Cognos(R)
Application Development Tools
PowerHouse(R) 4GL

VERSION 8.4E

QTP REFERENCE

COGNOS®

THE NEXT LEVEL OF PERFORMANCE™

Product Information

This document applies to PowerHouse(R) 4GL Version 8.4E and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos support Web site (http://support.cognos.com).

Copyright

Table of Contents

About this Book

Overview

This book is intended for experienced Cognos PowerHouse users who require a concise summary of QTP statements.

Chapter 1, "Introducing QTP", introduces QTP and the other Powerhouse components and utilities.

Chapter 2, "Processing Phases of QTP", provides information about the sequence of events in the phases of QTP processing.

Chapter 3, "QTP Statements", provides concise summaries and detailed information about QTP statements. Syntax summaries, detailed syntax discussions, and examples are provided for each QTP statement, where applicable.

Chapter 4, "QTP Tracer", describes QTP's debug tracing features. Information is provided about Tracer syntax and options as well as examples of Tracer output.

Conventions in this Book

This book is for use with MPE/iX, OpenVMS, UNIX, and Windows operating systems. Any differences in procedures, commands, or examples are clearly labeled.

In this book, words shown in uppercase type are keywords (for example, SAVE). Words shown in lowercase type are general terms that describe what you should enter (for example, filespec). When you enter code, however, you may use uppercase, lowercase, or mixed case type.

Getting Help

For more information about using this product or for technical assistance, visit the Cognos Global Customer Services Web site (http://support.cognos.com). This site provides product information, services, user forums, and a knowledge base of documentation and multimedia materials. To create a case, contact a support person, or provide feedback, click the **Contact Us** link at the bottom of the page. To create a Web account, click the **Web Login & Contacts** link. For information about education and training, click the **Training** link.

Cognos PowerHouse 4GL Documentation Set

PowerHouse 4GL documentation includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Install PowerHouse 4GL	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* book. This document provides step-by-step instructions on installing and licensing PowerHouse 4GL. Available in the release package or from the following website: http://support.cognos.com

Objective	Document
Review changes and new features	*Cognos PowerHouse 4GL & PowerHouse Web Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.cognos.com
Get an introduction to PowerHouse 4GL	*Cognos PowerHouse 4GL Primer*. This document provides an overview of the PowerHouse language and a hands-on demonstration of how to use PowerHouse. Available from the PowerHouse 4GL documentation CD or from the following website: http://powerhouse.cognos.com
Get detailed reference information for PowerHouse 4GL	Cognos PowerHouse 4GL Reference documents. They provide detailed information about PowerHouse rules and each PowerHouse component. The documents are • *Cognos PowerHouse 4GL PowerHouse Rules* • *Cognos PowerHouse 4GL PDL and Utilities Reference* • *Cognos PowerHouse 4GL PHD Reference* • *Cognos PowerHouse 4GL PowerHouse and Relational Databases* • *Cognos PowerHouse 4GL QDESIGN Reference* • *Cognos PowerHouse 4GL QUIZ Reference* • *Cognos PowerHouse 4GL QTP Reference* Available from the PowerHouse 4GL documentation CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

Cognos PowerHouse Web Documentation Set

PowerHouse Web documentation includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Start using PowerHouse Web	*Cognos PowerHouse Web Planning and Configuration book*. This document introduces PowerHouse Web, provides planning information and explains how to configure the PowerHouse Web components. **Important:** This document should be the starting point for all PowerHouse Web users. Also available from the PowerHouse Web Administrator CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

Objective	Document
Install PowerHouse Web	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* book. This document provides step-by-step instructions on installing and licensing PowerHouse Web. Available in the release package or from the following website: http://support.cognos.com
Review changes and new features	*Cognos PowerHouse 4GL & PowerHouse Web Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.cognos.com
Get detailed information for developing PowerHouse Web applications	*Cognos PowerHouse Web Developer's Guide*. This document provides detailed reference material for application developers. Available from the Administrator CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com
Administer PowerHouse Web	The *PowerHouse Web Administrator Online Help*. This online resource provides detailed reference material to help you during PowerHouse Web configuration. Available from within the PowerHouse Web Administrator.

Cognos Axiant 4GL Documentation Set

Axiant 4GL documentation includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Install Axiant 4GL	*Cognos Axiant 4GL Web Getting Started* book. This document provides step-by-step instructions on installing and licensing Axiant 4GL. Available in the release package or from the following website: http://support.cognos.com
Review changes and new features	*Cognos Axiant 4GL Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.cognos.com
Get an introduction to Axiant 4GL	*A Guided Tour of Axiant 4GL*. This document contains hands-on tutorials that introduce the Axiant 4GL migration process and screen customization. Available from the Axiant 4GL CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

Objective	Document
Get detailed reference information on Axiant 4GL	*Axiant 4GL Online Help*. This online resource is a detailed reference guide to Axiant 4GL. Available from within Axiant 4GL or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

For More Information

For information on the supported environments for your specific platform, as well as last-minute product information or corrections to the documentation, see the *Release and Install Notes*.

Chapter 1: Introducing QTP

Overview

This chapter introduces QTP, the PowerHouse transaction processor. It also provides overview information about other PowerHouse components and utilities.

About PowerHouse

PowerHouse 4GL is an application development environment that allows you to create business applications quickly and easily.

Components

PowerHouse 4GL is divided into the following separate, yet integrated components:

PowerHouse Dictionary

The PowerHouse dictionary is the foundation of PowerHouse applications. As the backbone of all PowerHouse systems, the PowerHouse dictionary stores definitions of the data used by your PowerHouse applications.

There are two dictionary types—PDC and PHD. PDC dictionaries exist as a single file and have a .pdc extension (**OpenVMS, UNIX, Windows**) or file code 655 (**MPE/iX**). PHD dictionaries exist as five indexed files and have a .phd extension. PHD dictionaries are OpenVMS-specific.

For more information about the PHD dictionary, see the reference manuals, *PHD Reference and PowerHouse Rules*. See also the section, "PowerHouse Dictionary on OpenVMS", in Chapter 1, "Introducing the PowerHouse Dictionary", in the *PDL Reference* book.

PDL

The PowerHouse Definition Language (PDL) allows you to create and maintain a PowerHouse dictionary.

PDL source code can be compiled in either the PDL or PHDPDL (**OpenVMS**) compiler.

PDL Compiler

PDL compiler is the component that compiles PDL source statements to a PowerHouse dictionary. Dictionaries generated with the PDL compiler have a .pdc extension (**OpenVMS, UNIX, Windows**) or file code 655 (**MPE/iX**).

PHDPDL Compiler (OpenVMS)

PHDPDL is an OpenVMS-specific component that compiles PDL source statements to a PowerHouse dictionary. Dictionaries generated with PHDPDL have a .phd extension.

PHD Screen System (OpenVMS)

PHD is a screen interface to PHD dictionaries. You can initiate PHD with the POWERHOUSE or POW command.

For more information about running PHD, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

QDESIGN and QUICK

QUICK is an interactive screen processor with a powerful development tool: QDESIGN. As a screen designer, you use QDESIGN to build data entry and retrieval screen systems. QUICK screens are used by data-entry operators and other end-users to process data quickly or to browse effortlessly through their files.

QUICK includes an interactive debugger that lets you analyze and control QUICK screens as they run.

QUIZ

QUIZ is the PowerHouse report writer. It takes the information you request and gives it a structure. Your information is automatically displayed in columns with headings. The key to the simplicity of QUIZ lies in its relationship with the data dictionary. QUIZ references the rules and standards defined in the data dictionary by the application designer when it formats your report.

QTP

QTP is a high-volume transaction processor. It gives you the power to change the data in your files in one sweep. Because QTP is easy to use and designed for fast, high-volume file updating, it should be used by someone who is familiar with the implications of updating active files.

QTP includes a trace facility that lets you debug QTP requests.

Utilities

PowerHouse also contains the following data dictionary utilities:

QSHOW

QSHOW is the data dictionary reporting program. It allows you to view and obtain cross-reference information about the contents of your PowerHouse dictionaries. It also allows you to generate PDL source for a PowerHouse dictionary.

QUTIL

QUTIL is a utility that creates and deletes non-relational files and databases.

ITOP (MPE/iX)

ITOP is an IMAGE to PDL conversion utility that generates PDL statements directly from an existing IMAGE database.

QCOBLIB (MPE/iX)

QCOBLIB is a utility that generates COBOL definitions from a PDL dictionary.

PHDMAINTENANCE (OpenVMS)

PHDMAINTENANCE creates and manages PHD dictionaries. It is also referred to as PHDMAINT.

