEase record entry procedures. Each time you choose **Edit>>Form Clear**, DataEase displays a new blank record, just as in normal record entry.

When you finish entering records and close the MEMBERS form, DataEase executes each command in the Control procedure until the end of the RESERVATION INVOICES procedure is reached.

When the Control procedure is completed, DataEase automatically returns to the form or menu from which the procedure was originally called.

The sample scripts included in this chapter are designed to illustrate many of the advanced capabilities of the DataEase Query Language. There may be more efficient and effective ways of achieving the same ends through the use of a well-placed derivation formula or in a Client-server environment. Some alternate solutions are presented in the next chapter.

Chapter 5 : Transaction Processing

Using DQL in a Client-Server Environment

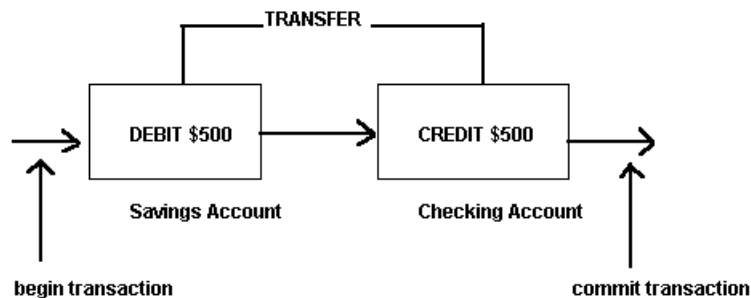
See **DQL 8** for more information on the DataEase SQL Commands.

See **DG A** for information on designing and running DataEase applications with a foreign database engine.

Using DataEase in a Client-server environment can significantly speed up performance by dividing data access and processing tasks between a high performance back-end database engine and a user-friendly, graphical, front-end client application. The enhanced performance provided by Client-server architecture is particularly useful for transaction processing, a special type of application in which large numbers of records are frequently updated, such as travel reservations, inventory control, or financial transactions.

SQL Procedural Commands
begin transaction
commit
rollback
current SQLCODE
current SQLCOUNT
current SQLMSGTXT
error messages off
error messages on
exec SQL
rollback
tran off
tran on

In high-volume transaction processing applications, data integrity is as important as speed. By default, DataEase treats any DQL Procedure that adds, modifies, or deletes data as a transaction. A transaction is a set of actions that must be processed as a single unit of work to maintain data integrity. For example, a procedure that withdraws money from a customer's savings account and deposits the withdrawn amount into the customer's checking account must be treated as a single transaction to maintain the correct balance in both of the customer's accounts.



Debiting the savings account and crediting the checking account must be treated as a single unit of work to maintain data integrity.

An application cannot process just one part of a transaction. When a computer goes down in the middle of a transaction, the application contains safeguards to rollback the data to the state it was in before the transaction began.

Using SQL Commands in a DQL Procedure

Whether or not you run DataEase in a Client-server environment, the special set of transaction-oriented commands (shown on the previous page) are automatically included in the DQL Script Editor in the Commands pick list. The example in this chapter explains how these commands are used to enhance the **Club ParaDEASE** reservations system described in the preceding chapter.

Note: If you haven't already done so, we recommend that you read at least the first section of Chapter 4 which describes the INPUT RESERVATIONS procedure, before you continue.

Using Explicit Transactions to Enhance a DQL Procedure

In Chapter 4 we created the INPUT RESERVATIONS procedure to automate the process of checking room availability and entering reservations. The original script from Chapter 4 is shown below.

```
if data-entry ROOMS_REQUIRED > any CLUB ROOMS with
    ( RESERVATION DATE = data-entry RESERVATION_DATE and CLUB ID =
    data-entry CLUB_ID ) VACANCIES then
    message " Not enough rooms are available. |
    Please choose another Club or date. " window .
else
while current status not = 1 do
    input using MEMBERS in "TEMPMEM" .
    case ( current status )
    value 1 :
        exit
    value 2 :
        modify records in CLUB ROOMS with
            ( CLUB ID = data-entry CLUB_ID and
            RESERVATION DATE = data-entry RESERVATION_DATE )
            VACANCIES := VACANCIES - data-entry ROOMS_REQUIRED .
        enter a record in RESERVATIONS
            copy all from TEMPMEM ;
            CLUB ID := data-entry CLUB_ID ;
            DATE:= data-entry RESERVATION_DATE ;
            ROOMS REQUIRED:= data-entry ROOMS_REQUIRED .
            record entry RESERVATIONS .
    others :
        message " You are not authorized to |
        modify or delete a record! " window .
    end -- case
end -- while
end -- if
```

Although the INPUT RESERVATIONS procedure can check to see if the required number of rooms are available before any RESERVATIONS data is entered, the procedure does not include the data integrity safeguards that online transaction processing normally requires. For example, if the RESERVATIONS form is locked when this procedure attempts to enter a reservation record, the procedure cannot automatically replace the number of rooms that were subtracted from the total rooms available at the destination club.

Modified INPUT RESERVATIONS Script

However, if you are running DataEase in a Client-server environment, the procedure can be rewritten to: (1) identify and control individual transactions, and (2) use the SQL database engine's transaction log to rollback an incomplete transaction. The Client-server version of the INPUT RESERVATIONS script might be written as shown below. The parts of the script that remain the same are shown in greyed text.

```
tran off
  if data-entry ROOMS_REQUIRED > any CLUB ROOMS
  with ( RESERVATION DATE = data-entry RESERVATION_DATE
  and CLUB ID = data-entry CLUB_ID) VACANCIES then
    message " Not enough rooms are available. |
    Please choose another Club or date. " window .
  else while current status not = 1 do
    input using MEMBERS in "TEMPMEM" .
    case ( current status )
    value 1 :
      exit
    value 2 :
      begin transaction
        modify records in CLUB ROOMS with
          ( CLUB ID = data-entry CLUB_ID and
          RESERVATION DATE =
          data-entry RESERVATION_DATE )
          VACANCIES := VACANCIES - data-entry ROOMS_REQUIRED .
        enter a record in RESERVATIONS
          copy all from TEMPMEM ;
          CLUB ID := data-entry CLUB_ID ;
          DATE := data-entry RESERVATION_DATE ;
          ROOMS REQUIRED := data-entry ROOMS_REQUIRED .
      if current SQLCODE = 0 then
        commit
        record entry RESERVATIONS .
      else
        rollback
        message " This transaction has failed. |
        No data has been changed or entered. " window .
      end -- if
    others :
      message " You are not authorized to |
      modify or delete a record! " window .
    end -- case
  end -- while
```

```

end -- if
tran on

```

Explanation of the Modified Script

Sew **DQL 8** for more information on the tran on and tran off commands.

In this section, we will look at each of the modifications and examine the effect they have on how DataEase processes the script.

The first new command is tran off. When a DQL Procedure is translated into SQL, DataEase inserts an implicit begin transaction command at the beginning of the script. By default, DataEase treats any procedure that includes a Processing command (e.g., enter a record, list records) as a transaction (i.e., a single unit of work).

The tran off command turns off the DataEase default transaction management and data locking facilities and switches DQL processing into tran off mode. In tran off mode, DataEase processes each record as a separate transaction. No DQL actions are grouped together as a transaction and no multi-user data locking rules are enforced. These restrictions let you control the beginning and end of individual transactions by placing your own begin transaction, commit, and rollback commands. In our example, we have inserted a tran off command so we can control exactly which actions constitute a transaction.

The next section of the script (shown below) remains the same. It checks room availability then opens the MEMBERS form using the input using command.

```

if data-entry ROOMS_REQUIRED > any CLUB ROOMS
  with ( RESERVATION DATE = data-entry RESERVATION_DATE
    and CLUB ID = data-entry CLUB_ID) VACANCIES then
    message " Not enough rooms are available. |
    Please choose another Club or date. " window .
  else
    while current status not = 1 do
      input using MEMBERS in " TEMPMEM " .
      case ( current status )
      value 1 :
        exit
      value 2 :

```

The begin transaction command which we inserted immediately after the value 2 statement marks the start of a transaction. When processing reaches a begin transaction command, DataEase treats all the following statements in the script as part of the same transaction until a commit, rollback, or another begin transaction command is reached.

See **DQL 8** for information on the begin transaction, commit, and rollback commands.

In our example, when the reservation clerk finishes filling in the information in the MEMBERS form and saves the record, DataEase sends a begin transaction command to the server. The begin transaction command combines the following two actions into a single transaction:

- Modify the matching record (the record with the same CLUB ID and DEPARTURE DATE) in CLUB ROOMS to decrement the number of available rooms.
- Enter a new RESERVATIONS record.

If one of these actions isn't completed successfully, DataEase tells the server to undo the other.

If you run the original procedure in a Client-server environment, the entire script is treated as a single transaction. For example, if you begin to enter a fifth reservation and the RESERVATIONS form is unavailable, all four of the previous reservations are rolled back.

The next three sections of the script (below) remain the same. They modify a record in CLUB ROOMS and enter a new record in RESERVATIONS.

```

modify records in CLUB ROOMS with
( CLUB ID = data-entry CLUB_ID and
  RESERVATIONDATE = data-entry RESERVATION_DATE)
VACANCIES := VACANCIES - data-entry ROOMS_REQUIRED .

enter a record in RESERVATIONS
copy all from TEMPMEM ;
CLUB ID:= data-entry CLUB_ID;
DATE:= data-entry RESERVATION_DATE ;
ROOMS REQUIRED:= data-entry ROOMS_REQUIRED .

```

See **DQL 8** for more information on current SQLCODE.

At the end of the previous series of actions, we've added the following statements:

```

if current SQLCODE = 0 then
commit
record entry RESERVATIONS .
else
rollback
message " This transaction has failed. |
      No data has been changed or entered. " window .
end

```


This section of the script uses an if Command to test the value stored in the current SQLCODE system-defined variable.

The current SQLCODE variable traps the error code the server generates when an error occurs during the processing of an SQL statement (DataEase sends an SQL INSERT command to the server when you use the enter a record command in a script). If the last SQL statement processes without error, current SQLCODE is set to zero.

The if command checks the value in the current SQLCODE variable for a value of zero. If the value is zero, DataEase commits all the previous actions that have occurred since the begin transaction command was issued. In our example, the following actions are committed (written to the database):

- Decrement the number in the VACANCIES field in CLUB ROOMS by the number entered in the ROOMS_REQUIRED field in the Data-entry form.
- Enter a new record in the RESERVATIONS form.

See the documentation that comes with your SQL database software to determine if your environment supports stored procedures.

If the two actions described on the previous page are successfully committed, DataEase opens the RESERVATIONS form so the reservation clerk can complete the data entry. When the reservation clerk closes the RESERVATIONS form, control reverts to the script.

```

        if current SQLCODE = 0 then
        commit
        record entry RESERVATIONS .
    else
        rollback
        message " This transaction has failed. |
            No data has been changed or entered. " window .
    end

```

If the value in the current SQLCODE variable is anything other than zero, an error has occurred. When an error occurs, the actions that follow the else command are executed: DataEase first rolls back the two data modifications, then displays the specified message text. The end command terminates the if statement.

The last new command is the tran on command. The tran on command is used to restore the DataEase default transaction management and data locking facilities we disabled at the beginning of the script with the tran off command.

Note: The tran on command at the end of this script is required only if additional statements follow. However, we recommend that you include this command to maintain correct syntax throughout the script.

SQL User Permissions

See **DG 8, A** for information on how DataEase security affects running DQL Scripts in a Client-server environment.

To execute a DQL Procedure that accesses foreign data, you must have a valid User Name in all the SQL databases in which referenced tables are stored and the required User Permissions in all the accessed tables. DataEase automatically translates its own User Security Levels stored in the Users form into the appropriate SQL User Permissions.

The table below shows the SQL User Permissions required to execute various DQL commands.

SQL User Permissions for DQL Procedures

DQL Command	Required SQL User Permission	Comments
list records	SELECT	N/A
enter a record	INSERT	N/A
modify records	UPDATE	N/A
delete records	DELETE	N/A
input using record entry	Depends on which processing key is pressed (F2 = INSERT, F3 = SELECT, F7 = DELETE, F8 = UPDATE)	N/A
install application	Depends on what operations the installation file performs (CREATE TABLE, INSERT, DELETE, UPDATE)	The install application command copies all the documents and data from another application into the current application (equivalent to choosing Application>>Utilities>>Install Application).
reorganize	N/A	You can only use the reorganize command on native DataEase tables, not on forms that access SQL tables.
application status	N/A	The application status command generates a report that displays the status of the application (equivalent to choosing Application>>Utilities>>Status).
lock unlock query selection	N/A	Many SQL engines do not support the lock, unlock, and query selection commands. See your Database Engine Information Guide for information on whether these DQL commands are supported.
lock db unlock db	N/A	The lock db command prevents DataEase users from accessing any form, table, report, or procedure in an application, but has no effect on users who are using another application (e.g., Excel) to access the SQL tables that are part of the current application.

Chapter 6 : DQL Tech Tips

Using DQL with the DataEase Native Engine

The following pages describe how you can use the DataEase Query Language to carry out some of the most commonly encountered database tasks.

How to Organize the Commands in a Simple Script	100
How to Choose the Primary Table in a Multi-Table Procedure	103
How to Combine Multiple Selection Criteria using and and or	104
How to Add Group Totals and Grand Totals to a Report	105
How to List Individual Field Values from a Related Table	107
How to Use Indexes to Improve DQL Processing Speed	109
How to Post Totals to Another Table	113
How to Calculate a Percentage	114
How to Count the Number of Groups in a Report	116
How to Count the Number of Records in a Group	118
How to Produce Multiple Printouts of Each Record in a Report	120
How to Create an Accounts Receivable Aging Report	122
How to Sort the Values in a Choice List Field	125
How to List the First Several Records in a Table	126
How to Find and List Duplicate Records	127

How to Organize the Commands in a Simple Script

See **DQL 8** for information on the for, list records, and end commands.

Question

How should I order the commands in a script when I'm using sorting, grouping, and statistical operators?

Solution

1. Begin with a for command to specify the Primary table for the report.
2. Use the list records command to list the fields you want to include.
3. List the grouped field(s) in groups with group-totals first under the list records command.
4. List the ordered field(s) in order (or in reverse) after the grouped field(s).
5. List the field(s) on which you wish to generate statistics using the conditional statistical operators or the relational statistical operators.

Example 1

```
for MEMBERS ;
    list records
        STATE in order ;
        LAST NAME in order ;
        FIRST NAME in order .
end
```

This script tells DataEase to: (1) sort the STATE field in order first, (2) then, within each state group, sort the LAST NAME field in order, and finally, (3) for each duplicate LAST NAME, sort the FIRST NAME in order.

Sample Output 1

The output for the above example can be formatted to look like this:

State Name

```
AL    Gould, Matthew
AL    Jones, Anita
AL    Jones, Thomas
AR    Notarnicola, Rosanne
```

Example 2

```
for MEMBERS ;
    list records
        LAST NAME in order ;
        FIRST NAME in order ;
        STATE in order .
end
```

In Example 2, since the LAST NAME field is listed first and the STATE field last, DataEase performs the sorts in a different sequence, creating a very different result.

SampleOutput 2

The output for Example 2 can be formatted to look like this:

Name	State
Adams, John	CT
Adams,Will	OK
Albert, Roland	OH
Anders, Jenna	MT

Example 3

```

for MEMBERS ;
    list records
    STATE in groups ;
    LAST NAME in order ;
    FIRST NAME in order .
end

```

Example 3 is identical to Example 1 except that the STATE field is now listed in groups rather than in order. The in groups command orders the values just like the in order command but each value is listed just once instead of repeatedly. Notice the difference in the appearance of the output shown on the next page.

SampleOutput 3

The output from Example 3 can be formatted to look like this:

State Name

AL

Gould, Matthew
 Jones, Anita
 Jones, Thomas

AR

Notarnicola, Rosanne
 Rubin, Benjamin
 Shoen, Cecilia
 Turner, Anne
 Turner, Patrick

Example 4

```

for CLUBS ;
    list records
    COUNTRY in groups with group-totals ;
    CLUB NAME in order ;
    mean of RESERVATIONS TOTAL DUE ;
    ROOMS : item sum .
end

```

This example tells DataEase to: (1) open the CLUBS table and process all the records, (2) group the records by country, (3) within each country group, sort the club names in ascending order (a-z), (4) calculate the average reservation total for each club, and (5) list the number of rooms at each club, the number of rooms in each country, and the total number of club rooms in **Club ParaDEASE**.

Discussion

DataEase processes the commands in a script in the order it encounters them. Therefore, as a general rule you must:

- Place any grouping commands before any ordering commands.
- Place any statistical operators after any grouping and/or ordering commands.

Tip

When you use multiple or nested for loops, you must terminate each for loop with an end command. Although DataEase doesn't require an end command in a simple script like the one shown on the previous page, we recommend that you get in the habit of using an end command to signal the end of each for loop.

How to Choose the Primary Table in a Multi-Table Procedure

Question

How do I choose the Primary table in a procedure that accesses more than one table?

Solution

You should typically choose the table that contains the fewest number of records as the Primary table for a report.

Example

```

    for MEMBERS ;
        list records
        MEMBER ID in groups ;
        all FAMILY MEMBERS LAST NAME ;
        all FAMILY MEMBERS FIRST NAME ;
        TOTAL DUE .
    end

```

This example tells DataEase to: (1) process each record in the MEMBERS table listing each MEMBER ID just once and then, (2) the FIRST NAME and LAST NAME of each FAMILY MEMBER included in the membership, and (3) list the total amount of dues paid for the membership.

Discussion

This script could also have used the FAMILY MEMBERS table as the Primary table listing any MEMBERS MEMBER ID and any MEMBERS TOTAL DUE to select data from the MEMBERS table. However, as a general rule, it is better to start on the one side of a one-to-many relationship since there are fewer records to process in the Primary table. Since Match fields in a relationship should always be indexed, the time it takes DataEase to retrieve records from a related table is minimized.

How to Combine Multiple Selection Criteria using and and or

Question

When should I choose **and** to combine multiple selection criterion and when should I use **or**?

Solution

1. Use **and** when you want to select records that meet each of a set of criteria.
2. Use **or** when you want to select records that meet one or more of a set of criteria.
3. When you want to combine criteria using both **and** and **or**, use parentheses to clarify the meaning to DataEase. The examples below illustrate each of the possible combinations.

Example 1

```
for FAMILY MEMBERS      with
AGE STATUS  =  "adult"  and STATE  =  "CO"  ;
```

Example 1 tells DataEase to select only those records that store both the values adult in the AGE STATUS field and CO in the STATE field.

Example 2

```
for FAMILY MEMBERS
with AGE STATUS  =  "adult"  or  STATE  =  "CO"  ;
```

In Example 2, only one of the specified conditions must be met for the record to qualify. When we use the **or** operator instead of the **and** operator, children living the state of Colorado qualify as do adults living in any other state.

Example 3

```
for FAMILY MEMBERS
with AGE STATUS = "adult" and (STATE = "CO" or STATE = "WY" )  ;
```

The parentheses in Example 3 clarify any ambiguity for DataEase. This example tells DataEase to find all the adults living in either Colorado or Wyoming.

Caution

You might think that to list members from both Colorado and Wyoming, you could use the **with** clause, with **STATE = "CO"** and **STATE = "WY"**. However, the **and** operator stipulates that all specified values must be found in a record for the record to be processed. Since an address cannot be in both Colorado and Wyoming, DataEase doesn't find any records to process. When you want to select records that store any one of several different values in the same field, you must use the **"or"** operator.

When you combine an **and** operator with an **or** operator in the same **with** clause, parentheses must be used to assure that no ambiguity exists. For example, without the parentheses, Example 3 would read **with STATE = "CO" or STATE = "WY" and AGE STATUS = "adult"**. Given this statement, DataEase can't determine whether you want to list: (1) only adults living in Colorado or Wyoming, or (2) all Colorado residents (regardless of age status) and any adult residents of Wyoming. DataEase is unable to resolve the ambiguity and will not therefore parse the script.

How to Add Group Totals and Grand Totals to a Report

Question

How do I get DataEase to generate both group subtotals and a grand total for the entire report?

Solution

1. Use the in groups command to list the grouped fields.
2. Use the statistical operators to generate the desired statistics on the appropriate field(s). You can use any combination of the statistical operators.
3. If you also want to list the individual field values, include the item operator with any other statistical operators.

Example

```
for RESERVATIONS ;
    list records
    MEMBER ID in groups with group-totals ;
    any MEMBERS LAST NAME ;
    CLUB NAME ;
    TOTAL DUE : item sum mean .
end
```

Sample Output

The output for this example can be formatted to look like this:

Member ID	Name	Club Name	Total Due
00001	Birnbaum		
		Playa Blanca	2,502.00
		Cancun	2,700.00
		Magic Isle	3,120.00
			=====
	sum		8,322.00
	mean		2,774.00
00002	Perrault		
		Caravelle	5,560.00
		Paradise Island	2,900.00
			=====
	sum		8,460.00
	mean		4,230.00

sum	16,780.00
mean	3,356.40

Tip

When you choose any of the automatically generated layout options, DataEase places each statistical field on the appropriate parent object. If you choose a custom layout, the item field should be placed on the Subform's Record object, while any aggregate fields should be placed both on the Subform's Form object (where it will display subtotals) as well as the primary Form object (where it will display a grand total for the report).

How to List Individual Field Values from a Related Table

Question

How can I list individual field values from a related table?

Solution

1. Use a for command to open the Primary table.
2. Use a list records command to list any fields from the Primary table to be grouped or ordered first.
3. Based on the Primary table you choose, use the relational operators any or all to list individual field values from a related table. The relational operator you choose depends on the direction in which you are navigating the relationship. Use:

any in a many-to-one (or one-to-one) direction when there is only one related record for DataEase to find.

all in a one-to-many direction when the related file stores multiple related records for each primary record.

Example 1

```
for MEMBERS ;
list records
MEMBER ID in groups ;
LAST NAME ;
all FAMILY MEMBERS FIRST NAME ;
all FAMILY MEMBERS AGE STATUS ;
TOTAL DUE . end
```

Example 1 tells DataEase to: (1) process all the MEMBERS records listing, (2) the MEMBER ID and LAST NAME of each member, (3) the FIRST NAME, LAST NAME, and AGE STATUS from each related record in FAMILY MEMBERS, and (4) the total membership fees.

Example 2

```
for FAMILY MEMBERS ;
list records
any MEMBERS MEMBER ID in groups ;
any MEMBERS LAST NAME ;
FIRST NAME ;
AGE STATUS ;
any MEMBERS TOTAL DUE .
end
```

Example 2 can be formatted to present the identical output as Example 1. The only difference is that this script uses the many side of the relationship as the Primary table for the report.

Sample Output

Both examples can be formatted to produce output that looks like this:

Member ID	LastName	Family Members	AgeStatus	MembershipTotal
00033	McKeag	Michael	adult	70.00
		Susan	adult	
00034	Rada	Amanda	adult	85.00
		Jerome	child	
		Clarence	adult	
00035	Carlson	Elizabeth	adult	35.00

Tip

While Example 2 is perfectly viable, you should notice a significant speed improvement running Example 1. Because Example 1 starts on the one side of the relationship, there are fewer records for DataEase to process.

How to Use Indexes to Improve DQL Processing Speed

Question

Will indexing the fields that I typically sort and use in selection criteria improve the processing speed of my scripts?

Solution

There is no simple yes or no answer to this question; however, understanding how DataEase processes a procedure can help you optimize your scripts. This Tech Tip tells you when and how DataEase uses a field's associated index file so you can make an informed decision as to when an index will speed up (or perhaps slow down) the processing time required to run each of your DQL Procedures.

DataEase uses a three phase process to execute a procedure that contains a Processing command (e.g., list records, modify records):

1. Record Selection phase – used only if there are selection criteria that includes an indexed field.
2. Qualification phase – used when there are selection criteria and one or more fields are not indexed.
3. Processing phase – used to process the data.

In a Processing procedure (one that includes a Processing command), the Processing phase is always used. Record selection and/or qualification are only used if the script includes a with statement. The availability and size of the indices in the selection criteria determine when and how DataEase processes a simple script.

Lets begin by taking a sample script and looking at how DataEase uses any available indexes.

Example

```
for FAMILY MEMBERS
with STATE = "CA" and AGE STATUS = "ADULT" ;
list records
FIRST NAME ;
LAST NAME in order .
end
```

Example 1

Depending on which fields are indexed and their placement in the script, DataEase may or may not use an associated index file. Using just the selection criteria from the above example, lets take a look at each possibility:

```
with STATE = "CA" and AGE STATUS = "ADULT" ;
```

If both the STATE and AGE STATUS fields are indexed, DataEase normally uses both indices to generate an array of record numbers that meet each criterion. The arrays are then compared (in memory) for matching record numbers. Since this example uses an and

operator, the intersection of the list is then processed by the list records portion of the script, as illustrated below:

STATE = "CA"	AGE STATUS = "adult"	115392121157	Record
Numbers for Processing	113688121157	11121157	

Example 2

In the second scenario, we use the identical with clause except that the and operator is replaced with the or operator:

```
with STATE = "CA" or AGE STATUS = "ADULT" ;
```

Once again, if both fields are indexed, DataEase uses both indices to generate two arrays of record numbers that meet each criterion. However, since we're using or instead of and, the union of the two lists is processed as shown on the next page.

STATE = "CA"	AGE STATUS = "adult"	115392121157	Record
Numbers for Processing	113688121157	1136538892121157	

Example 3

In the third scenario, using the and operator again, suppose the STATE field is indexed, but the AGE STATUS field is not:

```
with STATE = "CA" and AGE STATUS = "ADULT" ;
```

As before, DataEase uses the available index to generate an array of record numbers that meet the STATE = "CA" criterion. However, since the AGE STATUS field is not indexed, DataEase must qualify each of the records in the first array against the second criterion, as illustrated below:

STATE = "CA" and AGE STATUS = "adult"
115392121157
Each of the records selected in the array is qualified against the second criterion. Records that meet the second criterion are processed, but those that don't are discarded.

Example 4

In the last scenario, suppose again that we are using the or operator and that the STATE field is indexed, the AGE STATUS field is not:

```
with STATE = "CA" or AGE STATUS = "ADULT" ;
```

In this scenario, DataEase does not use the index on the STATE field. Since every record in the FAMILY MEMBERS table must be qualified for the second criterion, there is no benefit in building an array of the matching STATE records. It is much faster for DataEase to qualify each record for both criteria simultaneously. Those records that qualify are passed on to the Processing phase of the procedure.

Discussion

The order in which you list selection criteria can also have a dramatic impact on performance. If DataEase is going to use an index in selecting records, it automatically opens the index for the first field listed after the with command. For example, in the following with statement:

```
with STATE = "CA" and AGE STATUS = "adult"
```

if both fields are indexed, DataEase automatically opens the STATE field's index and builds the array of matching record numbers. If the second criteria's array is determined to be at least four times greater than the first (e.g., STATE = "CA" selects 1,000 records, while AGE STATUS = "ADULT" selects 6,000 records), DataEase does not take the time to build the second array. It is faster to simply qualify the first 1,000 records for the second criterion. If, however, the order of the fields is reversed (e.g., AGE STATUS = "ADULT" and STATE = "CA"), DataEase automatically opens and uses the index on the AGE STATUS field, forcing the comparison of the two arrays in memory and potentially slowing down performance. Therefore, you should always list the fields in your selection criteria in order from most discriminating to least discriminating.

Example 5

Now let's look at the original script and consider the use of indexes when you sort the procedure's output.

```
for FAMILY MEMBERS
  with STATE = "CA" and AGE STATUS = "ADULT" ;
  list records
  FIRST NAME ;
  LAST NAME in order .
end
```

In this example, we list the LAST NAME field in order. If the LAST NAME field is indexed, DataEase only uses the index if an index was not used in the Selection phase of processing (i.e., if neither the STATE field nor the AGE STATUS fields are indexed). If DataEase does use an index in selecting records, the index on the sorted field is not used.

When using an index for sorting the records, DataEase uses only the index on the primary sort level. For example, in the script shown above, if the FIRST NAME field was also indexed and listed in order, DataEase would use the index on FIRST NAME and ignore the index on LAST NAME.

Tip

Always index the Match (key) fields in a relationship. Both the Primary and foreign keys should be indexed.

Caution

Indexes are most effective in improving DQL processing speed when they select a relatively small number of records. Therefore, indexing a choice field with two or three choices, or a Yes/No field (where either selection is likely to select a large percentage of the records in the file) can be a particularly ineffective use of an index.

How to Post Totals to Another Table

Question

I can get DataEase to generate report subtotals but I do not know how to transfer those totals to a separate summary table.

Solution

1. Define a temporary variable of the same type as the field you are listing in groups.
2. Use a for command to specify the Primary table for the report.
3. Include a list records command. You only need to list the field that you are grouping. List the field in groups and terminate the list records command with a period.
4. Use an if command to check the status of the temporary variable.
5. Use the enter a record in command to enter a new record in the summary table each time the group changes.
6. Reassign the temporary variable for the next pass of the for loop.

Example

```

define temp  "LASTGRP"  Numeric String 5 .
for RESERVATIONS ;
  list records
  MEMBER ID in groups .
    if MEMBER ID not  =  temp LASTGRP then
      enter a record in RESERVATION SUMMARY
      MEMBER_ID := RESERVATIONS MEMBER ID ;
      TOTAL_SPENT := sum of RESERVATIONS with
        ( MEMBER ID  =  RESERVATION SUMMARY MEMBER_ID )
      TOTAL DUE .
    assign temp LASTGRP:= MEMBER_ID .
  end -- if
end -- for

```

This example tells DataEase to: (1) define a temporary variable called last grp as a 5 digit numeric string, (2) open the reservations table and process every record, and (3) list the MEMBER ID in groups. The next part of the script tells DataEase to: (4) using an if command, compare the value in the MEMBER ID field (in the record being processed) with the value in the LAST GRP variable. If the two values are different, DataEase enters a new record in the summary table (RESERVATION SUMMARY). The information entered in the new record includes the MEMBER ID and the sum of the total cost of all the reservations for that member. The last part of the script tells DataEase to: (5) assign the temp LAST GRP variable the current MEMBER ID. The first end command terminates the if command and the second end terminates the for loop.

How to Calculate a Percentage

Question

How can I calculate a percentage in a report?

Solution

It depends on the type of percentage you want to generate. You can:

- Use the conditional statistical operators to calculate what percent of the records in a group or the entire table contain a particular field value, as illustrated in Example 1 below.
- Use the relational statistical operators to generate a total that you can divide a field value or another sum by, as illustrated in Example 2 below.

Note: When you use relational statistical operators, your solution will benefit from the addition of one or more temporary variables to store the calculated value, as shown in Examples 2 and 3.

Example 1

```
for FAMILY MEMBERS ;
    list records
    MEMBER ID in groups ;
    AGE STATUS = "child" : percent .
end
```

This example tells DataEase to: (1) process all the records in the FAMILY MEMBERS table, (2) group the records by the MEMBER ID field, and (3) generate a percentage of how many of the members in each group are children.

Example 2

```
define temp "RM" number .
assign temp RM := sum of CLUBS ROOMS .
for CLUBS ;
    list records
    CLUB ID in groups ;
    ( ROOMS / temp RM ) * 100
. end
. end
```

Example 2 tells DataEase to: (1) process all the records in the CLUBS table, (2) group the records by MEMBER ID, and (3) divide the total number of rooms at each club by the total number of rooms at all the **Club ParaDEASE** clubs, then multiply the result by 100 to generate a percentage.

Note: In Example 2, we used a temporary variable to accumulate the total number of rooms for all clubs prior to opening the CLUBS table. We might have instead replaced the last line of the script with "rooms/sum of CLUBS ROOMS * 100". However, this syntax requires DataEase to generate the same total over and over again once for each record processed by the for loop.

Example 3

```

define temp "CLUB" numeric string 5 .
define temp "CLUBTOT" number .
define temp "KIDTOT" number .
for RESERVATIONS ;
  if temp CLUB not = CLUB ID then
    assign temp CLUB := CLUB ID .
    assign temp CLUBTOT := TOTAL DUE .
    assign temp KIDTOT := TOTAL CHILD PRICE .
  else
    assign temp CLUBTOT := temp CLUBTOT + TOTAL DUE .
    assign temp KIDTOT := temp KIDTOT + TOTAL CHILD PRICE .
  end
  list records
  CLUB ID in groups ;
  ( temp KIDTOT / temp CLUBTOT ) * 100 .
end

```

Example 3 tells us what percentage the total child's price is of the total of all reservations at each club. This script tells DataEase to: (1) define three temporary variables. temp CLUB will store the CLUB ID of the current record being processed. temp CLUBTOT will accumulate the total dollar amount of all the reservations for each club. temp KIDTOT will accumulate the total dollar amount of all the children's reservations at each club. The next part of the script tells DataEase to: (2) open the reservations table and process all the records, then (3) use an "if" command to check the value in temp CLUB. If this value is different to the CLUB ID value in the current record, assign CLUB the current CLUB ID. Simultaneously, assign the CLUBTOT variable the value in the TOTAL DUE field and the KIDTOT variable the value in the TOTAL CHILD PRICE field. If temp CLUB and the CLUB ID are the same, increment temp CLUBTOT and temp KIDTOT, (4) group and list the CLUB ID, and (5) divide temp KIDTOT by temp CLUBTOT and multiply the result by 100 to list what percentage the children's reservation total is of the whole total for each club.

How to Count the Number of Groups in a Report

Question

How can I count the number of groups in my reports?

Solution

1. Define a temporary variable to store the value in the grouped field. Use this variable to indicate when the group has changed.
2. Define a second temporary variable to accumulate the number of groups. Increment this variable each time the value in the first variable changes.
3. Use a for command to open the Primary table for the report.
4. Use an if Command to check the value in the first variable. If the grouped field value and the variable are different, update the variable to reflect the new group and simultaneously increment the second variable.
5. Use an end command to terminate the if Command.
6. Include a list records command to list the fields to be printed. The list should include the second temporary variable that accumulates the number of groups.

Example

```
define temp  "ST"   text 2 .
define temp  "KOUNT" number .
  for MEMBERS ;
    if temp ST not = STATE then
      assign temp ST := STATE .
      assign temp KOUNT := temp KOUNT + 1 .
    end -- if
    list records
    MEMBER ID ;
    LAST NAME ;
    STATE in groups ;
    temp KOUNT .
  end -- for
```

This example tells DataEase to: (1) define two temporary variables. The first should be the same field type as the field you are grouping. The second variable should be defined as a number since it will accumulate a numeric value. The next part of the script tells DataEase to: (2) open the MEMBERS table and check the value in the ST temporary variable. If the value in the ST variable is different than the value in STATE field in each record processed, assign the ST variable to the current STATE field value. At the same time, increment the KOUNT variable by one, (3) sort the records in ascending order by state, and (4) list the STATE field in groups, the MEMBER ID, LAST NAME, and temp KOUNT variable.

Tip

When you create the layout for this type of procedure, place the temp KOUNT variable on the primary Form object.

Caution

When you name a temp or global variable in a script, be sure to avoid using keywords and existing field names (e.g., use KOUNT or COUNTER rather than COUNT, as shown on the previous page).

How to Count the Number of Records in a Group

Question

How can I count the number of records in each group?

Solution

There are two alternative solutions:

1. You can use conditional statistical operators as shown in Example 1.
2. If you want to further manipulate that number (e.g., use it to calculate a percentage) or enter it in a summary file, you can use a temporary variable to accumulate the value.

Example 1

```
for MEMBERS ;
  list records
    STATE in groups ;
    LAST NAME in order ;
    "A"    =    "A"    : count .
end -- for
```

This example tells DataEase to: (1) open the MEMBERS table and process all the records, (2) sort the records in ascending order by state, (3) further sort the records for each state in order by last name, and (4) count every record in the file.

The line in Example 1, "A" = "A" : count, is used to count every record processed by the query. This line uses a conditional statistical operator to count all the records unconditionally. For each record in the table, the comparison "A" = "A" is always true, therefore, every record processed is counted. You can also use a field value (e.g., LAST NAME = LAST NAME : count; (for every record processed the last name is always equal to itself) to count the records, but the constant text value "A" is a more commonly used method for counting records.

Example 2

```
define temp "ST" text 2 .
define temp "KOUNT" number .
for MEMBERS ;
  if temp ST not = STATE then
    assign temp ST := STATE .
    assign temp KOUNT : = 1 .
  else
    assign temp KOUNT := temp KOUNT + 1 .
  end -- if
  list records
    STATE in groups ;
    LAST NAME in order ;
    temp KOUNT .
end -- for
```

Example 2 tells DataEase to: (1) create two temporary variables. The first holds the state value from the current record. The second is incremented each time the state changes. The next part of this script tells DataEase to: (2) open the MEMBERS table and process every record, and (3) compare the ST variable with the value in the state field. If the two values are different, update the ST variable to the state value in the current record and reinitialize the KOUNT variable. If the two values are the same, increment the KOUNT variable by one. The rest of the script tells DataEase to: (4) group the records by state. Within each state group, sort the records in ascending order by last name, and (5) list the state, last name and number of records processed.

Discussion

Both examples can be formatted to produce nearly identical output. As well as being simpler, Example 1 can be used to provide a grand total (i.e., the number of all the records processed by the query) as well as a subtotal for each group. Example 2 provides only group subtotals. To generate a grand total in Example 2 you must add another temporary variable.

Although Example 1 is much simpler and more straightforward, if you want to then post the totals to a separate file, you must follow Example 2.

Caution

When you name a variable in a script, be sure to avoid using keywords and existing field names (e.g., use KOUNT or COUNTER rather than COUNT, as shown above).

How to Produce Multiple Printouts of Each Record in a Report

Question

How can I print several copies of each record in a report? For example, I frequently need to print three or four mailing labels for each person.

Solution

1. Define a temporary variable to keep track of how many labels have been printed.
2. Use a for command to open the Primary table.
3. Assign the temporary variable a starting value. This should be done within the for loop.
4. Open a while loop within the for loop. This tells DataEase to continually loop through the list records command as many times as requested for each record processed. Use a list records command to list the desired fields.
5. Reassign the temporary variable to increment as each interior looping action is completed.

Example

```
define temp "KOUNT" Number .
  for MEMBERS ;
    assign temp KOUNT := 1 .
    while temp KOUNT <= 3 do
      list records
      FIRST NAME ;
      LAST NAME ;
      STREET ;
      CITY ;
      STATE ;
      ZIP .
      assign temp KOUNT := temp KOUNT + 1 .
    end -- while
  end -- for
```

This example tells DataEase to: (1) define a temporary variable (temp KOUNT) that will be used as a counter, and (2) use a for command to open the MEMBERS table. Since there are no selection criteria, process all the records, and (3) assign the temp KOUNT variable to one. Since the assign command is within the for loop, temp KOUNT is reassigned for each record processed. The next part of the script tells DataEase to: (4) use a while looping command to check the value in temp KOUNT. As long as the count is less than four, continually loop through the list records command, and (5) each time a record is listed, increment temp KOUNT by one. When temp KOUNT reaches 4, exit the while loop and repeat the entire process for each successive record until all the records have been processed.

Tip

Rather than hard code the number of times you want DataEase to loop through the list records, the value can be entered in a Data-entry form. The words DATA-ENTRY FIELDNAME would then replace the number 3 in the above example where fieldname represents the name of the Data-entry field.

If you reverse the placement of the while and for commands in the above example, DataEase runs the entire report as many times as requested. This solution has the same effect as requesting three copies of the report in the Windows Print dialog and checking the Collate Copies option.

Caution

To improve printing speed, whenever possible, avoid using the Collate Copies option in the Windows Print dialog. Avoiding this option is particularly important if your report output includes graphic elements.

How to Create an Accounts Receivable Aging Report

Question

How can I generate an aging report that shows a breakdown of accounts that are 0-30, 31-60, and over 60 days past due?

Solution

1. Use a for command to open the Primary table and specify any required selection criteria using a with clause.
2. Use a list records command to list the fields you want to print.
3. Under the list records command, list the field you want to group (e.g., CUSTOMER NO., MEMBER ID, STUDENT ID) in groups.
4. Use an if function to check each record to see which age group it falls into (e.g., 0-30, 31-60, 60+, or whatever age groups you want to establish).
5. Within each if function use the sum statistical operator to generate the total for each group.

Example 1

```
for INVOICES
  with PAID = NO ;
  list records
  MEMBER ID in groups ;
  LAST NAME ;
  if ( current date - INVOICE DATE < 30, AMT_DUE,
    blank) : sum ;
  if ( current date - INVOICE DATE between 31 to
    60, AMT_DUE , blank) : sum ;
  if ( current date - INVOICE DATE > 60, AMT_DUE ,
    blank) : sum .
end
```

This example tells DataEase to: (1) list all open invoice records grouped by MEMBER ID, (2) check the difference between today's date and the INVOICE DATE to determine the number of days the invoice is past due, and (3) if the records for each member have amounts less than 30 days past due, calculate the sum of those amounts. If the records for each member contain totals between 30 and 61 days past due, calculate the sum of those totals. Finally, if the records for each member contain totals more than 60 days past due, calculate the sum of those totals.

Sample Output 1

Member ID	Last Name	Current	30 - 60	Over60
00001	Birnbaum	0.00	0.00	2,502.00
00002	Perrault	2,720.00	2,900.00	0.00
00003	Christino	7,000.00	0.00	0.00
00004	Williams	0.00	0.00	0.00
00005	Riggs	0.00	3,120.00	0.00
sum		9,720.00	6,020.00	2,502.00

Example 2

```

define temp  "MEMID"  numeric string 5 .
define temp  "XTHIRTY"  number .
define temp  "XSIXTY"  number .
define temp  "XOVER"  number .
define temp  "ACC_THIRTY"  number .
define temp  "ACC_SIXTY"  number .
define temp  "ACC_OVER"  number .
for INVOICES with PAID = no ;
    if MEMID not = MEMBER ID then
        assign temp MEMID := MEMBER ID .
        assign temp XTHIRTY := sum of INVOICES named
            "under 30" with ( MEMBER ID = INVOICES
MEMBER ID
and PAID = NO and current date - INV_DATE < 30 )
AMT_DUE .
        assign temp XSIXTY := sum of INVOICES named
            "thirty-60" with ( MEMBER ID = INVOICES
MEMBER ID
and PAID = NO and current date - INV_DATE
between 31 to 60 ) AMT_DUE .
        assign temp XOVER := sum of INVOICES named
            "over60" with ( MEMBER ID = INVOICESMEMBER ID
and PAID = NO and current date - INV_DATE > 60 )
AMT_DUE .
        assign temp ACC_THIRTY := temp ACC_THIRTY + temp
            XTHIRTY .
        assign temp ACC_SIXTY := temp ACC_SIXTY + temp XSIXTY .
        assign temp ACC_OVER := temp ACC_OVER + temp XOVER .
    end
list records
MEMBER ID in groups with group-totals ;
LAST NAME ;
temp XTHIRTY ;
temp XSIXTY ;
temp XOVER ;
temp ACC_THIRTY ;
temp ACC_SIXTY ;
temp ACC_OVER .

```

Example 2 uses temporary variables to accumulate the individual totals for each member (broken down by day range) and report totals for each day range. This example tells DataEase to: (1) create seven separate temporary variables. The first will be used to track each instance of the group changing (in this example, it holds the current MEMBER ID). The next three variables are used to accumulate the day range values: one accumulates totals less than 30 days past due, the next, 31 to 60 days past due and the last, over 60 days past due. The last three variables accumulate report totals for each of the three day ranges. The next part of the script tells DataEase to: (2) use a for command to open the INVOICES table, and (3) under the for loop, assign each temporary variable whenever the current record represents a new group. This is accomplished by testing the MEMID variable with an if command to see if it contains the same value as the MEMBER ID in the current record. In the rest of the script, (4) whenever the MEMID variable is reassigned, the next three variables (XTHIRTY, XSIXTY and XOVER) are also reassigned. At the same time, the values in the last three variables are incremented to accumulate report totals for each category, and (5) list the MEMBER ID (in groups) and the values in each of the variables.

Sample Output 2

Example 2 can be formatted to produce the identical output as Example 1 shown on the previous page.

Tip

Example 2, though much more complex than Example 1, provides the added benefit of assigning the totals DataEase generates to temporary variables. This lets you post the totals to a separate summary file if desired.

When you choose a layout for the procedure, you should choose a tabular format (the default) and place the temporary variables in the layout.

How to Sort the Values in a Choice List Field

Question

When I try to list a Choice List field in order, DataEase always lists the values in the order they were originally entered in the choice list. How can I sort the text value of the choice rather than the choice number?

Solution

Whenever you sort a Choice List field, DataEase lists the values in order based on the number associated with the choice since the number is what is actually stored in the field. To list in alphabetical order, you must make DataEase treat the choice list field as text. To do this, use a text manipulation function such as jointext, and then sort on this.

Example

```
for MEMBERS ;
    list records
    jointext ( "" , FAVORITE ACTIVITY) in groups ;
    MEMBER ID ;
    LAST NAME .
```

End

This example tells DataEase to: (1) process every record in the MEMBERS table, (2) sort the records by the first word of text value of the choice, and (3) list the FAVORITE ACTIVITY, MEMBER ID, and LAST NAME of each member.

Tip

The easiest solution to this problem is to enter the choices in alphabetic or numeric order when you first create the table. Then maintain the order whenever the list is updated.

Caution

You must exercise extreme care when rearranging the choices in a pre-existing choice list. Failure to update a choice list correctly can result in lost or corrupted data.

How to List the First Several Records in a Table

Question

How can I list just the first ten records in a table?

Solution

1. Use a for command to open the Primary table for the report.
2. Use a list records command to list the fields you want to include in the report output.
3. Use an if command within the list records portion of the script to check the current item number.
4. Use the exit or break command to terminate processing when the current item number exceeds the number of records you want to process.

Example

```

for ACTIVITIES ;
    list records
    ACTIVITY ;
    PICTURE .
        if current item number > 10
            exit
        end -- if
end -- for

```

Tip

In the above example, we used the exit command to terminate DQL processing. If the script included additional processing commands (e.g., a second complete for loop), we would have used the break command instead of exit.

Caution

You may be tempted to test for the current item number in a with clause in script (e.g., with current item number <= 10). However, this solution doesn't work since the current item number variable doesn't begin accumulating its value until the processing phase of the procedure begins (e.g., the list records command in the above example).

How to Find and List Duplicate Records

Question

How can I find duplicate records in a table?

Solution

1. Define a temporary variable to store the key value for any duplicate record.
2. Use a for command to open the Primary table.
3. Use an if command to compare the key field to the temporary variable.
4. Use a list records command to list only those records whose key field matches the value stored in the temporary variable.
5. Use an end command to end the if command and then assign the temporary variable to the next key field value. Remember to use an end command to end the for loop.

Example 1

```
define temp " MEM " numeric string 5 .
for MEMBERS ;
  if MEMBER ID = temp MEM then
    list records
    MEMBER ID in order ;
    last name .
  end -- if
  assign temp MEM := MEMBER ID .
end -- for
```

This script tells DataEase to: (1) define a temporary variable to store the MEMBER ID for each record in the table, (2) sort the records in ascending order by MEMBER ID, and (3) process every record in the members table, comparing the MEMBER ID in the current record to the one stored in the temporary variable. If any records contain the matching MEMBER ID, (4) list the MEMBER ID and LAST NAME of the member, and (5) assign the temporary variable to the current MEMBER ID and repeat the process.

Example 2

```
for MEMBERS
  with count of MEMBERS with ( MEMBER ID = MEMBERS MEMBER ID ) >
  1 ;
  list records
  MEMBER ID in order ;
  LAST NAME . end
```

Example 2 tells DataEase to: (1) open the MEMBERS table, (2) for each record, count the number of records with the matching MEMBER ID number, and (3) list the MEMBER ID and LAST NAME of each record whose MEMBER ID matches the MEMBER ID stored in another record.

Discussion

Example 1 lists only the duplicates while Example 2 lists all the records that share the same MEMBER ID. Because Example 2 must count the records in the table repeatedly (once for each record in the table), this process can be quite slow, particularly on a very large table. For this reason, we recommend that you use Example 1 whenever possible. However, if you want to see the original record in addition to the duplicates, you must follow Example 2.

Tip

To avoid entering duplicate records, be sure to include a Primary key (a unique identifier for each row in the table) in each table. Your Primary key should always be defined as Unique, Indexed, and Required.

Caution

The key field in Example 1 must be listed either in order or in groups since the if command only evaluates the current MEMBER ID against the previous one.

Chapter 7 : SQL Tech Tips

Using DataEase with SQL Data

The following pages describe how you can write DQL which works efficiently with SQL engines.

How to.....

Control Transactions in a DQL Procedure	130
Join Tables in a DQL Script	131
Specify an Explicit Inner Join in a DQL Procedure	132
Join Tables Stored on Different Engines	133
Join Two Independent One-to-Many Relationships	134
Avoid Redundant Processing in a DQL Procedure	135
Improve Processing Speed by Creating a View from Which to Query the Data	136
Use Stored Procedures in a DQL Script	138
Replace Explicit DQL Locking Commands by Creating a Semaphore	139
Avoid Deadlocks	140
Detect Deadlocks	142
Prevent Deadlocks	143
Improve Performance by Avoiding Inconvertible DQL Expressions	145

How to Control Transactions in a DQL Procedure

Question

When I write a DQL Procedure that accesses SQL data, DataEase seems to treat the whole procedure as a single transaction. Is there any way around this?

Solution

1. Divide the procedure into logical transactions.
2. Use the DataEase begin transaction and commit commands to tell DataEase which portions of the script constitute those individual transactions.

Example

```

    for RESERVATIONS ;
    begin transaction
        enter a record in NEW_RESERVATIONS
        copy all from RESERVATIONS .
    commit
end -- for

```

This example tells DataEase to: (1) open the Native RESERVATIONS table and process all the records, (2) issue a begin transaction command to the server, (3) enter a new record in the NEW_RESERVATIONS table on the server, and (4) commit each record as it is entered.

Discussion

As a DQL Procedure modifies records, the modified records are immediately available for use as part of a condition or operation. However, there may be times when you want to save groups of changes as they are completed. In this case, you can use the DQL begin transaction and commit commands to divide the procedure into a group of smaller transactions. If you do not use the begin transaction and commit commands, DataEase treats the entire procedure as a single transaction.

For example, in the above sample script, if the server determines that a record is a duplicate, that individual transaction fails. Without the begin transaction and commit statements, if the insertion of record number 197 of 200 fails, all 196 previous insertions are rolled back.

The purpose of this script is twofold. This script allows you to enter duplicate records since DataEase doesn't require you to create a unique field in a Native table. This script also transfers the RESERVATIONS data from the Native table to a new table on the server and simultaneously eliminates duplicate records since they are not allowed on the server.

Tip

You may also find it useful to divide some of your larger procedures into a series of small procedures which are executed by a single control procedure.

How to Join Tables in a DQL Script

Question

How do I instruct DataEase to perform any necessary table joins when I write a DQL Procedure?

Solution

DataEase performs the join automatically. The type of script you define determines the type of join that DataEase performs on the server. Except for the following case, DataEase performs a left outer join on any tables that are joined in a script.

Example

When you define a script that contains nested for commands, with no intervening DQL statements, DataEase performs an inner join on the two tables.

```
for ACTIVITIES ;
  for MEMBERS
    list records
      ACTIVITIES ACTIVITY;
      LASTNAME
      MEMBERID .
  end
end
```

This example tells DataEase to: (1) process all the ACTIVITIES records, and (2) for each ACTIVITIES record, list the name of the activity and the LAST NAME and MEMBER ID from each related MEMBERS record.

The result of the join is a list of activities that currently have members who favor the activity. Any activities without members who list that activity as their favorite are not listed. Additionally, any members who haven't indicated a favorite activity are not listed.

How to Specify an Explicit Inner Join in a DQL Procedure

Question

Is there an easy way to force DataEase to perform an inner join when it would normally perform a left outer join?

Solution

When you write the script, create an ad hoc relationship to select the related records. Name the relationship and enter the keyword inner: before the relationship name, as demonstrated in the following example.

Example

```
for MEMBERS ;
list records
any ACTIVITIES named "inner: favorites" with ( ACTIVITY =
MEMBERS FAVORITEACTIVITY) ACTIVITY;
MEMBERID ;
LASTNAME .
end
```

This example tells DataEase to process all the MEMBERS records but list the ACTIVITY, MEMBER ID, and LAST NAME from only those MEMBERS records that have at least one related ACTIVITIES record.

Discussion

Whenever a procedure requires data from two different database tables, DataEase temporarily joins (merges) the two tables. Tables can be joined by an outer join that selects all parent (Primary form) records and all child (related form) records or an inner join that selects only those parent records with at least one matching child record.

By default, DataEase uses a left outer join to join two tables (selects all parent records regardless of whether any matching child records exist). To make DataEase use an inner join instead of a left outer join, simply insert the prefix inner: immediately before the unique relationship name following the named relational operator.

Tip

As an alternative, you can specify inner: as a prefix in the Optional Relationship Name field on the Relationships form.

How to Join Tables Stored on Different Engines

Question

How can I create a procedure that lists or manipulates data stored on two different types of database engines?

Solution

When you write a script like the example shown below, and the data is stored on two separate servers (regardless of engine type), DataEase must perform the join on the workstation. Depending on the number of records the script selects, performing a join on a workstation can have a significant impact on performance. If there are any selection criteria, the selection is performed by the SQL engine before the join is performed by DataEase at the workstation.

Example

```
for ACTIVITIES ;  
    list records  
    ACTIVITY in groups ;  
    all MEMBERS MEMBER ID.  
end
```

This example tells DataEase to: (1) open the ACTIVITIES table and group the records by activity, and (2) list the name of the activity and the last names of all the members who chose that activity as their favorite. The processing required to find the matching member records must be performed on the workstation because the MEMBERS and ACTIVITIES tables are stored on different servers.

Caution

Any time DataEase has to perform a join on a workstation, performance is affected. If the script selects a large number of records, the impact on performance can be significant. If you find that you are frequently joining tables across servers, you may find it worthwhile to move a table from one server to another to improve performance.

How to Join Two Independent One-to-Many Relationships

Question

What happens internally when I write a script that contains nested for commands that access two separate one-to-many relationships?

Solution

This scenario generates a disjoint report. A disjoint report combines information from two or more independent relationships. If your script requests data from two (or more) independent one-to-many relationships, DataEase performs the first join on the server. Any additional processing that is required is done at the workstation. Depending on the number of records the script selects, a disjoint report can have a significant impact on performance.

Example

```
for ACTIVITIES
    list records
    ACTIVITY in groups ;
    all MEMBERS LASTNAMEin order ;
    all CLUB ACTIVITIES CLUBNAMEin order .
end
```

This example tells DataEase to: (1) process all the ACTIVITIES records listing the activity in groups, and (2) for each activity, list the last names of all the members who favor that activity along with the names of all the clubs that offer the activity.

Discussion

The ACTIVITIES table is related to both the MEMBERS and the CLUB ACTIVITIES tables, but these two tables are not related to each other.

When this script is processed, DataEase joins ACTIVITIES and MEMBERS using the SQL engine, but the additional processing needed to join the ACTIVITIES and CLUB ACTIVITIES data must be performed at the workstation.

If one of the one-to-many relationships was instead a one-to-one relationship, in the last all statement or you used the highest of or any statement (under the list records command), and all the data was stored on the same server, both joins could be performed on that engine.

How to Avoid Redundant Processing in a DQL Procedure

Question

How do I avoid making DataEase request the same data repeatedly in a script?

Solution

When processing a procedure, DataEase is sometimes forced to request the same records more than once from the SQL engine. If you specify an action that causes DataEase to perform redundant processing, the impact on performance can be significant. Two examples that cause redundant processing are shown in this tip.

Example 1

```
for RESERVATION SUMMARY ;
    list records
    MEMBERID in groups ;
    TOTALSPENT;
    all RESERVATIONS LASTNAME;
    sum of RESERVATIONS TOTALDUE.
end
```

This script tells DataEase to: (1) process all the records in the RESERVATION SUMMARY table listing the MEMBER ID in groups and the TOTAL SPENT by that member, and (2) for each record in RESERVATION SUMMARY, find all the related records in RESERVATIONS, list the last name of each member, and calculate the total amount of all their open reservations. This script requests the same aggregate twice. TOTAL SPENT is a Virtual field on the RESERVATION SUMMARY form that is calculated by totaling the values in the TOTAL DUE field in all the related records in the RESERVATIONS table.

Example 2

```
for RESERVATION SUMMARY ;
    if TOTALSPENT > 5000 then
        for RESERVATIONS
            list records
            MEMBERID;
            LASTNAME;
            CLUBNAME.
        end -- for
    end -- if
end -- for
```

In Example 2, performance is reduced because the procedure contains a nested for command inside a DQL conditional statement. If a procedure contains a for or all command inside a case, if, or while statement, DataEase first tests each record against the condition and then requests the record again from the SQL engine when processing the nested action(s).

If you instead include the criteria in the if statement as selection criteria in the for statement (using a with clause), DataEase only has to issue one SELECT statement for the whole file instead of multiple selects for each record.

How to Improve Processing Speed by Creating a View from Which to Query the Data

Question

When I write a script that requests just a few columns from an SQL table that contains many columns, the processing speed seems to be extremely slow. How can I speed it up?

Solution

When you request data from an SQL table that contains many columns, regardless of how many columns you request, DataEase returns all the columns for each record selected. This can have a significant impact on performance.

If the script is one that you run frequently, or if you have several DQL Procedures that require the same or similar information from such a table, you will improve processing speed dramatically by creating a view that contains just those few columns and updating your scripts to reference the view rather than the original table. For example, let's assume that Club ParaDEASE buys a mailing list of potential members and sends a detailed questionnaire out to each individual on the list. Only those individuals who respond are then entered into a new SQL table called QUESTIONNAIRE. Assuming that the QUESTIONNAIRE table contains 200 columns, the two scenarios that follow would benefit greatly from the creation of separate views based on the QUESTIONNAIRE table.

Example 1

Suppose you want to run a procedure that produces a letter, thanking each respondent for their participation in the survey. This procedure requires only six of the 200 fields (e.g., FIRST_NAME, LAST_NAME, STREET, CITY, STATE, and ZIP). Performance would be greatly improved by creating a view that contains just the above-mentioned fields and writing the following script:

```
for Q_VIEW1
    list records
    FIRST_NAME;
    LAST_NAME;
    STREET ;
    CITY;
    STATE;
    ZIP in order .
end
```

Example 1 tells DataEase to select and list all the records from the view called Q_VIEW1, sorting them in zip code order. This script will perform noticeably faster than one that uses QUESTIONNAIRE, since there are only six columns for DataEase to return from the server.

Example 2

In the second scenario, let's suppose you want to produce a report that lists the percentage of Yes and No answers to each question on the questionnaire. Rather than create a DQL Procedure to access the QUESTIONNAIRE table and calculate the percentages using DQL, performance would be greatly improved by creating a view that generates the requested percentages and accesses the view in the script, as shown below:

```
for Q_VIEW2 ;
    list records
    Q1PERCENTY ;
    Q1PERCENTN ;
    Q2PERCENTY ;
    Q2PERCENTN;
    ...
end
```

This example tells DataEase to select all the records from a view called Q_VIEW2, listing the percentages that the server has already generated.

Tip

Views are also frequently used to improve performance in certain record entry situations. For example, views are recommended if you frequently use the dynamic lookup feature to retrieve related data.

A dynamic lookup brings back all the fields from the related table. If any of those fields are virtual calculations or lookups, many additional SELECT statements are issued, slowing data access time dramatically. A more efficient strategy is to create a view that displays just the fields necessary to select the appropriate record.

How to Use Stored Procedures in a DQL Script

Question

Since I am accessing data on Sybase SQL Server, there are a number of stored procedures I would like to be able to execute when I'm running a DQL Procedure. Is there an easy way to do that?

Solution

Use the exec SQL command within your script to execute the required stored procedure.

Example

```
exec SQL at CONN1 EXECUTE ARCHIVE_OLD_MEMBERS
WHERE EXPIRATION_DATE <': current date' ;
```

...where CONN1 is the Connect_ID that identifies the source server and database. This example tells DataEase to execute a stored procedure called ARCHIVE_OLD_MEMBERS.

Discussion

Some SQL engines, including Microsoft/Sybase SQL Server, let you define and save stored procedures (also called precompiled procedures). Stored procedures are collections of SQL statements that are compiled the first time they are executed and stored in a precompiled format so that subsequent executions of the procedure are very fast. If you are using DataEase with an SQL engine that saves stored procedures, you can use the exec SQL command to run a stored procedure.

Note: Since the execution of a stored procedure is almost instantaneous, stored procedures are an ideal way to enhance performance. We strongly recommend that you use stored procedures whenever possible.

To find out if your SQL engine supports stored procedures, see your DataEase Engine Information Guide and your SQL engine documentation.

Tip

When you use the DataEase Backup Application and Restore Application menu options (or the equivalent DQL commands), DataEase only backs up the application definition files and data. Any stored procedures or other server operations associated with the application are not backed up. For this reason, we recommend that you define all the stored procedures, rules, etc. that your application requires as DQL Procedures (using the exec SQL command).

Once these features exist as DQL Procedures, they will be backed up with the application and can be easily migrated to a different server or database. After you restore your application, you just have to run the appropriate DQL Procedures to recreate the performance-enhancing SQL features.

How to Replace Explicit DQL Table and Field Locking Commands in a Procedure by Creating a Semaphore

Question

Can I use DQL locking commands to lock data on my SQL engine?

Solution

The locking commands available in DQL control locking in Native DataEase forms only. When used with SQL tables, the DQL locking commands are ignored. If there are multiple users accessing the same SQL tables, all required locks are placed by the SQL engine. There are no DataEase options or DQL commands that let you explicitly control the locks placed by the SQL engine.

One way to lock a procedure for exclusive access is to define a semaphore. A semaphore is a flag used to regulate the number of users who can simultaneously access a network resource. The following examples describe how you can define a semaphore that lets only one user at a time run a procedure.

Example 1

Defining a DataEase-based Semaphore. If the only users who can access the SQL tables in the application are DataEase users signed on to the same application, you can use a DataEase-defined semaphore to lock a procedure for exclusive access. To define a DataEase-based semaphore:

1. Define a Native DataEase table. This table needs just one 1-character field.
2. Modify each procedure that you want to lock exclusively to include the following query statements:

```
lock file NATIVEFORM exclusive.
    ( Query statements for
      the procedure that you want to
      to lock exclusively)
```

```
unlock file NATIVEFORM. (or end the procedure)
```

...where NATIVEFORM is the Native table you defined in Step 1

When you include the lock and unlock commands in the query, only one user at a time can execute the procedure. The first user who runs the procedure places an exclusive lock on the Native DataEase table that you defined in Step 1

Any other users who try to run this procedure are locked out until the lock placed on NATIVEFORM is released (this happens when either the unlock command is executed or the procedure has finished running).

Example 2

Defining an SQL Engine-based Semaphore. If other DataEase applications and/or other front-end products will access the SQL tables in the current application, you must use an SQL engine-defined semaphore to lock a procedure for exclusive access. To define an SQL engine-based semaphore:

1. 1 Define an SQL table in a database that all users can access. Enter one row of data into the table.
2. 2 Modify each procedure that you want to lock exclusively to include the following query statements:

```
exec SQL at CONN1 BEGIN TRANSACTION ;
exec SQL at CONN1 UPDATE TABLEA SET COLUMN1 = COLUMN1 + 1;
    (Query statements for
    the procedure that you want to
    to lock exclusively)
exec SQL at CONN1 COMMIT TRANSACTION;
```

..where CONN1 is the name of the Connect_ID that identifies the SQL Server and database and TABLEA is the name of the SQL table you defined in Step 1 and COLUMN1 is a column_in this table.

When you include the exec SQL commands in the query, only one user at a time can execute the procedure. The first user who runs the procedure updates the single row in the SQL table that you defined in Step 1, thus locking the SQL table exclusively. Any other users who try to run this procedure are locked out until the exclusive lock on the table is released (this only happens when the COMMIT TRANSACTION command is issued to the server).

Deadlocks

The three Tech Tips that follow are concerned with deadlocks and how to prevent them.

A deadlock occurs when two users simultaneously try to perform an action that requires an exclusive lock on a resource which one user has already placed a shared lock on. When a deadlock occurs, the SQL engine grants the required exclusive lock to one of the users based on engine-specific criteria. For example, if you are using Microsoft SQL Server, the exclusive lock is granted to the user who has logged the most CPU time prior to the deadlock.

The three methods of dealing with the possibility of deadlocks that occur during transaction processing are:

- Query Standardization
- Deadlock Detection
- Deadlock Prevention

Each method is explained below and on the following pages.

How to Avoid Deadlocks by Duplicating the Order in Which Tables Are Accessed in Multiple Scripts

Query Standardization is the process of developing a standard order in which tables and rows are accessed and updated in a script.

When you define two or more procedures that update the same tables and can be executed at the same time, try to write each script so that all the tables are accessed and updated in the same order. If your scripts access tables in random order, deadlocks may occur.

If you standardize on the order in which tables are accessed in queries, you can solve many of the deadlock problems you may be experiencing.

How to Detect Deadlocks

Deadlock Detection is a technique in which a transaction is placed inside a while loop in order to ensure that processing breaks off after a set number of attempts if a deadlock is detected while DataEase is processing a transaction.

To detect whether a deadlock has occurred while DataEase is processing a transaction, you can set up a series of steps that verify whether the transaction is committed successfully or not. If the commit is successful, the procedure should run to completion; otherwise, the query should deal with the failure of the transaction. This technique ensures that a deadlock cannot last indefinitely.

The procedure for detecting deadlocks uses the current status and current SQLCODE variables and two temporary variables to determine the status of the update and commit and how many attempts were made to complete the transaction. The following example describes how to detect deadlocks in DQL Procedures.

Example

```
define temp  " COMSTAT  "  number .
define temp  " KOUNT    "  number .
assign temp COMSTAT := -1 .
assign temp KOUNT  := 0 .
while temp COMSTAT = -1 do
  begin transaction
  for SAVINGS ACCOUNTS ;
    if current SQLCODE = 1205 then
      break
    else
      modify records
      ACCOUNT BALANCE := ACCOUNT BALANCE - DEBITS .
    end -- 1st if
  commit .
  assign temp COMSTAT := current status .
  assign temp KOUNT := + 1 .
  end -- for
  if temp KOUNT > 10 then
    message " Sorry, DQL transaction
    can't be processed now. " window .
    exit .
  end -- 2nd if
end -- while
```

The first two lines in the query read:

```
define temp  "COMSTAT"  number .
define temp  "KOUNT"    number .
```

This part of the script tells DataEase to define two temporary variables. The first variable, COMSTAT, will be used to determine if the preceding DQL commit command is successful. The second variable, KOUNT, will be used to count the number of times DataEase attempts to commit the DQL transaction.

The next part of the query reads:

```
assign temp COMSTAT := -1 .
assign temp KOUNT := 0 .
```

This part of the script assigns a value of -1 to the COMSTAT variable and a value of 0 to the KOUNT variable. Subsequently, when DataEase processes the following while statement, the value of the COMSTAT variable is replaced by the value in the current status variable. The current status value is -1 if the transaction is not committed; 0 if it is committed.

The next part of the script reads:

```
while temp COMSTAT = -1 do
  begin transaction
  for SAVINGS ACCOUNTS ;
    if current SQLCODE = 1205 then
      break .
    else
      modify records
      ACCOUNT BALANCE := ACCOUNT BALANCE - DEBITS ;
    end -- if
  commit
  assign temp COMSTAT := current status .
  assign temp KOUNT := temp KOUNT + 1 .
```

This part of the script tells DataEase to set up a while loop that will keep trying to process the following DQL transaction until it is either committed or the value in the KOUNT variable exceeds 10. The current SQLCODE variable is used to break out of the for loop if a deadlock has occurred (1205 is the error code for deadlock on Microsoft SQL Server). As long as the value of the COMSTAT variable remains -1, the value of the KOUNT variable is increased by one each time DataEase processes the while statement.

When the DQL transaction is committed or the value in the KOUNT variable exceeds 10, DataEase breaks out of the while loop and continues processing the remainder of the query.

The lines:

```
begin transaction for SAVINGS ACCOUNTS ;
  if current SQLCODE = 1205 then
    break .
  else
    modify records
    ACCOUNT BALANCE := ACCOUNT BALANCE - DEBITS ;
  end -- if
commit
```


..are the actual DQL transaction that we are trying to process without running into a deadlock. The begin transaction and commit commands tell DataEase to process the entire for loop as a single transaction. The for statement tells DataEase to modify all the records in the SAVINGS ACCOUNT form by subtracting the DEBITS amount from the ACCOUNT BALANCE amount.

Since all the modifications are treated as a single transaction, if any single modification fails (in the case of a deadlock or any other reason), the entire transaction is rolled back. If the transaction cannot be committed, the COMSTAT variable retains the value of -1 and the KOUNT variable is incremented by one.

The next part of the script reads:

```

if temp KOUNT > 10 then
  message " Sorry, DQL transaction|
  can't be processed now. " window .
  exit .
end -- if
end -- while

```

This section tells DataEase to look at the value in the KOUNT variable each time the while statement is processed. If the value of the KOUNT variable exceeds 10, DataEase displays a message stating: Sorry, DQL transaction cannot be processed now.

If the value of the COMSTAT variable becomes zero before the value of the KOUNT variable exceeds 10, DataEase exits out of the while loop.

How to Prevent Deadlocks

To prevent a deadlock from occurring while DataEase is processing a transaction, you can use a technique called gating. Gating prevents deadlocks by requiring that each user who runs the procedure acquires an exclusive lock on a special table before he/she can execute the procedure. Although this technique prevents deadlocks, it reduces concurrency and should be used only where recovery from a deadlock cannot be ensured.

The procedure for preventing deadlocks uses the DQL exec SQL command to create a barrier action that blocks other users from accessing the data to be modified by the transaction.

Example

```
exec SQL at CONN1 BEGIN TRANSACTION ;
exec SQL at CONN1 UPDATE BARRIER_FORM
SET BLOCK_FIELD = BLOCK_FIELD + 1;
for SAVINGS ACCOUNTS ;
    modify records
    ACCOUNT BALANCE := ACCOUNT BALANCE - DEBITS .
end

exec SQL at CONN1 COMMIT TRANSACTION ;
```

...where CONN1 is the name of the Connect_ID that identifies the SQL Server and database where the data is stored.

The first two script statements read:

```
exec SQL at CONN1 BEGIN TRANSACTION ;
exec SQL at CONN1 UPDATE BARRIER_FORM
SET BLOCK_FIELD = BLOCK_FIELD + 1;
```

This part of the script uses the exec SQL command to initiate a transaction (but not the actual DQL transaction we want to protect against a deadlock). This preliminary SQL transaction updates the BLOCK FIELD in every record in a table named BARRIER FORM. This form should only contain one record.

Note: Once a user gets an exclusive lock on the BARRIER FORM, no other DataEase user can get past this barrier until the entire exec SQL transaction is committed. However, a user of another front-end product or database application may still be able to lock the table we want to modify before we gain access to it.

The next part of the script reads:

```
for SAVINGS ACCOUNTS ;
  modify records
  ACCOUNT BALANCE := ACCOUNT BALANCE - DEBITS .
end -- for
```

This is the actual DQL transaction that we are trying to process without running into a deadlock. The for statement tells DataEase to modify all the records in the SAVINGS ACCOUNT table by subtracting the DEBITS amount from the ACCOUNT BALANCE amount.

The next part of the query reads:

```
exec SQL at CONN1 COMMIT TRANSACTION ;
```

Once the DQL transaction is completed, this statement commits the SQL transaction and releases the lock on the BARRIER FORM. At this point, other DataEase users can access the BARRIER FORM.

Caution

Although the DQL transaction cannot be deadlocked by another DataEase user in the same application at this point, it may still be impossible to commit the transaction because a non-DataEase user has locked the data, or the server is temporarily down, etc.

How to Improve Performance by Avoiding Inconvertible DQL Expressions

Question

What DQL expressions, if any, should I avoid when writing scripts?

Solution

When you define a DQL Procedure or a Quick Report, you should try not to select or sort data based on inconvertible DQL expressions. Inconvertible DQL expressions are expressions that currently have no SQL equivalent, i.e., they cannot be directly converted to SQL expressions. If you use an inconvertible expression as part of the selection criteria in a DQL Procedure, DataEase uses the SQL engine to retrieve the records that meet all the criteria except the inconvertible expression(s). This set of records is then returned to the workstation where the Native DataEase engine completes the processing that includes the inconvertible expression(s). Depending on the number of records the script selects, using inconvertible expressions can have a significant impact on performance.

Expressions that include any of the following DQL symbols or words are inconvertible:

ampm	item	presentvalue	std. err.
cosh	julian	proper	tanh
count	lastfirst	rate	textpos
count of	lastw	sinh	timeampm
firstc	lower	spellcurrency	upper
firstlast	midc	spelldate	variance
firstw	midw	spellmonth	~ (Soundex Wild Card Symbol)
futurevalue	mod	spellnumber	
if	percent	spellweekday	
installment	period	std. dev.	

Example

```

for MEMBERS with LAST NAME = " ~ * SON " and
    TOTAL DUE > 35 ;
    list records
    LASTNAME;
    all FAMILY MEMBERS FIRSTNAME.
end

```

This example tells DataEase to: (1) select all the records in the MEMBERS table where the end of the last name sounds like SON (e.g., Hansen, Johnson, Wilson) and the membership dues are more than \$35.00, (2) list the last name of each selected member, and (3) list the first name from all the related records in the FAMILY MEMBERS table.

If the ~ wild card symbol does not have an equivalent operation on your SQL engine, DataEase first retrieves all the MEMBERS records that have membership dues greater than \$35.00 from the server. Then the Native DataEase engine must perform the processing that selects only the records that meet the "sounds like" selection criteria.

Chapter 8 : DQL Lexicon

Alphabetical Language Reference

The DQL Lexicon is a complete list of all the DataEase Query Language concepts, terms, and symbols presented in alphabetical order. In this lexicon, most entries begin on a separate page. However, some entries continue for two or more pages. Each term is explained in the following format (only relevant sections are included):

Type

Identifies the type of the term. DQL terms are divided into four main groups: commands, functions, operators, and symbols. Each of these groups contains several subordinate types.

Purpose

Explains the general purpose of the term.

Syntax

Demonstrates the proper format in which the term is used. For an explanation of the syntax format itself, see Typographical Conventions later in this chapter.

Returns

Specifies the type of value returned by a function or operator.

Usage

Provides additional details concerning the use of the term in general and special contexts.

LAN

Includes special considerations or restrictions that govern the use of the term when DataEase is running on a LAN (Local Area Network).

Example

Demonstrates the use of the term in a DQL statement or script. When a complete script is presented, an explanation of the script is included. If the script generates output, an example of how the resulting output might appear is included (see example below).

Last Name	First Name	Middle Initial
Andersen	Eric	M
Anderson	Eric	S

A list appears in the right margin showing where you can use a DQL keyword. For example, the figure below shows the lists for the count of and commit commands.

Use in a...	This list shows where you can use the COUNT OF operator.	Use in a ...	Some DQL terms such as COMMIT and ROLLBACK are used only when your procedure accesses data stored in a SQL database. Such terms are marked SQL ONLY.
DQL Script Formula QBF/QBM	The checkmarks indicate that you can use COUNT OF in DQL script, Derivation Formula, or Validation Formula, but not in Query By Form or Query By Model.	DQL Script Formula QBF/QBM SQL Only	

DQL Lexicon Typographical Conventions

Except as noted below, the typographical conventions used in this lexicon are the same as those used in the previous chapters. These conventions are explained in Chapter I, DQL Basics.