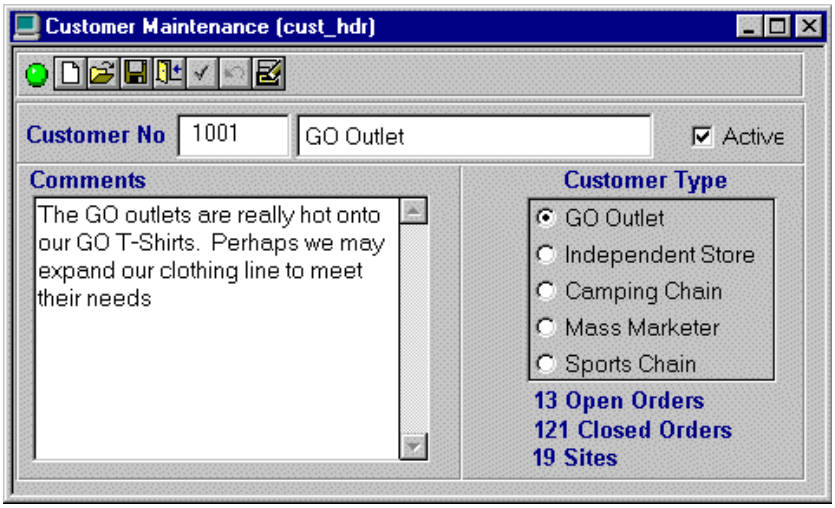
PHDADMIN (OpenVMS)

PHDADMIN is a run-time utility for administering security classes in PHD dictionaries.

PowerHouse-Related Products

Axiant 4GL

Axiant 4GL is a visual Windows-based development environment for creating PowerHouse applications. With Axiant 4GL, you can build applications that can be deployed in a variety of thin-client, fat-client, mobile, stand-alone, and server-only architectures. Axiant 4GL gives PowerHouse a Window-like user interface.



PowerHouse Web

Chapter 2: Processing Phases of QTP

Overview

This chapter provides information about

- the sequence of events in the phases of processing
- the timing of output and buffer initialization
- the effects of statement sequence
- how to test record status

About the Processing Phases

A QTP run is made up of one or more QTP requests that perform a set of related processes. Within a request there can be three phases:

- input
- sort
- output

Statements in the Input, Sort, and Output Phases

Each phase has several statements associated with it. The following example shows the order in which you should enter statements when constructing QTP runs and requests. It is not syntactically correct.

```
RUN name

REQUEST name
  Input statements
    ACCESS, CHOOSE, DEFINE, EDIT,
      SELECT IF, SELECT file IF
  Sort statements
    SORT, SORTED
  Output statements
    ITEM, DEFINE, TEMPORARY, SUBFILE, OUTPUT,
      SELECT file IF
```

Input Phase

In the input phase, QTP builds a set of transactions based on the input files. A transaction is formed by QTP when it is processing records. The transaction comprises a record from the primary record-structure and related records from the subordinate record-structure.

Note: The term "transaction" as used in this chapter refers to QTP transactions, not to relational transactions. That is, it is used in the PowerHouse, and not in the relational database sense.

The input phase of a QTP request lets you create a transaction set from one or more files, record-structures, subfiles, and/or relational tables.

The statements used to create a transaction set are the ACCESS, CHOOSE, DEFINE, EDIT, and SELECT statements.

Statement	Description
ACCESS	Declares • the record-structures that are read • the order in which the record-structures are read • optionally, how the linkages between record-structures are constructed
CHOOSE	Causes the retrieval to be by index value.
DEFINE	Names an expression. The expression is calculated every time the defined item is referenced during execution, or once if it gets its value from a PARM option.
EDIT	Validates items, record items, or values that have been entered in response to a prompt.
SELECT	Applies conditions to retrieval records which must be met for the record to be included in the transaction.

For more information about statements, see Chapter 3, "QTP Statements".

QTP's Intermediate File

Based on request requirements, QTP automatically determines whether the transactions selected during the input phase should be written first to an intermediate file or passed directly to the next processing phase. An intermediate file is always required if:

- a SORT statement has been specified (this doesn't apply to the SORTED statement)
- an OUTPUT statement references a record-structure in the access list
- an EDIT statement references a record item other than an item named in a CHOOSE statement with the PARM option

You may sometimes want to force QTP to use an intermediate file. For example, if an intermediate file isn't used and the process limit is exceeded, a partial update occurs. To force QTP to use an intermediate file, specify the SET SCRATCH statement. The default option, SET SCRATCH AUTO, can be used to reverse a previous SET SCRATCH statement.

MPE/iX, OpenVMS:	QTP uses one of two designated files as intermediate files. The designated file QTPSORT is used if a request contains a sort phase. Otherwise, the designated file QTPSCR is used.
UNIX	If a SORT is present, QTP creates two temporary work files: SIannnnn.tmp and SOannnnn.tmp, where nnnnn is the pid of the QTP process. Otherwise, the designated file QTPSCR is used.
Windows:	If a SORT is present, QTP creates temporary work files with names in the form of a random number having no prefix, suffix or extension. Otherwise, the designated file QTPSCR is used.

Sort Phase

In the sort phase, QTP passes the transaction file to the SORT program.

Many updating situations require that transactions be passed to the output phase in a specific sequence, either to create a sorted sequential output file, or to process related transactions as a group with special calculations and updating activities performed at the end (or beginning) of each group.

The SORT statement declares control breaks in the transaction set.

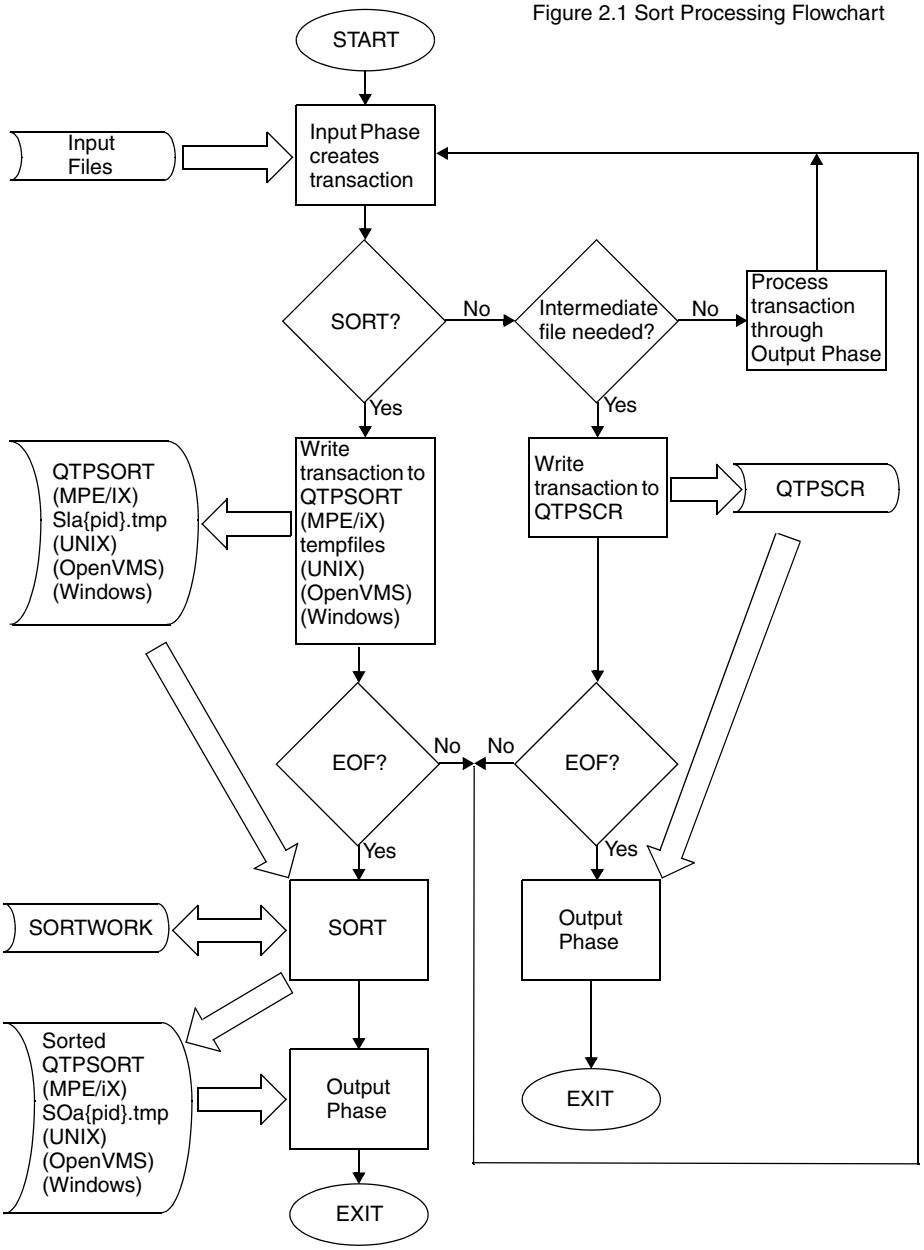
How QTP Sorts

- A transaction is created in the input phase consisting of all items from each file listed in the ACCESS list.
- If there is no SORT statement and an intermediate file is not required, the transaction is passed directly to the output phase.
- If there is a SORT statement, each transaction is passed to the sort utility. QTP passes all record items to the sort utility. Once all transactions have been passed to the sort utility, QTP issues a call to sort the records. The sorted transactions are processed in the output phase.

OpenVMS: The OpenVMS Sort Utility may write out these record complexes to one or more temporary work files in the SYS$SCRATCH directory, depending on the number of records. These are known as SORTWORK0 through SORTWORK9. When disk space is limited, or a large volume of data is being sorted, these files can be redirected to disks with available space by using the logical names, SYS$SCRATCH, SORTWORK0 and SORTWORK1. PowerHouse uses the default of two work files, consequently the logical names SORTWORK2 through SORTWORK9 are ignored by the OpenVMS Sort Utility.

The Sort Processing Flowchart provides a graphical representation of sort processing.

Figure 2.1 Sort Processing Flowchart



Output Phase

In the output phase, QTP reads through the sorted transaction set and performs initialization, processing, and output actions to update the output files.

The main statements used in the output phase to update files are the COMMIT AT, DEFINE, ITEM, OUTPUT, SELECT, and SUBFILE statements.

Statement	Description
COMMIT AT	Specifies when relational transactions involving relational databases are committed. If no COMMIT AT statement is specified, all relational transactions are committed according to the file lock duration.
DEFINE	Assigns a name to an expression or prompts for values at execution time. Evaluated when referenced.

Statement	Description
ITEM	Assigns values to RECORD, TEMPORARY, and GLOBAL TEMPORARY items. Options on the ITEM statement specify the initialization, calculation, and assignment of values.
OUTPUT	The primary statement controlling the output phase. It declares a record-structure as an output record-structure and specifies the output actions to be performed. These include the ADD, DELETE, UPDATE, and ADD UPDATE (or UPDATE ADD) activities.
SELECT	Applies selection conditions to records and transactions. The output phase uses only SELECT record-structure IF, which must be positioned after the OUTPUT statement.
SUBFILE	Tells QTP to create a self-describing file as an output file. The presence or absence of an AT or IF option on the SUBFILE statement determines when output to the subfile occurs.

Sequence of Events in the Output Phase

The output phase begins with a preliminary initialization step, followed by a double-loop structure and a final processing step.

In the preliminary initialization, all records and temporary items are initialized to default values, and all work areas are set up and cleared. Subfile buffers are initialized to binary zeros. Values in the buffers are subsequently changed only by the processing of FINAL options of ITEM statements for the subfile items. The double-loop structure consists of an outer loop for control break processing, and an inner loop for detail processing.

The logic contains sections that can carry out every available option. However, a QTP request may make use of only a few of the options. Therefore, to tailor the logic to the specifications in the source file, QTP builds internal tables. At parse time, QTP reads the source statements, decides what processing must be done at each point in the logic (based on the level of control break), and saves a record of those decisions in internal tables. In the output phase, the information in the internal tables directs the execution of the logic to perform processing appropriate to each level of control break.

For example, if a QTP request contains an OUTPUT statement with the AT sort-item option, the internal table contains a record for processing at that control break.

Timing of Output

In Figure 2.2 on (p. 24) there are five points at which output actions (ADD, DELETE, or UPDATE) can occur. The timing of output actions is controlled by the presence or absence of AT options in the OUTPUT and SUBFILE statements.

Timing option	When output occurs
AT INITIAL	at the beginning of the transaction set, before processing any transactions
AT START OF	at the beginning of a control group, before processing any transactions in the group
no AT option	for each transaction
AT "end of"	at the end of a control group, after processing all transactions in the group
AT FINAL	at the end of the transaction set, after processing all transactions

When QTP arrives at one of the five points where output actions can occur, it loops through the actions several times, first doing initializations, then subtotals and final values and updates that occur at that point. Each time QTP processes through the actions, the specified options determine the exact events.

AT START OF and AT "End Of" Initialization

If output actions (ADD, DELETE, or UPDATE) occur at the beginning of a control break (AT START OF) or at the end of a control break (AT), QTP initializes the buffers for this processing before beginning to process individual transactions.

AT START OF Processing, Output, and Commits

If output actions (ADD, DELETE, or UPDATE) or commits of relational transactions occur at the beginning of a control break (AT START OF), QTP performs all such processing and output followed by the commits, before beginning to process individual transactions.

The initialization, processing, and output for control breaks happen for each active control break in the request, from major to minor.

Detail (Transaction-Level) Initialization, Processing, Output and Commits

Once QTP handles all AT START OF output for the current control break, it

- initializes the record buffer for detail (transaction-level) processing and output
- performs all detail (transaction-level) processing, output, and lastly, commits of relational transactions
- reads the next transaction and determines whether or not a control break occurs (in which case, AT "end of" processing occurs) or whether to continue processing at the detail level

AT "End Of" Processing, Output, and Commits

If the last transaction read signals a control break, QTP performs all AT sort-item processing, output actions and lastly, commits of relational transactions.

If there are more transactions in the transaction set, a control break has been encountered. QTP repeats all the steps in the main loop.

If there are no more transactions in the transaction set, QTP proceeds to AT FINAL processing and output.

Control Breaks

How QTP Sets Control Breaks

As QTP reads through the transaction set, output phase processing occurs when a control break match is detected. A control break match exists when QTP arrives at a point in the transaction set that matches one of the timing options specified in the QTP source file. A particular control break is when a value in an item declared in a SORT or SORTED statement changes. If a break occurs and it matches the timing option, then there is a match. See Figure 2.3 on (p. 25).

All timing options don't refer to control breaks, so a control break match can exist at times other than control breaks. For example, a control break match for AT INITIAL processing exists at the beginning of a transaction set; a control break match for AT FINAL processing exists at the end of a transaction set. A control break match for detail processing occurs at each transaction.

Processing at Control Breaks

If QTP detects a control break at any level, a control break match exists for that level and for all lower-level control breaks. When QTP performs AT START OF processing and output for multiple control breaks, the highest-level active control break is processed first, followed by successively lower levels. When QTP performs AT "end of" processing and output for multiple control breaks, the lowest-level control break is processed first followed by successively higher levels.

Special Considerations for IMAGE Manual Masters (MPE/iX)

IMAGE requires an IMAGE manual master record to be on file before associated IMAGE detail records can be placed on file. When adding IMAGE manual master records at a control break, QTP places an initial record on file at the start of the break so that detail records that need the master record can be added at lower-level breaks or at the transaction level. The master record is then updated at the end of the break.

The index value must be specified using an INITIAL option on the ITEM statement unless automatic item initialization occurs. FINAL options are only performed when the record is updated at the end of the group.

When changing index or sort-item values of existing records, QTP automatically handles the deletion of the old and addition of the new record for IMAGE detail files. This is not done for IMAGE manual master files. In order to change the index value of a IMAGE manual master file, two OUTPUT statements are required; one to delete the old record and one to add the new record. This also holds for IMAGE with Critical Item Update. For example:

```
> OUTPUT CUSTOMER-MASTER DELETE
> OUTPUT CUSTOMER-MASTER ADD ALIAS NEW-MAST &
>  INITIALIZE FROM CUSTOMER-MASTER
> ITEM CUSTOMER-ID OF NEW-MAST INITIAL ALTKEYVAL
```

The INITIALIZE option directs QTP to initialize the new record from the old one which is to be deleted. The ITEM statement specifies the new index value.

If index values are being changed for IMAGE master and detail records, the master record must be deleted and added at a control break (the index of the master file), as in

```
> ACCESS MASTER LINK TO DETAIL
> CHOOSE IDX-ITEM PARM PROMPT 1 TIME
> DEFINE NEWINDEX =PARM
> EDIT NEWINDEX LOOKUP NOTON MASTER
> SORTED ON IDX-ITEM OF MASTER
> OUTPUT DETAIL UPDATE
> ITEM IDX-ITEM FINAL NEWINDEX
> OUTPUT MASTER DELETE AT IDX-ITEM
> OUTPUT MASTER ADD AT IDX-ITEM ALIAS M-ADD &
>   INITIALIZE FROM MASTER
>   ITEM IDX-ITEM OF M-ADD INITIAL NEWINDEX
```

Output at the Start of a Request

QTP reads the first transaction into the input buffer and moves it to the work area. If there are no transactions in the transaction set, the request terminates.

Before QTP begins processing for output at control breaks or individual transactions, it performs the following steps:

- preliminary initialization
- AT INITIAL and AT FINAL initialization
- AT INITIAL processing and output

Preliminary Initialization

All record buffers are initialized to blanks, zeros, or default values defined in the dictionary. Global temporary items are not touched. Redefined items are reinitialized in the order in which they're encountered. For a series of redefinitions, the last redefinition becomes the default. QTP sets up and clears all work areas, and initializes subfile buffers to binary zeros.

AT INITIAL and AT FINAL Initialization

The OUTPUT statements and ITEM statements for global temporary and temporary items control initialization in the output phase. Initialization associated with ITEM statements for record items occurs only when the corresponding OUTPUT statement is processed.

If output actions (ADD, DELETE, or UPDATE) occur at the beginning of the request (AT INITIAL) and/or at the end of the request (AT FINAL), QTP initializes the buffers for this processing before beginning to process individual control breaks or transactions.

AT INITIAL Processing and Output

See Figure 2.4 on (p. 26) for a graphic representation of the AT INITIAL process and the output phase.

See Figure 2.5 on (p. 27) for a graphic representation of processing of the = option.