Syntax Diagrams

In the syntax diagrams, items enclosed in brackets [] may or may not be included in the statement. If they are included, the quotation marks and parentheses must appear where shown. When multiple items are separated by a vertical bar |, any one of them may be used. For example, in the syntax statement:

```
any TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria ) ] FIELDNAME ; | .
```

You can use either a TABLENAME or a RELATIONSHIP in the script statement. A "UNIQUE RELATIONSHIP NAME" can be included or not. If selection criteria are included in the statement, the criteria must be enclosed in parentheses. A FIELDNAME must be included at the end of the statement, followed by either a comma or semicolon.

In the syntax diagrams, the terms shown in lower case (any, named, and with) are DQL keywords. The words shown in upper case (FORMNAME, TABLENAME, RELATIONSHIP, UNIQUE RELATIONSHIP NAME, and FIELDNAME) are generic terms. In an actual script you replace such terms with specific data by typing in the actual database names or selecting the appropriate items from the Script Editor pick lists.

Most examples in this lexicon are based on the MEMBERS and RESERVATIONS tables, included in the **Club ParaDEASE** sample application.

Symbols

Symbols are tools that clarify a script's meaning, perform basic math operations, and help retrieve data when you cannot recall the exact values you want a query to retrieve. The symbols listed below can be used in Field Derivation Formulas as well as in a DQL script.

There are four groups of DQL symbols as specified below:

- Punctuation symbols are used to clarify, separate, and conclude various operations specified in a script. The double dash symbol precedes comments used to annotate a script.
- Math symbols are used to perform arithmetic operations on numeric values.
- Wild card symbols are used to specify unknown characters in selection criteria text values.
 - * (asterisk) represents any number of characters
 - ? (question mark) represents a single character
 - ~ (tilde) represents a sound pattern
- Comparison symbols are used to compare one value to another. The assignment operator symbol is used to assign a value to a field or variable.

For a full explanation of any DQL symbol, see its corresponding entry in this section. The symbols are presented first in the lexicon, beginning on the next page.

+ (addition)

Type

Arithmetic Symbol

Purpose

The + symbol tells DataEase to add Value 2 to Value 1

The result is the sum of the addition.

Syntax

VALUE 1 + VALUE 2

Returns

A numeric value.

Example

TOTAL DUE := ADULT FEES + CHILD FEES ;

- (subtraction)

Type

Arithmetic Symbol

Purpose

The - symbol tells DataEase to subtract Value 2 from Value 1.

The result is the remainder of the subtraction.

Syntax

VALUE 1 - VALUE 2

Returns

A numeric value.

Example

SALE PRICE := RESERVATION PRICE-DISCOUNT;

/ (division)

Type

Arithmetic Symbol

Purpose

The / symbol tells DataEase to divide Value 1 by Value 2

The result is the quotient of the division.

Syntax

VALUE 1 / VALUE 2

Returns

A numeric value.

Usage

When typing a fraction (10 / 12, for example), DataEase requires that you type a space before and after the / symbol.

Example

```
MONTHLY FEES := ANNUAL FEES / 12 ;
```

*** (multiplication)**

Type

Arithmetic Symbol

Purpose

The * symbol tells DataEase to multiply Value 1 by Value 2

The result is the product of the multiplication.

Syntax

VALUE 1 * VALUE 2

Returns

A numeric value.

Example

```
CHILD FEES := NO OF CHILDREN * 15.00
```

* (asterisk)

Type

Wild Card (Character Pattern) Symbol

Purpose

The * asterisk wild card character can be used to substitute for any number of unspecified characters in an alphanumeric string. It can be used up to twice in a string to show that an unspecified number of characters may have been omitted. The symbol can be used in any logical comparison or record selection context.

Example 1

```
for MEMBERS with LAST NAME =   "~Read"   ;
    list records
        LAST NAME in order ;
        FIRST NAME .
End
```

This script tells DataEase: List the records for members whose last name ends with the letters son. The output for this script, in alphabetical order by LAST NAME, might look as follows:

Last Name

Anderson
Carlson
Christenson
Jackson
Johannsson

Example 2

```
for MEMBERS with LAST NAME =   "*s?n"   ;
    list records
        LAST NAME in order .
end
```

This script tells DataEase: List the members whose last name ends with the letters s and n, with an unknown value between them. The report output for this script might look as follows:

Last Name

Andersen
Anderson
Carlson
Christenson
Jackson
Johannsson

? (question mark)

Type

Wild Card (Single Character) Symbol

Purpose

The ? wild card character is used to substitute for exactly one unspecified character in an alphanumeric string. It can, however, be used repeatedly in a field to specify the exact number of characters that have been omitted. The symbol can be used in any logical comparison or record selection context.

Example 1

```
for MEMBERS with LAST NAME = "Anders?n" ;
  list records
    LAST NAME in order ;
    FIRST NAME ;
    MIDDLE INITIAL .
end
```

This script tells DataEase: list the records for members named Andersen or Anderson (it also lists members named Andersan, Andersin, or Andersun, if any exist). The output for this script, in alphabetical order by LAST NAME, might look as follows:

Last Name	First Name	Middle Initial
Andersen	Eric	M
Anderson	Eric	S

Example 2

```
for MEMBERS with LAST NAME = "W?s*" ;
  list records
    LAST NAME in order .
end
```

This script tells DataEase: list the members whose last name contains a W followed by a single character, then s, then an unknown number of unknown characters. The output from this script might look as follows:

Last Name
Wasserman
Washburn
West
Wyshner

~ (tilde)

Type

Wild Card (Sound Pattern) Symbol

Purpose

The ~ wild card character is used to find data that "sounds like" a specified text string. The ~ symbol can be used before any number of characters or words; it locates all the records that have a similar sound to the specified string. In making comparisons with the ~, DataEase matches the consonant pattern; vowels and spaces between words are ignored. Consonants with similar sounds (such as f and ph) are treated as the same character. The ~ symbol can be used in any logical comparison or record selection context and may be used in a text string after another wild card symbol (^ or ?).

Example

```
for MEMBERS with LAST NAME = "~Read" ;
  list records
    LAST NAME in order ;
    FIRST NAME .
end
```

This script tells DataEase: list the records for members whose last name sounds like Read. The report output for this script, arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	First Name
Rada	Amanda
Reardon	Paul
Redzepi	Zudi
Reede	Shannon
Rhode	Gertrude
Rhodes	Gisela
Ride	Sheryl

: (colon)

Type

Punctuation Symbol

Purpose

A colon is inserted between a field name and one or more statistical operators when you want to include statistics in the report output.

Syntax

```
FIELDNAME : statistical operators ;|.
```

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item sum .
end
```

This script tells DataEase: list the MEMBERS records showing each member's LAST NAME, TOTAL DUE, and the total of all the TOTAL DUE values at the end of the report. The report output for this script, arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	Total Due
Adams	85.00
Albert	120.00
Anders	120.00
Andersen	70.00
Anderson	115.00
Archer	155.00
Baldwin	100.00
 TOTAL:	 \$765.00

() (parentheses)

Type

Punctuation Symbol

Purpose

Parentheses are used as separators in selection criteria and function arguments and to clarify the sequence of mathematic operations. The use of parentheses is especially important when combining selection criteria using the and and or operators or when clarifying the order of evaluation of otherwise ambiguous expressions and operations.

Example

```
for MEMBERS with TOTAL DUE > 75 and
  ( STATE = "NY" or STATE = "NJ" ) ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    STATE .
end
```

With the parentheses placed as shown, this script tells DataEase to list all the members from either New York or New Jersey whose TOTAL DUE is greater than \$75

The report output from this script, arranged alphabetically by LAST NAME, might look as follows:

Last Name	Total Due	State
Baldwin	100.00	NY
Crandall	85.00	NY
Fitzpatrick	100.00	NY
Morales	115.00	NY
Morrison	100.00	NJ
Parker	105.00	NJ
Rutschow	100.00	NY
Stone	100.00	NY

Without parentheses, this script is ambiguous (it could be interpreted as all New York members whose total due is greater than \$75 and all New Jersey members).

In the absence of parentheses, the default order of evaluation is: multiplication and division operations, addition and subtraction operations, comparison operators, and finally, the and and or operators. When operations of equal priority are involved, the expression is evaluated from left to right. When evaluating expressions in nested parentheses, the innermost expressions are evaluated first.

. (period)

Type

Punctuation Symbol

Purpose

A period marks the end of an action that may or may not be followed by other actions.

In a script, you must insert a period after:

- Any DQL Control command.
- Any assign, break, define, or exit Procedural command and after each action in a case, if, or while command.
- The last item in a list records, modify records, delete records, or enter a record Processing command.
- Except when the period is used as a decimal point in a numeric value, you must insert a space between an action item and the terminating period.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE .
end
```

This script tells DataEase: list each member's LAST NAME and TOTAL DUE. The report output, arranged in alphabetical order by LAST NAME, might look like this:

Last Name	Total Due
Adams	85.00
Albert	120.00
Anders	120.00
Andersen	70.00
Anderson	115.00
Archer	155.00
Baldwin	100.00
Beauchamp	35.00

; (semicolon)

Type

Punctuation Symbol

Purpose

A semicolon is used to separate items in a script action. The semicolon is always used after a for command that specifies the script's Primary table; after ad hoc selection criteria are specified; and after each output item except the last in a list records, modify records, or enter a record command.

The semicolon is used after a for command that specifies a script's Primary table; it is not used with a nested for command that specifies a table other than the Primary table.

Syntax

```
FIELDNAME|CONSTANT VALUE ;
```

Example

```
for MEMBERS with TOTAL DUE > 180 ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    STATE .
end
```

This script tells DataEase: For members whose TOTAL DUE is greater than \$180, list each member's LAST NAME, TOTAL DUE, and STATE. The report output, arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	Total Due	State
Christino	280.00	VA
Perrault	215.00	CT
Stafford	185.00	AZ
Strachan	205.00	OR

" " (quotation marks)

Type

Punctuation Symbol

Purpose

Quotation marks are always used to enclose a text constant. Quotes are also used to enclose the name of a variable in a define command, to enclose an custom relationship name when it is first specified in a script by the named operator, and to enclose the name of a Document, Table, or Import specified in a Control Command.

Syntax

```
"TEXT CONSTANT" define global|temp  "VARIABLE NAME"  .
sum of MEMBERS named  "HIGH VOLUME"  with
    TOTAL DUE  > 500  ;
run procedure  "MEMBERSHIP LIST"  .
```

Example

```
for MEMBERS with STATE  =  "NY"  ;
  list records
    LAST NAME in order  ;
    TOTAL DUE  ;
    STATE  .
End
```

This script tells DataEase: For members who live in New York, list each member's LAST NAME, TOTAL DUE, and STATE. The report output, arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	Total Due	State
Baldwin	100.00	NY
Callahan	70.00	NY
Cheng	65.00	NY
Cohen	70.00	NY
Crandall	85.00	NY
Fitzpatrick	100.00	NY
Meier	70.00	NY
Morales	115.00	NY

:= (assignment operator)

Type

Operator Symbol

Purpose

The assignment operator symbol is used whenever the value of a variable or field is assigned or modified.

Syntax

```
FIELDNAME : = ASSIGNED VALUE .
```

```
assign global|temp VARIABLE NAME : = ASSIGNED VALUE .
```

Example 1

```
LAST NAME : = data-entry LAST NAME .
```

This statement tells DataEase to modify the LAST NAME field by copying the value entered in the LAST NAME field on the Data-entry form.

Example 2

```
assign temp DISCOUNT : = RESERVATIONS TOTAL DUE * 0.15.
```

This script statement assigns a value to a temporary variable named DISCOUNT. The value assigned to the variable is 15% of the TOTAL DUE value in the current RESERVATIONS record.

Note: Do not confuse this symbol with the equal (=) sign which is used to compare one value to another.

< (less than)

Type

Comparison Operator Symbol

Purpose

The <(less than) symbol is used to compare one value to another. It specifies that the value to the left of the symbol is less than the value to the right of the symbol.

Syntax

VALUE 1 <VALUE 2

Example

```
for MEMBERS with TOTAL DUE < 100 ;
```

This script statement selects MEMBERS records that have a value less than 100 in the TOTAL DUE field.

<= (less than or equal to)

Type

Comparison Operator Symbol

Purpose

The <=(less than or equal to) symbol is used to compare one value to another. It specifies that the value to the left of the symbol is less than or equal to the value to the right of the symbol.

Syntax

VALUE 1 <= VALUE 2

Example

```
for MEMBERS with TOTAL DUE <= 500 ;
```

This script statement selects MEMBERS records that have a value less than or equal to 500 in the TOTAL DUE field.

= (equals)

Type

Comparison Operator Symbol

Purpose

The = (equals) symbol is used to compare one value to another. It specifies that the value to the left of the symbol is equal to the value to the right of the symbol.

Syntax

VALUE 1 = VALUE 2

Example

```
for MEMBERS with TOTAL DUE = 100 ;
```

This script statement selects MEMBERS records that have a value equal to 100 in the TOTAL DUE field.

> (greater than)

Type

Comparison Operator Symbol

Purpose

The > (greater than) symbol is used to compare one value to another. It specifies that the value to the left of the symbol is greater than the value to the right of the symbol.

Syntax

VALUE 1 > VALUE 2

Example

```
for MEMBERS with TOTAL DUE > 100 ;
```

This script statement selects MEMBERS records that have a value greater than 100 in the TOTAL DUE field.

>= (greater than or equal to)

Type

Comparison Operator Symbol

Purpose

The >= (greater than or equal to) symbol is used to compare one value to another. It specifies that the value to the left of the symbol is greater than or equal to the value to the right of the symbol.

Syntax

VALUE 1 >= VALUE 2

Example

```
for MEMBERS with TOTAL DUE >= 100 ;
```

This script statement selects MEMBERS records that have a value greater than or equal to 100 in the TOTAL DUE field.

- - (comment)

Type

Punctuation Symbol

Purpose

The - - (comment) symbol is used to insert annotative comments in a script. Comments have no effect during the execution of the procedure.

Syntax

```
-- COMMENT TEXT
```

Example

```
for MEMBERS with count of RESERVATIONS >= 7 ;  
  list records  
    LAST NAME ;--    THIS SCRIPT LISTS MEMBERS WHO  
    MEMBER ID . --    TAKE THE MOST VACATIONS  
end
```

This script selects MEMBERS records that have at least seven matching RESERVATIONS records and lists the members' LAST NAME and MEMBER ID values. The comment to the left of the script has no effect on processing. DataEase ignores text that follows the double dash comment symbol.

abs (absolute value)

Type

Math Function

Purpose

The abs function converts a numeric value to a positive, unsigned numeric value.

Syntax

```
abs ( NUMERIC VALUE )
```

Returns

A numeric value.

Usage

The numeric value in a Math function can be a constant value (as shown below), a variable, a field value or an expression.

The abs function can only be used with a numeric value. A non-numeric value always returns a value of 0.

Examples

```
abs ( -453 )
```

Returns: 453

```
abs ( 4.53 )
```

Returns: 4.53

acos (arccosine)

Type

Trigonometric Function

Purpose

The acos function calculates the arccosine of a numeric value. The result is an angle expressed in radians between 0 and π .

Syntax

`acos (NUMERIC VALUE)`

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

The valid range of the input value is -1 to 1

Examples

`acos (0.5)`

Returns: 1.047195755

`acos (-0.8)`

Returns: 2.49809154

ad hoc relationship

Type

Concept

Purpose

Like a predefined relationship, an ad hoc relationship is a relationship between two sets of records, but an ad hoc relationship is created as a script is processed instead of being specified on the Relationships form during the Form Definition process. An ad hoc relationship lets you easily access records in another table while processing a procedure. However, because the ad hoc relationship isn't stored as a part of the database, it must be redefined in each procedure.

An ad hoc relationship is created by combining a relational (or relational statistical) operator with the name of an unrelated table, or the name of a predefined relationship to which new selection criteria are added.

When you create an ad hoc relationship, the named operator is used to give the set of related records a unique name. This lets DataEase identify each distinct set of records selected from a given table.

Unlike predefined relationships, ad hoc relationships can also include comparisons based on a range, such as "between Current Date – 60 to Current Date", as well as fields where the values are not equal.

Syntax

```
relational operator TABLENAME | RELATIONSHIP
[named "UNIQUE RELATIONSHIP NAME" ]
[with ( selection criteria) ] FIELDNAME;|.
```

Usage

Once an ad hoc relationship is created with certain selection criteria, the criteria cannot be changed during the remainder of the script. If you want to select another set of records, you must create a new ad hoc relationship and give it a unique name using the named operator. The FIELDNAME is required for all relational and relational statistical operators except count of.

Example

```
for MEMBERS with TOTALDUE > 100 ;
  list records
    LASTNAME in order ;
    sum of RESERVATIONS named "SPRING" with
      ( RESERVATION DATE between 03/21/95 to 06/20/95 )
    TOTALDUE.
end
```

Since there is a predefined relationship between MEMBERS and RESERVATIONS based on the MEMBER ID field, the statement:

```
sum of RESERVATIONS named "SPRING" with
  ( RESERVATION DATE between 03/21/95 to 06/20/95 )
TOTALDUE .
```


..creates an ad hoc relationship by adding the additional RESERVATION DATE selection criteria to the existing predefined relationship.

This script tells DataEase: (1) Select all the MEMBERS records that have a TOTAL DUE greater than \$100, (2) find all these members' reservations in the related RESERVATIONS table that are dated between March 21 and June 20, 1995, and (3) for each record selected from the MEMBERS table, list the member's last name and the total cost of that member's Spring reservations.

If the predefined relationship used a custom relationship name and the script specified the tablename instead of the custom relationship name, it would be necessary to restate the predefined relationship criteria. For example, the statement above would read:

```

        sum of RESERVATIONS named  "SPRING"  with
( MEMBERID      = MEMBERS MEMBER ID  and
  RESERVATIONDATE between 03/21/95 to 06/20/95 )
TOTALDUE.

```

If we also want DataEase to list the sum of these members' Summer reservations, we have to create a new ad hoc relationship and give it a unique name as shown below:

```

for MEMBERS with TOTALDUE  > 100 ;
  list records
    LASTNAME in order ;
    sum of RESERVATIONS named  " SPRING "  with
      ( RESERVATIONDATE between 03/21/95 to 06/20/95 )
TOTALDUE ;
    sum of RESERVATIONS named  " SUMMER "  with
      ( RESERVATIONDATE between 06/21/95 to 09/20/95 )
      TOTAL DUE .
end

```

all

Type

Relational Operator

Purpose

The all operator selects every record in a related table that matches the current record being processed.

Syntax

```
all TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] FIELDNAME; | .
```

Returns

The specified value from every matching record in the related table. The type of value returned is the same as that of the specified field. For example, if you are selecting text fields, the returned value is text.

Usage

The all operator can be only used to specify report output items in a list records command. It cannot be used to specify selection criteria.

Example 1

```
for MEMBERS with TOTAL DUE > 100 ;
  list records
    LAST NAME in groups ;
    all RESERVATIONS RESERVATION ID ;
    all RESERVATIONS TOTAL DUE .
end
```

This script tells DataEase: (1) Select all the MEMBERS records that have a TOTAL DUE greater than \$100, (2) find all these members' reservations in the related RESERVATIONS table, and (3) for each record selected from the RESERVATIONS table, list the RESERVATION ID and TOTAL DUE.

The output from this script, arranged in groups by LAST NAME (from the MEMBERS table), might look as shown below:

Last Name	Reservation ID	Total Due
Albert	00197	4,760
	00359	3,220
Anders	00015	4,420
	00421	2,290
	00298	2,480
Anderson	00077	4,320
Archer	00085	4,796
Bennington	00002	5,662
Bickford	00141	2,800
	00356	3,650
Christino	00139	5,450

Example 2

Example 2 demonstrates how you can use the all operator to access information from two or more related tables on the same relationship level. The relational operators can also be used to navigate from one table to a third table (which is not directly related) by using an intermediate relationship, as shown in the sixth line of the script.

```
for ACTIVITIES
  list records
    ACTIVITY in groups ;
    all CLUB ACTIVITIES CLUB NAME ;
    all MEMBERS LAST NAME ;
    all MEMBERS all FAMILY MEMBERS FIRST NAME .
end
```

This script tells DataEase: (1) Select all the ACTIVITIES records and group them by ACTIVITY, (2) find all the related records in CLUB ACTIVITIES and list the name of each club that offers the activity, (3) find all the related records in MEMBERS and list the last name of each member who favors the activity, and (4) find all the related MEMBERS records (for each ACTIVITIES record), and for each MEMBERS record, find all the related FAMILY MEMBERS records and list the value in the FIRST NAME field.

The output from this script, arranged in groups by ACTIVITY, might look as follows:

Activity: Baseball

Club Name	Last Name	First Name
Playa Blanca	Connelly	John
Cancun		Erin
Punta Cana		Patrick
Huatulco		Mary
St. Lucia	Rada	Amanda
Columbus Island		Jerome
		Clarenct
	Ruggiero	Donato
		Anna
	DiLorenzo	Lawrence
		Daniel
		Gregory
		Mary Anne
	Stomboulis	Christos
		Helen
	Walsh	Ivan
		Emma
		Charlotte
		Ann Marie

ampm

Type

Time Function

Purpose

The ampm function evaluates a value expressed in the 24-hour time format (HH:MM:SS) and returns the appropriate abbreviation AM (before noon), or PM (after noon).

Syntax

```
ampm( TIME VALUE)
```

Returns:

A text value (either AM or PM).

Usage:

Time values from midnight (00:00:00) to (11:59:59) return AM. Time values from noon (12:00:00) to (23:59:59) return PM.

Examples:

```
ampm( 08: 15: 25)
```

Returns: AM

```
ampm( 15: 30: 50)
```

Returns: PM

and

Type

Logical Operator

Purpose

The and operator combines two or more sets of selection criteria.

Syntax

```
SELECTION CRITERIA 1 and SELECTION CRITERIA 2 [and SELECTION CRITERIA
3 . . .]
```

Returns

The values in records that satisfy all of the selection criteria statements.

Usage

The and operator requires that a record meet all the specified selection criteria to be processed. The or operator requires that a record meet any of the specified criteria to be processed. When selection criteria are combined with both the and and or operators in one statement, the criteria must be enclosed in parentheses to clarify the meaning.

Example 1

```
for MEMBERS with STATE = "NY"
and TOTAL DUE > 500
and LAST NAME between "A*" to "L*" ;
```

This statement tells DataEase to process only those MEMBERS records that contain NY in the STATE field, a value greater than 500 in the TOTAL DUE field, and a LAST NAME value that begins with any letter between A and L (inclusive). Only records that satisfy all three criteria are processed.

Example 2

```
for MEMBERS with ( STATE = "NY" or STATE = "NJ" )
and TOTAL DUE > 500 ;
```

This statement tells DataEase to process only those MEMBERS records that contain either NY or NJ in the STATE field and a value greater than 500 in the TOTAL DUE field.

Example 3

```
for MEMBERS with STATE not = "NY" and STATE not = "NJ" ;
```

This statement tells DataEase to process all the members records except those that contain either NY or NJ in the STATE field.

Again, only records that satisfy all sets of criteria are processed.

any

Type

Relational Operator

Purpose

The any operator selects the first record in a related table that matches the current record being processed. Since any returns the value from just the first related record, it is typically used when you are accessing the one side of a many-to-one relationship.

Syntax

```
any TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] FIELDNAME ;|.
```

Returns:

The specified value from the first matching record in the related table. The type of value returned is the same as that of the specified field.

Usage:

The any operator can be used to:

- Specify output items in a list records command.
- Specify selection criteria.
- Select related records from more than one table on each relationship level (i.e., Secondary tables, Tertiary tables, etc.). See Primary Table for details on relationship levels.

Example

```
for FAMILY MEMBERS ;
    list records
        any MEMBERS LAST NAME in order ;
        FIRST NAME;
        DATE OF BIRTH.
end
```

This script tells DataEase: (1) Select all the FAMILY MEMBERS records, (2) for each record selected from the FAMILY MEMBERS table, list the LAST NAME from the (one) related MEMBERS record, and (3) list the first name and birthdate of each family member. The output, sorted by LAST NAME, might look as follows:

Last Name	First Name	Date of Birth
Adams	Will	09/23/60
Adams	Becca	04/28/90
Adams	Annalee	08/12/63
Albert	Roland	12/07/48
Albert	Rhonda	12/10/78
Albert	Lori	06/27/82
Albert	Kay.	11/03/50
...

application status

Type

Control Command

Purpose

The application status command generates a report that displays the status of the Documents, defined Imports, Records, or Servers included in the current application.

Syntax

```
application status [DOCUMENTS | IMPORTS | RECORDS | SERVERS]
```

Usage

When processing reaches an application status command, DataEase automatically locks the current application. It then runs a pre-defined system report describing the current state of the Documents, Imports, Records, or Servers in your application. Depending on which parameter you specify, the status report includes one or more of the following:

- The name of each document.
- The DOS filename and file size of each document.
- The number of existing records in each table.
- The number of deleted records in each table.
- The name of each procedure.
- The DOS filename and file size of each procedure.
- The name of each import specification in the selected directory.
- The DOS filename and file size of each import specification.
- The name of each server linked to the current application and the name of the database(s) you access on each server.

Example

```
record entry "MEMBERS" . application status records .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter or update member records, and (2) when the user closes the MEMBERS form, run the pre-defined system report that displays information about the number of existing and deleted Records stored in the application.

Note: If you specify the application status command without including any of the optional parameters listed above, DataEase runs the status report that displays the status of Records by default.

asin (arcsine)

Type

Trigonometric Function

Purpose

The asin function calculates the arcsine of a numeric value. The result is an angle expressed in radians between $-\pi/2$ to $\pi/2$.

Syntax

`asin(NUMERIC VALUE)`

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

The valid range of the input value is -1 to 1.

Example

`asin(1)`

Returns: 1.570796

`asin(-0.50)`

Returns: -0.5235988

assign

Type

Procedural Command

Purpose

The optional assign command is used to give a value to a temporary or global variable. (You must define a variable before you can assign it a value.) A variable is used to store a value, such as a text string or a calculated result, that can change during the processing of a procedure. By specifying the variable's name, the stored value can be used like any other value in a script.

The status of a variable can be global (denoted by the keyword global) or temporary (denoted by the keyword temp). A temporary variable maintains its value only during the current procedure. A global variable can pass its value from one procedure to another. To pass a value from one procedure to another, the global variable must be defined identically in each procedure.

Syntax

```
assign global|temp VARIABLE NAME := ASSIGNED VALUE .
```

Usage

The assign command is followed by:

- The status of the variable (global or temporary).
- The name of the variable (without quotation marks).
- The assignment operator.
- The value assigned to the variable.
- A period.

Example

```
define temp "DISCOUNT" Number .
for RESERVATIONS with TOTAL DUE > 2000 ;
  assign temp DISCOUNT := RESERVATIONS
    TOTAL DUE * 0.15 .
  modify records
    TOTAL DUE := TOTAL DUE - temp DISCOUNT .
end
```

This script tells DataEase: (1) Create (define) a temporary variable called DISCOUNT to store a number while processing the current script, (2) find all the RESERVATIONS records that have a value greater than 2000 in the TOTAL DUE field, (3) give (assign) the DISCOUNT variable a number value determined by multiplying the TOTAL DUE on each reservation by 15%, and (4) modify these RESERVATIONS records by subtracting the value of the DISCOUNT variable from the value in the TOTAL DUE field.

atan (arctangent)

Type

Trigonometric Function

Purpose

The atan function calculates the arctangent of a numeric value. The result is an angle expressed in radians between $-p/2$ to $p/2$.

Syntax

`atan(NUMERIC VALUE)`

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

Examples

`atan(500.3)`

Returns:1.56879753

`atan(-359.4)`

Returns:-1.5680139

atan2 (arctangent 2)

Type

Trigonometric Function

Purpose

The atan2 function calculates the arctangent of Value 1 divided by Value 2

The result is an angle expressed in radians between -p and p.

Syntax

```
atan2( NUMERIC VALUE 1, NUMERIC VALUE 2)
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

Examples

```
atan2(3,127)
```

Returns:0.02361766

```
atan2(1, 1)
```

Returns:0.7854

backup db (backup database)

Type

Control Command

Purpose

The backup db command creates a backup copy of the current DataEase application. When processing reaches a backup db command, DataEase automatically locks the current application. It then displays a dialog asking you to specify the drive on which the backup copy should be stored and how you want to handle any errors that occur during the backup.

Syntax

```
backup db .
```

Usage

When you backup a database using the backup db command, DataEase copies the database using a special format; therefore, the backup copy can only be restored using the DataEase restore db command or **Application>>Utilities>>Restore**.

When you backup and restore a database, all records that have been deleted since the last backup and restore operation are permanently erased.

LAN

On a LAN (Local Area Network), if another user is currently using any resource required by the backup db command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

```
record entry  "MEMBERS"  .
run procedure  "PRINT INVOICES"  .
backup db .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter new member records, (2) when the user closes the MEMBERS form, run the procedure named PRINT INVOICES, and (3) after running the PRINT INVOICES procedure, make a DataEase backup copy of the current application.

begin transaction

Type

Procedural Command

Purpose

The begin transaction command is used to mark the start of a unit of work called a transaction. A transaction can be a whole procedure or any part of a procedure that enters or modifies data (a procedure that includes an enter a record, modify records, or delete records command). When processing reaches a begin transaction command, DataEase treats the statements that follow as part of the same transaction until it reaches a commit, rollback, or another begin transaction command. A procedure can contain any number of begin transaction commands.

Syntax

```
begin transaction
```

Usage

When a DQL Procedure is translated into SQL, DataEase inserts an implicit begin transaction command at the beginning of the procedure. If you insert an explicit begin transaction command outside a for loop or conditional statement, it is ignored when the procedure is translated into SQL. If you insert an explicit begin transaction command inside a for loop or conditional statement, it is interpreted as a savepoint when the procedure is translated into SQL.

The commit command is used to end a transaction and save all the modified data. Once a transaction is committed, it cannot be undone by a rollback command.

Example 1

```
for RESERVATION AGENTS ;
  begin transaction
    for DAILY RESERVATIONS with ( POSTED = NO) ;
      enter a record in YEARLY RESERVATIONS
        copy all from DAILY RESERVATIONS .
      modify records
        POSTED = YES .
    end
    if sum of YEARLY RESERVATIONS AMOUNT > 80000
      then
        modify records
          CHRISTMAS BONUS := YES .
      end
    commit .
  end
```

The procedure in Example 1 contains three operations that are treated as a single transaction. The first operation enters a record in the YEARLY RESERVATIONS table using the values in the DAILY RESERVATIONS table. The second operation modifies the POSTED field in the DAILY RESERVATIONS table to indicate that the record was posted to the YEARLY RESERVATIONS table. The third operation modifies the records in the RESERVATION AGENTS table whose yearly sales total is greater than \$80,000.00. When processing reaches the commit command, the updates to the parent record and both child records are saved together. If any part of the transaction fails, the entire transaction is rolled back.

Example 2

```

for RESERVATIONS ;
  begin transaction
    modify records in MEMBERS
      ACCOUNT BALANCE := ACCOUNT BALANCE +
        RESERVATIONS TOTALDUE .
    modify records in RESERVATION AGENTS
      DAILY TOTAL := DAILYTOTAL +
        RESERVATIONS TOTALDUE .
    modify records in CLUB ROOMS
      VACANCIES := VACANCIES -
        RESERVATIONS ROOMSREQUIRED .
    modify records
      POSTED:= YES .
  commit .
  if current SQLCODE not = 0 then
    list records
      RESERVATION ID .
  end
end
end

```

The procedure in Example 2 contains four modify operations that are treated as a single transaction. The first operation updates a record in the MEMBERS table using the value in the TOTAL DUE field in the RESERVATIONS table. The second operation updates a record in the RESERVATION AGENTS table using the value in the TOTAL DUE field in the RESERVATIONS table. The third operation updates a record in the CLUB ROOMS table using the value in the ROOMS REQUIRED field in the RESERVATIONS table. The fourth operation updates the current record by setting the POSTED field to YES.

When processing reaches the commit command, all four of the modifications are committed together. If any part of the transaction fails, the entire transaction (all four modifications) is rolled back. The current SQLCODE variable is set to zero if the commit is successful and to an SQL engine-specific error code if the commit fails. If DataEase is unable to commit all the modifications for a specific order, the order number is listed in the procedure output.

between

Type

Comparison Operator

Purpose

The between operator is used to indicate that a value falls within a specified range. It specifies that the test value is greater than or equal to Value 1 and less than or equal to Value2.

Syntax

TEST VALUE between VALUE 1 to VALUE 2

Returns

All records for which the comparison is true.

Example

```
for MEMBERS with TOTAL DUE between 0 to 499 ;  
  list records  
    LAST NAME ;  
    MEMBER ID .  
end
```

This statement selects MEMBERS records that have a value between 0 and 499 (inclusive) in the TOTAL DUE field.

blank

Purpose

The blank keyword is used to assign a null value to a field or variable, or to compare a null value to the value in a field or variable.

Syntax

blank

Usage

A blank value is a null value, not a zero. When statistics are calculated on a field, blank values are not included in the calculation. Zero field values are included when generating statistics.

Field or variable values can be compared to blank, and blank can be assigned as a value to any field or variable.

Example

```
for MEMBERS with TOTAL DUE = blank ;  
    delete records .  
end
```

This script tells DataEase: Delete the MEMBERS records that have a blank value in the TOTAL DUE field. It might be used to delete inactive MEMBERS records.

break

Type

Procedural Command

Purpose

The break command is used to immediately stop the action of a for or while command and continue processing the rest of the procedure.

When processing reaches a break command, DataEase immediately abandons the current action. It resumes processing the procedure with the first action listed after the corresponding end command.

Syntax

```

        ACTION 1 .
        ACTION 2 .
        ACTION 3 .
        break .
end
        NEXT COMMAND
```

Usage

The break command is always followed by a period.

Example

```

for MEMBERS ;
    list records
        TOTAL DUE in reverse ;
        LAST NAME .
        if current item number > 100 then
            break .
end
end
for RESERVATIONS ;
    delete records with DATE < 01/01/90 .
end
```

This script tells DataEase: (1) List each member's TOTAL DUE and LAST NAME, arranging the output from highest to lowest TOTALDUE value, and (2) when one hundred MEMBERS records have been processed, stop listing records and proceed to the next command (delete RESERVATIONS records with DATE before January 1, 1990).

call menu

Type

Control Command

Purpose

The call menu command opens the specified menu document.

Syntax

```
call menu "MENU NAME" .
```

Usage

The call menu command is used to display a user-defined menu at any point during a Control Procedure.

When the menu is displayed, you can make selections from it and perform the associated DataEase operations normally.

When you close the menu by double-clicking its Control menu box (or by clicking on a custom button designed for this purpose), the Control Procedure resumes with the action following the call menu command.

User-defined menus are called by the name specified when the menu document was created. In a call menu command, the menu name must be enclosed in quotation marks unless it is specified as a variable.

Example

```
record entry "MEMBERS" .  
run procedure "PRINT INVOICES" .  
call menu "MAIN MENU" .  
backup db .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter new records, when the user closes the MEMBERS form, (2) run the PRINT INVOICES Procedure, when the procedure finishes processing, (3) display the MAIN MENU document, and when the operator closes the menu, (4) backup the application.

call program

Type

Control Command

Purpose

The call program command runs the specified DOS program.

Syntax

```
call program "PROGRAM NAME [OPTIONAL ARGUMENT]" .
```

Usage

The call program command is used to call and run another program at any point during a Control Procedure.

The called program can be any .BAT, .PIF (shortcut), .COM, or .EXE program. The command must specify the drive and directory on which the called program is stored if it is not stored in the current directory.

The command can include optional arguments. The command can be a constant (Example 1) or an expression involving functions and variables (Example 2). The program name and arguments must be enclosed in quotation marks unless specified as a variable.

When the called program has been executed, DataEase resumes processing the Control Procedure with the action following the call program command.

Example 1

```
run procedure "MONTHLY RESERVATIONS" .
call program "C: \LOTUS\123W.EXE RESVDATA.WK3" .
```

This script tells DataEase: (1) Run the predefined MONTHLY SALES Procedure which saves a summary of the monthly reservations to a disk file, and (2) call the Lotus 1-2-3 program and load the file created by the previous procedure for further processing.

Example 2

```
define temp "CHART" TEXT .
assign temp CHART := "RESVDATA.WK3" .
define temp "PROG" TEXT .
assign temp PROG := "C: \LOTUS\123W.EXE" .

run procedure "MONTHLY SALES" .
call program jointext ( temp PROG,temp CHART ) .
```

This script does exactly the same thing as the simpler one above, but shows the proper use of an expression as an argument for the call program command.

The call program command requires a space between the program name and any optional argument. Because the jointext function does not automatically insert a space between the PROG and CHART variables in the example above, a blank space is included inside the quotation marks after the program name, 123W.EXE, stored in the PROG variable.

Note: The arguments of the jointext function (PROG, CHART) are not enclosed in quotes because they are variables.

case

Type

Procedural Command

Purpose

The case command functions like an if-then-else statement with multiple ifs. It tells DataEase to compare an expression to a series of values and execute a different action (or group of actions) based on which comparison is true.

When processing reaches a case command, DataEase compares the expression that follows the case command to each of the statements specified by the keyword value.

If the first comparison is true, DataEase executes all the actions between that value statement and the next value statement.

If the first comparison is not true, the case expression is compared to the next value statement. This comparing of values continues in top-to-bottom sequence from one value statement to the next until a true comparison is made.

If none of the specified value comparisons are true, DataEase executes the actions specified after the keyword others. If the others keyword is not present, DataEase executes none of the specified actions. It continues processing the remainder of the script.

As soon as the actions following any single statement are executed, processing passes to the first action following the end command for the case structure.

Syntax

```
case (  EXPRESSION)
    value COMPARISON 1 :
        ACTION SERIES 1 .
    [value COMPARISON 2 :
        ACTION SERIES 2 .]
    [value COMPARISON 3 :
        ACTION SERIES 3 .]
    [others :
        DEFAULT ACTION SERIES .]
end
```

Usage

The case command requires a case expression, at least one value statement, and an end command. Subsequent value statements, actions, and the others keyword are optional. If others is used, it must follow the last value statement.

The case expression must be enclosed in parentheses. It can be a field, a variable, or any other expression except a boolean expression (true or false). Each comparison value can be a constant, a variable, an expression, or a range of values.

An action can specify any valid procedure command including another if, while, or case command. Each nested command must be completed before processing passes to the first action after the end command (see the nested actions entry in this Lexicon).

When case commands are nested, the first end corresponds to the last preceding case, and each case command must be matched with a corresponding end command. This rule applies to all nested Procedural commands (including case, for, if, and while).

When you select case from the command pick list, DataEase automatically enters the opening ((parenthesis).

Example

```

value  "FRANK"  :
    call menu  "MAIN MENU"  .
value  "TOM"    :
    call menu  "CLUB ADMINISTRATION"  .
value  "CAROL"  :
    run procedure  "MAINTENANCE"  .
others:
    record entry  "MEMBERS"  .
end

```

This script tells DataEase: (1) If the current user is Frank, display the MAIN MENU document, (2) if the current user is Tom, display the CLUB ADMINISTRATION menu document, (3) if the current user is Carol, run the MAINTENANCE procedure, and (4) if the current user is anyone other than Frank, Tom, or Carol, display the MEMBERS form.

ceil

Type

Math Function

Purpose

The ceil function rounds up a numeric value to the next integer.

Syntax

```
ceil( NUMERIC VALUE)
```

Returns

An integer value.

Usage

The numeric value in a Math function can be a constant value (as shown below), a variable, a field value or an expression.

Examples

```
ceil(5.000)
```

Returns:5

```
ceil(5.001)
```

Returns:6

comment

Type

Symbol

Purpose

The -- (comment) symbol is used to insert annotative comments in a script. Comments have no effect during the execution of the procedure.

Syntax

```
-- COMMENT TEXT
```

Usage

DataEase treats each character between the double dash symbol and the end of the line as part of the comment.

To continue a comment on more than one line, use the double dash symbol at the beginning of each line.

Example

```
for MEMBERS with TOTAL DUE >= 500 ;
    list records
        LAST NAME ;
        TOTAL DUE .
end      -- THIS QUERY LISTS THE LARGEST FAMILIES.
        -- IT SHOULD BE PRINTED ON THE TENTH DAY
        -- OF EACH MONTH.
```

This script statement selects MEMBERS records that have a value greater than or equal to 500 in the TOTAL DUE field and lists the members' LAST NAME and TOTAL DUE values. The comment at the end of the script has no effect on processing. DataEase ignores text that follows the double dash comment symbol.

commit

Type

Procedural Command

Purpose

The commit command is used to mark the end of a transaction. A procedure can contain any number of commit commands.

Syntax

```
commit .
```

A transaction can be all or any part of a DQL Procedure that changes data and must be processed as a single unit to maintain integrity (such as a procedure that transfers money from a customer's savings account into his/her checking account). When accessing data in an SQL database, DataEase usually treats a DQL Procedure as a single transaction by default and commits all the changes made to the data simultaneously at the end of the procedure. When you're running a shared application on a network, treating a whole procedure as a single transaction may reduce concurrency. For this reason, DataEase provides the tran off command (to turn transactions off) and the commit command to divide a procedure into several smaller transactions.

Usage

The commit command is used with the begin transaction command to divide a procedure into several transactions. By defining separate transactions within a DQL Procedure, it's possible to rollback any partially completed changes that leave data in an inconsistent state and to recover from system or user-generated errors.

A commit command can be used anywhere in a procedure. When DataEase converts a script into SQL, the position of a commit command in the script determines how DataEase processes it. If the commit command is outside a for loop or conditional statement, DataEase translates the DQL commit into an SQL COMMIT command (this commits all changes made to the data since the last commit or begin transaction command was processed). If the commit is inside a for loop or conditional statement, DataEase inserts the SQL SAVEPOINT command.

The commit command is often followed by a conditional statement that uses the current status, current SQLCODE, or current SQLCOUNT variable to determine if the preceding transaction was committed successfully (see Example 2).

Example 1

```
for RESERVATIONS ;
  begin transaction
    modify records
      TOTAL DUE := TOTAL DUE - AMOUNT PAID .
  commit .
  If DEPARTURE DATE < current date - 90 then
    begin transaction
      modify records
        TOTAL DUE := TOTAL DUE * LATE PENALTY .
    commit .
  end
end
```


This procedure contains two transactions. The first transaction modifies the RESERVATIONS records by subtracting the AMOUNT PAID from the TOTAL DUE. The second transaction modifies the records of past due accounts (that is, reservations that are not paid 90 days after the RESERVATION DATE) by multiplying the TOTAL DUE by a LATE PENALTY factor. Each transaction is committed as it is completed. The commit commands tell DataEase to post the changes in the appropriate table on the server and then continue processing the procedure.

Example 2

```
for RESERVATIONS ;
  begin transaction
    modify records in CLUB ROOMS with
      ( CLUB ID = RESERVATIONS CLUB ID and
        RESERVATION DATE = RESERVATIONS RESERVATIONDATE)
      VACANCIES := VACANCIES - ROOMSREQUESTED .
  commit .
  if current SQLCODE not = 0 then
    message " Last transaction was unsuccessful. |
      Not enough rooms available. |
      Changes have been rolled back . " window.
  else      begin transaction
    modify records
      CONFIRMED := YES .
    commit .
  end
end
```

This procedure uses the current SQLCODE variable to verify that the first transaction was successfully committed before beginning the second transaction. If the current SQLCODE is not equal to zero (i.e., the first transaction was not successfully committed), the SQL engine automatically rolls back the changes and displays a message that notifies the user that the transaction failed. If the current SQLCODE is zero, the first transaction is committed and DataEase continues processing.

Comparison Operators

Type

Operator

Purpose

Comparison Operators are used to compare one value to another in record selection criteria, math formulas, and in other text and numeric expressions.

The DQL uses seven Comparison Operators:

- = Both sides of the comparison have the same value.
- < The value on the left side of the comparison is less than the value on the right side.
- > The value on the left side of the comparison is greater than the value on the right side.
- <= The value on the left side is less than or equal to the value on the right side.
- >= The value on the left side is greater than or equal to the value on the right side.
- between.....to The value lies within the specified range (inclusive).
- not Reverses the meaning of the operator that immediately follows it.

Examples

```
for MEMBERS with TOTAL DUE > 500
  and STATE not = " NY " ;
for RESERVATIONS with RESERVATIONDATE between 06/01/95
  to 06/31/95 ;
```

Conditional Statistical Operators

Type

Operator

Purpose

Conditional Statistical Operators generate statistical information about specific conditions that occur in a set of records.

The DQL uses three Conditional Statistical Operators: item, count, and percent.

- item returns a YES or NO answer indicating if the comparison is true or false.
- count returns the number of true responses within all the specified records as well as specified groups.
- percent returns the percentage of true responses (the number of true responses divided by the total number of records processed, multiplied by 100). percent also works at the group level.

Usage

In a script, the Conditional Statistical Operator is inserted after a list item that is compared to a specified value. The operator is separated from the comparison by a colon.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    TOTAL DUE > 100 : item count percent .
end
```

This script tells DataEase: (1) Process all the MEMBERS records, (2) list each member's LAST NAME and TOTAL DUE, (3) for each member, display a YES or NO answer indicating if the member's TOTAL DUE is greater than \$100, (4) display the total number of members whose TOTAL DUE is greater than \$100, and (5) display the number of members that have a TOTALDUE greater than \$100 as a percentage of all the members.

The output from this script arranged in alphabetical order by LAST NAME might look as follows:

Last Name	Total Due	Total over \$100
Adams	85.00	NO
Albert	120.00	YES
Andersen	120.00	YES
Anderson	70.00	NO
Archer	115.00	YES
Baldwin	100.00	YES
Beauchamp	35.00	NO
Beauchamp	85.00	NO
Beecher	85.00	NO
Bennington	135.00	YES

Bickford	135.00	YES
Birnbaum	65.00	NO
Blake	85.00	NO
Borusiewicz	100.00	NO
...
Count:	44	
Percent:	23%	

Constant Value

Type

Concept

Purpose

A constant value is a value that does not change while a procedure is being processed (in contrast to a variable, a value that can change during processing).

Constant values are used in selection criteria comparisons, math formulas, proper names, and other text and numeric expressions.

Usage

In a script, a constant value can be used wherever its value type is allowed. The value types of constants are the same as the basic field value types: text, number, numeric string, date, time, and dollar.

The format for typing constant values is as follows:

- Enclose Text constants in double quotes ("TEXT").
- Use slashes in Date constants (01/01/2001).
- Use colons in Time constants (09:30:00).
- Do not type formatting characters in numeric strings.
- Do not type commas in Number constants.

Example

```
for MEMBERS with LAST NAME = " SPENCER "  
and ZIP CODE > 90000 ;
```

In this query statement, both the last name SPENCER and the zip code 90000 are constant values.

Control Procedure

Type

Concept

Purpose

A Control procedure is a procedure that lets you link other procedures together with or without conditional processing actions specified by Procedural commands.

A Control procedure can link any number of procedures together and can automatically initiate several actions, including run procedure, backup db (database), install application, etc.

Usage

A Control procedure is used primarily to link Processing procedures together. It can include any combination of Control commands, Processing commands, and Procedural commands.

For a full explanation of any of the DQL commands, see the individual command entry in this lexicon.

Example

```
if current user name = "PAUL" then
    call menu "RECREATIONAL ADMIN" .
    run procedure "MAILING LIST" .
else
    call menu "MAIN MENU" .
end
```

This Control procedure tells DataEase: If the current user is Paul, (1) Display the RECREATIONAL ADMIN menu, (2) run the procedure that generates an up-to-date mailing list, and (3) if the current user is not Paul, display the main menu.

copy all from

Type

Processing Command

Purpose

The copy all from command copies values in identically named fields from one table to another.

Syntax

```
copy all from TABLENAME|data-entry ;|.
```

Usage

The copy all from statement must specify a Source table and must be immediately preceded by a modify records or enter a record command. The Source table can be any relationship previously specified in the script. The procedure's Data-entry form can also be used as the Source.

The Target table is specified at the end of the preceding modify records or enter a record command. The Target table can be any table you specify; the default is the Primary table specified in the most recent for statement. Data is copied into each Target field from the identically named field in the Source table.

When you use the copy all from command, data is transferred only between fields with the same fieldname. Non-matching fields are ignored.

Example

Assume we've defined a target form called CATALOG LIST. This form contains only those members who receive a special Christmas promotional catalog.

```
for MEMBERS with
  any RESERVATIONS TOTAL DUE > 1500 ;
  enter a record in CATALOG MEMBERS
  copy all from MEMBERS .
end
```

This script tells DataEase: (1) Select all the MEMBERS records that have any related RESERVATION record with a TOTAL DUE greater than \$1500, and (2) for each MEMBERS record selected, enter a record in the CATALOG MEMBERS table and copy all the information from the source MEMBERS record into the target CATALOG MEMBERS record, based on matching field names.

cos (cosine)

Type

Trigonometric Function

Purpose

The cos function calculates the cosine of an angle expressed in radians. The value returned ranges between -1 and 1.

Syntax

```
cos ( NUMERIC VALUE )
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

Examples

```
cos ( 3.1415928 )
```

Returns: -1

```
cos ( -22 )
```

Returns: .9271838

cosh (hyperbolic cosine)

Type

Trigonometric Function

Purpose

The cosh function calculates the hyperbolic cosine of an angle expressed in radians.

Syntax

```
cosh ( NUMERIC VALUE )
```

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value or an expression.

Returns

A numeric value.

Examples

```
cosh ( 2.34 )
```

Returns: 5.23878166

```
cosh ( -5.4 )
```

Returns: 110.70547

count

Type

Conditional Statistical Operator

Purpose

The count operator counts the number of records in the table being processed. It can also be used to count the number of times a condition specified by a comparison of two values is true. The result usually appears as a statistic in the summary area at the end of a report or group.

Syntax

```
CONDITION : count [other statistical operators];|.
```

Usage

To count all the records that meet a specified condition, enter :count after the condition as shown in Example 1. . When applied to a field, it counts how many times that field has a value. To count all the records processed by a script, use a statement that is true for every record, such as "A" = "A". This method is shown in Example 2.

Example 1

```
for MEMBERS with STATE = "NJ" ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    TOTAL DUE > 40 : count .
end
```

This script tells DataEase: (1) List the MEMBERS records showing each member's LAST NAME and TOTAL DUE, and (2) count the number of members who have an TOTAL DUE greater than \$40. The output from this script might look as follows:

Last Name	Total Due
Beauchamp	35.00
Carley	50.00
Fairchild	50.00
Giovanelli	70.00
Morrison	100.00
Parker	105.00
Count (> 40):	5

Example 2

```
for MEMBERS with STATE = "NJ" and TOTAL DUE > 40 ;  
  list records  
    LAST NAME in order ;  
    TOTAL DUE ;  
    "A" = "A" : count .  
end
```

Example 2 produces output identical to Example 1, except that Beauchamp is omitted because her TOTAL DUE is less than \$40. The count statistic is again 5, but in this case it represents all the records that are processed (because all records processed satisfy the count condition).

count of

Type

Relational Statistical Operator

Purpose

The count of operator counts how many records in a related table match the specified selection criteria. The result can appear as a list item in the detail area of a report or as a statistic in the summary area at the end of a report.

There's an important difference between the conditional statistical operator count and the relational statistical operator count of. count finds the number of records that satisfy a specified condition among the records being processed. count of calculates the number of matching records related to the records being processed by the script.

Syntax

```
count of TABLENAME|RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] ;|.
```

Example

```
for MEMBERS with STATE = "CA" ;
    list records
        LASTNAME in order ;
        TOTAL DUE ;
        TOTAL DUE > 100 : item count ;
        count of RESERVATIONS with ( TOTAL DUE > 1500) .
end
```

This script tells DataEase: (1) Process all the MEMBERS records of members living in California, (2) list the LAST NAME and TOTAL DUE field from each MEMBERS record, (3) for each member, display a YES or NO answer indicating if the member's TOTAL DUE is greater than \$100, (4) count the total number of members whose TOTAL DUE is greater than \$100 and display this total as a statistic at the end of the report output (this is generated by the count operator), and (5) count the number of related RESERVATIONS records that have a TOTAL DUE greater than \$1500, and display this number as a list item for each member (this is generated by the count of operator).

count of

This script uses data from two different fields named TOTAL DUE, one in the MEMBERS table (containing membership fee data), and the other from the RESERVATIONS table (containing reservation price data).

The output from this script might look as follows:

Last Name	Total Due	Total Due > 100	Count of RESERVATIONS with TOTAL DUE > 1500
Broadrick	\$100.00	NO	1
Gauthier	\$140.00	YES	1
Goetz	\$70.00	NO	1
Jackson	\$35.00	NO	0
Kowalski	\$70.00	NO	1
Manzi	\$100.00	NO	1
McKenna	\$85.00	NO	1
Schwartz	\$70.00	NO	1
Steinberg	\$120.00	YES	2
Sullivan	\$70.00	NO	1
Young	\$105.00	YES	1
Number of MEMBERS with TOTAL DUE > 100 :		3	

In the output above, the count of totals for each member are displayed as list items in the Report Detail Area while the count total is displayed at the bottom of the output in the Report Summary Area. This is accomplished by placing the count of total on the Record object in the procedure layout and the count total on the Form object.

current

See also **commit**, **data-entry**, **exec SQL**, **global**, **input using**, **temp**.

Type

Variable component

Purpose

The **current** keyword lets you access any of the eleven system defined variables summarized below and on the following page.

Variables Used with the current Keyword

current date	The date recorded by the system clock (a date value). The year part is a two digit number.
Current extended date	The date recorded by the system clock (a date value). The year part is a four digit number.
current time	The time recorded by the system dock (a time value).
current page number	In DataEase, this command is replaced by the Page Number application variable. current page number is included only to maintain compatibility with applications created in character-based versions of DataEase. See DG 6 for information on creating application variables.
current item number	The number of the current record. Records are automatically counted as they are processed by the procedure (a numeric value).
current user name	The name used to sign onto the application (a text value).
current user level	The current user's security level (a numeric value from 1 to 7).
current computername	The name of the workstation running the procedure when using DataEase in a multi-user environment (a text value set by the DENAME environment variable).
current status	The value of the processing action selected following the DQL input using command or the status of the last DQL command that modifies data. By testing the current status variable, you can control subsequent processing following an input using command or check for errors following a DQL command that modifies data. See the input using command for an example of current status used with the input using command.

The following variables are used only in applications that access data on an SQL database:

current SQLCODE	The SQL Error Code number generated by the server when an error occurs during the processing of an SQL statement (e.g., CREATE, INSERT, UPDATE, or DELETE). current SQLCODE is set to zero if the last SQL statement is processed without error. Server-specific error codes are
------------------------	--

explained in your database engine documentation. See the commit and exec SQL commands for examples.

current SQLCOUNT

The number of rows processed by the procedure. See the exec SQL command for an example.

current SQLMSGTXT

The SQL Message Text generated by the server, in addition to any errors and warnings (a server can return a **current SQLCODE** value of zero and still generate SQL Message Text). DataEase displays the first 50 characters of the message. Server-specific messages are explained in your database engine documentation. See the exec SQL command for an example.

Note: See input using for examples demonstrating the use of the **current status** variable. See commit for examples that use **current SQLCODE** and **current SQLCOUNT**. See exec SQL for examples using **current SQLCODE**, **current SQLCOUNT**, and **current SQLMSGTXT**.

When current status is used with the input using command, it returns a numeric value from 1 to 4 depending on which menu selection (or icon) the user chooses after entering data.

When current status is used with an enter a record, modify records, delete records, or commit command, it returns 0 (zero) if the command is processed successfully and -1 (negative 1) if the command fails.

Syntax

current variable name

Example

```
list records
  current date ;
  current user name .
for MEMBERS with TOTAL DUE > 175 ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    current item number end
```

This script tells DataEase: (1) Select the MEMBERS records that have a TOTAL DUE value greater than \$175, (2) list the LAST NAME and TOTAL DUE values and current item number (record number) from each record that is processed, and (3) list the current date (today's date) and the current user name (the name of the user who generated the report).

The output from this script might look as follows:

Report generated by: George McGrath Date: 01/20/99

Item #	Last Name	Total Due
1	Perrault	215.00
2	Christino	280.00
3	Stafford	185.00
4	Strachan	205.00

data-entry

Type

Keyword

Purpose

The data-entry keyword identifies the Data-entry form as the source of the specified value.

Syntax

data-entry FIELDNAME

Usage

A Data-entry form is a form used to collect data and processing specifications from a user at the beginning of a procedure. Values retrieved from a Data-entry form are used in a script exactly like any other value or expression. The keyword data-entry is used in place of a table name when specifying a value entered in a Data-entry form.

Example

```
for MEMBERS with MEMBERID    =    data-entry MEMBERID    ;
  modify records
    copy all from data-entry
    ADDRESS := data-entry NEW ADDRESS .
end
```

This script might be used to post address changes into the MEMBERS table. It tells DataEase: (1) Find the MEMBERS record that has the same MEMBER ID as the MEMBER ID specified on the Data-entry form, and (2) modify the MEMBERS record by copying the value from the NEW ADDRESS field in the Data-entry form to the ADDRESS field in the MEMBERS table.

date

Type

Date Function

Purpose

The date function constructs a date value from three separate numeric values.

Syntax

```
date( MONTH, DAY, YEAR)
```

...where;

MONTH is the number of the month (1-12),

DAY is the day of month (1-31), and

YEAR is the last two digits of year, or the full four digit year, depending on the type of date field.

Returns

A date value in the Short Date format specified in the Windows Control Panel. For example, if you specify the United States default date format in Control Panel, the date value is returned in the order MM/DD/YY.

Usage

If any input value is invalid (e.g., month > 12 or day > 31), DataEase calculates a future date, for example, 6,45,95 becomes 07/15/95

Decimal values are automatically truncated.

Example 1

```
date(7,1,99)
```

Returns:07/01/99

Example 2

```
FIRSTDAY := date ( month ( TODAY ) , 1, year ( TODAY) ) ;
LASTDAY := date ( ( month ( TODAY ) + 1) , 1, year ( TODAY) ) - 1
.
```

This portion of a script uses a variable named TODAY (which holds the current date) and two other date functions, month and year, to find the first and last days of the current month.

The result, if the current date is July 4th, 1995, is: FIRSTDAY = 07/01/2001 LASTDAY = 07/31/01 This routine is accurate for any date.

Note: If your system is configured for an international date format other than United States, the date value may be returned in the order DD/MM/YY or YY/MM/DD. See your Microsoft Windows documentation to find the date format that corresponds to your Windows Control Panel country setting.

Regardless of how your system is configured, the three numerical values this function requires are always input in the order MONTH, DAY, YEAR.

day

Type

Date Function

Purpose

The day function extracts the day of the month (1-31) from a date value.

Syntax

`day(DATE VALUE)`

Returns

An integer value between 1 and 31

The date format selected in Windows Control Panel changes the date sequence but does not affect which value is returned by a Date function.

Examples

`day(12/31/01)`

Returns:31 (United States format)

`day(31/12/01)`

Returns:31 (Australian, English Canadian, South American, most European formats)

`day(01/12/31)`

Returns:31 (Austrian, French Canadian, Taiwanese, South Korean, some European formats)

db status (database status)

Type

Control Command

Purpose

The db status command generates a report that displays the status of the current database. In DataEase, this command is replaced by the application status records command and is included to maintain compatibility with applications created in the character-based version of DataEase. If you run a procedure that contains the db status command, DataEase executes the application status report that displays information about the records that exist in the application.

define

Type

Procedural Command

Purpose

The define command is used to create a global or temporary variable. A variable is used to store a value such as a text string or a calculated result that can change during the processing of a script. By specifying the variable's name, the stored value can be used like any other value in a script.

The status of a variable can be global (denoted by the keyword global) or temporary (denoted by the keyword temp). A temporary variable maintains its value only during the current script. A global variable can pass its value from one script to another when several scripts are integrated within a Control Procedure. To pass a value from one script to another, a variable must be defined identically in each script.

Syntax

```
define [global|temp] "VARIABLE NAME" TYPE [LENGTH] .
```

Usage

- The define command is followed by:
- The status of the variable (global or temporary).
- The name of the variable (enclosed in quotation marks).
- The type of the variable (any field type except Choice, currency, or Yes/No, Sequenced ID, or Memo).
- The length of the variable (optional, see below).
- A period. The define command requires a concluding period.

If you do not specify temp or global in the define statement, DataEase automatically creates a temporary variable.

The length specifies the number of characters for Text and Numeric String variables. The default length is 25; you can specify any length up to 255 characters. Number variables are always 14 digits long. Date and Time variables are 8 characters long. The default length is used if you do not specify a length in the define command.

Note: You can define a variable as any field type except Choice, Dollar, or Yes/No.

Example

```
define temp  "DISCOUNT"  Number .
for RESERVATIONS with TOTAL DUE  > 1500 ;
  assign temp DISCOUNT :=
    RESERVATIONS TOTAL DUE *  0.15 .
  modify records
    TOTALDUE := TOTAL DUE - temp DISCOUNT .
end
```

This script tells DataEase: (1) Create (define) a temporary variable called DISCOUNT in which to store a number while processing the current script, (2) find all the RESERVATIONS records that have a value greater than 1500 in the TOTAL DUE field, (3) give (assign) the DISCOUNT variable a value determined by multiplying the TOTAL DUE on each invoice by 15%, and (4) modify these RESERVATIONS records by subtracting the value of the DISCOUNT variable from the value in the TOTAL DUE field.

delete records

Type

Processing Command

Purpose

The delete records command deletes records in the specified table.

Syntax

```
delete records in FORMNAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] .
```

Usage

You must end the delete records command with a period (.). When you delete records in the Primary table, the keyword in is omitted (see Example 1).

Example 1

```
for members with ( highest of
    RESERVATIONS DATE < 01/01/99) ;
    delete records .
end
```

This script deletes records in the Primary table (MEMBERS) whose most recent invoice (in the related RESERVATIONS table) is dated prior to January 1st, 1994.

When you delete records in a table other than the Primary table, the keyword in precedes the table name (as shown in Example 2).

Example 2

```
for MEMBERS ;
    delete records in RESERVATIONS named " OUTDATED "
        with RESERVATIONS DATE < 01/01/99) .
end
```

This script deletes records in the related (RESERVATIONS) table that are dated prior to January 1st, 1999.

The named operator is used to assign a unique relationship name to the group of outdated records in the related table.

Note: The delete records command marks records as deleted, but does not physically delete them from disk until the specified table is reorganized. If records are frequently deleted, reorganizing the table will increase performance.

do

Type

Command component

Purpose

The do keyword is a component of the while command syntax.

When processing reaches a while command, DataEase evaluates the condition that follows the keyword while. If the specified condition is true, DataEase executes all the actions that follow the keyword do until it reaches the corresponding end command. DataEase then reevaluates the original condition. If the condition is still true, DataEase executes the do action series again. If the condition is false, processing passes to the first action following the end command for the while statement.

Syntax

```
while CONDITION do
    ACTION 1 .
    [ACTION 2 . ACTION 3 .]
end
```

Example

```
define temp "CRUISE TICKET NUM" Number .
assign temp CRUISE TICKET NUM := 0 .
while temp CRUISE TICKET NUM <=1000 do
    temp CRUISE TICKET NUM := temp CRUISE TICKET NUM + 1 .
    list records
        jointext ( "Cruise Boarding Pass No. " ,
            temp CRUISE TICKET NUM) .
end
```

This script tells DataEase: (1) Create (define) a temporary variable called CRUISE TICKET NUM, (2) give (assign) an initial value of 0 to the CRUISE TICKET NUM variable, (3) print a series of labels joining the words Cruise Boarding Pass No. to the number that is the current value of the CRUISE TICKET NUM variable, (4) while printing the labels, increment the variable by one each time a new label is printed, and (5) when the value of the variable exceeds 1000, stop printing labels.

The while command tells DataEase to reevaluate the value of the variable each time it prints a label. As long as that value is less than or equal to 1000, DataEase prints another label. When the value of the variable exceeds 1000, DataEase stops performing the action following the do keyword.

else

Type

Procedural Command component

Purpose

The else keyword is a component of the if command syntax.

The if command executes one of two different actions (or series of actions) based on whether the specified condition is true or false. When processing reaches an if command, DataEase evaluates the condition that follows the keyword if. If the specified condition is true, DataEase executes all the actions which follow the keyword then until processing reaches the corresponding end or else command. If the specified condition is false, DataEase executes all the actions that follow the keyword else until processing reaches the corresponding end command. If there is no else, DataEase jumps directly to the statement following the end command.

Syntax

```
if CONDITION then
    ACTION 1 .
    [ACTION 2 .
    ACTION 3 .]
[else
    ACTION 1 .
    [ACTION 2 .
    ACTION 3 .]]
end
```

Example

```
for MEMBERS ;
    if highest of RESERVATIONS DATE < 01/01/99 then
        delete records
    else
        delete records in RESERVATIONS
            with ( DATE < 01/01/99) .
    end
end
```

This script tells DataEase: (1) For each MEMBERS record, find the most recently dated record in the related RESERVATIONS table, (2) if the most recent reservation is dated before January 1st, 1999, delete the MEMBERS record, and (3) if a member's most recent reservation is dated on or after January 1st, 1999, delete all of that member's RESERVATIONS records dated prior to 1999.

Note that the first end marks the end of the if command that selects which records to delete. The second end marks the end of the for command that selects the records from the Primary table.

end

Type

Procedural Command

Purpose

The end command marks the end of the span of control of other processing and/or procedural commands.

Syntax

```

    ACTION 1 .
    ACTION 2 .
    .
    .
    ACTION N .
end
```

Usage

An end command is required following the last specified action invoked by a for, case, if, or while command.

When multiple procedural commands are used in a script, the proper positioning of end commands is critical to correct processing. Each end command refers to the most recent procedural command in the script. Each procedural command controls processing of the script until it reaches its matching end command (see Example 2).

Example 1

```

for MEMBERS ;
  list records
    LAST NAME ;
    TOTAL DUE .
end
```

This script tells DataEase: Process all the MEMBERS records and list each member's LAST NAME and TOTAL DUE in the report output.

Example 2

```

for MEMBERS ;
  if highest of RESERVATIONS  DATE < 01/01/99 then
    delete records
  else
    delete records in RESERVATIONS
      with (  DATE < 01/01/99)
  end
  if highest of RESERVATIONS DATE  >= 01/01/99 then
    list records
      LAST NAME in order ;
      highest of RESERVATIONS  DATE .
  end
end

```

This script tells DataEase: (1) For each MEMBERS record, find the most recently dated record in the related RESERVATIONS table, (2) if the most recent reservation is dated before January 1st, 1999, delete the MEMBERS record, (3) if a member's most recent reservation is dated on or after January 1st, 1999, delete all of that member's RESERVATIONS records dated prior to 1999, and (4) for all members with reservations dated on or after January 1st 1999, list the members in order by LAST NAME and show the date of each member's most recent reservation.

Notice that for each if command in the script, there is a corresponding end command. The first end marks the end of the if command that selects which records to delete. The second end marks the end of the if command that selects which records to include in the report output. The third end marks the end of the for command that selects the records from the Primary table.

enter a record

Type

Processing Command

Purpose

The enter a record command adds a record to the specified table.

Syntax

Enter a record in TABLENAME

Usage

The enter a record command must specify a target table and the field values to be entered into the table. Alternatively, you can specify another table from which to copy field data using the copy all from command (see Example 2, below). As data is entered in the table, automatic error checking is performed, including checks for uniqueness, required fields, range checks, etc. Any errors are logged to an exception file.

Example 1

```
for MEMBERS with TOTAL DUE  > 150 ;
  enter a record in CATALOG MEMBERS
    NAME 1 := MEMBERS FIRST NAME ;
    NAME 2 := MEMBERS LAST NAME ;
    HOME ADDRESS := MEMBERS ADDRESS ;
    HOME CITY := MEMBERS CITY ;
    HOME STATE := MEMBERS STATE ;
    ZIP CODE := MEMBERS ZIP CODE .
end
```

This script tells DataEase: (1) Select all the MEMBERS records that have an TOTAL DUE greater than \$150, and (2) for each MEMBERS record selected, enter a record in the CATALOGMEMBERS table, copying the information from each field in the MEMBERS table to the assigned field in the CATALOG MEMBERS table.

Example 1 simply copies information from the MEMBERS table into the target table. If the source table and target table have identical field names, you can accomplish the same result with the simpler script shown in Example 2.

Example 2

```
for MEMBERS with TOTAL DUE  > 150 ;
  enter a record in CATALOG MEMBERS
    copy all from MEMBERS .
end
```

error messages off

Type

Procedural Command

Purpose

In its default mode of operation, DataEase displays all error messages generated by DataEase and your SQL database engine. The **error messages off** command turns off these system-generated error messages.

Once DataEase processes an **error messages off** command, no system-generated error messages are displayed until it reaches an **error messages on** command or the end of the whole procedure.

Syntax

```
error messages off .
```

Usage

A DQL Procedure can contain any number of **error messages off** and **error messages on** commands.

The **error messages off** and **error messages on** commands can be used in both Control procedures and Processing procedures.

The **error messages off** command has no effect on messages generated by the DQL message command, or messages generated by triggers, stored procedures, defaults, rules, etc. stored on the server.

Example

```
error messages off .
record entry "MEMBERS" .
run procedure "PRINT RESERVATIONS" .
error messages on .
run procedure "UPDATE CRUISES" .
application status records .
```

This script tells DataEase: (1) Turn off system-generated error messages, (2) display the MEMBERS form so the operator can enter new member records, (3) when the operator finishes entering records, run the PRINT RESERVATIONS procedure, (4) turn on system-generated error messages, (5) run the UPDATE CRUISES procedure, and (6) display the status of the records in the current application.

See also **error messages on**.

error messages on

Type

Procedural Command

Purpose

The **error messages on** command turns on system-generated error messages that have been disabled by the **error messages off** command.

Once DataEase processes an **error messages off** command, no system-generated error messages are displayed until it reaches an **error messages on** command or the end of the whole procedure.

Syntax

```
error messages on.
```

Usage

A DQL Procedure can contain any number of **error messages off** and **error messages on** commands.

The **error messages off** and **error messages on** commands can be used in both Control procedures and Processing procedures.

Example

```
error messages off.
record entry  MEMBERS" .
run procedure  "PRINT RESERVATIONS" .
error messages on.
run procedure  "UPDATE CRUISES" .
application status records.
```

This script tells DataEase: (1) Turn off system-generated error messages, (2) display the MEMBERS form so the operator can enter new member records, (3) when the operator finishes entering records, run the PRINT RESERVATIONS procedure, (4) turn on system-generated error messages, (5) run the UPDATE CRUISES procedure, and (6) display the status of the records in the current application.

See also **error messages off**.

exec SQL

Type

Procedural Command

Purpose

The exec SQL command lets you connect to a specified server, embed an SQL statement in a DQL Procedure, and terminate a connection to a specified server. A procedure may contain any number of exec SQL commands. Each SQL statement must be preceded by the exec SQL command and followed by a period or semicolon.

Embedded SQL statements must use the correct syntax for the target server. When processing reaches an exec SQL command, DataEase SQL passes the SQL statement to the current server with no mediation whatsoever (DataEase does not provide any interactive prompts or check the syntax for you).

Syntax

The exec SQL command is divided into three parts:

1. The first part connects to the server using the following syntax:

```
exec SQL connect CONNECT_ID to ENGINE_NAME SERVER_NAME
      DATABASE_NAME as USER_NAME PASSWORD .
```

2. The second part executes a user-defined SQL statement against a previously established connection using the following syntax:

```
exec SQL at CONNECT_ID ANY SQL STATEMENT [: VARIABLE NAME] ;
```

3. The third part disconnects from a server using the following syntax:

```
exec SQL DISCONNECT CONNECT_ID .
```

Usage

To connect to a specific server and database, follow the exec SQL command with:

- CONNECT_ID – a user-defined name for the connection.
- ENGINE_NAME – the text description of an enabled SQL Engine. This description must exactly match the text description in the Engine Type drop-down list in the Database Links dialog (e.g., Oracle, Other Engines via ODBC).
- SERVER_NAME – the name of the server where the database is stored.
- DATABASE_NAME – the name of an existing database on the specified server.
- USER_NAME – the name of a valid user Logon ID for the specified server. DataEase uses this User Name to log on to the server.
- PASSWORD – the password associated with the specified USER_NAME.

Note: For connections that use the Other Engines via ODBC link option, you must ensure that the ODBC Datasource name is a single word without special characters. In this case the ODBC Datasource name is used for both the SERVER_NAME and the DATABASE_NAME.

To send embedded SQL statements to the server, follow the exec SQL at command with any combination of:

Any valid dynamic SQL statement.

Any DataEase variable name(s) (e.g., current date, temp"SQLTEXT").

:VARIABLE NAME - the name of any valid DataEase variable (temp, global, data-entry, or current) or DataEase field, preceded by a colon.

To terminate a connection to an SQL server, follow the exec SQL command with the keyword disconnect and the CONNECT_ID.

When you use exec SQL to insert SQL statements in a DQL Procedure, you must use the names of the SQL tables and columns, not the corresponding DataEase Form and Field Names. An SQL Table Name or Column Name must not include embedded spaces.

You can combine DQL and exec SQL commands in the same script, but when you use the SQL INSERT, DELETE, or UPDATE commands within a DQL for loop, you must be especially careful about how you use the SQL COMMIT command. Incorrect usage may lead to unexpected results (e.g., you may be locked out or you may not be able to list records that were modified by the exec SQL statement).

All DQL variables (including current, data-entry, temp, and global variables) can be used in conjunction with embedded SQL statements to pass values needed for additional processing. These variables are substituted in the SQL statement when the DQL script executes.

If you use a variable or Field Name in an exec SQL statement, you must precede the variable or Field Name with a colon. The example below shows a colon used before the variable TAX RATE:

```
exec SQL at connect1 UPDATE RESERVATIONS
SET TOTAL_DUE = SUBTOTAL + ( SUBTOTAL* : TAX RATE) ;
```

If you are passing a variable that must normally be enclosed in quotes (for example, a date value used as part of a comparison in SQL), enclose the entire variable (including the colon) within single quotes, as shown:

```
exec SQL at connect1 DELETE FROM CATALOG_MEMBERS
WHERE EXPIRATIONDATE<': current date' ;
```

Example

This sample script demonstrates the use of the exec SQL command as well as the current SQLCODE, current SQLCOUNT, and current SQLMSGTXT variables.

The exec SQL command must precede each SQL command in a DQL script. Each exec SQL statement must be followed by a period.

```
exec SQL connect CONN1 to ORACLE t: oraserv default SCOTT TIGER .
exec SQL at CONN1 DELETE FROM MEMBERS WHERE
OVERDUE_90 = "Y" and PAY_PROCESS = "N" ;
  if current SQLCODE not = 0 then
    exec SQL at CONN1 COMMIT ;
    message jointext ( current SQLCOUNT, " Members deleted. " )
window .
  else
    exec SQL at CONN1 ROLLBACK ;
    message " Delete from Members failed; all changes rolled back.
" window.
    message current SQLMSGTXT window .
  end
exec SQL disconnect CONN1 .
```

The first exec SQL statement.

```
exec SQL connect CONN1 to ORACLE t: oraserv default as SCOTT
TIGER.
```

...logs on to the Oracle server (defined by the Oracle connect string t:oraserv) and connects to the default database via the UserID SCOTT and the Password TIGER. This connection is identified as CONN1.