If output actions (ADD, DELETE, or UPDATE) or commits of transactions occur at the beginning of the request (AT INITIAL), QTP performs all such processing, output and lastly commits before beginning to process individual control breaks or transactions.

The OUTPUT, SUBFILE, and ITEM statements for global temporary and temporary items control processing in the output phase. Processing associated with ITEM statements for record items occurs only when the corresponding OUTPUT or SUBFILE statement is processed.

Processing for the OUTPUT statement begins with the processing of the = option and summary operations of ITEM statements for the record-structure. These are processed in the sequence in which the ITEM statements appear in the request. These ITEM statements are processed regardless of any conditions associated with the OUTPUT statement. ITEM statements with an = option or summary operations are only processed if there's a control break match. In addition, if an ITEM statement contains an IF option, the statement is only processed if the specified condition is true. Figure 2.5 on (p. 27) illustrates this logic.

After processing the = option and summary operations of the ITEM statements for the record-structure, no further processing of the OUTPUT statement occurs unless there's a control break match. If there's a match, any IF condition of the OUTPUT statement is tested. If the condition is satisfied, or if there's no IF option, processing of the OUTPUT statement continues. At this point, any record-retrieval errors detected during initialization are signaled as file-output errors and processing stops. If there are no errors, the FINAL options of the ITEM statements for the record-structure are processed. The output action is then performed. If specified, the commits of transactions are then done.

Processing for the SUBFILE statement is similar to OUTPUT statement processing, with one exception. Because the = option and summary operations are not valid in ITEM statements for subfile items, that step is bypassed.

For an ITEM statement for a global temporary or temporary item, any = option or summary operation is processed.

Initialization associated with the OUTPUT statement proceeds as follows:

- QTP checks whether a control break match exists in view of the last transaction that was read. A control break match exists when QTP arrives at a point in the transaction set that matches one of the timing options specified in the QTP source file.
- If there's no AT option in the OUTPUT statement, a control break match for that statement occurs after each transaction is read. If a control break match exists, the output action specified in the OUTPUT statement is checked.
- If a control break match doesn't exist and the output action in the OUTPUT statement is UPDATE or UPDATE ADD, any RESET options on the ITEM statement for the record-structure are processed.

If the action specified in the OUTPUT statement is UPDATE, DELETE, ADD UPDATE, or UPDATE ADD, QTP tries to retrieve a data record from the output record-structure that corresponds to the transaction.

If the output action is either UPDATE or DELETE, and if data record retrieval fails, an error is flagged. (The error is actually signaled only when the output action is executed as part of processing. This method takes into account the possibility of conditional output.)

If the output action is either ADD UPDATE or UPDATE ADD and data record retrieval fails, or if no data record retrieval is attempted because the output action is ADD, then the record buffer is initialized to zeros, spaces, and data dictionary initial values. The INITIAL and RESET options in the ITEM statements for the record-structure are then processed.

INITIAL and RESET options in the ITEM statements for global temporary and temporary items are processed. For ITEM statements corresponding to global temporary and temporary items, the INITIAL option and the RESET AT INITIAL option are equivalent.

Initialization caused by ITEM statements for record items occurs as part of initialization for the corresponding OUTPUT statement. When processing the INITIAL and RESET options for record items, all ITEM statements for record items in the file are processed in the sequence in which they're encountered. For record items, the RESET and RESET AT options are performed regardless of the output action. The NORESET option prevents item reinitialization.

If the RESET option specifies a value, that is, the TO option is used, and NORESET is also used, the value is set for the first transaction but not subsequently. For example, if RESET TO 5 NORESET is specified, the item is set to the value 5 for the first transaction but is not subsequently reinitialized.

Output for Control Breaks and Individual Transactions

Once QTP has finished its preliminary initialization, it enters a main loop in which it handles initialization, processing, and output for all control breaks. Within this main loop is a subordinate loop that handles initialization, processing, and output for individual transactions.

Output at the End of a Request

After QTP has finished processing for control breaks and individual transactions, it performs the following steps:

- AT FINAL processing and output
- process FINAL options on ITEM statements for GLOBAL TEMPORARY items

AT FINAL Processing, Output, and Commits

If output actions (ADD, DELETE, or UPDATE) or commits of relational transactions occur at the end of the request (AT FINAL), QTP processes them and performs the actions only after processing all the transactions in the transaction set. The commits are the last action taken.

Processing FINAL Options on ITEM Statements for GLOBAL TEMPORARY Items

If the request contains ITEM statements that reference global temporary items, QTP processes the ITEM statements only at the end of the request. These ITEM statements aren't performed if the request isn't executed (due to conditional execution), nor are they performed if no transactions are processed in the output phase. To be performed, the request must be executed and at least one transaction must be read in the output phase.

Effects of Statement Sequence

The processing sequence must be considered when you mix item statements for global temporary and temporary items with item statements for record items, as in

```
.
.
.
> temporary t1
> temporary t2
>   item t1 = t1 + 1
> output testfile update
>   item t2 = t2 + 2
>   item tot1 of testfile final t1
>   item tot2 of testfile = t2
.
.
.
```

Figure 2.2 on (p. 24) shows that, given this source file, the processing sequence is as follows:

```
ITEM T1 = T1 + 1               process temporary item
ITEM TOT2 OF TESTFILE = T2     process = option
ITEM TOT1 OF TESTFILE FINAL T1 process final option
```

```
OUTPUT TESTFILE UPDATE          output action
ITEM T2 = T2 + 2                process temporary item
```

The record-structure TESTFILE is updated before the ITEM statement for item T2 is evaluated. This means that the value of item TOT2 written to the TESTFILE record-structure is the value of item T2 prior to the execution of the ITEM statement for item T2. If the ITEM statement for item T2 is placed before the OUTPUT statement, T2 will be processed before the ITEM statements for the OUTPUT statement, then the value of item TOT2 written to the TESTFILE record-structure is the value of item T2 after the execution of the ITEM statement for item T2.

When performing calculations and accumulations it is easy to use temporary items over which you have complete control. The value of the temporary items should be moved to the record using the FINAL option. (In the previous example, the = option is only used with the item TOT2 to illustrate the processing sequence.)

Testing Record Status

To determine what's happened in the output phase, you can test the status of the output record-structure record buffer. (You can test record status in the input phase, but its usefulness is limited.) The predefined conditions alteredrecord, deletedrecord, and newrecord can be used, as in this example:

```
> access salesmaster
> output customermaster update
>   item currentbalance final &
>     (currentbalance + salestotal)
>   item datelastmod final sysdate &
>     if alteredrecord of customermaster
> go
```

Consider that the item salestotal in the input record-structure salesmaster has a value other than zero. This means that:

- When it's added to the value item CURRENTBALANCE of the output record-structure, the content of the output-file record buffer changes and the record status is set to "changed". The predefined condition, ALTEREDRECORD, becomes true.
- The output record-structure is updated only if the values in the record change, so if the predefined condition, ALTEREDRECORD, becomes true, an update occurs.
- The last ITEM statement tests the predefined condition, ALTEREDRECORD. If true, and an update is to be performed, then the statement changes the value of item DATELASTMOD. The change in the date's value and the update output action are both caused by a change in some other part of the record.
- Without the test on ALTEREDRECORD, the date could change even if the value of CURRENTBALANCE doesn't change.

This table shows the possible record status settings for the add, delete, and update output actions at each stage of processing.

Timing	ADD	UPDATE	DELETE
immediately after buffer initialization	new, unchanged, undeleted	new, unchanged, undeleted	new, unchanged, undeleted
immediately after successful record retrieval	n/a	old, unchanged, undeleted	old, unchanged, undeleted
immediately after unsuccessful record retrieval	n/a	new, unchanged, undeleted	new, unchanged, undeleted

Timing	ADD	UPDATE	DELETE
after processing ITEM statements	new, changed, undeleted	old, changed, undeleted	n/a
after output action	new, changed, undeleted	old, changed, undeleted	old, unchanged, deleted

Note: The following record status combinations don't appear in the table, and never occur in QTP:

- new, changed, deleted
- new, unchanged, deleted
- old, changed, deleted

Therefore, deleted can only appear with old, unchanged.

When the record buffer is initialized to spaces, zeros, and data dictionary initial values, the record status is set to new, unchanged, undeleted (that is, the predefined condition, NEWRECORD, is true, and ALTEREDRECORD and DELETEDRECORD are false).

Any ITEM statement that causes a change in the content of the record buffer, including automatic item initialization, causes QTP to set the record status to changed (the predefined condition, ALTEREDRECORD, is true).

For the DELETE and UPDATE output actions, QTP tries to retrieve the relevant data record from the output record-structure. If the retrieval is successful, QTP sets the record status to old, unchanged, undeleted (the predefined condition, NOT NEWRECORD, is true). If the retrieval is unsuccessful, record status remains new, unchanged, undeleted.

After an ADD output action, record status is set to new (the predefined condition NEWRECORD is true). After a DELETE output action, QTP sets the record status to deleted (the predefined condition, DELETEDRECORD, is true). In the case of the UPDATE output action, only changed records are updated and the output action has no effect on record status.

Record status is not reset to new, unchanged, undeleted until the record buffer is reinitialized following output. The timing of reinitialization depends on whether output occurs at detail transaction time or at the beginning or end of a control group.