The second exec SQL command,

```
exec SQL at CONN1 DELETE FROM MEMBERS WHERE
OVERDUE_90 = "Y" and PAY_PROCESS = "N" ;
```

..uses the SQL DELETE command to delete all records for all members whose payments are more than 90 days in arrears and not currently being processed. The SQL statement is applied to connection CONN1. What follows is a DQL if...then...else statement:

```
if current SQLCODE = 0 then
```

This if statement tells DataEase to check the value in the current SQLCODE variable before continuing to process the script. If the value of the current SQLCODE is zero (indicating no errors have occurred), the script continues processing and executes the third exec SQL command:

```
exec SQL at CONN1 COMMIT ;
```

This command commits the changes to the database permanently. DataEase displays a message on the screen telling the user how many MEMBERS records were actually deleted. If current SQLCODE returns a value not equal to zero (indicating an SQL error), the fourth exec SQL command is executed:

```
exec SQL at CONN1 ROLLBACK ;
```

The ROLLBACK command cancels the record deletions before they are permanently saved to the database. DQL messages inform the user that the transaction has failed and display the text of the returned SQL error message.

The final exec SQL command:

```
exec SQL disconnect CONN1 .
```

..disconnects DataEase from the server and ends processing of the script.

exit

Type

Procedural Command

Purpose

The exit command is used to immediately terminate a script.

Syntax

```
exit .
```

Usage

The exit command can be used anywhere in a script. When it is executed, the script immediately stops processing and returns control to the user or calling procedure.

Example

```
for MEMBERS ;
  list records
    TOTAL DUE in reverse ;
    LAST NAME .
  if current item number > 5 then
    exit .
  end
end
```

This script tells DataEase: (1) List the five members with the highest TOTAL DUE values, and (2) when the fifth record is processed, terminate the script. Note that two end commands are required to fulfill the DQL syntax requirements even though the exit command terminates processing.

The output from this script, arranged in descending order on the TOTAL DUE field, might look as follows:

Last Name	Total Due
Christino	\$280.00
Perrault	\$215.00
Strachan	\$205.00
Stafford	\$185.00
Jones	\$175.00

exp

Type

Scientific Function

Purpose

The exp function calculates the exponential value of a numeric value.

Syntax

`exp (NUMERIC VALUE)`

Returns

A numeric value equal to e^x where $e = 2.71828183$

Usage

The numeric value in a Scientific function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

`exp (2)`

Returns:7.3890561

`exp (-3)`

Returns:.0497871

export

Type

Processing Command

Purpose

The export command exports data from the current application to an external file. The export command functions like choosing **File>>Export** in User View.

Syntax

```
export to "FILENAME "
.form headerTEXT | FIELDNAME
.items@f[FORM NUMBER, FIELD NUMBER, FIELD LENGTH]@f[FORM NUMBER,
FIELD NUMBER, FIELD LENGTH]...
.form trailerTEXT
.end
```

In this syntax example, the brackets are required; optional parameters are shown in italics.

Usage

The export command must be preceded in a script by a list records command that specifies the source table and the fields to be exported. The numeric parameters included in the export command refer to the sequential listing of the form(s) and field(s) included in the preceding list records command.

The export command lets you include the following formatting commands: `.form header`, `.items`, `.form trailer`, and `.end`. Items listed under the `.form header`, `.form trailer`, and `.end` commands print only once in the output file DataEase generates. Items listed under the `.items` formatting command, print multiple times, once for each record processed.

Example 1

```
for MEMBERS ;
  list records
    MEMBER ID in order ;
    CARDNO. ;
    TOTALDUE .
end
export to "MEMBDATA.TXT" .
.form header
Member ID~ Card No.~ Total Due
.items
@f[1,1]~ @f[1,2]~ @f[1,3]
.end
```

This example generates a variable length delimited ASCII file. Each record begins on a new line and each field value is separated by a tilde (~) character.

This example tells DataEase: (1) process all the MEMBERS records, listing the MEMBER ID in order, the credit CARD NO., and the total membership dues, (2) export the same data to a file called MEMBDATA.TXT, (3) include a header record in the export file that lists the column names.

Example 2

```

for MEMBERS ;
  list records
    LAST NAME .
    for RESERVATIONS
      list records
        TOTALDUE .
    end
  end
end
export to "a: \MEMTOTAL.TXT" .
.form header members .items
.form header reservations
@f[1,1]~ @f[2,1]
.end

```

This example tells DataEase to: (1) process all the records in the MEMBERS table, listing the LAST NAME of each member, (2) for each MEMBERS record processed, list the TOTAL DUE value from all the related RESERVATIONS records, and (3) export the data to a variable length delimited ASCII file called MEMTOTAL.TXT located on disk drive "A". The syntax @f[2,1] refers to form 2 (RESERVATIONS), field 1 (TOTAL DUE).

Example 3

```

for MEMBERS ;
  list records
    MEMBERID in order ;
    CARDNO. ;
    TOTALDUE .
  end
end
export to "MEMBDATA.TXT" .
.form header
.items
@f[1,1,5]@f[1,2,20]@f[1,3,7]
.end

```

This example generates a fixed length ASCII file and includes the same data as in Example 1. The third digit in the field specification indicates the length of the field.

firstc

Type

Text Function

Purpose

The firstc function extracts a specified number of characters (n) from the beginning (left) of a text value.

Syntax

```
firstc( TEXT VALUE, n)
```

Returns

A text string n characters in length.

Usage

Leading blanks are included in the count of characters (n).

Examples

```
firstc ( "Sapphire International Ltd." , 8)
```

Returns: Sapphire

```
firstc ( "Club ParaDEASE" , 6)
```

Returns: Club P

```
firstc ( LAST NAME, 4)
```

Returns: The first four characters in the LAST NAME field for every record that is processed. If a record contains the value Birnbaum in the LAST NAME field, the function returns Birn.

firstlast

Type

Text Function

Purpose

The firstlast function converts a text value from the format: LastName, FirstName[M.] to the format: FirstName[M.]LastName. The [M.] is an optional middle initial (or name).

Syntax

```
firstlast( TEXT VALUE)
```

Returns

A text value.

Usage

This function is used when names are stored in a single field. The contents of the field are rearranged so the first word (i.e., the last name) becomes the last word in the returned string. If the name is followed by a comma, the comma is deleted.

Examples

```
firstlast ( "Anthony, Susan B." )
```

Returns: Susan B. Anthony

```
firstlast ( "Eliot, T.S." )
```

Returns: T.S. Eliot

```
firstlast ( FULL NAME)
```

Returns: The value in the FULL NAME field in the format shown above for every record that is processed. If a record contains the value Holmes, Sherlock in the LAST NAME field, the function returns Sherlock Holmes.

firstw

Type

Text Function

Purpose

The firstw function extracts a specified number of words from the beginning (left) of a text value.

Syntax

```
firstw( TEXT VALUE, n)
```

Returns

A text value n words in length.

Usage

firstw(FIELDNAME, n) returns the first n words in the field, including intervening punctuation symbols. Leading spaces are treated as delimiters. Trailing spaces are ignored. If there are n or fewer words in the field, firstw returns the original text value.

Examples

```
firstw( "Sapphire International Ltd." , 2)
```

Returns: Sapphire International,

```
firstw( "Club ParaDEASE" , 1)
```

Returns: Club

```
firstw( ADDRESS , 2)
```

Returns: The first two words in the ADDRESS field for every record that is processed. If a record contains the value 540 Avenida de los Delfines in the ADDRESS field, the function returns 540 Avenida.

floor

Type

Math Function

Purpose

The floor function rounds down a numeric value to the next lowest integer.

Syntax

```
floor( NUMERIC VALUE)
```

Returns

An integer value.

Usage

The numeric value in a Math function can be a constant value, a variable, a field value, or an expression.

Examples

```
floor( 5.000)
```

Returns: 5

```
floor( 5.999)
```

Returns: 5

```
floor ( ( current date - BIRTHDATE) /365.25)
```

Returns: The result of the age calculation. The operation contained in the interior set of parentheses is performed first and returns an age value expressed in days. The exterior calculation converts the age in days to years. If the current date is 05/30/2001 and the value in the BIRTHDATE field is 07/02/62, the function returns 38.

for

Type

Processing Command

Purpose

The for command is used in a script to specify the table from which records are selected for processing and the criteria used to select those records.

Usage

The for command is the most frequently used word in the DataEase Query Language. Most Processing procedures begin with this command.

A script can have multiple for commands when records must be selected from different tables (see Examples 1 and 2). for commands can be nested to alter the sequence of program actions (see Example 3).

The for command is always terminated by an end command. Actions specified between a for command and its corresponding end command are executed once for each record selected by the for command.

Syntax

```
for TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] ;
```

The for command usually requires a semicolon after the TABLENAME (and selection criteria, if any). However, when a for command is nested within another for command, only the outermost for command requires a semicolon (see Example 3).

Example 1

Example 1 shows how to use a for command to access a single table.

```
for MEMBERS with TOTAL DUE > 175 ;
    list records
        LAST NAME in order ;
        TOTAL DUE .
end
```

This script tells DataEase: (1) Select the MEMBERS records that have a value greater than \$175 in the TOTAL DUE field, and (2) for each selected record, list the member's LAST NAME and TOTAL DUE. The output from this script, arranged in alphabetical order by LAST NAME, might look like this:

Last Name	Total Due
Christino	280.00
Perrault	215.00
Stafford	185.00
Strachan	205.00

Example 2

Example 2 shows how to use sequential for commands without nesting.

```

for MEMBERS with TOTAL DUE > 200 ;
  list records
    LASTNAME in order .
end

for RESERVATIONS ;
  list records
    LASTNAME in order ;
    RESERVATION I D ;
    TOTAL DUE .
end

```

This script tells DataEase: (1) Select the MEMBERS records that have a value greater than \$200 in the TOTAL DUE field, (2) for each selected record, list the member's LAST NAME, (3) select all the RESERVATIONS records, and (4) for each RESERVATIONS record, list the LAST NAME, RESERVATION ID and TOTAL DUE.

When a script contains two or more for statements that list records, DataEase prints or displays all records requested by the first for statement first, then displays all records requested by the second for statement, and so on. The output for the script in Example 2 might look like this:

Last Name

Christino
Perrault
Stafford
Strachan

Last Name	Reservation ID	Total Due
...
Christenson	00011	3,360.00
Christino	00139	5,450.00
Christino	00259	4,570.00
Christino	00765	6,490.00
Chu	00113	2,780.00
Chu Cipriano	00541	3,480.00
Clark	00052	2,660.00
...	00194	6,290.00

Example 3

Example 3 shows how to use nested for commands.

Although you can display data from multiple tables using sequenced for statements, as shown in Example 2, a script may run faster and generate more useful output when it contains nested for statements as shown in Example 3:

```
for MEMBERS with TOTAL DUE    > 200 ;
  for RESERVATIONS
    list records
      MEMBERS LAST NAME in groups ;
      RESERVATIONID    in order ;
      TOTAL DUE .
    end
  end
```

In this script, the second for command, which is nested within the first, selects all RESERVATIONS records related to the selected MEMBERS records. For each of those RESERVATIONS records, DataEase performs all the actions between the second for and its corresponding end.

This query tells DataEase: (1) Select the MEMBERS records that have a value greater than \$200 in the TOTAL DUE field, (2) find all the related RESERVATIONS records for each MEMBERS record selected in Step 1, (3) for each selected MEMBERS record, list the member's LAST NAME, and (4) for each selected RESERVATIONS record, list the RESERVATION ID and the TOTAL DUE.

Notice that in Example3, there is no semicolon after the second for because it is nested within another for command. In contrast, both for statements in Example 2 require a semicolon because neither statement is nested inside another for statement. The output from this script, arranged in alphabetical order by LAST NAME, might look as shown on the next page.

Last Name	Reservation ID	Total Due
Christino	00139	5,450.00
	00259	4,570.00
	00765	6,490.00
Perrault	00150	2,900.00
	00445	2,790.00
	00671	3,010.00
Stafford	00098	5,712.00
Strachan	00044	3,740.00
	00337	4,120.00

The in groups operator tells DataEase to processes all the records for each member together as a group and to list the members in alphabetical order. The in order operator tells DataEase to arrange the data for each member in ascending order by RESERVATION ID.

Note: The in groups operator always precedes the in order operator in a script.

Functions

Type

Concept

Purpose

The 58 DataEase functions execute calculations, manipulate text, and perform other time-saving data-entry operations. There are nine groups of functions, briefly summarized below.

- The if function returns one of two values based on its evaluation of a specified condition as true or false.
- Date functions convert a date into a number representing the day, month, year, day of the week, or day of the year on which the specified date falls.
- Spell functions convert a date or numerical value into its equivalent text value.
- Time functions extract the hour, minutes, or seconds from a 24-hour format time value or assign the appropriate AM or PM suffix.
- Text functions are used to concatenate, truncate, or manipulate a text value.
- Financial functions calculate either the beginning value, end value, interest rate, installment amount, or number of payment periods for a financial transaction, given the values of the other four variables.
- Scientific functions raise a base value to a power, derive natural and base 10 logs, or derive the square root of a value.
- Trigonometric functions convert a numeric value into a value expressed in radians.
- Math functions perform common mathematical rounding operations or return the absolute value of a specified numeric value. The random function generates a pseudo random number.

For a full explanation of any DQL function, see the entry listed under the function's name in this Language Reference.

futurevalue

Type

Financial Function

Purpose

The futurevalue function calculates the final value of a financial transaction given the presentvalue, installment payment, interest rate, and number of payment periods.

Syntax

```
futurevalue( presentvalue, installment, rate, periods)
```

Returns

A numeric value (the final value after all payments are made).

Usage

If the installment payment increases the value of the investment (e.g., an annuity), the installment must be expressed as a positive amount. If the payment decreases the value of the investment (e.g., loan amortization), the installment must be expressed as a negative amount.

The interest rate, installment payments, and time periods must be based on the same terms. For example, if payments are made monthly on a five-year loan, the number of periods is (12*5) or 60. The interest rate must be expressed in the same terms (the monthly interest rate is the annual rate divided by 12). The interest rate must also be expressed as a whole number (10% is 10, not 0.10).

Example 1

```
futurevalue( 10000, 100, 10 / 12, 120)
```

Returns: 47,554.91

This example calculates the futurevalue (accumulated balance) of a monthly savings plan, if you start with \$10,000 and add \$100 every month for 12 years at an annual interest rate of 10%.

Example 2

```
futurevalue( 10000, 1200, 10, 12)
```

Returns: 57,045.42

This example calculates the futurevalue (accumulated balance) of an annual savings plan, starting with \$10,000 and adding \$1200 each year for 12 years at an annual interest rate of 10%. Both examples assume that installments are paid at the start of each period.

Note: When typing fractions like 10 / 12, DataEase requires that you type a space before and after the / division symbol.

DataEase financial functions are derived from the formula shown below. (The double asterisks (meaning "raise to the power") cannot be used in a script).

$$\text{futurevalue} = \text{principal} * ((1 + (\text{rate}/100)) ** \text{periods}) + (\text{installment}/(\text{rate}/100)) * (((1 + (\text{rate}/100)) ** \text{periods}) - 1)$$

global

Type

Keyword

Purpose

The keyword global specifies a global variable.

A variable is used to store a value such as a text string or a calculated result that can change during the processing of a procedure. By specifying the variable's name in a script, the stored value can be used like any other value.

The status of a variable can be global (denoted by the keyword global) or temporary (denoted by the keyword temp).

A temporary variable can hold a value only while processing the current script.

A global variable can pass its value from one script to another within a Control procedure. To pass a value from one script to another, a variable must be defined identically in each script.

Syntax

```
global VARIABLE NAME
```

Usage

When processing a Control procedure that links two or more Processing procedures, each global variable with the same name and type is considered the same variable and can pass its value from one Processing procedure to another.

Example

```
define global  "INCOME"   Number .
define global  "EXPENSE S" Number .
define global  "PROFI T"  Number .
run procedure  "MONTHLY TOTALS" .
if global INCOME > global EXPENSES then
    run procedure  "PAYROLL BONUS" .
else
    run procedure  "REGULAR PAYROLL" .
end
```

This script is a Control procedure that tells DataEase: (1) Create (define) three global variables called INCOME, EXPENSES, and PROFIT to store three different numbers while processing the script, (2) run the MONTHLY TOTALS procedure. The MONTHLY TOTALS procedure (shown below) calculates the total income and expenses for the month and subtracts the latter from the former to determine the amount of profit. As each of these totals is calculated, it is assigned to the appropriately named global variable, (3) if the INCOME variable is greater than the EXPENSES variable, run the PAYROLL BONUS procedure using the value stored in the PROFIT variable to calculate each employee's bonus (a percentage of the profit based on the employee's salary level), and (4) if the INCOME variable is not greater than the EXPENSES variable, run the REGULAR PAYROLL procedure.

The MONTHLY TOTALS Processing procedure looks like this:

Example

```
define global "INCOME" Number .
define global "EXPENSES" Number .
define global "PROFIT" Number .
assign global INCOME := sum of RESERVATIONS
    with ( DATE between 06/01/01 to 06/30/01) TOTAL DUE .
assign global EXPENSES := sum of EXPENDITURES
    with ( DATE between 06/01/01 to 06/30/01) TOTAL COST . assign
global PROFIT : = global INCOME - global EXPENSES .
end
```

The key feature of this example is the fact that the global variables are identically defined in the Processing procedure which calculates the value of each variable and in the Control procedure which passes these values from one Processing procedure to another.

Grouping

Type

Concept

Purpose

Grouping lets you process records with the same field value together.

Usage

There are two DQL operators that control grouping: `in groups` and `in groups with group-totals`. They perform identically; “`in groups with group-totals`” is included for compatibility with previous versions of DataEase.

The `in groups` operator causes records with identical values in a specified field to be processed as a single group. The groups are automatically sorted in ascending order. For example, French Polynesian clubs are listed together as a group followed by the Haitian clubs, and so on.

Example 1

```
for CLUBS ;
  list records
    COUNTRY in groups ;
    CLUB NAME in order ;
    CITY .
end
```

The output generated by the script in Example 1 is grouped on the **COUNTRY** field (shown in bold):

Country	Club Name	City
French Polynesia	Bora Bora	Papeete
	Moorea	Moorea
Haiti	Magic Isle	La Pointe aux Sables
Mexico	Cancun	Yucatan
	Chichen Itza	Chichen Itza
	Cholula	Cholula
	Coba	Coba
	Huatulco	Oaxaca
	Ixtapa	Guerrero
	Playa Blanca	Jalisco
	Sonora Bay	Sonora
	Teotihuacan	Villa de Teotihuacan
	Uxmal	Uxmal

Example 2

The in groups operator groups records and generates statistical totals for each group if any other statistic is requested by the script.

```
for CLUBS ;
  list records
    COUNTRY in groups ;
    CLUB NAME in order ;
    CITY ;
    "A" = "A" : count .
end
```

In this example, the count statistic allows DataEase to calculate the group total for each country. The line:

```
"A" = "A" : count .
```

..tells DataEase to count every record that is processed.

The output includes the group level statistics (shown in bold):

Country	Club Name	City
French Polynesia	Bora Bora	Papeete
	Moorea	Moorea
Group Total 2		
Haiti	Magic Isle	La Pointe aux Sables
Group Total 1		
Mexico	Cancun	Yucatan
	Chichen Itza	Chichen Itza
	Cholula	Cholula
	Coba	Coba
	Huatulco	Oaxaca
	Ixtapa	Guerrero
	Playa Blanca	Jalisco
	Sonora Bay	Sonora
	Teotihuacan	Villa de Teotihuacan
	Uxmal	Uxmal
Group Total 10		
...
Total CLUBS: 25		

There are two key points to remember when using Grouping:

Grouping automatically sorts groups in ascending alphabetical or numerical order (e.g., French Polynesian clubs before the Haitian clubs).

Grouping operators must precede Sorting operators (in order and in reverse) in a script.

highest of

Type

Relational Statistical Operator

Purpose

The highest of operator finds the highest value in a specified field in all matching records in a related table. The result can appear as a list item in the detail area of a report or as a statistic in the summary area at the end of each group or the end of the report.

Syntax

```
highest of TABLENAME|RELATIONSHIP [named "UNIQUE RELATIONSHIP NAME"
] [with ( selection criteria) ] FIELDNAME ;|.
```

Returns

A value of the same type as the specified field. If a Text field is specified, DataEase returns the highest ASCII value. If a Choice field is specified, DataEase returns the value in the highest numbered choice (not necessarily the highest ASCII value).

Example

```
for MEMBERS ;
  list records
    LASTNAME in order ;
    highest of RESERVATIONS TOTAL DUE .
end
```

This script tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, (2) for each MEMBERS record, find all the related records in the RESERVATIONS table (those that have a matching MEMBER ID), and (3) list the highest TOTAL DUE in the set of matching RESERVATIONS records.

The output from this script, arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	Highest of Reservations Total Due
Adams	\$3000.00
Albert	\$4760.00
Anders	\$4420.00
Andersen	\$2100.00
...	...

If you also want to include the highest TOTAL DUE among this group of invoices, change the fourth line of the query to read:

```
highest of RESERVATIONS TOTAL DUE : item max .
```

Note: There's an important difference between the statistical operator max and the relational statistical operator highestof. max finds the highest value in the specified field among all the records being processed. highest of finds the highest value among the records related to the records being processed.

hours

Type

Time Function

Purpose

The hours function extracts the hour from a time value expressed in a 24-hour format.

Syntax

```
hours( TIME VALUE)
```

Returns

An integer value from 00 to 23.

Examples

```
hours(12:00:00) {midday}
```

Returns:12

```
hours(23:59:00) {one minute before midnight}
```

Returns:23

Note: The bracketed text above (e.g., midday) is used to clarify the time of day only; it is not one of the function's parameters.

if Command

Type

Procedural Command

Purpose

The if command executes one of two different actions (or series of actions) based on whether a specified condition is true or false.

When processing reaches an if command, DataEase evaluates the condition that follows the keyword if.

If the specified condition is true, DataEase executes all the actions which follow the keyword then until processing reaches the corresponding end or else command.

If the specified condition is false, DataEase executes all the actions that follow the keyword else (if present) until processing reaches the corresponding end command.

Syntax

```
if CONDITION then
    ACTION 1 .
    [ACTION 2 .
    .
    .
    ACTION N .]
[else
    ACTION 1 .
    [ACTION 2 .
    .
    .
    ACTION N . ]]
end
```

Usage

A script may contain multiple if commands nested within one another (see Example 2). An additional if command can appear after either then or else.

Each if command must contain at least one true action (following the keyword then). When an if statement contains an else command, at least one action must follow the keyword else.

Example 1

```
for MEMBERS ;
    if highest of RESERVATIONS DATE <01/01/99 then
        delete records .
    else
        delete records in RESERVATIONS
        with ( DATE < 01/01/99) .
    end
end
```

This script tells DataEase: (1) For each MEMBERS record, find the most recently dated record in the related RESERVATIONS table, (2) if the most recent invoice is dated before January 1st, 1999, delete the MEMBERS record, and (3) if a member's most recent invoice is dated on or after January 1st, 1999, delete all of that member's RESERVATIONS records dated prior to 1999.

The first end marks the end of the if command that selects the records to delete. The second end marks the end of the for command that selects the records from the Primary table.

Example 2

Nested if Command:

```
for RESERVATIONS ;
  if TOTAL DUE    > 3000 then
    modify records in MEMBERS
    STATUS := PREFERRED .
    if DATE between 01/01/99 to current date then
      enter a record in CATALOG MEMBERS
      copy all from MEMBERS .
    end
  end
end
end
```

This script tells DataEase to divide the RESERVATIONS records into two groups and process them as follows: (1) The records with a TOTAL DUE greater than \$3000 are modified by changing the value in the STATUS field to PREFERRED, and (2) the records with a TOTAL DUE greater than 3000 and a DATE between the start of 1994 and today's date are modified and also copied into the CATALOG MEMBERS table.

In this example, the second if command, which selects a subgroup of the records selected by the first if command, is nested within the first if command.

if Function

Type

Conditional Function

Purpose

The if function evaluates a specified condition and returns one of two specified values based on whether the condition is true or false.

Syntax

```
if( CONDITION, TRUE VALUE, FALSE VALUE)
```

Returns

The true value if the specified condition is true. The false value if the specified condition is false.

Usage

In a field Derivation formula, you can use the if function to:

- derive a field value based on one of two specified values or expressions depending on whether a condition is true or false (see Example 1).

In a Validation formula, you can use the if function to:

- check a field's validity based on one of two specified expressions depending on whether a condition is true or false (see Example 2).

When you use an if function in a Derivation or Validation formula, the true value and false value parameters must be of the same data type as the field being derived or validated.

In a script, you can use the if function to:

- assign one of two specified values to a field or variable based on whether the condition is true or false (see Example3).
- hide or show a field, calculated value, or text string in a procedure's output depending on whether a condition is true or false (see Example 4).

A script, Derivation formula, or Validation formula may have multiple if function statements nested within one another.

Example 1

Using the if function in a Derivation formula:

The following example shows the if function used to derive the value in a Text field. The function returns one of two text values depending on the current time in the computer's system clock.

```
if( ampm( current time)    =    "AM" , " in the morning " ,
    "in the afternoon" )
```

Because this Derivation formula returns a value for a Text field, the true value ("in the morning") and the false value ("in the afternoon") are both text expressions.

The following example shows the if function used in a Derivation formula for a Number field named DISCOUNT. The formula returns one of two DISCOUNT values depending on the value in another field, TOTAL DUE.

```
if( TOTAL DUE >= 500, TOTAL DUE * 0.15, TOTAL DUE * 0.03)
```

This formula evaluates the value in the TOTAL DUE field. If that value is greater than or equal to \$500 (making the condition true), the formula sets the value of DISCOUNT to TOTAL DUE * 0.15. If the value of TOTAL DUE is less than \$500 (making the condition false), the formula sets the value of DISCOUNT to TOTAL DUE * 0.03.

By nesting if statements inside one another, you can define a Derivation formula that evaluates multiple conditions and returns one of several values. The example below shows how one if statement can be nested inside another to return one of three DISCOUNT values depending on the value in the TOTAL DUE field:

```
if( TOTAL DUE >= 500, TOTAL DUE * 0.15,
if( TOTAL DUE >= 200, TOTAL DUE * 0.08, TOTAL DUE * 0.03) )
```

This formula evaluates the value in the TOTAL DUE field. If that value is greater than or equal to \$500 (making the condition true), the formula sets the value of DISCOUNT to the TOTAL DUE multiplied by 0.15. If the value is less than \$500 (making the condition false), a second if statement evaluates TOTAL DUE to determine if its value is greater than or equal to \$200. If this second condition is true, the value of DISCOUNT is set to the TOTAL DUE multiplied by 0.08. Otherwise DISCOUNT is set to TOTAL DUE * 0.03.

Example 2

Using the if function in a Validation formula:

The example below shows how the if function can be used in a Validation formula to conditionally apply one of two validation criteria to a field named CREDIT PURCHASES.

```
if( OVERDUE DAYS < 90, <=5000, <50)
```

This Validation formula evaluates the value in a field named OVERDUE DAYS (this field tracks a customer's past-due invoices). If that value is less than 90 (meaning the customer has no invoices over 90 days past due), then any value up to \$5000 is valid in the CREDIT PURCHASES field. If the value of OVERDUE DAYS exceeds 90, then only up to \$50 can be saved in the CREDIT PURCHASES field.

Example 3

Using the if function to assign a value in a DQL script:

The examples below show how to use the if function in a DQL script to conditionally assign a value to a field or variable depending on whether a condition is true or false.

```
modify records in RESERVATION AGENTS
BONUS := if( SALES > 80000, 1000, 0) .
```

This script assigns a value to the BONUS field in all RESERVATION AGENTS records. If an employee generated more than \$80,000 in sales last quarter, the value in the BONUS field is set to 1000. Otherwise the value of BONUS is set to zero.

The following line shows the if function used to assign a value to a temporary variable in a DQL script:

```
assign temp BONUS := if( SALES > 80000, 1000, 0) .
```

Example 4

You can use the if function to conditionally display fields, calculated values, or text strings in the output of a DQL procedure. The script below uses the if function to display the SALARY field in a procedure's printed output only if the procedure is run by a specific user:

```
for EMPLOYEES ;
  list records
    LAST NAME in order ;
    FIRST NAME ;
    JOB TITLE ;
    if( current user name = "Moe" , SALARY, BLANK) .
end
```

This script tells DataEase: (1) For each record in the EMPLOYEES table, list the LAST NAME, FIRST NAME, and JOB TITLE, (2) evaluate the current user name system variable, and if the current user is Moe, list each employee's SALARY along with his/her other data, (3) otherwise, do not list the SALARY field.

import

Type

Control Command

Purpose

The import command imports a data file into the current application. It can be used to import data at any point during a procedure. After executing the import, the data can be used in the same procedure.

Syntax

```
import  "IMPORT SPECIFICATION FILENAME"  .
```

Usage

The import command can only be used to execute a previously defined Import Specification. The filename that follows the import command is the filename of the Import Specification. This name must be eight or fewer characters with no intervening spaces.

LAN

On a LAN (Local Area Network), if another user is currently using any resource required by the import command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

```
import  "MAILING LIST"  .  
run procedure  "PRINT CONFIRMATIONS"  .
```

This script tells DataEase: (1) Execute the MAILING LIST Import Specification, and (2) when the import is completed, run the PRINT CONFIRMATIONS procedure.

in

Type

Keyword

Purpose

The in operator tells DataEase to use a table other than the script's Primary table when executing a list records, delete records, modify records, or enter a record command.

Syntax

```
in TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ]
```

Usage

When the in operator is used without selection criteria, DataEase performs the action on all related records in the specified table. The in operator should not be used to process records in the script's Primary table (see the Caution below).

Example 1

```
    for MEMBERS ;
    modify records in RESERVATIONS
        TOTALDUE := TOTALDUE - DISCOUNT .
end
```

This script discounts every related RESERVATIONS record each time a MEMBERS record is processed. Since every reservation is related to only one member, each record is discounted only once.

Example 2

```
    for RESERVATIONS ;
    modify records
        TOTALDUE := TOTALDUE - DISCOUNT .
end
```

This script correctly modifies records in the Primary table. The in keyword is omitted from a list records, modify records, or delete records command when the action is performed on the Primary table.

Caution

Be careful not to use the following construction in a script:

```
    for RESERVATIONS ;
    modify records in RESERVATIONS
        TOTAL DUE := TOTAL DUE - DISCOUNT .
end
```

This example demonstrates an incorrect use of the in operator. This script discounts every RESERVATIONS record each time a record is processed (i.e., each reservation is discounted as many times as there are records to process).

in groups

Type

Grouping Operator

Purpose

The in groups operator groups records and generates statistical totals for each group if any other statistic is requested by the query.

Note: the variation “in groups with group-totals” is included for compatibility with previous version of DataEase. Either version will include statistics if they are required

Syntax

```
FIELDNAME in groups [with group-totals] ;
```

Usage

The in groups operator can be used only on list items following the list records command. It cannot be used with the enter a record, modify records, or delete records commands.

In the procedure output, the group identifier appears only once at the beginning of a group. If statistics are specified for any output fields, group subtotals for those fields are included in the output.

An item to be processed in groups with group-totals should be listed before all other output items in a script, including items to be sorted in order. When a field is listed in groups with group-totals, DataEase also sorts the groups in order (e.g., when grouping MEMBERS by STATE, all California members are listed before any Colorado members).

Caution

Grouping operators cannot be used with the all relational operator (i.e., if RESERVATIONS is the Primary table, you cannot use the query statement: all MEMBERS STATE in groups.

Example

```
for MEMBERS ;
list records
  STATE in groups ;
  LASTNAME in order ;
  TOTALDUE : item sum .
end
```

This script tells DataEase: (1) Process all MEMBERS records with the same value in the STATE field as a group, (2) display the group identifier (STATE) once at the beginning of each group, (3) within each group, arrange the members alphabetically by LAST NAME, and (4) list each member's TOTAL DUE, the subtotal for each STATE group, and the sum of all the TOTAL DUE amounts combined. The output from this query might look as follows:

State	Last Name	Total Due
...
DC		
	Dowling	\$115.00
	Schmidt	\$85.00
	Spinelli	\$100.00
	Group Total:	\$300.00
DE		
	Gross	\$35.00
	Stromboulis	\$70.00
	Group Total:	\$105.00
...
	Grand Total:	\$405.00

in order

Type

Sorting Operator

Purpose

The in order operator tells DataEase to process records and display the output in sequence from lowest to highest value in the specified field. (e.g. 01011, before 01012, before 01013, etc.).

Syntax

```
FIELDNAME in order ;|.
```

Usage

The in order operator can be used only on list items following the list records command. It cannot be used with the enter a record, modify records, or delete records commands.

The in order operator can be used on any type of field. A Time, Date, or Number field is ordered from lowest to highest value. A Choice field is sorted by Choice number. A Text field is sorted in alphabetical order.

DataEase orders alphanumeric values in standard ASCII sequence and makes no distinction between upper and lower case.

If a script specifies an item to be processed in groups, that item should be listed before all other output items in a script, including items to be sorted in order. When a field is listed in groups, DataEase also sorts the groups in order (e. g., when grouping MEMBERS by STATE, all Alabama members are listed before any Alaska members).

Caution

Sorting operators cannot be used in conjunction with the all relational operator (e.g., if RESERVATIONS is the Primary table, you cannot use the statement:

```
all MEMBERS LAST NAME in order ;
```

Example

```

for RESERVATIONS ;
  list records
    STATE in groups ;
    ZIP CODE in order ;
    LAST NAME in order .
end

```

This script tells DataEase: (1) Process all the RESERVATIONS records with the same value in the STATE field together as a group, (2) display the group identifier STATE once at the beginning of each group, (3) within each group, arrange the members in order from lowest to highest ZIP CODE, and (4) within each ZIP CODE, arrange the members in alphabetical order by LAST NAME. A portion of the output from this script might look as follows:

State	Zip Code	Last Name
...
DC		
	20007-3947	Spinelli
	20036-0000	Dowling
	20036-8749	Schmidt
DE		
	19810-3729	Stromboulis
	19901-3445	Gross
...

in reverse

Type

Sorting Operator

Purpose

The in reverse operator tells DataEase to process records and display the report output in reverse sequence from highest to lowest value in the specified field.

Syntax

```
FIELDNAME in reverse ;|.
```

Usage

The in reverse operator can be used only on list items following the list records command. It cannot be used with the enter a record, modify records, or delete records commands.

The in reverse operator can be used on any type of field. A Time, Date, or Number field is ordered from highest to lowest value. A Choice field is sorted by Choice number from highest to lowest. A Text field is sorted in reverse alphabetical order.

DataEase orders alphanumeric values in standard ASCII sequence and makes no distinction between upper and lower case. Sorting operators cannot be used in conjunction with the all relational operator (e.g., if RESERVATIONS is the Primary table, you cannot use the statement:

```
all MEMBERS LAST NAME in reverse ;
```

Example

```
for MEMBERS ;
  list records
    STATE in groups ;
    ZIP CODE in reverse ;
    LAST NAME in order .
end
```

This script tells DataEase: (1) Process all the MEMBERS records with the same value in the STATE field together as a group, (2) display the group identifier STATE once at the beginning of each group, (3) within each group, arrange the members in highest to lowest ZIP CODE code order, and (4) within each ZIP CODE code, arrange the members in alphabetical order by LAST NAME. A portion of the output from this script might look as follows:

State	Zip Code	Last Name
DC		
	20036-8749	Schmidt
	20036-0000	Dowling
	20007-3947	Spinelli
DE		
	19901-3445	Gross
	19810-3729	Stromboulis

input using

Type

Processing Command

Purpose

The input using command provides all the facilities of record entry but allows DQL to process records before they are entered into the database. It can be used at any point in a procedure.

Syntax

```
input using TABLENAME into "TEMPFORM" via form EXISTINGFORM .
```

In the input using command syntax, FORMNAME is the name of a table. TEMPFORM is the name DataEase uses to reference an input using record. You can assign any name you want for this purpose except the name of a form in the same application. The first time that the TEMPFORM name is specified in the script, it must be enclosed in quotation marks.

The optional "via form EXISTINGFORM" displays the form document EXISTINGFORM, which is the name of an existing form that uses as its main table the table specified by TABLENAME.

If no form name is specified, DataEase displays the form that has the same name as the table, usually the form that defines the table.

There is no limit to the number of input using commands you can use in a script; however, each DataEase form must be assigned a unique TEMPFORM in the input using statement.

Usage

The input using command tells DataEase to display a specified record entry form. When the specified form is displayed, you can either enter or modify a record (in Form View), or scroll through the form (in Form or Table View) and select any existing record. You can also use the **Goto>>Related Form** menu option (or F10 key) to view records in related forms.

When you execute a processing action (e.g., **File>>Close**, **Edit>>Save As New Record**), the input record is copied into memory and a number that corresponds to the processing key is stored in the current status variable. DataEase then resumes processing the procedure, using the input using record data just like values entered in a Data-entry form. The current status value can be used to control subsequent processing. The table below summarizes the current status values returned by each processing action:

This Menu Selection	Performs This Action	And Returns This Value
File>>Close	Close the active document	1
Edit>>Save As New Record	Save a new record	2
Edit>>Save	Modify an existing record	3
Edit>>Delete	Delete a record	4

Like a Data-entry form, the input using command is used to enter data for use in a procedure (rather than to permanently store the data in a table in the database). Unlike a Data-entry form, however, more than one input using statement can be used to enter data at any point during a procedure.

The input using command can only hold one record in memory at a time for a given TEMPFORM name. Therefore, if you want to save the contents of an input using record, you must use the enter a record command in the script to save the current record before processing another. Example 2 demonstrates the use of the input using command and current status variable within a loop to process multiple records.