Record Status of Subfiles

The record status of a subfile can be tested. The timing of status resetting for a subfile depends on the timing of output to it. This is determined by the presence or the absence of an AT option in the SUBFILE statement. ITEM statements with the FINAL option are processed and cause the record status to be set to changed. Other options in ITEM statements for the subfile are invalid. Without ITEM statements, the status is always NEW, UNCHANGED, UNDELETED. Subfiles are always NEW. Record status settings for subfiles declared in an OUTPUT statement are set in the same manner as any other file.

Flowcharts

The following flowcharts provide a graphical representation of QTP processing:

- Figure 2.1 Sort Processing Flowchart. See (p. 16).
- Figure 2.2 Output Phase Flowchart. See (p. 24).
- Figure 2.3 Initialization Flowchart. See (p. 25).
- Figure 2.4 AT INITIAL Process and Output Flowchart. See (p. 26).
- Figure 2.5 Processing = option Flowchart. See (p. 27).

Figure 2.2 Output Phase Flowchart

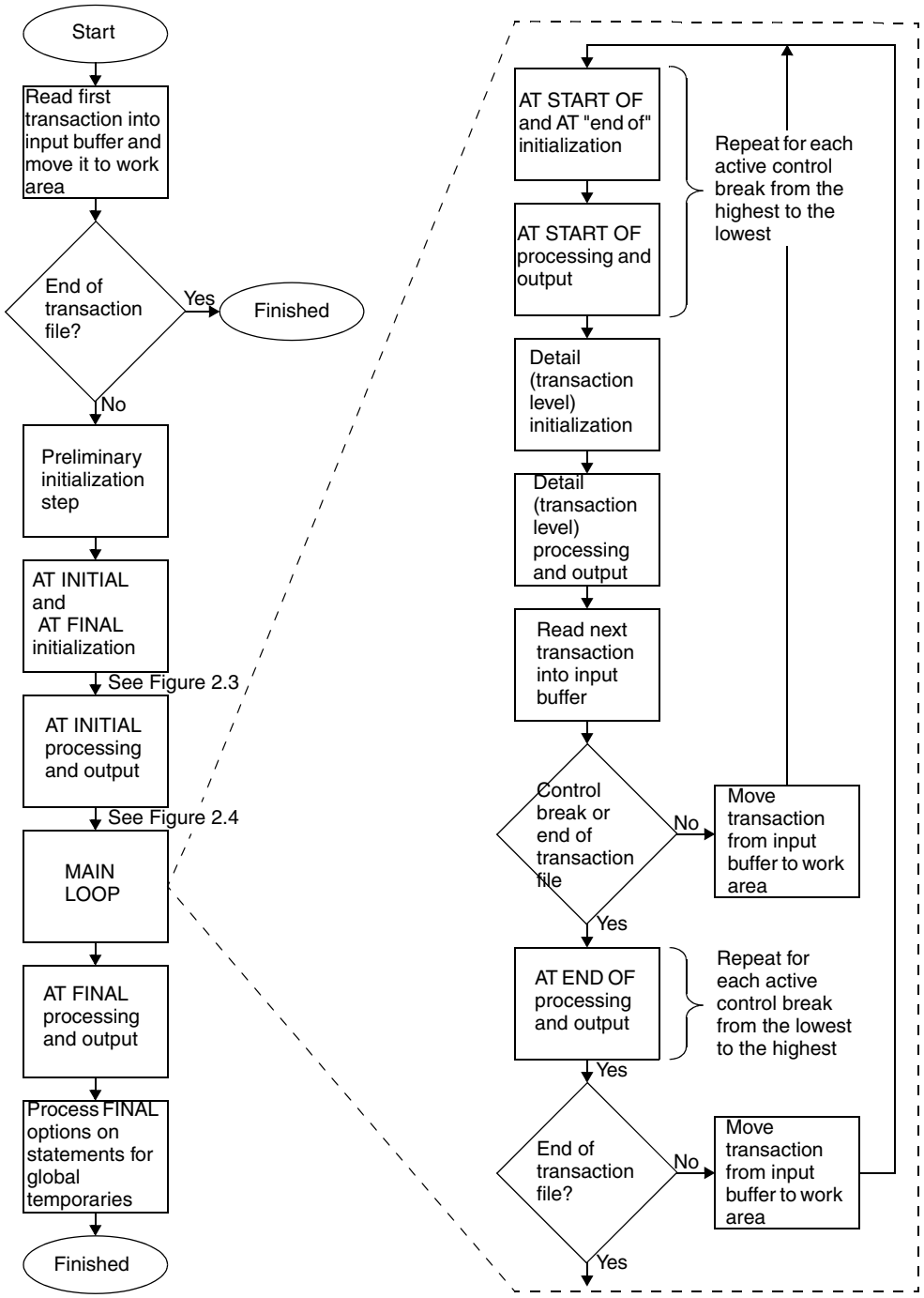


Figure 2.3 Initialization Flowchart

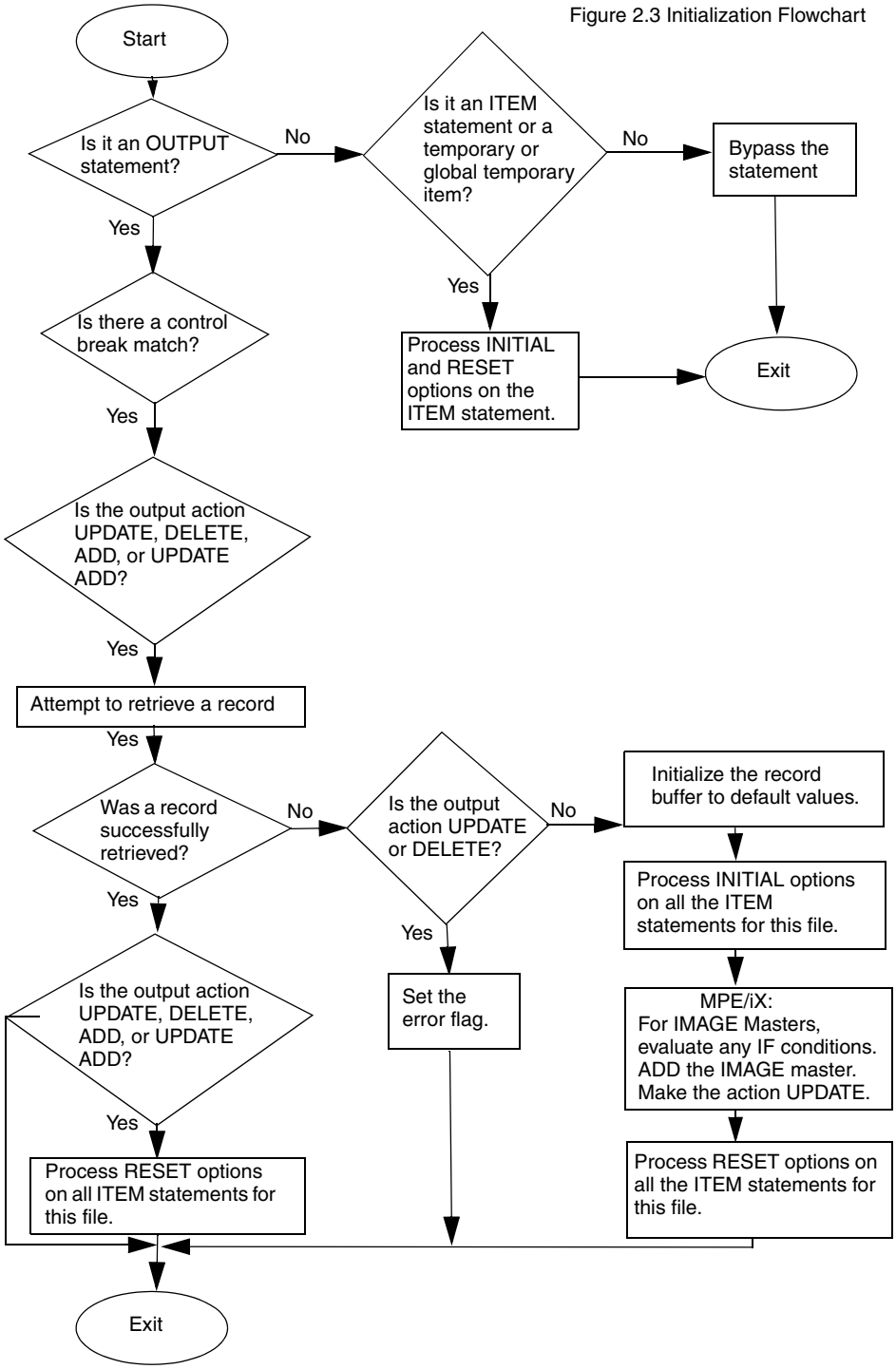


Figure 2.4 AT INITIAL Process and Output Flowchart

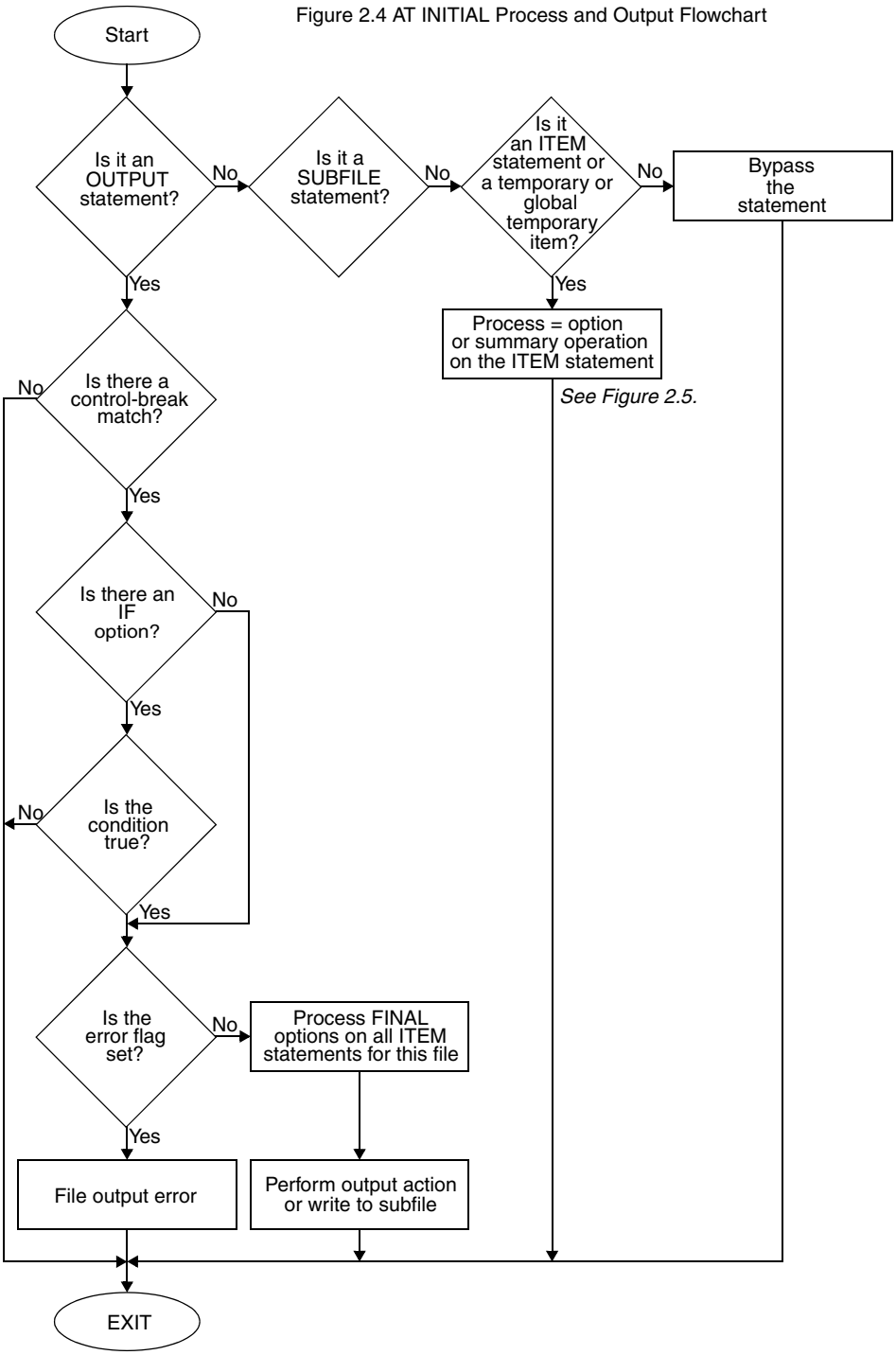
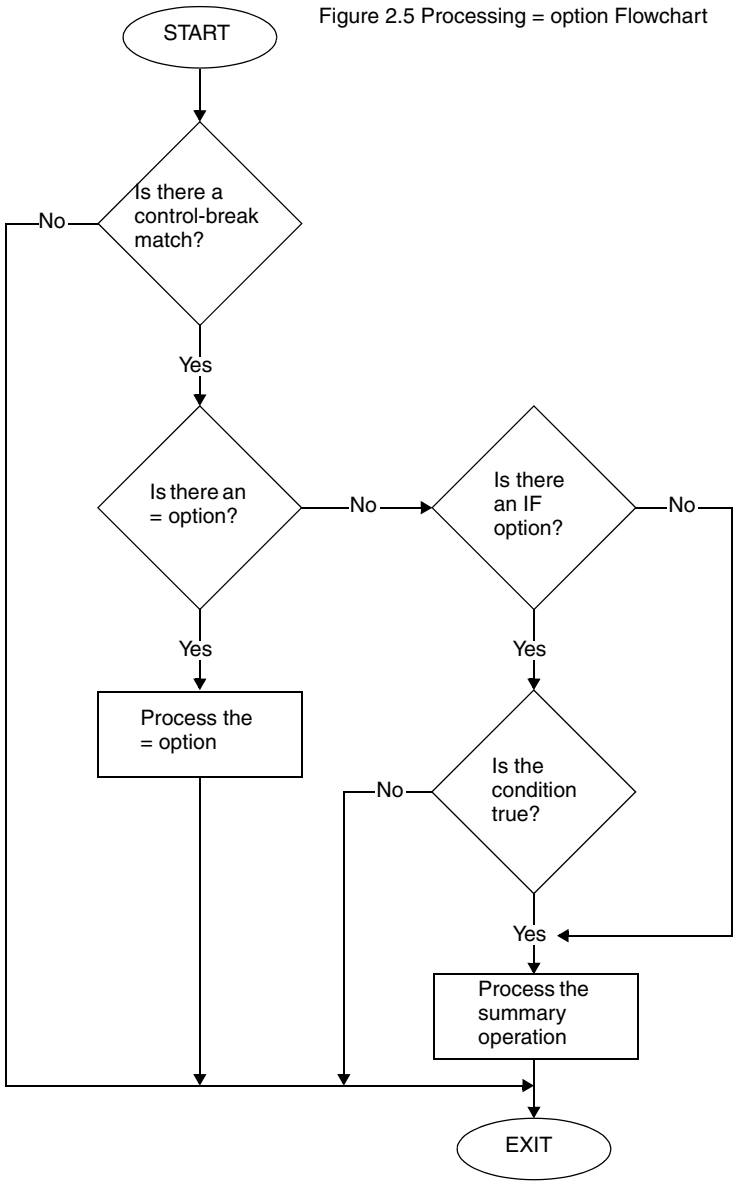


Figure 2.5 Processing = option Flowchart



Chapter 3: QTP Statements

Overview

This chapter describes each QTP statement in detail. For each statement, you'll find

- detailed syntax descriptions
- detailed statement discussions
- examples

Summary of QTP Statements

Statement	Purpose
ACCESS	Specifies the input record-structures and their logical relationships.
BUILD	Compiles and saves a QTP run.
[SQL] CALL	Calls a stored procedure from the specified database.
CANCEL	Cancels the current QTP run specifications.
CHOOSE	Extracts data from an indexed file, relational table or view, or SQL cursor by item value.
COMMIT AT	Specifies when commits are done for transactions involving relational databases.
[SQL] DECLARE CURSOR (query-specification)	Defines a set of data as a run-time view.
[SQL] DECLARE CURSOR (stored procedure)	Calls a stored procedure in an Oracle or Sybase database.
DEFINE	Assigns a name to an expression or prompts for values at execution-time.
[SQL] DELETE	Removes rows in a table in a database.
DISPLAY	Displays a message.
EDIT	Validates record items, or entries made in response to a PARM prompt.
EXECUTE	Executes a compiled QTP run.
EXIT	Ends a QTP session.
GLOBAL TEMPORARY	Creates a global temporary item for the duration of the run that does not exist in the data dictionary.

Statement	Purpose
GO	Initiates execution of a QTP run.
[SQL] INSERT	Adds new rows to a table.
ITEM	Assigns values to record items, global temporary items, or temporary items.
OUTPUT	Defines output record-structures and output actions.
QSHOW	Runs QSHOW from QTP.
query-specification (SELECT)	Defines a collection of rows that will be accessible when the cursor is opened.
QUIT	Ends a QTP session.
REQUEST	Initiates a QTP request.
REVISE	Edits the current temporary save file or a specified file.
RUN	Initiates a QTP run.
SAVE	Saves the current QTP source statements in a file.
SELECT	Applies selection conditions to records and transactions.
SET	Overrides default QTP settings.
SET LOCK	Overrides default QTP settings.
SHOW	Displays all record-structures or items.
SORT	Sorts transactions in a desired sequence and defines control breaks.
SORTED	Defines control breaks for records known to be in sorted order.
SUBFILE	Directs output to a subfile.
TEMPORARY	Creates a temporary item that does not exist in the data dictionary.
[SQL] UPDATE	Updates rows in a table.
USE	Processes QTP source statements that are contained in a file.

ACCESS

Specifies the input data structures and their logical relationships.