Because input using can only hold one record in memory, the form must be displayed in Form View when a processing key is pressed. Table View is used for selection purposes only.

Example 1

```
input using MEMBERS into "TEMPMEMBER" .
for RESERVATIONS with ( MEMBER ID =
    TEMPMEMBER MEMBER ID) ;
list records
    RESERVATION ID ;
    TOTAL DUE ;
    DATE .
end
```

This script tells DataEase: (1) Display the MEMBERS form, (2) when the user selects a specific MEMBERS record and presses F2, store that record in memory (using the name TEMPMEMBER to reference it), (3) find all the RESERVATIONS records with the same MEMBER ID as the MEMBER ID in the TEMPMEMBER record, and (4) for each related reservation, list the RESERVATION ID, the TOTAL DUE, and the RESERVATION DATE. The output from this script might look as follows:

Reservation ID	Total Due	Reservation Date
00164	\$2780.00	08/17/92
00312	\$3150.00	04/30/93
00515	\$2942.00	12/20/93

Example 2

```
while current status not = 1 do
input using MEMBERS into "TEMPMEMBER" .
case ( current status )
value 1 : exit .
value 2 : enter a record in MEMBERS
    copy all from TEMPMEMBER .
value 3 : modify records in MEMBERS with
    ( MEMBER ID = TEMPMEMBER MEMBER ID)
    copy all from TEMPMEMBER .
value 4 : delete records in MEMBERS with
    ( MEMBER ID = TEMPMEMBER MEMBER ID) .
end
end
```

The script in Example 2 tells DataEase: (1) Display the MEMBERS form, (2) each time the operator enters a record and executes a processing action (e.g., **Edit>>Delete**), check the value of the current status variable. If current status =1 (the user chose **File>>Close** when the input using record was on the screen), stop script processing without entering a record in the MEMBERS table and close the procedure. If current status = 2 (the user chose **Edit>>Save**

As New Record when the input using record was on the screen), copy the input using record into the MEMBERS table.

If current status = 3, (the user chose **Edit>>Save** when the input using record was on the screen), modify the record in the MEMBERS table with a MEMBER ID (a Unique field) that is the same as the MEMBER ID on the input using record. If current status = 4, (the user chose **Edit>>Delete** when the input using record was on the screen), delete the record in the MEMBERS table with a MEMBER ID (a Unique field) that is the same as the MEMBER ID on the input using record, and (3) display the MEMBERS form again so the user can enter, modify, or delete another record.

Example 3

```
while current status not = 1 do
  input using MEMBERS into "TEMPMEMBER" .
  case ( current status )
  value 1 : exit .
  value 2 : if TEMPMEMBER MEMBER ID <= 20000 then
    enter a record in MEMBERS
    copy all from TEMPMEMBER .
  else
    message " Invalid ID. Re-enter the record. " .
  end
  others : message " You are not authorized to
    modify or delete records. " .
end
end
```

This script tells DataEase: (1) Display the MEMBERS form, (2) each time the user enters a record and presses a processing key, check the value of the current status variable. If current status = 1, stop script processing without entering a record in the MEMBERS table. If current status = 2, verify that a valid MEMBER ID (<= 20000) was entered in the input using record. If it is valid, copy the input using record into the MEMBERS table. If it is not valid, display an error message. If current status = 3 or 4, display a message telling the user that he/she is only authorized to enter records, and (3) display the MEMBERS form again so the user can enter another record.

Using the input using Command with Multiforms

When using the input using command on a Multiform, DataEase treats Main forms and Subforms differently. If you enter, modify, or delete a Main form record, the information is copied into the specified temporary form where it can be validated and processed by the script.

If you enter or modify a Subform record, however, the new or modified record is entered directly into the Subform in the database. This is because DataEase is unable to hold all of the entered or modified Subform records in memory (up to 5000 Subform records can be entered or modified at once) and still continue processing the script.

If the user chooses **Edit>>Save As New Record** to enter a Multiform record, you can use enter a record to enter the record into the Main form (the Subform records were automatically entered when the user pressed F2). If the script tests the Main form for validity (e.g., MEMBER ID >= 20000 in Example 3) and it fails, you must use the delete records command to remove the Subform records that were entered.

If the user chooses **Edit>>Save** to modify a Multiform record, you can use the modify records command to modify the Main form record (the Subform records are automatically modified). If the script tests the Main form for validity (e.g., MEMBER ID >= 20000 in Example 3) and it fails, it is up to the user to remember which Subform records were modified and then go back to the Subform and undo the modifications. DataEase cannot automatically undo modified Subform records.

If you delete a Multiform record using the input using command, the records are not directly deleted from the table. You must use the script to verify that a valid record was entered and then use the delete records command to explicitly delete the appropriate Main and Subform records.

Example 4

```
while current status not = 1 do
  input using RESERVATIONS into "TEMPRESV" .
  case ( current status )
  value 1 : exit .
  value 2 :
    if TEMPRESV RESERVATION ID <= 50000 then
      enter a record in RESERVATIONS
      copy all from TEMPRESV .
    else
      message " Invalid ID. Re-enter the record. " .
      delete records in RESERVATION DETAIL
      with ( RESERVATION ID =
        TEMPRESV RESERVATION ID ) .
    end
  others : message " You are not authorized to
    modify or delete records. If you have
    modified Subform records, you must
    manually delete the modifications. " .
  end
end
```

This script tells DataEase: (1) Display the RESERVATIONS form, (2) each time the user enters a record and presses a processing key, check the value of the current status variable. If current status = 1, stop script processing without entering a record in the RESERVATIONS table. If current status = 2, verify that a valid RESERVATION ID was entered in the input using record. If the RESERVATION ID is valid, copy the input using record into the RESERVATIONS table. If the RESERVATION ID is not valid, display an error message and delete the records that were automatically entered in the Subform (RESERVATION DETAIL). If current status = 3 or 4, display a message telling the user that he/she is only authorized to enter records, and (3) display the RESERVATIONS form again so the user can enter another record.

install application

Type

Control Command

Purpose

The install application command lets you copy the documents and data from another DataEase application into the current application. You can also use this option to convert and install documents and data created in another database or spreadsheet program.

The install application command functions like choosing the **Application>>Utilities>>Install**.

Syntax

```
install application "[ INSTALLATION FILENAME ]" .
```

The filename that follows the install application command is the name of the Installation Command file containing specific instructions about the documents and data to be installed. This filename must be enclosed in quotation marks and must contain eight or fewer characters with no intervening spaces. Although it is not necessary to include the filename extension, DataEase automatically looks for a file with the extension .DIW.

If you want DataEase to prompt the user for the Installation Command filename when the procedure is run (when processing reaches the install application command), omit the filename from the script. You still must include a pair of quotation marks even if you omit the filename.

LAN

On a LAN (Local Area Network), if another user is currently using any resource required by the install application command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals. When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

```
install application "A: CATALOG" .
run procedure "MAILING LIST" .
```

This script tells DataEase: (1) Install the CATALOG application from the disk in drive A: into the current application, and (2) when the new application is installed, run the MAILING LIST procedure.

installment

Type

Financial Function

Purpose

The installment function calculates the periodic installment payment required for a financial transaction given the presentvalue, futurevalue, interest rate, and number of payment periods.

Syntax

```
installment( presentvalue, futurevalue, rate, periods)
```

Returns

A numeric value (the installment payment amount).

Usage

If the installment payment increases the value of the investment (e.g., an annuity), the numeric value returned by the function is expressed as a positive amount. If the payment decreases the value of the investment (e.g., loan amortization), the numeric value returned by the function is expressed as a negative amount. The interest rate, installment payments, and time periods must be based on the same terms. For example, if payments are made monthly on a five-year loan, the number of periods is (12 * 5) or 60. The interest rate must be expressed in the same terms (the monthly interest rate is the annual rate /12). When typing fractions like 10 / 12, DataEase requires that you type a space before and after the / division symbol.

Example 1

```
installment( 15000, 0, 10 / 12, 60 )
```

Returns:-318.71

This example calculates the monthly installment payment required to pay off a \$15,000 loan in 5 years at an annual interest rate of 10%.

Example 2

```
installment( 0, 15000, 10 / 12, 60)
```

Returns:193.71

This example calculates the monthly installment amount required to accumulate \$15,000 in 5 years at an annual interest rate of 10%.

Note: DataEase financial functions are derived from the formula shown below. (The double asterisks (meaning "raise to the power") cannot be used in a script).

$$\text{futurevalue} = \text{principal} * ((1 + (\text{rate}/100)))^{**} \text{periods} + (\text{installment}/(\text{rate}/100)) * (((1 + (\text{rate}/100)))^{**} \text{periods} - 1)$$

item (Statistical Operator)

Type

Statistical Operator

Purpose

The item statistical operator tells DataEase to display the specified field value from each processed record in addition to any specified statistics. The result appears as a list item in the detail area of a procedure's output.

Syntax

```
FIELDNAME : item [other statistical operators] ;|.
```

Usage

When you create a report using Query by Model, each specified field value is automatically included in the report output. In a similar way, when you generate report output using a DQL procedure, if no other statistics are specified for a field, the field value is automatically included in the report output. However, if statistics are specified on a field, the field value is included in the output only if the field name is followed by the item operator.

There is an important difference between the item statistical operator and the item conditional statistical operator (described on the following page). The item statistical operator tells DataEase to include the list item in the report output when other statistics are specified. The item conditional statistical operator tells DataEase to evaluate a comparison and return a yes or no answer indicating if the comparison is true or not.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item min max sum .
end
```

This script tells DataEase: (1) Process all the MEMBERS records in alphabetical order by LAST NAME, (2) list each member's TOTAL DUE (item), (3) list the smallest TOTAL DUE (min), (4) list the largest TOTAL DUE (max), and (5) list the total of all the TOTAL DUE amounts combined (sum).

The output from this script might look as follows:

Last Name	Total Due
Adams	\$85.00
Albert	\$120.00
Anders	\$120.00
Andersen	\$70.00
Anderson	\$115.00
...	...
Minimum Total Due	\$35.00
Maximum Total Due	\$280.00
Sum Total Due	\$18,190.00

item (Conditional Statistical Operator)

Type

Conditional Statistical Operator

Purpose

The item conditional statistical operator returns a yes or no answer indicating if the specified comparison is true or false.

Syntax

```
COMPARISON STATEMENT :  item [other conditional statistical operators] ; | .
```

Usage

In a script, the item operator is placed after a list item that is compared to a specified value. The item operator is separated from the comparison by a colon.

There is an important difference between the item statistical operator (described on the preceding page) and the item conditional statistical operator. The item statistical operator tells DataEase to include the list item in the report output when other statistics are specified. The item conditional statistical operator tells DataEase to evaluate a comparison and return a yes or no answer indicating if the comparison is true or not.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE :  item min max sum ;
    TOTAL DUE > 100 :  item .
end
```

This script tells DataEase: (1) Process all the MEMBERS records in alphabetical order by LAST NAME, (2) list each member's TOTAL DUE (item), (3) list the smallest TOTAL DUE (min), (4) list the largest TOTAL DUE (max), (5) list the total of all the TOTAL DUE amounts combined (sum), and (6) for each member specify a yes or no answer indicating if the member's TOTAL DUE is greater than \$100 (item conditional statistical operator).

Although the fourth line in the script tells DataEase to include the TOTAL DUE value in the report output, you must repeat the fieldname as part of the comparison statement to generate the YES or NO data in the output.

The output from this script might look as follows:

Last Name	Total Due	Total Due over \$100
Adams	\$85.00	NO
Albert	\$120.00	YES
Anders	\$120.00	YES
Andersen	\$70.00	NO
Anderson	\$115.00	YES
...

Minimum Total Due	\$35.00
Maximum Total Due	\$280.00
Sum Total Due	\$18190.00

jointext

Type

Text Function

Purpose

The jointext function combines two separate text values into one.

Syntax

```
jointext( TEXT VALUE1, TEXT VALUE2)
```

Returns

A text value up to 255 characters in length.

Usage

When two text values are joined, leading spaces are maintained but trailing spaces are deleted. To join three or more fields, you can use additional jointext functions as either or both parameters.

Examples

```
jointext( "Sapph" , "ire" )
```

Returns: Sapphire

```
jointext( jointext( "Sapph" , "ire" ) , "International" )
```

Returns: Sapphire International.

In the above example, there is a space at the start of the string International.

```
jointext( PRODUCT , " for Windows" )
```

Returns: DataEase for Windows

In the above example, there is a space at the start of the string for Windows. The PRODUCT field holds the value DataEase.

julian

Type

Date Function

Purpose

The julian function converts a date value to a Julian date.

Syntax

```
julian( DATE VALUE)
```

Returns

A five-digit integer in the format YYDDD.

Usage

YY are the last two digits of the year. DDD is the position of the day in the year (with January 1st = 001; and December 31 = 365, except in leap year when it is 366).

Examples

```
julian( 07/04/99)
```

Returns: 99185

```
julian( 12/31/99)
```

Returns: 99365

lastc

Type

Text Function

Purpose

The lastc function extracts a specified number of characters from the end (right) of a text value.

Syntax

```
lastc( TEXT VALUE, n)
```

Returns

A text value n characters in length.

Usage

lastc(FIELDNAME, 1) returns the last character in the field. Trailing spaces are ignored. If there are n or fewer characters in the field, lastc returns the original text value.

Examples

```
lastc( "DataEase" , 4)
```

Returns: Ease

```
lastc( "Club ParaDEASE" , 4)
```

Returns: EASE

```
lastc( FIRST NAME, 4)
```

Returns: The last four characters in the FIRST NAME field for every record that is processed. If the FIRST NAME field contains the value Roger, the function returns oger.

lastfirst

Type

Text Function

Purpose

The lastfirst function converts a text value from the format: First Name [M.] Last Name to the format: Last Name, First Name [M.].

Where [M.] is an optional middle initial.

Syntax

```
lastfirst( TEXT VALUE)
```

Returns

A text value.

The contents of the field are rearranged so that the last word (i.e., the last name) becomes the first word in the returned value. A comma is automatically inserted after the last name.

Usage

The name of this function and the firstlast function refer to the format in which the names are stored in a single field.

Examples

```
lastfirst( Susan B. Anthony)
```

Returns: Anthony, Susan B.

```
lastfirst( T.S. Eliot)
```

Returns: Eliot, T.S.

```
lastfirst( FULL NAME)
```

Returns: The value in the FULL NAME field in the format shown above for every record that is processed. If a record contains the value Roger Birnbaum in the FULL NAME field, the function returns Birnbaum, Roger.

lastw

Type

Text Function

Purpose

The lastw function extracts a specified number of words from the end (right) of a text value.

Syntax

```
lastw( TEXT VALUE, n)
```

Returns

A text value n words in length.

Usage

lastw(FIELDNAME, n) returns the last n words in the field including intervening spaces and punctuation symbols. If there are n or fewer words in the field, lastw returns the original text value.

Examples

```
lastw( "Sapphire International Ltd." , 2)
```

Returns: International Ltd.

```
lastw( "Club ParaDEASE" , 1)
```

Returns: ParaDEASE

```
lastw( STREET, 2)
```

Returns: The last two words in the STREET field for every record that is processed. If a record contains the value "540 Avenida de los Delfines" in the STREET field, the function returns "los Delfines".

length

Type

Text Function

Purpose

The length function counts the number of character positions in a text value.

Syntax

```
length( TEXT VALUE)
```

Returns

An integer value.

Usage

The length function counts all characters, including leading and trailing spaces.

Examples

```
length( "Sapphire International Ltd." )
```

Returns: 27

```
length( "Columbus Island" )
```

Returns: 15

```
length( COMPANY NAME)
```

Returns: The number of characters in the COMPANY NAME field for every record that is processed. If the COMPANY NAME field contains the value Club ParaDEASE, Inc., the function returns 20.

list records

Type

Processing Command

Purpose

The list records command tells DataEase which items to display in the procedure output for each record processed by a script. These items are called list items. Although the most common list item is a fieldname, a list item can also be a constant, a variable, or any other expression you want to include in the report output.

Syntax

```
for TABLENAME | RELATIONSHIP
  [with selection criteria] ;
  list records
    FIELDNAME | VARIABLE NAME | LITERALS ;|.
end
list records [in FORMNAME | RELATIONSHIP
  [named "UNIQUE RELATIONSHIP NAME" ]
  [with( selection criteria) ]]
```

Usage

Each list item must be followed by a semicolon except the last, which is followed by a period. To sort, group, or generate statistics on a list item specified in a list records command, insert the appropriate operator after the list item. You cannot list or modify more than 255 fields in a single DQL Procedure.

Example 1

```
for MEMBERS ;
  list records
    STATE in groups ;
    LAST NAME in order ;
    TOTAL DUE : item min max sum .
end
```

This script tells DataEase: (1) Process all the MEMBERS records with the same value in the STATE field together as a group, (2) display the group identifier (STATE) once at the beginning of each group, (3) within each group, arrange the members in alphabetical order by LAST NAME, (4) list each member's TOTAL DUE (item), (5) list subtotals of each specified statistic for each STATE group (in groups with group-totals), (6) list the smallest TOTAL DUE (min), (7) list the largest TOTAL DUE (max), and (8) list the sum of all the TOTAL DUE amounts combined (sum). The output might look as follows:

State	Last Name	Total Due
...
KY	Bouchard	\$100.00
	Denofsky	\$70.00

	Steiner	\$115.00
	Group Total	\$285.00
LA	Orsini	\$85.00
	Rodriguez	\$100.00
	Simpson	\$85.00
	Group Total	\$270.00
...
Minimum Total Due		\$280.00
Maximum Total Due		\$35.00
Sum Total Due		\$18,190.00

Example 2

Although list records is usually used with the for command to list data, you can also use list records to list text literals, values stored in variables, and other values. For example, the script below prints **Club ParaDEASE** cruise boarding passes. It lists a text literal and values stored in a variable, joined together using the jointext function:

```
define temp  "CRUISE_TICKET_NUM"  Number .
assign temp CRUISE_TICKET_NUM := 0 .
while temp CRUISE_TICKET_NUM  < 1000 do
    temp COUNTER := temp CRUISE_TICKET_NUM + 1 .
    list records
        jointext( " Boarding Pass No.  " , CRUISE_TICKET_NUM ) .
end
```

The example above tells DataEase to: (1) Create a temporary variable named CRUISE_TICKET_NUM and to assign it an initial value of zero, (2) evaluate the condition following the word while, and if the value of CRUISE_TICKET_NUM is less than 1000, execute the actions following the word do until the end command, and (3) reevaluate the condition (CRUISE_TICKET_NUM < 1000), and if it is still true, repeat the actions until the condition becomes false.

Each time the actions are executed, the list records command tells DataEase to print the text string, "Boarding Pass No.", followed by the current value in the CRUISE_TICKET_NUM variable. Notice that the text string "Boarding Pass No." includes a single trailing space so the ticket number prints in the correct location.

The output for this procedure might look as follows:

```
Boarding Pass No. 1
Boarding Pass No. 2
Boarding Pass No. 3
Boarding Pass No. 4
```

lock

Type

Processing Command

Purpose

The lock command is functional only when using DataEase on a LAN (local area network). The lock command prevents users from viewing, modifying, adding, or deleting data in forms and records being used by a report/procedure.

The lock command overrides the default multi-user locking rules assigned by the database administrator. To override the default locking rules, the first statement of the script must be either lock all files or unlock all files. If neither of these commands appears at the start of the script, the default locking rules govern LAN functioning until a lock or unlock command appears in the script.

All lock and unlock commands are automatically terminated at the end of a script.

Syntax

```
lock all files .
lock file TABLENAME .
lock selected record [shared|exclusive] .
```

Usage

The lock all files command locks all tables referenced by the script. Users can view records in the locked tables but cannot add, delete or modify data in these tables.

The lock file command locks a specified table for the duration of the procedure or until the same table is specified in an unlock command. When an individual table or record is locked, you must also specify the type of lock (shared or exclusive). Shared lets other users view records in the locked table but prohibits those users from entering, modifying, or deleting records in that table. Exclusive prevents all access to the locked table, including viewing records.

The lock selected record command functions like lock file except it only locks the record that is currently being processed. When DataEase finishes processing a record, it is automatically unlocked. Use this command only within a for command loop (see Example 2). If you use it outside a for loop, no records are selected; therefore nothing is locked.

Example 1

```
lock file RESERVATIONS shared .
for MEMBERS with
  ( sum of RESERVATIONS TOTAL DUE  > 3000)  ;
  list records
    LAST NAME ;
    TOTAL DUE .
end
```

This script tells DataEase: (1) Lock the RESERVATIONS table so other users can view records but cannot add, delete or modify RESERVATIONS records during the processing of this script, (2) select all the MEMBERS records whose related RESERVATIONS records have a combined TOTAL DUE greater than \$3000, and (3) for each record selected, list the member's LAST NAME and TOTAL DUE.

Notice that locking the RESERVATIONS table during the processing of this report prevents other users from entering new records that might alter the sum of calculation.

Example 2

```
for MEMBERS ;  
  lock selected record shared .  
  list records  
    LAST NAME ;  
    TOTAL DUE ;  
    mean of RESERVATIONS TOTAL DUE .  
end
```

This script tells DataEase: (1) Select all the MEMBERS records, (2) lock each record as it is selected so other users can view it but cannot add, delete or modify the record while it is being processed, and (3) for each record selected, list the member's LAST NAME, TOTAL DUE, and the mean TOTAL DUE of all the related RESERVATIONS records.

In this example, all MEMBERS records can be modified except the record that is currently being processed.

lock db (lock database)

Type

Control Command

Purpose

The lock db command is only functional when using DataEase on a LAN (local area network). This command completely prohibits other users from accessing the current application.

Syntax

```
lock db .
```

Usage

The lock db command can be used to preserve the integrity of data during an extended Control procedure or to prevent a conflict when installing a new or revised application into the application.

The lock db Control command differs from the lock Processing command in several ways:

It locks the whole application rather than specified tables or individual records.

There is no shared option for the lock db command: no other user can view records or access any table in the application for any purpose.

The lock db command is not automatically terminated at the end of the procedure. Once an application is locked by the lock db command, it remains locked until an unlock db command is executed, or the user who initiated the lock db command exits from DataEase.

LAN

On a LAN (local area network), if another user is currently using any resource required by the lock db command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

```
lock db .
run procedure  "ARCHIVE OLD RESERVATIONS"  .
run procedure  "DELETE INACTIVE MEMBERS"    .
run procedure  "UPDATE ACTIVE MEMBERS"      .
unlock db .
```

This script tells DataEase: (1) Lock the current application, (2)run the ARCHIVE OLD RESERVATIONS procedure, (3) run the DELETE INACTIVE MEMBERS procedure, (4) run the UPDATE ACTIVE MEMBERS procedure, and (5) unlock the application.

log

Type

Scientific Function

Purpose

The log function computes the natural base e logarithm of a numeric value.

Syntax

```
log( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Scientific function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
log( 3)
```

Returns: 1.098612

```
log( 0.5)
```

Returns -0.693147

log10

Type

Scientific Function

Purpose

The log10 function computes the common base 10 logarithm of a numeric value.

Syntax

```
log10( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Scientific function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
log10( 3)
```

Returns: 0.477121

```
log10( 0.5)
```

Returns: -0.301031

lower

Type

Text Function

Purpose

The lower function converts each letter in a specified text value into lower-case type.

Syntax

```
lower( TEXT VALUE)
```

Returns

The specified text value with all the letters in lower case.

Examples

```
lower( "DataEase" )
```

Returns: dataease

```
lower( "Club ParaDEASE" )
```

Returns: club paradease

```
lower( LAST NAME)
```

Returns: The value in the LAST NAME field in lower-case type for every record that is processed. If a record contains the value Birnbaum in the LAST NAME field, the function returns birnbaum .

lowest of

Type

Relational Statistical Operator

Purpose

The lowest of operator finds the smallest value in the specified field in all matching records in a related table. The result can appear as a list item in the detail area of a report or as a statistic at the end of each group or at the end of the report.

Syntax

```
lowest of TABLENAME|RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ] ]
    [with ( selection criteria) ] FIELDNAME ;|.
```

Returns

A value of the same type as the specified field. If a Text field is specified, DataEase returns the lowest ASCII value. If a Choice field is specified, DataEase returns the value in the lowest numbered choice (not necessarily the lowest ASCII value).

Usage

There's an important difference between the statistical operator min and the relational statistical operator lowest of. min returns the lowest value in the specified field among the records being processed. lowest of returns the lowest value in the specified field among the records related to the records being processed.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    lowest of RESERVATIONS TOTAL DUE .
end
```

This script tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, (2) for each MEMBERS record selected, find the related records in the RESERVATIONS table (those that have the same MEMBERID), and (3) list the lowest TOTAL DUE value for each of the matching RESERVATIONS records.

The output from this script arranged in alphabetical order by LAST NAME, might look as follows:

Last Name	Lowest of Reservations Total Due
Adams	\$2740.00
Albert	\$4100.00
Anders	\$3690.00
Anderson...	\$1720.00...

If you also want to include the smallest TOTAL DUE among this group of reservations, change the fourth line of the query to read:

```
lowest of RESERVATIONS TOTAL DUE : item min.
```

max

Type

Statistical Operator

Purpose

The max operator finds the highest value in a specified field among all the records that are processed. The result appears in the statistical summary areas of the report.

Syntax

```
VALUE : max [other statistical operators] ;|.
```

Returns

A value of the same type as the specified value. If the specified value is a Text field, max returns the highest ASCII value.

Usage

max finds the highest value in the specified field among the records being processed.

There's an important difference between the statistical operator max and the relational statistical operator highestof. max finds the highest value in the specified field among the records being processed. highestof finds the highest value in the specified field among the records related to the records being processed.

Example

```
for MEMBERS with TOTAL DUE > 150 ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item max sum .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$150 in the TOTAL DUE field, (2)list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE and include the largest TOTAL DUE amount and the sum of all the TOTAL DUE amounts in the report output.

The output from this script might look as follows:

Last Name	Total Due
Archer	\$155.00
Christino	\$280.00
Forrest	\$175.00
Jones	\$175.00
Morelli	\$155.00
Perrault...	\$215.00...
Max Total Due:	\$280.00
Sum Total Due:	\$2050.00

mean

Type

Statistical Operator

Purpose

The mean operator finds the average value in a specified field among all the records that are processed. The result appears in the statistical summary areas of the report.

Syntax

```
VALUE : mean [other statistical operators] ;|.
```

Returns

A numeric value.

Usage

The mean operator calculates the average of all the values in the specified field (the sum of the values divided by the number of values processed). Blank fields are ignored in the calculation. Fields that have a value of zero are included in the calculation.

There's an important difference between the statistical operator mean and the relational statistical operator mean of. mean finds the average value in the specified field among the records being processed. mean of finds the average value in the specified field among the records related to the records being processed.

Example

```
for MEMBERS with TOTAL DUE > 150 ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item mean sum .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$150 in the TOTAL DUE field, (2) list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE and include the mean TOTAL DUE amount and the sum of all the TOTAL DUE amounts in the report output.

The output from this query might look as follows:

Last Name	Total Due
Archer	\$155.00
Christino	\$280.00
Forrest	\$175.00
Jones	\$175.00
Morelli	\$155.00
Perrault...	\$215.00...
Mean Total Due:	\$186.36
Sum Total Due:	\$2050.00

mean of

Type

Relational Statistical Operator

Purpose

The mean of operator calculates the average value in a specified field in all matching records in a related table. The result can appear as a list item in the detail area of a report or as a statistic in the summary area at the end of each group or at the end of the report.

Syntax

```
mean of TABLENAME|RELATIONSHIP
      [named "UNIQUE RELATIONSHIP NAME" ]
      [with ( selection criteria) ] FIELDNAME ;|.
```

Returns

A numeric value.

Usage

The mean of operator calculates the average of all the values in the specified field (the sum of the values divided by the number of values processed). Blank fields are ignored. Fields that have a value of zero are included in the calculation.

There's an important difference between the statistical operator mean and the relational statistical operator mean of. mean finds the average value in the specified field among the records being processed. mean of finds the average value in the specified field among the records related to the records being processed.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    mean of RESERVATIONS TOTAL DUE .
end
```

This script tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, (2)for each MEMBERS record processed, find all the related records in the RESERVATIONS table (those that have the same MEMBERID), and (3) list the mean TOTAL DUE for each member from the set of related RESERVATIONS records.

The output from this script might look as follows:

Last Name	Mean of Reservations Total Due
Adams	\$2780.00
Albert	\$4430.00
Anders	\$4055.00
Anderson...	\$1910.00...

If you also want to include the mean TOTAL DUE among this group of reservations, change the fourth line of the script to read:

```
mean of RESERVATIONS TOTAL DUE : item mean .
```


message

Type

Procedural Command

Purpose

The message command tells DataEase to display a specified message. The message text can be 255 characters in length. You can tell DataEase to display the message on the status bar or in a standard Windows message box. Each message can be programmed to accept user input. DataEase provides five optional parameters (described below) that let you customize the message window.

Syntax

```
message "MESSAGE TEXT" [window | pause] [ "MESSAGE TITLE TEXT" ];
[Icon]; [Buttons]; [Beep] .
```

Usage

Message text can contain a field name, current system variable, a relational operator, a function, and/or a text constant enclosed in quotation marks. To divide the message into multiple lines, insert a vertical bar character (|) to indicate a new line (see Example 1). If you do not specify the window or pause parameter, DataEase displays the message on the status bar by default.

When you choose window, DataEase displays the message text in a standard Windows message box. DataEase automatically pauses and the user is required to press any key to acknowledge the message to resume processing. . Pause also displays a window, and is included for compatibility with previous versions of DataEase.

If you include the message title text parameter, DataEase displays the entered text on the message box title bar. If you exclude this parameter, DataEase displays DataEase Message as a default title.

To include the icon parameter, specify a number which represents one of the Windows icons shown in the table below.

Message Command Icon Parameters

Icon	Numeric Value	Icon Name
	1	Information Icon
	2	Exclamation Icon
	3	Stop Icon
	4	Question Icon

If you exclude the icon parameter, DataEase automatically displays the Windows information icon by default.

To include the buttons parameter, specify a number which represents one of the button combinations shown in the table below.

Message Command Button Parameters

To display this button combination...	...enter this number as the buttons parameter.
OK, CANCEL	1
RETRY, CANCEL	2
ABORT, RETRY, IGNORE	3
YES, NO	4
YES, NO, CANCEL	5

If you exclude the buttons parameter, DataEase displays the OK, CANCEL button combination by default.

When the user clicks a button displayed in the message box, DataEase returns a value to the current status system-defined variable. The current status values associated with each of the buttons are displayed in the table below.

Message Command Current Status Values

When you

click this button...	...DataEase Returns this value to the current status variable.
OK	1
CANCEL	2
ABORT	3
RETRY	4
IGNORE	5
YES	6
NO	7

You can use the value stored in the current status variable to perform conditional processing of the remainder of your script as shown in Example 2.

To include an audible warning sound, specify a number which represents one of the standard Windows beep sounds, as shown in the table on the following page.

Message Command Beep Parameters

To play the sound associated with this Windows icon... ...enter this number.

System Default	1
Asterisk	2
Information	3
Exclamation	4
Question	5
Hand	6
Stop	7
OK	8

If you exclude the beep parameter, DataEase does not provide a default.

When you choose pause, the message is displayed on the status bar. If you include any of the optional parameters with the pause option, DataEase displays an error when you choose **Script>>Check DQL**.

Example 1

```
message  "Sorry. A high security level is
         required|to run this procedure"  window .
```

This message is displayed in a Windows message box. There are line breaks after the words Sorry and required. By default, DataEase includes the title DataEase Message, the Windows information icon, and the OK, CANCEL button combination.

Example 2

```

for MEMBERS ;
  if EXPIRATION DATE < currentdate then
    message jointext ( " Delete " , jointext
      ( LAST NAME , " record? " ) ) window " Delete
      Inactive Member Records " ; 4; 5; 4.
    if current status = 6 then-- Yes button
      delete records.
      message " Record Deleted! " pause .
    else
      if current status = 7 then-- No button
        message " Finding next record. " pause .
      else
        if current status = 2 then-- Cancel button
          message " Cancelling procedure. " pause .
          exit .
        end
      end
    end
  end
end
end
end

```

This example tells DataEase: (1) Process all the records in the MEMBERS table. (2) Check each record to see if the membership has expired as indicated by the EXPIRATION DATE. If the membership has lapsed, display the message: "Delete LAST NAME record?" in a message box that includes the message title "Delete Inactive Member Records", the Question icon, the YES, NO, CANCEL button combination, and plays the exclamation sound. (3) If The user clicks YES, delete the current record and display the message "Record Deleted!" on the status bar, (4) If the user clicks NO, abandon the current record and display the message "Finding next record." on the status bar. (5) If The user clicks CANCEL, display the message "Cancelling procedure." on the status bar and exit the script.

midc

Type

Text Function

Purpose

The midc function extracts a specified number of characters from the middle of a text value.

Syntax

```
midc( TEXT VALUE, m, n)
```

...where;

m is the starting position in the text value and

n is the number of characters to extract.

Returns

A text value n characters in length.

Usage

midc(FIELDNAME, m, n) returns n characters beginning at character position m (inclusive). Spaces and punctuation symbols are included.

Examples

```
midc( "DataEase" , 4, 3)
```

Returns: aEa

```
midc( "Club ParaDEASE" , 6, 3)
```

Returns: Par

```
midc( LAST NAME, 4, 2)
```

Returns: Two characters beginning at the fourth character position in the LAST NAME field for every record that is processed. If a record contains the value Birnbaum in the LAST NAME field, the function returns nb.

midw

Type

Text Function

Purpose

The midw function extracts a specified number of words from the middle of a text value.

Syntax

```
midw( TEXT VALUE, m, n)
```

...where

m is the starting position in the text value and

n is the number of characters to extract.

Returns

A text value n words in length.

Usage

midw(FIELDNAME, m, n) returns n words beginning at word number position m (inclusive). Spaces and punctuation symbols are included.

Examples

```
midw( "Sapphire International Ltd." , 2, 1)
```

Returns: International

```
midw( "Club ParaDEASE Catalog" , 2, 2)
```

Returns: ParaDEASE Catalog

```
midw( STREET, 2, 3)
```

Returns: Three words beginning at the second word position in the STREET field for every record that is processed. If a record contains the value "540 Avenida de los Delfines" in the STREET field, the function returns "Avenida de los".

min

Type

Statistical Operator

Purpose

The min operator finds the smallest value in a specified field among all the records that are processed. The result appears in the statistical summary areas of the report.

Syntax

```
VALUE : min [other statistical operators] ;|.
```

Returns

A value of the same type as the specified value. If the specified value is a Text field, min returns the lowest ASCII value.

Usage

There's an important difference between the statistical operator min and the relational statistical operator lowest of. min returns the lowest value in the specified field among the records being processed. lowest of returns the lowest value in the specified field among the records related to the records being processed.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item min .
end
```

This script tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, and (2) list each member's TOTAL DUE and include the smallest TOTAL DUE amount in the report output.

The output from this script might look as follows:

Last Name	Total Due
Archer	\$120.00
Christino	\$210.00
Forrest	\$75.00
Jones	\$95.00
Morelli	\$155.00
Perrault	\$185.00
Min Total Due:	\$75.00

minutes

Type

Time Function

Purpose

The minutes function extracts the minutes from a time value expressed in a 24-hour format.

Syntax

```
minutes( TIME VALUE)
```

Returns

An integer value from 00 to 59.

Examples

```
minutes( 12: 00: 00)
```

Returns: 00

```
minutes( 23: 59: 00)
```

Returns: 59

mod (modulus)

Type

Scientific Function

Purpose

The mod (modulus) function returns the remainder when one numeric value is divided by a second numeric value. If both values are integers, an integer value is returned. Otherwise, a decimal value is returned.

Syntax

```
mod( NUMERIC VALUE 1, NUMERIC VALUE 2)
```

Returns

A numeric value.

Usage

The numeric value in a Scientific function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
mod( 8, 3)
```

Returns: 2

```
mod( 12.6, 1.6)
```

Returns: 1.4

Modify Records

Type

Processing Command

Purpose

The modify records command modifies records in a specified table.

Syntax

```
modify records [in TABLENAME|RELATIONSHIP] [named "UNIQUE
RELATIONSHIP NAME" ] [with ( selection criteria) ] FIELDNAME :=
MODIFIED VALUE ;|.
```

Usage

The modify records command must specify the field values to be changed in the Target table. The Target table may be the Primary table specified at the start of a for command, or a related table specified within a for loop.

When you modify records in the Primary table, the keyword in is omitted and no punctuation is used after the modify records command (see Example 1). If no selection criteria are specified, all the records in the Primary table are modified.

When you modify records in a table other than the Primary table, the keyword in precedes the table name, and the command ends with the appropriate punctuation mark (a semicolon or a period - see Example 2). If no selection criteria are specified, all the records in the specified table are modified.

Source data used to modify records may come from the Primary table, a Data-entry form, system variable, or calculation. Alternatively, you can use the copy all from command to specify a Source table from which to copy field data.

As data is entered in the Target table, automatic error checking is performed. Errors are logged to an error file.

You cannot list or modify more than 255 fields in a single DQL Procedure.

Example 1

```
for MEMBERS with ( any RESERVATIONS TOTAL DUE > 3000) ;
  modify records
    TOTAL DUE := ( TOTAL DUE - data-entry DISCOUNT) .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have any related RESERVATIONS record with a TOTAL DUE greater than \$3000, and (2) modify each MEMBERS record by subtracting the DISCOUNT amount entered in the Data-entry form from the value in the TOTAL DUE field in the MEMBERS table.

Example 2

```
for MEMBERS with STATE = "NY" ;
  modify records in RESERVATIONS
    TOTAL DUE := TOTAL DUE + NY SALES TAX .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have NY in the STATE field, and (2) for each MEMBERS record that is processed, find all the related records in the RESERVATIONS table. Modify each related RESERVATIONS record by adding the NY SALES TAX amount to the value in the TOTAL DUE field.

month

Type

Date Function

Purpose

The month function extracts the month (1- 12) from a date value.