Syntax

ACCESS primary-data-structure [ALIAS name]
[LINK {direct-linkage|indexed-linkage|TO cursor
[sql-substitution...]} [ALIAS name] [OPTIONAL]
[{AND|LINK}
{direct-linkage|indexed-linkage|TO cursor
[sql-substitution...]} [ALIAS name] [OPTIONAL]]...]

primary-data-structure

Name of the primary data structure, which can be:

- a cursor defined in a DECLARE CURSOR statement
- a record-structure named in the dictionary
- a subfile
- a table or view defined in a relational database

cursor [sql-substitution...]

A cursor is the name of a set of data defined by the PowerHouse SQL DECLARE CURSOR statement.

An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default sql-substitutions, WHERE and ORDERBY, will be inserted in generated SQL statements even if the corresponding substitution-variables do not exist on a DECLARE CURSOR statement.

The syntax for an sql-substitution is:

substitution-variable (text)

For more information about substitutions, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

Limit: Any sql-substitutions must appear before any other options.

record-structure [IN file]

The name of a record-structure and, optionally, the name of the file to which it belongs. Both must be declared in the data dictionary. A file name adds clarity if the file name differs from the record-structure name, as in coded record-structures.

*subfilespec

Specifies the name of an existing subfile, prefixed by an asterisk (*).

Subfilespec may be a file equation name (**MPE/iX**), a logical name (**OpenVMS**), or an environment variable name (**UNIX, Windows**).

For more information about subfiles, see (p. 149).

table [IN database]

Names the table or view in a relational database and, optionally, the name of the database to which it belongs. The database must be attached to the current data dictionary.

If an IN database qualifier is not used, QTP first assumes that the primary record is a record-structure defined in the data dictionary, or a cursor defined in a DECLARE CURSOR statement.

If the record-structure exists, QTP uses the first file containing the record-structure. If the first assumption fails and you are running QTP using the SEARCH option of the **subdictionary** program parameter, QTP searches every attached relational database for tables or views with the specified name. If there is one table or view with the specified name, QTP uses it. If there is more than one, or there is no table or view with the name, QTP issues an error message.

For more information about the **subdictionary** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

ALIAS name

Assigns an alternative name to a data structure. Once the alias is assigned, subsequent references to the data structure must use this name.

LINK

Specifies that the linkage to the subordinate data structure is hierarchical. The first related data structure must be linked hierarchically. If more than one data record exists for the subordinate data structure, a transaction is generated for each data record in the subordinate data structure. The data for higher-level record-structures is repeated in each of the generated transactions.

direct-linkage

Specifies the linkage to a particular data record of the subordinate data structure, which can be:

- a record-structure named in the dictionary
- a subfile

Direct-linkage has the form:

TO RECORD numeric-expression OF {record [IN file]|*subfilespec}

numeric-expression

A numeric-expression that produces a value that corresponds to a record number in the subordinate record-structure. The record number isn't verified until execution-time.

OF {record [IN file]|*subfilespec}

Direct-linkage is valid only for record-structures in direct files, relative files, or non-indexed subfiles. A subfilespec must be the qualified or unqualified name of an existing subfile or a portable subfile dictionary and it must be prefixed by an asterisk.

indexed-linkage

Specifies the linkage via an item or column, or items, or columns either to

- the subordinate record-structure for hierarchical linkage, or
- the related record-structure for parallel linkage

Indexed-linkage in QTP has the form:

[expression[,expression]...] [VIAINDEX indexname]
TO [linkitem[,linkitem]... OF]
{record [IN file]|*subfilespec|table [IN database]}

expression[,expression]...

Specifies an expression. QTP uses the result of the expression to find associated data records in the linked record-structure. Each expression must result in the value for a single item. The item must exist in a record-structure that has been declared in an ACCESS statement list.

Each expression must result in the value for a single item. The item must exist in a record-structure that has been declared in an ACCESS statement list.

Limit: Up to 255 expressions can be specified.

VIAINDEX indexname

Specifies either

- the name of an index of the subordinate or related record-structure for an indexed file

- the name of an index that has been defined for the subordinate or related relational table

Using indexname forces data records to be retrieved in index order; otherwise, the order is determined by the file system.

Limit (**MPE/iX**): VIAINDEX is ignored for IMAGE, except when the index specified is an OMNIDEX or a B-Tree index.

linkitem[,linkitem]... OF

If the subordinate or related record-structure is in an indexed or IMAGE (**MPE/iX**) file, the first linkitem must be the first segment of an index in that record-structure.

If the first linkitem is also the initial segment of a multiple-segment index, then subsequent linkitems can be used to specify additional segments of the multiple-segment index. The number of linkitems must be less than or equal to the number of segments in the index.

If the subordinate or related record-structure refers to a table or view, any item (column) in the table or view can be a linkitem.

Limit: There can be up to 255 linkitems. **MPE/iX:** IMAGE does not support linkage via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. All segments of the index must be specified.

If you specify both the VIAINDEX option and linkitems, the linkitems must match consecutive initial segments of the index. There can be additional segments in the index that aren't used in the linkage; in this case, linkage is by means of only the specified segments instead of the entire index, but retrieval is in index order.

The expression list specifies which items and/or columns from previous record-structures in the ACCESS statement list will be used in the link. The linkitem list or the indexname specifies which items or columns in the subordinate or related record-structure will be used in the link. If you explicitly specify both sides of the link, the two sides must match in the number of values, and in the type and size of the values.

TO cursor-name [sql-substitution...] [AND TO cursor-name [sql-substitution...]

When the linkage refers to a cursor, only the "LINK TO" or "AND TO" syntax is allowed. Linkage criteria can be specified within the cursor declaration, or by using sql-substitutions.

The syntax for an sql-substitution is:

substitution-variable (**text**)

For more information about sql-substitutions and substitution-variables, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

AND

Specifies that the record-structures on either side of the AND keyword are in a parallel relationship and are on the same level in the linkage hierarchy.

When a data record from one record-structure is read, a data record from the other record-structure is also read. If one record-structure has fewer data records than the other, a dummy record for that record-structure is included in the transaction. The content of this dummy record depends on whether the record is from a relational database table, whether null value support is enabled in the PowerHouse dictionary, and whether the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified.

If the record is from a relational database table and null value support is enabled, and the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified, then the data record is initialized to null values. If null value support is disabled or the record is not from a relational database table, then the data record is initialized to zeroes, spaces, and dictionary initial values. Also, if the **noinitnulls** program parameter or the INITIALIZE NULLS OFF resource file statement is specified explicitly or by default, then the data record is initialized to zeroes, spaces, and dictionary initial values. Regardless, the linkitem is initialized to the value used for linkage.

If the linkage to one record-structure is by unique index or record number, the data record, if it exists, is assumed to exist for all related data records in the parallel record-structure and the data is repeated in subsequent transactions.

OPTIONAL

Specifies that the construction of the transaction continues even if a related data record isn't found in the subordinate record-structure. A dummy record-structure is added to the transaction. The content of this dummy record depends on whether the record is from a relational database table, whether null value support is enabled in the PowerHouse dictionary, and whether the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified.

If the record is from a relational database table and null value support is enabled, and the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified, then the data record is initialized to null values. If null value support is disabled or the record is not from a relational database table, then the data record is initialized to zeroes, spaces, and dictionary initial values. Also, if the **noinitnulls** program parameter or the INITIALIZE NULLS OFF resource file statement is specified explicitly or by default, then the data record is initialized to zeroes, spaces, and dictionary initial values. Regardless, the linkitem is initialized to the value used for linkage.

For parallel relationships, specify OPTIONAL for the record-structure you want if construction of the transaction is to continue when there are no related data records in any of the parallel record-structures. When any record-structure in a parallel linkage is assigned as OPTIONAL, all parallel record-structures in the related group are treated as optional.

Limit: Cannot be used with the primary record-structure in the ACCESS statement.

Discussion

Each QTP request requires an ACCESS statement. The ACCESS statement declares

- the record-structures that are read
- the order in which the record-structures are read
- optionally, how the linkages between record-structures are constructed

The ACCESS statement causes QTP to open the files that contain the record-structures named in the ACCESS statement.

Limit: A maximum of 31 record-structures, including subfiles, can be declared in a request. A maximum of 63 record-structures can be declared in a run. There can be a maximum of 1,023 items per record structure. The maximum record size is 32, 767 bytes. The maximum size of all input records plus all defined items used in the input phase is 32,767 bytes.

This list describes terms used in the rest of this discussion:

Term	Definition
compound record	Consists of a data record from the primary record-structure and one data record from each of the related "linked to" record-structures.
data record	A single occurrence of a record in a file or database. QTP retrieves one or more data records from the subordinate record-structure for each data record in the primary record-structure.
dummy record	A record with zeros, spaces, and dictionary initial values.
initial subset	Either the first segment alone or the first segment followed by one or more consecutive segments.
linkage	Describes the different types of relationships that can occur between two or more record-structures.
primary record structure	The first record-structure named in the ACCESS statement.

Term	Definition
record-structure	A collection of elements that relate to a particular activity. A record-structure describes the structure of the data records that make up a file. A file can contain one or more record-structures.
transaction	Consists of a data record from the primary record-structure and one data record from each of the related "linked to" record-structures; together they form a compound record. QTP builds a transaction for each data record in the primary record-structure.