Syntax

```
month( DATE VALUE)
```

Returns

An integer value between 1 and 12

The date format selected using the International option in the Windows Control Panel changes the date sequence but does not affect which value is returned by a Date function.

Examples

```
month( 12/31/99)
```

Returns: 12 (North American format)

```
month( 31/12/99)
```

Returns: 12 (European format)

```
month( 99/12/31)
```

Returns: 12 (Metric format)

named

Type

Relational Operator

Purpose

The named operator is used to give a unique name to each relationship based on different selection criteria so DataEase can distinguish multiple relationships between the same two tables.

Syntax

```
RELATIONSHIP named "UNIQUE RELATIONSHIP NAME" [with ( selection
criteria) ]
```

Usage

During a script, whenever you add or change the selection criteria of a relationship (predefined or ad hoc), you create a new ad hoc relationship. If two tables are related by more than one relationship, each of these relationships must be assigned a unique name, either on the Relationships form or by using the named operator in a script, so DataEase can distinguish between them.

A unique relationship name is an arbitrary name. The name you assign to the relationship is analogous to the custom relationship name used on the Relationships form. It can be any name you like as long as no other relationship in the script (or predefined relationship) has the same name. The name is placed in quotes at the time it's first defined; no quotation marks are used if the name appears in subsequent statements.

When you use the named operator to distinguish a relationship and specify its selection criteria, the new criteria are added to any previously defined criteria, including the criteria on which a predefined relationship was based. Thus, if you add new criteria to a relationship and use the named operator to distinguish it, you don't need to restate all of the criteria that defines the original relationship. Once a relationship is assigned a unique name, you can use the new relationship name alone to reference it without restating the criteria.

Example 1

```
for MEMBERS ;
    modify records in RESERVATIONS named "ACTIVE"
        with ( DATE > 01/01/1999 )
        TOTAL DUE := TOTAL DUE - data-entry DISCOUNT .
    delete records in RESERVATIONS named " OUTDATED "
        with ( DATE < 01/01/1998 ) .
end
```

This script tells DataEase: (1) Process all the MEMBERS records, (2) for each record processed, find all the related RESERVATIONS records, (3) divide the related RESERVATIONS records into two groups: one group named ACTIVE (dated after January 1st, 1999), and a second group named OUTDATED (dated before January 1st, 1998), (4) modify the records in the ACTIVE relationship by subtracting the DISCOUNT value entered in the Data-entry form from the RESERVATIONS TOTAL DUE, and (5) delete the records in the OUTDATED relationship.

There is a predefined relationship between MEMBERS and RESERVATIONS. During this script, we are creating two different ad hoc relationships between MEMBERS and RESERVATIONS by adding selection criteria based on the value in the DATE field. The named operator is used to assign a unique relationship name to each group of records so DataEase can distinguish between these two relationships.

Since DataEase continues to apply the match field criteria specified in the predefined relationship between the MEMBERS and RESERVATIONS (that is, MEMBERS MEMBER ID = RESERVATIONS MEMBER ID), you don't need to respecify this match criteria when each new ad hoc relationship is named.

Subsequently, as long as the criteria remains unchanged, you can use the relationship name alone without restating the relationship criteria. For example, if after modifying the records in the ACTIVE relationship you want to list the RESERVATION ID and TOTAL DUE of each ACTIVE reservation record, the script would look like this:

Example 2

```
for MEMBERS ;
  modify records in RESERVATIONS named  "ACTIVE"
    with ( DATE  > 01/01/99 )
  TOTAL DUE := TOTAL DUE - data-entry DISCOUNT  .
  list records in ACTIVE
    RESERVATION ID in order ;
    TOTAL DUE  .
end
```

Nested Actions

Type

Concept

Purpose

A Nested Action is an action that is invoked by another action.

There are two types of nested actions: (1) A nested loop action performs the same series of actions on each record that is processed (see Example 1), and (2) a nested conditional action is most often used to select subgroups of records and perform different actions on each group (see Example 2).

Example 1

```
for MEMBERS ;
  for RESERVATIONS
    list records
      MEMBERS LASTNAME in groups ;
      RESERVATION ID ;
      TOTAL DUE .
    end
  end
end
```

This script tells DataEase to perform the same series of actions on each record selected by the for command. In this example, the second for command, which retrieves the RESERVATIONS records related to the MEMBERS record that is currently being processed, is nested within the first for command.

Although DataEase does not require it, it is easier to follow the logic if you indent nested actions as shown in these examples.

Example 2

```
for RESERVATIONS ;
  if TOTAL DUE > 2000 then
    modify records in MEMBERS
      STATUS := PREFERRED .
    if DATE between 01/01/99 to current date then
      enter a record in CATALOG MEMBERS
      copy all from MEMBERS .
    end
  end
end
end
```

This script tells DataEase to select two groups of RESERVATIONS records: (1) All the records with an TOTAL DUE greater than \$2000 (the related MEMBERS records get modified), and (2) the records with an TOTAL DUE greater than \$2000 and a DATE between the start of 1994 and today's date (the related MEMBERS records get modified and copied into the CATALOG MEMBERS table).

In this example, the second if command, which selects a subgroup of the records selected by the first if command, is nested within the first if command.

not

Type

Comparison Operator

Purpose

The not operator reverses the meaning of any comparison operator it precedes (e.g., STATE not = (equals) " NY" means any state except New York).

Syntax

```
VALUE 1 not comparison operator VALUE 2
```

Usage

The not operator's effect can often be accomplished by stating the comparison in a different way. For example, < (less than) can be substituted for not >= (not equal or greater than). Whenever possible, comparisons should be stated as a positive expression.

Examples

```
TOTAL DUE not > 500 ;
```

means the value in the TOTAL DUE field is less than or equal to 500.

```
STATE not = "NY" and STATE not = "NJ" ;
```

means any state except New York or New Jersey.

Caution

When you combine two or more comparisons using the not operator, you must use and in the comparison. If you use or in conjunction with not, as shown below, every record in the file is processed.

```
STATE not = "NY" or STATE not = "NJ" ;
```

In this example, records that do not meet one condition will meet the other condition.

Operators

Type

Concept

Purpose

Operators are used to manipulate variables and to tell DataEase how to carry out certain Processing and Control Procedures (such as in what order to process records, what statistics to generate, etc.). DQL includes seven types of operators:

- **Comparison Operators** are used to compare two values and process a record depending on the result of the comparison. Examples include <= (less than or equal to), < (less than), = (equal to), > (greater than), >= (greater than or equal to), between, and not.
- **Grouping/Sorting Operators** tell DataEase how to group and sort records when you process data using the list records command. Examples include in groups, in order, and in reverse.
- **Relational Operators** all and any are used to list records related to the record currently being processed. all tells DataEase to list all records related to the current record; any tells DataEase to list only the first related record in the database.
- **Statistical Operators** summarize the values of a specified field for all records processed. Examples include sum, mean, max, min, and std.dev.
- **Conditional Statistical Operators** summarize statistical information about a specific condition that occurs in a set of records. Examples include count, percent, and item.
- **Relational Statistical Operators** summarize information about fields in a set of related records. Examples include count of, highest of, lowest of, mean of, and sum of.
- **General Operators** include the arithmetic operators *(asterisk) ,/ ,+ (addition) , and - (subtraction) , the assignment operator (:=) used to assign a value to a field or variable, and the logical operators (and and or) that let you select records based on multiple criteria, and execute Procedural Commands based on more than one condition.

There is a separate entry for each individual operator in this language reference.

or

Type

Logical Operator

Purpose

The or operator combines two sets of selection criteria or comparison statements.

Syntax

```
SELECTION CRITERIA 1 or SELECTION CRITERIA 2
```

Returns

The values in records that satisfy either of the selection criteria statements.

Usage

The or operator requires that a record meet either of the specified criteria to be processed (as opposed to the and operator which requires that a record meet all the specified selection criteria to be processed).

When selection criteria are combined with both the and and or operators in one statement, the criteria must be enclosed in parentheses to clarify the meaning.

Examples

```
for MEMBERS with STATE = "NY" or TOTAL DUE > 200 ;
```

This statement tells DataEase: Process only those MEMBERS records that contain NY in the STATE field or have a value greater than \$200 in the TOTAL DUE field. A record is processed if it satisfies either of these criteria.

```
for MEMBERS with ( STATE = "NY" or STATE = "NJ" )
and TOTAL DUE > 200 .
```

This statement tells DataEase: Process only those MEMBERS records that contain either NY or NJ in the STATE field and a value greater than \$200 in the TOTAL DUE field.

In this case, only records that satisfy both sets of criteria are processed.

Caution

When you combine two or more comparisons using the not operator, you must use and in the comparison. If you use or in conjunction with not, as shown below, every record in the file is processed.

```
STATE not = "NY" or STATE not = "NJ" ;
```

In this example, records that do not meet one condition will meet the other condition.

others

Type

Command component

Purpose

The others keyword is an optional component of the case command syntax. When processing a case command, DataEase compares the expression that follows case to each of the statements specified by the keyword value. If none of the specified value comparisons are true, DataEase executes the actions specified after the keyword others. As soon as the actions following any value or others statement are executed, processing passes to the first action following the end command for the case statement.

Syntax

```
case ( EXPRESSION)
    value COMPARISON 1 :
        ACTION SERIES 1 .
    [value COMPARISON 2 :
        ACTION SERIES 2 .
    .
    .
    value COMPARISON N :
        ACTION SERIES N .]
    [others :
        DEFAULT ACTION SERIES .]
end
```

Usage

The case command requires a case expression, one comparison value, and an end command. Subsequent value statements, actions, and the others keyword are optional. If others is used, it must follow all the specified comparison values.

Example

```
case ( current user name)
    value "FRANK" :
        call menu "MAIN MENU" .
    value "CAROL" :
        run procedure "PRINT RESERVATIONS" .
    others :
        record entry "MEMBERS" .
end
```

This script tells DataEase: (1) If the current user is Frank, display the ADMINISTRATION menu, (2) if the current user is Carol, run the PRINT RESERVATIONS procedure, and (3) if the current user is anyone other than Frank or Carol, display the MEMBERS Record Entry form.

output

Type

Procedural Command

Purpose

The output command is included solely for compatibility with previous versions of Dataease that were unable to have multiple list records statements. The entry is included here for completeness.

The output command tells DataEase which items to display in the procedure output for each record processed by a scriptScript. The items you output can include a fieldname, constant value, a variable or any other expression you want to include in the layout of your report.

Syntax

```
outputFIELDNAME | "TEXT" ;|. end
```

Usage

The output command lets you specify fields and/or text to be included in the report. Text values must be enclosed in quotation marks. Each item listed in the output command must be followed by a semicolon except the last, which is followed by a period. (.)

A script can contain any number of output commands. For each output command, DataEase automatically creates a separate subform object in the body of the layout.

Example

```
for EMPLOYEE ;
  list records
    FIRST NAME ;
    LAST NAME
    STATUS .
  if STATUS = exempt then
    output :
      401K ;
      HEALTH PLAN ;
      STOCK OPTIONS .
    end
  else
    output :
      ANNUAL BONUS ;
      OVERTIME RATE .
    end
  end
end
```

This example tells DataEase: (1) Process every EMPLOYEE record, listing the first and last name and employment status, (2) if the employee's status is exempt, output the values in the 401K, HEALTH PLAN, and STOCK OPTIONS fields, and (3) if the employee's status is anything other than exempt (in this case hourly), output the values in the ANNUAL BONUS and OVERTIME RATE fields.

percent

Type

Conditional Statistical Operator

Purpose

The percent operator calculates the percentage of records that satisfy a condition specified by a comparison of two values.

Syntax

CONDITION : percent [other conditional statistical operators]

Returns

A numeric value.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    TOTAL DUE ;
    TOTAL DUE > 100 : item percent .
end
```

This query tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, (2) list each member's TOTAL DUE, (3) for each member, display a YES or NO answer indicating if the TOTAL DUE is over \$100, and (4) display the percentage of records that have a TOTAL DUE value greater than \$100 in the report output.

The output from this query might look as follows:

Last Name	Total Due	Total Due greater than \$100
Adams	\$85.00	NO
Albert	\$120.00	YES
Anders	\$120.00	YES
Andersen	\$70.00	NO
Anderson	\$115.00	YES
Archer	\$155.00	YES
Baldwin	\$100.00	NO
Beauchamp...	\$35.00...	NO...

Total Due greater than \$100 = 22%

periods

Type

Financial Function

Purpose

The periods function calculates how long a financial transaction requires to reach a future target value given the presentvalue, futurevalue, installment payment, and interest rate.

Syntax

```
periods( presentvalue, futurevalue, installment, rate)
```

Returns

A numeric value (the number of periods needed to reach the target futurevalue of the investment).

Usage

If the installment payment increases the value of the investment (e.g., an annuity), the installment is expressed as a positive amount. If the payment decreases the value of the investment (e.g., loan amortization), the installment is expressed as a negative amount.

The interest rate, installment payments, and time periods must be based on the same terms. To determine periods as a number of months, the rate is expressed as the annual rate / 12. To determine periods as a number of weeks, the rate is expressed as 7*annual rate / 365.

Examples

```
ceil( periods( 10000, 0, -150, 8 / 12) )
```

Returns: 89

This example calculates how many months it takes to repay a loan of \$10,000 at 8% annual interest with monthly installment payments of \$150. The ceil function rounds up the result to the next integer.

```
ceil( periods( 0, 10000, 250, 10 / 12) )
```

Returns: 35

This example calculates how many months it takes to accumulate \$10,000 in savings, if you begin with no money in the account, make monthly deposits of \$250, and earn 10% annual interest.

Note: DataEase financial functions are derived from the formula shown below. (The double asterisks (meaning "raise to the power") cannot be used in a script).

$$\text{futurevalue} = \text{principal} * ((1 + (\text{rate}/100)) ** \text{periods}) + (\text{installment}/(\text{rate}/100)) * (((1 + (\text{rate}/100)) ** \text{periods}) - 1)$$

When typing fractions like 8 / 12, DataEase requires that you type a space before and after the / division symbol.

power

Type

Scientific Function

Purpose

The power function raises a numeric value to a specified power.

Syntax

```
power( BASE, EXPONENT)
```

Returns

A numeric value equal to the base raised to the exponent.

Usage

DataEase does not support ** as a symbol for exponentiation. Use the power function instead.

Examples

```
power( 12, 2)
```

Returns: 144

```
power( 6, 3)
```

Returns: 216

```
power( 8, 1 / 3)
```

Returns: 2

presentvalue

Type

Financial Function

Purpose

The presentvalue function calculates the required starting amount for a financial transaction given the futurevalue, installment payment, interest rate, and number of payment periods.

Syntax

```
presentvalue( futurevalue, installment, rate, periods)
```

Returns

A numeric value (the initial value before any installment payments are made).

Usage

If the installment payment increases the value of the investment (e.g., an annuity), the installment is expressed as a positive amount. If the payment decreases the value of the investment (e.g., loan amortization), the installment is expressed as a negative amount.

The interest rate, installment payments, and time periods must be based on the same terms. For example, if the periods are expressed in months, the monthly interest rate is expressed as the annual rate/12.

Example 1

```
presentvalue( 0, -150, 7.9 / 12, 48)
```

Returns: 6156.12

This example calculates the auto loan amount you can afford to borrow at an interest rate of 7.9% if you want to repay the loan in 48 monthly payments of \$150.

Example 2

```
presentvalue( 125000, 150, 10 / 12, 240)
```

Returns: 1514.00

This example calculates the required starting balance for a savings plan designed to accrue a \$125,000 balance in 20 years based on monthly payments of \$150 and an annual interest rate of 10%.

Note: DataEase financial functions are derived from the formula shown below. (The double asterisks (meaning "raise to the power") cannot be used in a script).

```
futurevalue = principal * ( ( 1 + ( rate/100 ) ) * * periods ) +
( installment/( rate/100 ) ) * ( ( ( 1 + ( rate/100 ) ) * * periods )
- 1)
```

When typing fractions like 7.9 / 12, DataEase requires that you type a space before and after the / division symbol.

Primary Table

Type

Concept

Purpose

The first table specified in a script is called the Primary table for the script. The Primary table should be the table that holds the key data you want to view or manipulate.

Usage

One table can be related to another table by either a predefined or an ad hoc relationship. A table related to the Primary table is called a Secondary table. A table related to a Secondary table is called a Tertiary table, etc. There can be any number of such relationship levels.

When processing the records in a table, the data in another table can be accessed if a relationship exists between these tables. A relational operator (all, any, count of, highest of, lowest of, mean of, or sum of) is used to access the data in the related table.

Procedural Commands

Type

Concept

Purpose

The DQL Procedural commands are used to organize and control the flow of actions in a procedure.

Usage

Procedural commands can be used in both Processing procedures and Control procedures.

Example

```
case ( spellweekday ( weekday( current date) )
  value "MONDAY" :
    run procedure "LAST WEEK SUMMARY" .
  value "FRIDAY" :
    run procedure "PRINT RESERVATIONS" .
  others :
    call menu "MAIN MENU" .
end
```

This script tells DataEase: (1) If the current date is a Monday, run the LAST WEEK SUMMARY procedure, (2) if the current date is a Friday, run the PRINT RESERVATIONS procedure, and (3) if the current date is any other day of the week, display the MAIN MENU.

Processing Procedure

Type

Concept

Purpose

A DQL Processing procedure can enter, delete, and modify records, as well as create screen, disk, or printed output based on the script instructions, the data stored in database tables, and additional data input during the execution of the procedure.

Usage

The Processing commands (shown at right) are primarily used to create a Processing procedure. These commands can also be used in a Control procedure.

Example

```
for MEMBERS ;  
  list records  
    LAST NAME in order ;  
    TOTAL DUE .  
end
```

This script tells DataEase: (1) Process all the MEMBERS records, and (2) for each record processed, list the LAST NAME and the TOTAL DUE values and arrange the output in alphabetical order by LAST NAME.

Note: When accessing data in an SQL database, DataEase treats a Processing procedure as a single transaction, unless you use the DQL begin transaction and commit commands to divide the procedure into multiple transactions.

proper

Type

Text Function

Purpose

The proper function converts the first letter in each word in a text value to uppercase.

Syntax

```
proper( TEXT VALUE)
```

Returns

The specified text value with the first letter in each word capitalized.

Usage

Any letter preceded by a space, hyphen, ampersand, or a period is capitalized by the proper function as is the first letter in a parenthesized value. Other punctuation, including the apostrophe, does not act as a delimiter for the proper function.

If you enter a capital letter in a position that is not the first letter in the word or specified value, it remains in uppercase when the value is processed by the proper function.

To capitalize the first character of each word and force all other characters to lower-case, use the lower function in conjunction with the proper function as follows:

```
proper ( lower ( LAST NAME) )
```

The proper function is commonly entered as the Derivation Formula for Text fields during Form Definition. For example, if the derivation for the LAST NAME field is:

```
proper(LAST NAME)
```

DataEase follows the rules listed above to capitalize the values entered in the LAST NAME field.

Examples

```
proper( "playa blanca" )
```

Returns: Playa Blanca

```
proper( "sapphire international" )
```

Returns: Sapphire International

```
proper( FULL NAME)
```

Returns: The value in the FULL NAME field in the proper format for each record processed. For example, if a record contains the value roger birnbaum, the function returns Roger Birnbaum.

query selection

Type

Processing Command

Purpose

The query selection command is functional only on a LAN (local area network). This command is used to override DataEase's default locking rules. It should only be used by a technical expert who is very familiar both with DataEase and the LAN environment in use. If you do not have expertise in LANs, you can rely on the default locking rules to lock your data appropriately.

Before DataEase begins processing records for a script, it must select the records to be processed. The query selection command tells DataEase how to set locks prior to the start of processing if records are being selected or sorted on an indexed field. If records are not being selected or sorted on an indexed field, the default locking rules apply.

Syntax

```
query selection lock files|records|nothing .
```

The three syntax options that are used in conjunction with the query selection command are explained below.

The **query selection lock files** command overrides the default Record Selection locking rules and locks the table(s) from which records are being selected. Other users cannot add, modify, or delete records in the table. They can view records in the table if the selected records will only be listed by the procedure (not modified or deleted). The table remains locked until the end of the procedure.

The **query selection lock records** command overrides the default Record Selection locking rules and locks any records that have been selected for processing. Other users cannot modify or delete these records. They can view the records if the selected records will only be listed by the procedure (not modified or deleted). Other users can view, modify, or delete records in the same table(s) that have not been selected. Once processing begins, the default locking rule applies.

The **query selection lock nothing** command overrides the default Record Selection locking rules and unlocks the table(s) from which records are being selected. Other users can view, modify, or delete records that have been selected for processing. Once processing begins, the default locking rule applies.

The lock nothing command can be useful in situations where you need to keep data available to many users even while running a procedure that uses that data. However, with this command in effect, it is possible for a user to change a record that has already been selected for processing.

Caution

The query selection lock records command should be used with great care. It should only be used for procedures that select very few records from very few tables. If this command is used for many records, most LANs cannot support the number of locks required. If this command is used incorrectly, the integrity of your data can be compromised.

Note: If a lock all files command occurs as the first command in a script, it overrides the default locking rules and any locking commands specified later in the script. A lock all files command locks all files used by the procedure until an unlock command is encountered or the procedure ends, regardless of subsequent commands. See the lock command earlier in this chapter for more information.

Usage

The query selection command overrides the LAN default Record Selection locking rules assigned by the database administrator. A query selection command is automatically terminated at the end of a script.

When using lock records on a DOS network, we recommend that you process a small number of records. Otherwise, your network may run out of locks and cause your workstation (or the network server) to lock up and require rebooting.

Example

```
query selection lock nothing .
for MEMBERS with BONUS POINTS > 500 ;
    modify records
        TOTAL DUE := ( TOTAL DUE * data-entry DISCOUNT ) .
end
```

This script tells DataEase: (1) Override the default Record Selection locking rule so other users can view, add, delete, or modify the data in the MEMBERS table while records are selected to be processed by this procedure, (2) select all the MEMBERS records whose BONUS POINTS are greater than 500, and (3) modify each selected record by multiplying the TOTAL DUE amount by the value in the DISCOUNT field on the Data-entry form.

In this example many records may be selected to be processed. However, users can still access the MEMBERS table while the procedure is processing. It is unlikely that any member's BONUS POINTS will fall below 500 while the script is being processed.

random

Type

Math Function

Purpose

The random function returns a random decimal value between 0 and 1 (inclusive). No value is required in the function's argument.

Syntax

```
random( )
```

Returns

A number between 0 and 1 (inclusive).

The result of the random function is really a pseudo-random number calculated by a formula, not a true random number.

Usage

The following steps are used to generate a random integer using the random function:

- Determine the range (number of possibilities) in which the random number must fall. For example, if the desired range is between 10 and 100, the range is 90.
- Determine starting value (smallest integer) of the range. In the example 10-100, the starting value is 10.
- Use the following formula to generate a random integer:

$$(\text{random}() * \text{RANGE}) + \text{STARTING VALUE}$$

Examples

```
floor( random( ) * 9)
```

Returns: A random integer between 0 and 9.

```
floor( random( ) * 90) + 10)
```

Returns: A random integer between 10 and 100.

In these examples, floor rounds down the result to the nearest integer.

rate

Type

Financial Function

Purpose

The rate function calculates the interest rate of a financial transaction given the presentvalue, futurevalue, installment payment, and the number of payment periods.

Syntax

```
rate( presentvalue, futurevalue, installment, periods)
```

Returns

A numeric value (the interest rate).

Usage

If the installment payment increases the value of the investment (e.g., an annuity), the installment is expressed as a positive amount. If the payment decreases the value of the investment (e.g., loan amortization), the installment is expressed as a negative amount.

The interest rate, installment payments, and time periods must be based on the same terms. If payments are made monthly, periods is computed as the number of years * (multiplied by) 12 and rate is computed as 1/12 of the annual rate.

Example 1

```
12 * rate( 12000, 0, -289, 48)
```

Returns: 7.3

This example calculates the annual interest rate on a \$12,000 car loan to be repaid in 48 monthly payments of \$289.

Example 2

```
12 * rate( 0, 20000, 250, 60)
```

Returns: 10.2

This example calculates the annual interest rate required to accumulate \$20,000 in savings, if you begin with no money in the account and make monthly payments of \$250 over a period of 5 years.

Note: DataEase financial functions are derived from the formula shown below. (The double asterisks (meaning "raise to the power") cannot be used in a script).

$$\text{futurevalue} = \text{principal} * ((1 + (\text{rate}/100)) ** \text{periods}) + (\text{installment}/(\text{rate}/100)) * (((1 + (\text{rate}/100)) ** \text{periods}) - 1)$$

When typing fractions like 10 / 12, DataEase requires that you type a space before and after the / division symbol.

record entry

Type

Control Command

Purpose

The record entry command tells DataEase to display a specified Record Entry form.

Syntax

```
record entry "FORMNAME" .
```

Usage

The record entry command opens a specified form at any point in a Control procedure and allows records to be entered or modified in that form. You can enter as many records as you like, using either Form View or Table View.

Unlike the input using command, the record entry command provides all the functions available in User View, including the ability to add, modify, and delete records.

When you finish entering records and close the form, the Control procedure resumes with the action following the record entry command.

Example

```
record entry "MEMBERS" .  
run procedure "PRINT RESERVATIONS" .  
application status records .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter view, enter, modify, or delete member records, (2) when the user finishes entering records, run the PRINT RESERVATIONS procedure, and (3) after running the procedure, display the status of the records in the current application.

Note: This command can also be used to open any document (form, menu, report or procedure). It acts like the button action “document open”.

Relational Statistical Operators

Type

Operator

Purpose

Relational Statistical Operators are used to generate statistical information on the values in the specified field in a set of records in a related table (i.e., a set of records that are related to the record currently being processed, such as all the reservations for an individual member).

Syntax

```
relational statistical operator TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria ) paren ] FIELDNAME ;
```

Usage

- To generate statistics on a field in a related table:
- Insert the Relational Statistical Operator followed by the name of the related table or the name of the relationship.
- Assign a unique name to the relationship if necessary using the named operator.
- Insert any desired selection criteria in parentheses.
- Insert the field name followed by a semicolon or a period, depending on the location of the statement in the script.

The Relational Statistical Operators are: count of, highest of, lowest of, mean of, and sum of.

The count of operator counts the number of related records, not field values. The field name is omitted at the end of the count of statement.

The highest of and lowest of operators can be used on both Text and Number fields. highest of returns the highest ASCII field value in the set of related records. lowest of returns the lowest ASCII field value.

The sum of and mean of operators are used on Number fields. sum of returns the total of the field values in the set of related records. mean of returns the average of the field values in the set of related records.

Each Relational Statistical Operator is treated as a separate entry in this Language Reference.

Relationships

Type

Concept

Purpose

A Relationship is a link between two groups of records whether in the same table or in two different tables. When two groups of records are related, you can access information in one group while processing records in the other group. DataEase employs two types of relationships.

In a predefined relationship, the two groups of records are related by a link entered on a Relationships form. The link is created by specifying one or more fields that hold the same information in both groups. Because this relationship is stored in the database, it can be used in different procedures whenever necessary.

An ad hoc relationship is a relationship that is defined while creating a script. Although an ad hoc relationship can be used like a predefined relationship to access information in another group of records, the relationship isn't stored as a part of the database. Therefore, it can only be used in the procedure in which it is defined.

Usage

In a script, a value in a related table or group of records is called a relationship value. A relationship value can be used in a script wherever you can use a field, a constant value, or a variable (such as in selection criteria, as a list item, or as a value assigned to a temporary or global variable).

Once a relationship is established, the DQL Relational Operators let you access and manipulate values in the related table. For example, if the script is processing MEMBERS records, the statement:

```
sum of RESERVATIONS TOTAL DUE ;
```

..sums the TOTAL DUE field for each related reservation (that is, for every RESERVATIONS record with the same MEMBER ID as the MEMBERS record currently being processed).

The use of each Relational Operator is covered in a separate entry in this Language Reference.

reorganize

Type

Control Command

Purpose

The reorganize command reorganizes the specified table by updating the table's indices and physically erasing deleted records from disk.

The reorganize command functions like selecting **Application>>Utilities>>Reorganize**.

Syntax

```
reorganize  "TABLENAME"  [UNCLUSTERED|CLUSTERBY FIELD1 [ ;  
FIELD2...]] .
```

Usage

The reorganize command, used by itself, reorganizes the specified table as described above. If the table contains a clustered index, the records are reorganized according to the table-defined cluster order.

The tablename must be enclosed in quotes.

When a table is reorganized, all records that have been deleted since the last reorganization are permanently erased and all indices used to organize the data in the table are recreated.

If the table has become inconsistent (signaled by an error message), the reorganize command reorganizes the records in the table and returns the table to a consistent status.

The reorganize unclustered command overrides any table-defined cluster order. DataEase reorganizes the table in unclustered order. Any records entered since the table was last reorganized are left in the order they were entered.

The reorganize cluster by command lets you override the table-defined cluster order. The table is reorganized in order according to the field name(s) you specify. DataEase prohibits you from including any fields specified in the table-defined cluster order.

LAN

On a LAN (Local Area Network), if another user is currently using any resource required by the reorganize command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example 1

```
reorganize  "MEMBERS"  .  
run report  "PRINT INVOICES"  .
```

This script tells DataEase: (1) Reorganize the MEMBERS table, and (2) when the reorganization is completed, run the PRINT INVOICES procedure.

Example 2

```
reorganize  "DEPARTURE ZONES"  cluster by DEPARTURE ZONE .
```

This script tells DataEase to reorganize the DEPARTURE ZONES table. Cluster the data in order by the DEPARTURE ZONE field.

restore db (restore database)

Type

Control Command

Purpose

The restore db command recreates a database. When processing reaches a restore db command, DataEase displays a series of prompts asking you to specify the drive on which the database should be restored and how you want to handle any errors that occur during the restore procedure.

The restore db command functions like selecting **Application>>Utilities>>Restore**.

Syntax

```
restore db .
```

Usage

When you backup an application using the backup db command, DataEase copies the application using a special format. The backup copy can only be used after it is restored using the DataEase restore db command or Application menu option.

When you backup and restore an application, all records that have been deleted since the last backup and restore operation are permanently erased.

LAN

On a LAN (Local Area Network), if another user is currently using any resource required by the restore db command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

```
record entry  "MEMBERS"  .
run procedure  "PRINT INVOICES"  .
backup db .
restore db .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter new member records, (2) when the user closes the MEMBERS form, run the PRINT INVOICES procedure, (3) make a backup copy of the current application, and (4) when the backup procedure is complete, begin the procedure to restore the application from the backup copy.

rollback

Type

Procedural Command

Purpose

The rollback command is used to cancel a partially completed transaction when processing data stored in an SQL database. A procedure may contain any number of rollback commands.

The DQL commit and rollback commands are used to divide a procedure into multiple transactions (a transaction can be any command or procedure that changes data). By defining separate transactions within a DQL Procedure, it's possible to rollback partially completed changes that may leave the data in an inconsistent state.

When processing reaches a rollback command, DataEase immediately cancels all changes made to the data since the last commit command was processed. If a transaction accesses more than one server, the rollback command automatically cancels the transaction changes on each of the active servers. Once a transaction is committed, it cannot be undone by a rollback command.

The corresponding SQL command, ROLLBACK TRANSACTION, is used to cancel an embedded exec SQL statement in a DQL Procedure.

Syntax

```
rollback .
```

Usage

A rollback command can be used anywhere in a procedure.

The rollback command is often preceded by a conditional if statement. The if condition either checks the current status or current SQLCODE variable to see if the transaction was successful, or tests whether a specific business rule has been violated. If the last transaction was not fully completed or the business rule was violated, the rollback command tells DataEase to cancel the partially completed changes. Otherwise, processing continues.

If you do not include a rollback command in a procedure, DataEase automatically issues a rollback command at the end of any procedure that is interrupted before completion (due to a system malfunction, or when you voluntarily abort because of a resource conflict).

Example :

```
for RESERVATIONS with POSTED = NO ;
begin transaction
  enter a record in INVOICES
  copy all from RESERVATIONS .
  modify records in MEMBERS
    RESERVATIONSTOTAL := RESERVATIONSTOTAL + RESERVATIONS TOTAL
DUE ;
if any MEMBERS RESERVATIONS TOTAL > 20,000 then
  rollback .
  modify records in MEMBERS
    RESERVATIONSTOTAL := RESERVATIONSTOTAL - RESERVATIONS TOTAL
DUE ;
message " No member can have an account
  balance over $20,000. This reservation
  has been canceled. " .
```

```

else
    modify records
        POSTED = YES .
    commit .
    message " Member and Reservation Information
        updated. " .
    end
end
end

```

This first part of this procedure contains a transaction that enters a record in a table that owns an SQL table (INVOICES) and modifies a record in a native DataEase table (MEMBERS). The script enters a record in the INVOICES table for each record in the RESERVATIONS table that has not been posted by entering all the information for each unposted reservation into an invoice.

The script then adds the reservation's total to the member's total. If the member's cumulative reservation total exceeds \$20,000, the reservation is canceled, the modifications made to the SQL table are rolled back, and the cost of the reservation is subtracted from the member's total due.

The second part of the transaction modifies the current RESERVATIONS record by setting the POSTED field to yes. This part of the script is executed only if the first part of the transaction is successful. Once the second part is completed, the entire transaction is committed.

Note: When a transaction fails and data changes are rolled back, only the last group of modifications is canceled. Earlier transactions that have already been committed are not canceled. Carefully placed commit and rollback commands in your DQL procedures can minimize the work that must be repeated if an error occurs during processing.

run procedure

Type

Control Command

Purpose

The run procedure command opens the specified procedure document and executes its DQL instructions.

Syntax

```
run procedure  "PROCEDURE NAME"  .
```

Usage

The run procedure command automatically opens and runs a previously defined procedure document. When the procedure finishes processing, the Control Procedure automatically resumes with the action following the run procedure command.

The procedure name can be a constant or any expression (including functions) that returns a text string specifying the desired procedure name.

Example

```
record entry  "MEMBERS"  .  
run procedure  "PRINT INVOICES"  .  
application status records .
```

This script tells DataEase: (1) Display the MEMBERS form so the user can enter new member records, (2) when the user closes the MEMBERS form, run the PRINT INVOICES procedure, then (3) display the status of the records in the current database.

Secondary table

Type

Concept

Purpose

A table related to the Primary table is called a Secondary table.

Usage

One table can be related to another table by either a predefined or an ad hoc relationship. A table related to the Primary table is called a Secondary table. A table related to a Secondary table is called a Tertiary table, etc. There can be any number of such relationship levels.

When processing the records in a table, the data in a Secondary table can be accessed if a predefined relationship exists between the tables or if an ad hoc relationship has been established in the current script. A relational operator (all, any, count of, highest of, lowest of, mean of, or sum of) or nested for command is used to access the data in the related table.

seconds

Type

Time Function

Purpose

The seconds function extracts the seconds from a time value expressed in 24-hour format.

Syntax

```
seconds( TIME VALUE ) paren
```

Returns

An integer value from 0 to 59.

Examples

```
seconds( 09: 15: 34)
```

Returns: 34

```
seconds( 02: 53: 20)
```

Returns: 20

```
seconds( CHECK-OUT T IME)
```

Returns the seconds value in the CHECK-OUT TIME field for every record that is processed. If a record contains the value 07:30:25, the function returns 25.

Selection Criteria

Type

Concept

Purpose

In a script or report, you can tell DataEase to process or display all the records in a specified table, or only some of the records. The limiting factors that determine which records are selected for processing are called selection criteria.

The selection criteria tell DataEase to compare the value in the specified field in each record to a comparison value that you specify in a script or selection filter. When the procedure or report is run, the records that meet the defined selection criteria are processed. Records that do not meet the criteria are not processed.

Usage

There are two situations in which selection criteria must be enclosed in parentheses:

Criteria must be enclosed in parentheses when used with a relational operator to select records based on the values stored in a table other than the Primary table. (See Example 2.)

When two sets of selection criteria are combined with the and and or operators, the criteria must be enclosed in parentheses to clarify the meaning. (See Example 3.)

Example 1

The statement:

```
for MEMBERS with TOTAL DUE > 100 ;
```

..tells DataEase to process only the MEMBERS records that have a value greater than 100 in the TOTAL DUE field.

Example 2

The statement:

```
for MEMBERS with ( sum of RESERVATIONS TOTAL DUE > 5000 ) ;
```

..tells DataEase to process only the MEMBERS records whose related RESERVATIONS have a combined TOTAL DUE that is greater than 5000.

Example 3

The statement:

```
for MEMBERS with ( STATE = "NY" or STATE = "NJ" )
and TOTAL DUE > 200 ;
```

..tells DataEase to process MEMBERS records for only the members who live in either New York or New Jersey and who have a TOTAL DUE greater than 200.

sin

Type

Trigonometric Function

Purpose

The sin function calculates the sine of an angle expressed in radians.

Syntax

```
sin( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
sin( 2.53)
```

Returns: 0.574172

```
sin( -3)
```

Returns: -0.14112

sinh

Type

Trigonometric Function

Purpose

The sinh function calculates the hyperbolic sine of an angle expressed in radians.

Syntax

```
sinh( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
sinh( 2.53)
```

Returns: 6.2369235

```
sinh( -4.4)
```

Returns: -40.71929

Sorting

Sorting lets you process records and generate the resulting output in either ascending or descending order based on the values in the specified field.

The in order operator sorts records in sequence from least to greatest value based on the type of field as follows:

Text: alphabetical order (A-Z)

Number: least to greatest numerical value

Date: earliest to latest date

Time: earliest to latest time

Choice: lowest to highest numbered choice

Example

```
for CLUBS ;
  list records
    CLUB NAME in order .
end
```

The output from this script might look as follows:

Club Name

Bora Bora
Buccaneer's Creek
Cancun...
Uxmal

The in reverse operator sorts records in greatest to least sequence. For example, if we change the script to:

```
for CLUBS ;
  list records
    CLUB NAME in reverse .
end
```

The output is reversed:

Club Name

Uxmal
Turkoise
Teotihuacan...
Bora Bora

spellcurrency

Type

Spell Function

Purpose

The spellcurrency function spells the monetary equivalent of a numeric value.

Syntax

```
spellcurrency( NUMERIC VALUE)
```

Returns

A text value.

Examples

```
spellcurrency( 36.25)
```

Returns: Thirty Six Dollars and 25 Cents

```
spellcurrency( 210.00)
```

Returns: Two Hundred Ten Dollars and 00 Cents

```
spellcurrency( TOTAL DUE)
```

Returns: The spelled-out currency amount in the TOTAL DUE field for every record processed. If a record contains the value 905.75, the function returns Nine Hundred and Five Dollars and 75 Cents.

Note: the actual currency used is determined by which regional version of DataEase you have purchased. UK users, for example, should see the words “pounds” and “pence” instead of “dollars” and “cents”.

spelldate

Type

Spell Function

Purpose

The spelldate function spells a date value in common form.

Syntax

```
spelldate( DATE VALUE)
```

Returns

A text value.

Examples

```
spelldate( 07/01/01)
```

Returns: July 1, 2001

```
spelldate( 12/31/00)
```

Returns: December 31, 2000

```
spelldate( highest of RESERVATIONS DATE )
```

Returns: The spelled-out value in the most recent reservation's DATE field. If the most recent RESERVATIONS record contains the value 10/31/01 in the DATE field, the function returns October 31, 2001.

spellmonth

Type

Spell Function

Purpose

The spellmonth function spells the name of the month that corresponds to a numeric value from 1 (January) to 12 (December).

Syntax

```
spellmonth( NUMERIC VALUE)
```

Returns

A text value.

Usage

The input value must be an integer between 1 and 12 (inclusive).

Examples

```
spellmonth( 7)
```

Returns: July

```
spellmonth( month( 12/31/99)
```

Returns: December

```
spellmonth( month( highest of RESERVATIONS DATE ) )
```

Returns: The name of the month that corresponds to the value in the most recent reservation's DATE field. If the most recent RESERVATIONS record contains the value 10/31/01 in the DATE field, the function returns October.

spellnumber

Type

Spell Function

Purpose

The spellnumber function spells the integer portion of a numeric value.

Syntax

```
spellnumber( NUMERIC VALUE)
```

Returns

A text value

Usage

spellnumber(999999999) returns Nine Hundred Ninety Nine Million, Nine Hundred Ninety Nine Thousand, Nine Hundred Ninety Nine.

Examples

```
spellnumber( 3.45)
```

Returns: Three

```
spellnumber( 323.45)
```

Returns: Three Hundred Twenty Three

```
spellnumber ( RESERVATION ID )
```

Returns: The spelled-out number in the RESERVATIONID field for every record that is processed. If a record contains the value 10251, in the RESERVATION ID field, the function returns Ten Thousand Two Hundred Fifty One.

spellweekday

Type

Spell Function

Purpose

The spellweekday function spells the name of the day that corresponds to a numeric value from 1 (Monday) to 7 (Sunday).

Syntax

```
spellweekday( NUMERIC VALUE)
```

Returns

A text value.

Examples

```
spellweekday( 5)
```

Returns: Friday

```
spellweekday( weekday( 12/31/99)
```

Returns: Sunday

```
spellweekday( weekday( highest of RESERVATIONS DATE ) )
```

Returns: The name of the day that corresponds to the value in the most recent reservation's DATE field. If the most recent RESERVATIONS record contains the value 10/31/99 in the DATE field, the function returns Tuesday.

sqrt (square root)

Type

Scientific Function

Purpose

The sqrt function calculates the square root of a numeric value.

Syntax

```
sqrt ( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Scientific function can be a constant value, a variable, a field value, or an expression. The sqrt function should only be used on a positive value. If the input value is negative, the result is unpredictable.

Examples

```
sqrt ( 9)
```

Returns: 3

```
sqrt ( 16)
```

Returns: 4

```
sqrt ( FLOOR SPACE)
```

Returns: The square root of the value in the FLOOR SPACE field for every record processed. If a record contains the value 16000 in the FLOOR SPACE field, the function returns 40.

Statistical Operators

Type

Operator

Purpose

Statistical Operators (shown at right) are used to generate statistical information on numeric fields that appear as list items in a script.

Syntax

FIELDNAME : statistical operator

Usage

To generate statistics on a field:

- Insert the fieldname followed by a colon.
- After the colon, insert the appropriate statistical operators.
- After the last operator, insert a semicolon or a period, depending on the location of the operator in the script.

Statistical Operators can be used on any type of numeric field (Currency, Integer, Fixed Point, and Floating Point). They cannot be used on Numeric String fields.

You may use as many operators as you want on a given field, one after another, separating each by a space.

If you use a statistical operator and you want the field value listed in addition to the statistical information, you must include the item operator after the field name.

The statistical operators are: item, max, mean, min, std. dev., std. err., sum, and variance. Each statistical operator is treated as a separate entry in this Language Reference.

std.dev. (standard deviation)

Type

Statistical Operator

Purpose

The std.dev. operator calculates the standard deviation (the square root of the variance) in a set of data. The result usually appears as a statistic in the summary area at the end of a report.

Syntax

```
FIELDNAME|VARIABLE :  std.dev.      [other statistical operators] ; |
.
```

Returns

A numeric value.

Usage

Standard deviation is used as an indicator of variability in a set of data (68% of the data set is contained in the first standard deviation).

The std.dev. operator can also be used when creating a report using Query by Model. To generate the standard deviation in Query by Model, highlight the column whose standard deviation you want to calculate, then select std.dev. in the Summarize pick list.

Example

```
for MEMBERS with TOTAL DUE  > 175 ;
  list records
    LAST NAME in order ;
    TOTAL DUE :  item sum std.dev. .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$175 in the TOTAL DUE field, (2) list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE amount, the sum of these amounts, and the standard deviation in the report output.

The output from this script might look as follows:

Last Name	Total Due
Christino	\$280.00
Perrault	\$215.00
Stafford	\$185.00
Strachan	\$205.00
Sum Total Due:	\$885.00
Std. Dev. in data set:	\$41.10

std.err. (standard error)

Type

Statistical Operator

Purpose

The std.err. operator calculates the standard error (the standard deviation divided by the square root of the number of items) in a set of data. The result usually appears as a statistic in the summary area at the end of a report.

Syntax

```
FIELDNAME|VARIABLE : std.err.
    [other statistical operators] ; | .
```

Returns

A numeric value.

Usage

Standard error indicates the variability in a subset of data; if the standard error of the mean of two data sets overlap, the subsets may be considered to be members of the same data set.

The std.err. operator can also be used when creating a report using Query by Model. To generate the standard error in Query by Model, highlight the column whose standard error you want to calculate, then select std.err. in the Summarize pick list.

Example

```
for MEMBERS with TOTAL DUE > 175 ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item sum std.dev. std.err. .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$175 in the TOTAL DUE field, (2) list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE and include the standard deviation, standard error, and the sum of all the TOTAL DUE amounts in the procedure output. The output from this script might look as follows:

Last Name	Total Due
Christino	\$280.00
Perrault	\$215.00
Stafford	\$185.00
Strachan	\$205.00
Sum Total Due:	\$885.00
Std.Dev. in data set:	\$41.10
Std. Err. in data set:	\$20.55

sum

Type

Statistical Operator

Purpose

The sum operator adds the values in a specified field among all the records that are processed.

Syntax

```
FIELDNAME | VARIABLE : sum [other statistical operators] ; | .
```

Returns

A numeric value

Usage

The value returned by the sum operator differs depending on where its corresponding Summary field appears in the document layout. The effect of the position of the Summary field on the sum value is summarized in the table below.

Location	Sum
Document Header or Footer	N/A
Main form object	Grand total
Group Form object	Group total

Example

```
for MEMBERS with TOTAL DUE > 175 ;
  list records
    LAST NAME in order ;
    TOTAL DUE : item sum .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$175 in the TOTAL DUE field, (2) list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE and include the sum of all the TOTAL DUE amounts in the report output.

The output from this script might look as follows:

Last Name	Total Due
Christino	\$280.00
Perrault	\$215.00
Stafford	\$185.00
Strachan	\$205.00
Sum Total Due:	\$885.00

Note: There's an important difference between the statistical operator sum and the relational statistical operator sum of. sum returns the total of the values in the specified field among the records being processed. sum of returns the total of the values in the specified field among the records related to the records being processed.

sum of

Type

Relational Statistical Operator

Purpose

The sum of operator adds the values in a specified field in all matching records in a related table. The result can appear as a list item in the detail area of a report or as a statistic in the summary area at the end of each group or at the end of the report.

Syntax

```
sum of TABLENAME | RELATIONSHIP
    [named "UNIQUE RELATIONSHIP NAME" ]
    [with ( selection criteria) ] FIELDNAME ; | .
```

Returns

A numeric value.

Example

```
for MEMBERS ;
  list records
    LAST NAME in order ;
    sum of RESERVATIONS TOTAL DUE .
end
```

This script tells DataEase: (1) Process all the MEMBERS records and list each member's LAST NAME in alphabetical order, (2) for each MEMBERS record processed, find all the related records in the RESERVATIONS table (those that have the same MEMBERID), and (3) list the sum of the TOTAL DUE field for the set of RESERVATIONS records that match the current MEMBERS record.

The output from this script might look as follows:

Last Name	Sum of Reservations
	Total Due
Adams	\$3000.00
Albert	\$4760.00
Anders	\$4420.00
Andersen	\$2100.00
Anderson	\$4320.00
Archer...	\$4796.00...

If you want to include the sum of this whole group of reservations, change the fourth line of the query to read:

```
sum of RESERVATIONS TOTAL DUE : item sum .
```

Note: There's an important difference between the statistical operator sum and the relational statistical operator sum of. sum returns the total of the values in the specified field among the records being processed. sum of returns the total of the values in the specified field among the records related to the records being processed.

tan (tangent)

Type

Trigonometric Function

Purpose

The tan function calculates the tangent of an angle expressed in radians.

Syntax

```
tan( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
tan( 2.53)
```

Returns: -0.701292

```
tan( -1.89)
```

Returns: -3.025665

tanh (hyperbolic tangent)

Type

Trigonometric Function

Purpose

The tanh function calculates the hyperbolic tangent of an angle expressed in radians.

Syntax

```
tanh( NUMERIC VALUE)
```

Returns

A numeric value.

Usage

The numeric value in a Trigonometric function can be a constant value (as shown below), a variable, a field value, or an expression.

Examples

```
tanh( 2.53)
```

Returns: 0.987388

```
tanh( 1.45)
```

Returns: -0.895692

temp

Type

Keyword

Purpose

The keyword temp specifies a temporary variable.

Syntax

```
define temp VARIABLE NAME datatype .
assign temp VARIABLE NAME := value .
```

Usage

A variable is used to store a value such as a text string or a calculated result that can change during the processing of a procedure. By specifying the variable's name in a script, the stored value can be used like any other value.

The status of a variable can be global (denoted by the keyword global) or temporary (denoted by the keyword temp).

Although a global variable can pass its value from one procedure to another, a temporary variable holds its value only while the current procedure is processing.

Note: Because the value in a variable is frequently accumulated as each record is processed, you cannot sort (e.g., using in order or in groups) the value stored in a variable.

Example

```
define temp "DISCOUNT" Number .
for RESERVATIONS with TOTAL DUE > 4000 ;
  assign temp DISCOUNT := ( TOTAL DUE * 0.15) .
  modify records
    TOTAL DUE := ( TOTAL DUE - temp DISCOUNT ) .
end
```

This script tells DataEase: (1) Create (define) a temporary variable called DISCOUNT to store a number while processing the current script, (2) find all the RESERVATIONS records that have a value greater than \$4000 in the TOTAL DUE field, (3) give (assign) the DISCOUNT variable a number value determined by multiplying the TOTAL DUE on each invoice by 15%, and (4) modify these RESERVATIONS records by subtracting the value of the DISCOUNT variable from the value in the TOTALDUE field.

textpos

Type

Text Function

Purpose

The textpos function returns the first position of a substring in a specified text value.

Syntax

```
textpos( "TEXT VALUE" | FIELDNAME, "SUBSTRING" )
```

Returns

An integer value that indicates the position where the substring begins in the specified text value.

Usage

The textpos function is not case sensitive (no distinction is made between upper and lower case letters).

If the substring is contained in the specified text value, the starting position of the first (leftmost) character in the substring is returned.

If the substring is not contained in the text value, the function returns 0. Intervening spaces and punctuation symbols are included in the calculation. Trailing spaces are ignored.

Examples

```
textpos( "Buccaneer's Creek" , "can" )
```

Returns: 4

```
textpos( "Sapphire International" , "national" )
```

Returns: 15

```
textpos( CLUB NAME, "bus" ) ,
```

Returns: The first position of the string bus in every record that is processed. If a record contains the value Columbus Island in the CLUB NAME field, the function returns 6.

Note: textpos interprets wildcard symbols like "*" as a character, rather than a wildcard. It can therefore be used to detect the presence of such wildcard characters in a text string

timeampm

Type

Time Function

Purpose

The timeampm function converts a time value from a 24-hour format to a 12-hour format and appends the appropriate AM or PM designation.

Syntax

```
timeampm( TIME VALUE)
```

Returns

A text value representing the 12-hour clock time with a suffix of either AM or PM. Converts time values from midnight (00:00:00) to (11:59:59) to AM. Converts time values from noon (12:00:00) to (23:59:59) to PM.

Examples

```
timeampm( 13: 53: 12)
```

Returns: 01: 53: 12 PM

```
timeampm( 11: 10: 30)
```

Returns: 11: 10: 30 AM

```
timeampm( DEPARTURE TIME)
```

Returns the value in the DEPARTURE TIME field in the 12-hour clock format and adds the correct suffix for every record that is processed. If a record contains the value 19:30:25 in the DEPARTURE TIME field, the function returns 07:30:25 PM.

tran off

Type

Control Command

Purpose

The tran off command is used to turn off the DataEase default transaction management and data locking facilities and switch DQL processing into tran off mode when accessing data in an SQL database.

DQL Procedures can be run in either tran on mode (the default mode) or tran off mode.

In tran on mode, DataEase treats any procedure that adds, modifies, or deletes data as a transaction (the procedure is processed as a single unit of work) as long as there are no explicit begin transaction and commit commands. When DataEase is processing a procedure in tran on mode, it automatically provides the necessary commands to lock and unlock data and ensure that each transaction is either fully completed or rolled back.

In tran off mode, DataEase processes each record as a separate transaction. No DQL actions are grouped together as a transaction and no multi-user data locking rules are enforced.

Syntax

```
tran off .
```

Usage

A DQL Procedure can contain any number of tran off and tran on commands. By using the tran on and tran off commands to turn the DataEase transaction management facilities on and off, you can link regular Processing procedures and Transaction Processing procedures together in the same Control procedure (see the example on the next page).

Once DataEase enters tran off mode, it remains in this mode until it reaches a tran on command or the end of the whole procedure.

Example

```
tran off .
record entry  "RESERVATIONS"  .
tran on .
run procedure  "PRINT RESERVATIONS"  .
```

This script tells DataEase: (1) Turn off the default transaction management and data locking facilities, (2) display the RESERVATIONS form so the user can enter new reservations, (3) when the user finishes entering records, turn the transaction management facilities back on, and (4) run the PRINT RESERVATIONS Processing procedure.

Note: Because it reduces the amount of data locking the server must perform, the tran off command can significantly improve the performance of a procedure. However, DataEase provides no concurrency control when processing records in tran off mode.

For this reason, we recommend that procedures that contain tran off commands should only be run when no one else is accessing the SQL tables referenced in the script.

tran on

Type

Control Command

Purpose

The tran on command is used when accessing data in an SQL database to turn on the DataEase default transaction management and data locking facilities after they have been disabled by the tran off command.

DQL Procedures can be run in either tran on mode (the default mode) or tran off mode.

In tran on mode, DataEase treats any procedure that adds, modifies, or deletes data as a transaction (the procedure is processed as a single unit of work) as long as there are no explicit begin transaction and commit commands. When DataEase is processing a procedure in tran on mode, it automatically provides the necessary commands to lock and unlock data and ensure that each transaction is either fully completed or rolled back.

In tran off mode, DataEase processes each record as a separate transaction. No DQL actions are grouped together and no multi-user data locking rules are enforced.

Syntax

```
tran on .
```

Usage

A DQL Procedure can contain any number of tran on and tran off commands. By using the tran on and tran off commands to turn the DataEase transaction management facilities on and off, you can link regular Processing procedures and Transaction Processing procedures together in the same Control Procedure (see the example on the next page).

Note: DataEase is always in tran on mode at the start of a DQL Procedure and remains in this mode until processing reaches a tran off command.

Example

```
tran off .
record entry  "RESERVATIONS"  .
tran on .
run procedure  "PRINT RESERVATIONS"  .
```

This script tells DataEase: (1) Turn off the default transaction management and data locking facilities, (2) display the RESERVATIONS form so the operator can enter new reservations, (3) when the user finishes entering records, turn the transaction management facilities back on, and (4) run the PRINT RESERVATIONS Processing procedure.

Note: Because it reduces the amount of data locking the server must perform, the tran off command can significantly improve the performance of a procedure. However, DataEase provides no concurrency control when processing records in tran off mode.

For this reason, we recommend that procedures that contain tran off commands should only be run when no one else is accessing SQL tables referenced by the script.

Transaction Processing

Type

Concept

Purpose

Transaction processing is a technique that lets you enter, modify, and delete records in multiple tables using input from a single form. A transaction processing procedure lets you update the information in the database without having to physically enter the data into each form in record entry mode. Because it saves time and reduces the possibility of keying errors, transaction processing is widely used for point-of-sale applications to automate customer billing and inventory maintenance.

Usage

With DataEase, each transaction is entered using a Data-entry form or an input using command. After the data is entered, it is processed by a predefined DQL Procedure that automatically posts the information into the appropriate related tables.

unlock

Type

Processing Command

Purpose

The unlock command is functional only when running DataEase on a LAN (Local Area Network). The unlock command unlocks locked tables or records allowing other users full access to the data.

Syntax

```
unlock all files .  
unlock file TABLENAME .  
unlock selected record .
```

Usage

The unlock all files command unlocks all tables referenced by the procedure. If unlock all files is the first statement in the script, the default locking rules are overridden and all data is locked by individual records. You may lock tables on a selective basis using the lock file command.

The unlock file command unlocks the specified table for the duration of the procedure, or until the same table is specified in a lock command. In a script that begins with a lock all files statement, the unlock file command lets you unlock tables selectively, while the other tables referenced by the script remain locked.

The unlock selected record command unlocks the record that is currently being processed. If the default locking rules are overridden by an unlock all files or unlock file command, DataEase automatically locks a record from the time it is selected for processing until the next record is selected. Often, significant processing not involving the locked record may take place. By using the lock and unlock selected record commands, you can reduce the amount of time a record remains locked.

LAN

The default LAN Multi-User Locking Options resolve conflicts that arise when users attempt to view, modify, or delete records in a table while data in that table is being accessed by a procedure. If another user is currently using a resource required by the unlock command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Example

In order to override the default locking rules, the first statement of a script must be either lock all files or unlock all files. If neither of these commands appears at the start of the script, the default locking rules govern LAN functioning until a lock or unlock command appears in the script. All lock and unlock commands are automatically terminated at the end of a script.

```
lock file RESERVATIONS shared .

for MEMBERS with ( sum of
  RESERVATIONS TOTAL DUE  > 5000 )  ;
  lock selected record .
  list records
    LAST NAME ;
    TOTAL DUE .
  unlock selected record .
end
for MEMBERS with ( highest of
  RESERVATIONS DEPARTURE DATE < 01/01/99 )  ;
  delete records .
end
```

This script tells DataEase: (1) Lock the RESERVATIONS table so other users can view records but cannot add, delete or modify RESERVATIONS records during the processing of this procedure, (2) select all the MEMBERS records whose related RESERVATIONS records have a combined TOTAL DUE greater than \$5000, (3) lock each MEMBERS record as it is selected for further processing. For each record selected, list the member's LAST NAME and TOTAL DUE, and (4) after each record is listed, unlock the record before continuing to the next part of the script.

Locking the RESERVATIONS table at the start of this procedure prevents other users from entering new records that might alter the sum of calculation. Locking and unlocking each MEMBERS record as it is listed prevents the TOTAL DUE information from being modified by another user.

unlock db (unlock database)

Type

Control Command

Purpose

The unlock db command is functional only when running DataEase on a LAN (Local Area Network). The unlock db cancels the effect of the lock db command; it unlocks a locked database.

Syntax

```
unlock db .
```

Returns

A confirmation message appears on the screen as soon as the database is unlocked.

LAN

On a LAN, if another user is currently using a resource required by the unlock db command, DataEase displays a Resource Conflict message. While this message is displayed, DataEase automatically tries to execute the command at brief intervals.

When the required resource becomes available, DataEase automatically resumes processing and executes the rest of the procedure.

Usage

Once a database is locked by the lock db command it remains locked until an unlock db command is executed or until the user who initiated the lock db command exits from DataEase.

Example

```
lock db .
record entry  "MEMBERS"  .
run procedure  "PRINT RESERVATIONS"  .
unlock db .
```

This script tells DataEase: (1) Lock the current database, (2) display the MEMBERS form so an operator can enter new member records, (3) when the operator finishes entering records, run the PRINT RESERVATIONS procedure, and (4) unlock the current database.

Note

There's an important difference between the unlock Processing command and the unlock db Control command. The unlock Processing command can be used to unlock individual forms or records during a Processing procedure. The unlock db command is used to cancel the effect of the lock db Control command; it unlocks the whole database.

upper

Type

Text Function

Purpose

The upper function converts each letter in a text value to uppercase.

Syntax

```
upper( TEXT VALUE)
```

Returns

The specified text value with all the letters in uppercase.

The upper function only affects the alphabetical letters a-z. Numeric values are not converted to their Shift key equivalents.

Examples

```
upper( "DataEase" )
```

Returns: DATAEASE

```
upper( "Cancun" )
```

Returns: CANCUN

```
upper( LAST NAME)
```

Returns the value in the LAST NAME field in uppercase type for every record processed. If a record contains the value Birnbaum in the LAST NAME field, the function returns BIRNBAUM.

value

Type

Command component

Purpose

The value keyword is a component of the case command syntax. When processing a case command, DataEase compares the expression that follows case to each of the statements specified by the keyword value in the order they appear in the script. When DataEase reaches the first true comparison, it executes all the actions between that value statement and the next.

If none of the specified value comparisons is true, DataEase executes the actions specified after the keyword others. As soon as all actions following any value or others statement are executed, processing passes to the first action following the end command for the case statement.

Syntax

```
case ( EXPRESSION)
    value COMPARISON 1 :
        ACTION SERIES 1 .
    [value COMPARISON 2 :
        ACTION SERIES 2 .
    .
    .
    value COMPARISON N :
        ACTION SERIES N . ]
    [others :
        DEFAULT ACTION SERIES . ]
end
```

Usage

The case command requires a case expression, one comparison value, and an end command. Subsequent value statements, actions, and the others keyword are optional. If others is used, it must follow all the specified comparison values.

Example

```
case ( current user name )
    value "FRANK" :
        call menu " SITE ADMINISTRATION " .
    others :
        call menu " MAIN MENU " .
end
```

This script tells DataEase: (1) If the current user is Frank, display the SITE ADMINISTRATION menu, and (2) if the current user is anyone other than Frank, display the MAIN MENU.

variable

Type

Concept

Purpose

A variable is a substitute for another value.

In a script, a variable can be substituted for any expression that returns a value or for any type of field except a Choice field. Often a variable is used to store a numeric value that can change during the processing of a procedure (like a running total when adding a column of figures). The stored value can be used like any other value in the script; it is referenced by the variable's name.

Usage

The name of a variable can be any unique name, up to 20 characters in length. The name must be enclosed in quotes when it first appears, and be preceded by either the global or temp keyword.

The status of a variable is either current, global, or temporary.

The current variables are system-defined variables that hold commonly used system values such as current date, current time, current user name, etc. Because these variables are system-generated, you don't need to define them in a script prior to using them.

A global variable can pass its value from one procedure to another. In order to do this, the variable must be defined identically in each procedure in which it is used.

A temporary variable can store a value only during a single procedure.

There is a separate entry in this Language Reference for each of these groups.

Note: Because the value in a variable is typically accumulated as each record is processed, you cannot successfully sort (e.g., using in order or in groups) the value stored in the variable.

variance

Type

Statistical Operator

Purpose

The variance operator calculates the average of the squared deviations from the mean (the average of the squared difference between each item value and the mean value) in a set of data. The result usually appears as a statistic in the summary area at the end of each group or the end of the report.

Syntax

```
FIELDNAME|VARIABLE : variance [other
    statistical operators] ; | .
```

Returns

A numeric value.

Usage

Variance is used as an indicator of variability in a set of data.

The variance operator can also be used when creating a report using Query by Model. To generate the variance in Query by Model, highlight the column whose variance you want to calculate, then select variance in the Summarize pick list.

Example

```
for MEMBERS with TOTAL DUE > 175 ;
    list records
        LAST NAME in order ;
        TOTAL DUE : item mean sum variance .
end
```

This script tells DataEase: (1) Process all the MEMBERS records that have a value greater than \$175 in the TOTAL DUE field, (2) list the members by LAST NAME in alphabetical order, and (3) list each member's TOTAL DUE and include the mean, sum, and variance of all the TOTAL DUE amounts in the report output.

The output from this query might look as follows:

Last Name	Total Due
Christino	\$280.00
Perrault	\$215.00
Stafford	\$185.00
Strachan	\$205.00
Mean Total Due:	\$221.25
Sum Total Due	\$885.00
Variance in data set:	\$1689.58

weekday

Type

Date Function

Purpose

The weekday function converts a date value to an integer that corresponds to the day of the week from 1 (Monday) to 7 (Sunday).

Syntax

```
weekday( DATE VALUE)
```

Returns

An integer value from 1 to 7.

Examples

```
weekday( 12/31/99)
```

Returns: 5

```
spellweekday ( weekday( 12/31/99) )
```

Returns: Friday

```
weekday( DATE)
```

Converts the value in the DATE field into an integer from 1 to 7 for every record processed. If a record contains the value 10/31/00 (a Sunday), the function returns 7.

while

Type

Procedural Command

Purpose

Like the if command, the while command executes a series of actions based on whether a specified condition is true. Unlike the if command, the while command creates a processing loop in which actions are executed repeatedly as long as the specified condition remains true.

When processing reaches a while command, DataEase evaluates the condition that follows the keyword while. If the specified condition is true, DataEase executes all the actions that follow the keyword do until it reaches the corresponding end command.

DataEase then reevaluates the original condition. If the condition is still true, DataEase executes the do action series again. If the condition is false, processing passes to the first action following the end command for the while statement.

Syntax

```
while CONDITION do
    ACTION 1 .
    [ACTION 2 .
    .
    .
    ACTION N .]
end
```

Usage

Frequently, a while command specifies an initial condition that is modified during each pass through the loop (for example, a counter). Any action that modifies the value to be reevaluated must be included in the loop (i.e., it must precede the end command that terminates the loop.)

Example

```

define temp  "CRUISE TICKET NUM"  Number .
assign temp CRUISE TICKET NUM := 0 .

while temp CRUISE TICKET NUM < 1000 do
  temp CRUISE TICKET NUM := temp CRUISE TICKET NUM + 1 .
  list records
    jointext (  " Boarding Pass No.:  " ,
      temp CRUISE TICKET NUM)  .
end

```

This script tells DataEase: (1) Create (define) a temporary variable called CRUISE TICKET NUM, (2) give (assign) an initial value of 0 to the CRUISE TICKET NUM variable, (3) print a series of labels joining the words Boarding Pass No.: to the number currently stored in the CRUISE TICKET NUM variable, (4) while printing the labels, increment the variable by one each time a new label is printed, and (5) when the value of the variable reaches 1000, stop printing labels.

The while command tells DataEase to reevaluate the value of the variable each time it prints a label. As long as that value is less than 1000, DataEase prints another label. When the value of the variable reaches 1000, DataEase stops performing the action following the do keyword.

with

Type

Operator

Purpose

The with operator specifies the criteria for selecting records in a table or a relationship.

Syntax

```
with ( SELECTION CRITERIA)
```

Usage

The with operator tells DataEase to process some rather than all the records in the specified table. The records that are selected for processing are those that satisfy the selection criteria that follow the with operator.

Example 1

```
for MEMBERS with TOTAL DUE > 150 ;
```

This statement tells DataEase to process only those MEMBERS records that contain a value greater than \$150 in the TOTAL DUE field.

Example 2

```
delete records in RESERVATIONS named "OUTDATED"  
with ( DATE < 01/01/99) .
```

This statement tells DataEase to delete the records in the "OUTDATED" relationship (the RESERVATIONS records that are dated before 01/01/1999).

year

Type

Date Function

Purpose

The year function extracts the year from a date value.

Syntax

```
year( DATE VALUE)
```

Returns

An integer value from 0 to 99 (inclusive). The date format selected in Windows Control Panel changes the date sequence but does not affect which value is returned by a Date function.

If the year function is used on an extended date field or variable, it will return a four digit numeric string from 1776 to 9999.

Examples

```
year( 12/31/99)
```

Returns: 99 (most North American formats)

```
year( 31/12/99)
```

Returns: 99 (most European formats)

```
year(99/12/31)
```

Returns: 99 (Metric format)

```
year( DATEFIELD)
```

Converts the year portion of the value in the DATE field into an integer from 0 to 99 for every record processed. If a record contains the value 12/31/99 in the DATE field, the function returns 99.

yearday

Type

Date Function

Purpose

The yearday function converts a date value to the Julian day of the year (1-366).

Syntax

```
yearday( DATE VALUE)
```

Returns

An integer value from 1 to 366 (inclusive). The date format selected in Windows Control Panel changes the date sequence but does not affect which value is returned by a Date function.

Examples

```
yearday( 12/31/99)
```

Returns: 365 (most North American formats)

```
yearday( 31/12/99)
```

Returns: 365 (most European formats)

```
yearday( 95/12/31)
```

Returns: 365 (Metric format)

```
yearday( DATE)
```

Converts the day portion of the value in the DATE field into an integer from 1 to 366, representing the Julian date, for every record processed. If a record contains the value 12/31/00 (a Leap Year) in the DATE field, the function returns 366.

yearweek

Type

Date Function

Purpose

The yearweek function calculates the week number (1-53) of a date value.

Syntax

```
yearweek( DATE VALUE)
```

Returns

An integer value from 1 to 53 (inclusive). The date format selected in Windows Control Panel changes the date sequence but does not affect which value is returned by a Date function.

Usage

This yearweek function always counts January 1st-7th as Week1, January 8th-14th as Week 2, etc. In a non-Leap Year, the only day in Week 53 is December 31st. In a Leap Year, December 30th is also counted in Week 53 (all dates following February 28 are affected by leap year).

Examples

```
yearweek( 01/01/99)
```

Returns: 1

```
yearweek( 12/30/99)
```

Returns: 52 (most North American formats)

```
yearweek( 30/12/99)
```

Returns: 52 (most European formats)

```
yearweek( 99/12/30)
```

Returns: 52 (Metric format)

```
yearweek( DATE)
```

Converts the day portion of the value in the DATE field into an integer from 1 to 53, representing the week of the year, for every record processed. If a record contains the value 12/30/96 (a Leap Year) in the DATE field, the function returns 53.

Index

- - (comment), 165
- (subtraction), 152
- " " (quotation marks), 161
- () (parentheses), 158
- * (asterisk), 154
- * (multiplication), 153
- . (period), 159
- / (division), 153
- : (colon), 157
- := (assignment operator), 162
- ; (semicolon), 160
- ? (question mark), 155
- ~ (tilde), 156
- + (addition), 152
- < (less than), 163
- <= (less than or equal to), 163
- = (equals), 163
- > (greater than), 164
- >= (greater than or equal to), 164
- abs (absolute value), 166
- acos (arccosine), 167
- ad hoc relationship, 168
- all, 170
- ampm, 172
- and, 173
- any, 174
- application status, 175
- asin (arcsine), 176
- assign, 177
- atan (arctangent), 178
- atan2 (arctangent 2), 179
- backup db, 180
- begin transaction, 181
- between, 183
- blank, 184
- Body of a Procedure, 30
- break, 185
- Calculate a Percentage, 114
- call menu, 186
- call program, 187
- case, 188
- ceil, 190
- Check DQL Option, 25
- Clearing a DQL, 41
- Client-Server Environment, 91
- Combining SQL and DQL, 92
- Commands, 14
- comment, 191
- Comments, 19
- commit, 192
- Comparison Operators, 194
- Conditional Statistical Operators, 195, 296
- Constant Value, 197
- Control Commands**, 68
- Control Procedure, 198
- Control Procedures, 68
- copy all from, 199
- Copying Scripts, 23
- Copyright, 10
- cos (cosine), 200
- cosh (hyperbolic cosine), 200
- count, 201
- count of, 203, 204
- Create a Report, 55
- Creating a Data-Entry Form, 47
- Creating a DQL, 23
- current, 205
- Cursor Movement, 37
- Data Access Restrictions, 26
- Data Entry Methods, 51
- Database Objects, 17
- data-entry, 207
- Data-Entry Form, 32
- date, 208
- day, 209
- db status, 210

- Deadlocks, 141
- define, 211
- delete records, 213
- Deleting a DQL, 27
- Detecting Deadlocks, 143
- Displaying a Data-Entry Form, 48
- Displaying Pick Lists, 39
- do, 214
- DQL Basics, 12
- DQL Help, 20
- DQL Script Menu, 38
- DQL Security, 26
- DQL Tech Tips, 99
- DQL Toolbar, 45
- DQL View Menu, 45
- Duplicate Records, 127
- else, 215
- end, 216
- enter a record, 218
- error messages off, 219
- error messages on, 220
- Errors, 64
- exec SQL, 221
- Executing a DQL, 27
- exit, 225
- exp, 226
- export, 227
- firstc, 229
- firstlast, 230
- firstw, 231
- floor, 232
- for, 233
- Formatting DQL Output, 53
- Functions, 16, 236
- futurevalue, 237
- General Operators**, 296
- global, 238
- Grouping, 240
- Grouping/Sorting Operators**, 296
- Help, 20
- highest of, 242
- hours, 243
- if Command, 244
- if Function, 246
- import, 249
- in, 250
- in groups, 251
- in order, 253
- in reverse, 255
- Inner Joins, 132
- input using, 256
- Insert ASCII into DQL, 40
- install application, 260
- installment, 261
- Interactive Pick Lists, 36
- item (Conditional Statistical Operator), 263
- item (Statistical Operator), 262
- Joining Tables, 131
- jointext, 264
- julian, 265
- lastc, 266
- lastfirst, 267
- lastw, 268
- length, 269
- list records, 60, 270
- Loading a DQL, 27
- lock, 272
- lock db (lock database), 274
- log, 275
- log10, 275
- lower, 276
- lowest of, 277
- many-to-one**, 55
- max**, 278
- mean, 279
- mean of, 280
- message, 281
- Message Command, 282
- midc, 285
- midw, 286
- min, 287
- minutes, 288
- mod (modulus), 289
- Modify Records, 290

- month, 291
- Multiple Printouts, 120
- Multiple Selection Criteria, 104
- named, 292
- Nested Actions, 294
- not, 295
- one-to-many**, 55
- Operators, 15, 296
- or, 297
- others, 298
- output, 299
- Page Footer, 31
- Page Header, 31
- percent, 300
- periods, 301
- power, 302
- presentvalue, 303
- Preventing Deadlocks, 146
- Primary Table, 304
- Printing a Procedure Definition, 34
- Procedural Commands**, 68, 305
- Procedure Layout, 65
- Processing Commands**, 68
- Processing Procedure, 306
- proper, 307
- query selection, 308
- random, 310
- rate, 311
- record entry, 312
- Relational Operators, 62, 296
- Relational Statistical Operators**, 296, 313
- Relationships, 17, 314
- reorganize, 315
- Resizable Panels, 36
- restore db, 316
- rollback, 317
- run procedure, 319
- Saving, 67
- Saving a Data-Entry Form, 52
- Saving a DQL, 24
- script**, 21
- Script, 29
- Script Editor, 23
- Script Preferences Dialog, 41
- Search and Replace, 43
- Search Dialog Options, 42
- Searching for a Text String, 42
- Secondary table, 320
- seconds, 321
- Security, 26
- Selecting Records, 57
- Selection Criteria, 58, 322
- sin, 323
- sinh, 323
- Sorting, 324
- Sorting and Grouping, 60
- Specifying Selection Criteria, 58
- spellcurrency, 325
- spelldate, 326
- spellmonth, 327
- spellnumber, 328
- spellweekday, 329
- SQL Tech Tips, 129
- SQL User Permissions, 98
- sqrt (square root), 330
- Start a New DQL, 22
- Statistical Operators, 331
- std.dev. (standard deviation), 332
- std.err. (standard error), 333
- Stored Procedures, 138
- sum, 334
- sum of, 335
- Summary Footer, 32
- Summary Header, 32
- Symbols, 18, 151
- tan (tangent), 336
- tanh (hyperbolic tangent), 336
- temp, 337
- textpos, 338
- timeampm, 339
- Trademark, 10
- tran off, 340
- tran on, 341

Transaction Processing, 91, 342
Typographical Conventions, 13
unlock, 343
unlock db, 345
upper, 346
Using the Script Editor, 35
value, 347
Values, 18
variable, 348

Variables, 18
variance, 349
weekday, 350
while, 351
with, 353
year, 354
yearday, 355
yearweek, 356