Viewing Available Record-structures and Items

To display the record-structures that are available at execution-time during retrieval, enter

```
> SHOW FILES
```

To display the items that are available in the accessed record-structures at execution-time during retrieval, enter

```
> SHOW ITEMS
```

Notes

All files opened for previous requests in the current QTP session are closed when an ACCESS statement is encountered.

An ACCESS statement remains in effect until it is canceled using the CANCEL statement or until the next REQUEST statement is entered. An ACCESS statement is no longer in effect after a successful "GO" is executed.

Default Linkage

In QTP, there are two forms of linkage: direct-linkage or indexed-linkage. If you don't specify both sides of the linkage, QTP can usually construct a default linkage.

QTP links data records in indexed files by matching values for identically-named items in the record-structures. In the following ACCESS statement, default linkage is by the item EMPLOYEE which is an item in both the STOCKS and EMPLOYEES record-structures:

```
> ACCESS STOCKS LINK TO EMPLOYEES
```

The item EMPLOYEE in the record-structure EMPLOYEES (the record-structure being linked to) must be defined as a segment in an index. The item EMPLOYEE in the record-structure STOCKS (the record-structure before the keyword TO) does not have to be a segment in an index.

The following table shows how QTP attempts to construct a default linkage:

"From" side of link	"To" side of link	What QTP does
expression(s)	---	This combination isn't allowed if the subordinate or related record-structure has more than one index. If it has one index, QTP attempts to match that index's segments, in order, to the values(s) specified by the expression(s). There must be at least as many segments as expressions. If there are more segments than expressions, QTP ignores the extra segments.
---	linkitem(s)	QTP attempts to match the linkitems to identically named items in any record-structure that has previously been declared in the ACCESS statement list. Usually QTP searches the record-structures in the order that they were declared. However, if parallel linkage has already been established between a parallel driver record-structure and one or more related record-structures, then QTP looks in the parallel driver record-structure first.

"From" side of link	"To" side of link	What QTP does
---	indexname	QTP uses all the segments of the index as linkitems and attempts to construct a linkage using the procedure described above for linkitems. QTP uses the index to determine the order of data record retrieval.
---	indexname and linkitem(s)	QTP uses the specified linkitems (which aren't necessarily all the items in the index, as long as they are consecutive initial segments of the index) and attempts to construct a linkage as described above for linkitems. QTP uses the index to determine the order of data record retrieval.
---	---	There must be at least one index defined for the subordinate or related record-structure. For hierarchical linkage, QTP examines each index in the subordinate or related record-structure, in the order that the indexes were defined (the same order in which QSHOW reports them), to find one whose segments all match, by name, items in previous record-structures in the ACCESS statement list. The previous record-structures are checked, in the order in which they were declared. If this fails, QTP uses the first index whose first segment matches an item in a previous record-structure, and constructs the linkage from that first item, plus as many consecutive items as possible. The matching criteria are the same for parallel linkage as for hierarchical linkage, but QTP examines the indexes of the related record-structure in a more particular order. If a linkage has already been established between the parallel driver record-structure and one or more related record-structures in parallel, QTP tries to use the same item from the parallel driver record-structure again to match an index in the new related record-structure. If this doesn't work, QTP tries to match an index in the new related record-structure to any item in the driver record-structure. If that also fails, QTP tries to establish linkage by means of the procedure described above for hierarchical linkage.

How Linkage Works

Each time you retrieve a primary data record in a linkage, you retrieve all related data records from the linked record-structures.

Each record-structure named in the ACCESS statement can be linked to one or more record-structures. Each pair of record-structures is linked by a specific item and value. You can use any item in the primary record-structure to link to another record-structure. However, in the record-structure being linked to, you must use a segment that's defined in an index unless you are specifying linkage using record numbers.

Types of Relationships

When linking any two record-structures, there are two types of data relationships:

- one-to-one
- one-to-many

One-to-One Relationships

A one-to-one relationship between two record-structures occurs when one or more linkitems from a primary record-structure is in a unique index in another. QTP only treats an index as unique if all segments of the index are used in the linkage. If you only link to the first few segments of a unique index, QTP assumes a repeating link and therefore a one-to-many linkage.

For example, assume that an application contains a file called EMPLOYEES and another called POSITIONS, with the following record-structures:

EMPLOYEES

EMPLOYEE	LASTNAME	FIRSTNAME	**POSITIONCODE**	CITYBRANCH
1000	Abra	Margaret	**pgm III**	Toronto East

In the example below, the item POSITIONCODE in POSITIONS is defined as a segment in a unique index.

POSITIONS

POSITIONCODE	POSITIONTITLE	SALARY	CHARGEOUT
pgm III	Sr Programmer	4200.00	110.00

If you enter

```
> ACCESS EMPLOYEES LINK TO POSITIONS
```

you define a linkage that uses the item POSITIONCODE in EMPLOYEES to link to the segment POSITIONCODE in POSITIONS. Because POSITIONCODE is in a unique index in POSITIONS, there is only one matching record in POSITIONS for each record in EMPLOYEES.

If a segment is defined in an index of type UNIQUE in the data dictionary, QTP only retrieves one data record for each segment value.

One-to-Many Relationships

A one-to-many relationship between two record-structures occurs when one or more linkitems from one record-structure is in a repeating (non-unique) index of another.

For example, assume that an application contains a file called EMPLOYEES and another called BILLINGS, with the record-structures shown below.

Here, EMPLOYEE in EMPLOYEES is defined as a segment in a unique index:

EMPLOYEES

EMPLOYEE	LASTNAME	FIRSTNAME	POSITIONCODE	CITYBRANCH
1000	Abra	Margaret	pgm III	Toronto East

Here, EMPLOYEE in BILLINGS is defined as a segment in a repeating index:

BILLINGS

EMPLOYEE	PROJECT	HOURS
1000	5000	20.00
1000	5002	20.00

If you enter

```
> ACCESS EMPLOYEES LINK TO BILLINGS
```

you define a linkage that uses the item EMPLOYEE in EMPLOYEES to link to the segment EMPLOYEE in BILLINGS. Because EMPLOYEE is in a repeating index in BILLINGS, there may be more than one matching record in BILLINGS for each record in EMPLOYEES.

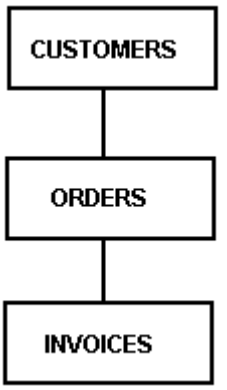
Specifying Linkage in a One-to-Many Relationship

In PowerHouse, there are two ways to specify linkage in a one-to-many relationship:

- hierarchical
- parallel

Hierarchical Linkage

A hierarchical linkage can be shown as a chain of record-structures:



In this example, CUSTOMERS can have one or many related ORDERS and each ORDER can have one or more related INVOICES. The relationship between CUSTOMERS and INVOICES exists only through ORDERS.

Defining a Hierarchical Linkage

You can define a hierarchical linkage with the LINK TO option. When you define this type of linkage, each record-structure is related to a record-structure that precedes it in the ACCESS statement.

For example,

```
> ACCESS CUSTOMERS LINK TO ORDERS LINK TO INVOICES
```

defines a linkage between CUSTOMERS and ORDERS, and between ORDERS and INVOICES.

In this example, the relationship between CUSTOMERS and INVOICES only exists through ORDERS:

CUSTOMERS

ACCOUNTNUMBER	CUSTOMER
1010	Kane Contractors

ACCOUNTNUMBER is a segment in a repeating index in ORDER:

ORDERS

ORDERNUMBER	ACCOUNTNUMBER	PARTNUMBER
5001	**1010**	1001
5002	**1010**	1002

ORDERNUMBER is a segment in a repeating index in INVOICES:

INVOICES

INVOICENUMBER	ORDERNUMBER
2001	**5001**
2002	**5001**
2003	**5002**

How Hierarchical Linkage Works

When you execute the request, QTP

1. Reads the first record from CUSTOMERS.
2. Reads the first record from ORDERS that has a value for ACCOUNTNUMBER equal to the value of ACCOUNTNUMBER in the CUSTOMERS record.
3. Reads the first record from INVOICES that has a value for ORDERNUMBER equal to the value of ORDERNUMBER in the ORDERS record.
4. Processes the transaction.

After processing the transaction, QTP

1. Reads the next data record from INVOICES when the value for ORDERNUMBER equals the value of ORDERNUMBER in ORDERS.
2. Processes the new transaction.
3. Continues to process all records from INVOICES when the value for ORDERNUMBER is equal to that of the ORDERS record.

When no more INVOICES records exist for the current ORDERS record, QTP

1. Reads the next ORDERS record when the value for ACCOUNTNUMBER is equal to that of the CUSTOMERS record.
2. Reads a new INVOICES record with an ORDERNUMBER equal to that of the ORDERS record.
3. Processes the transaction.
4. Continues to report all records from INVOICES when the value for ORDERNUMBER is equal to that of the ORDERS record.

When no more ORDERS records exist for the current CUSTOMERS record, QTP

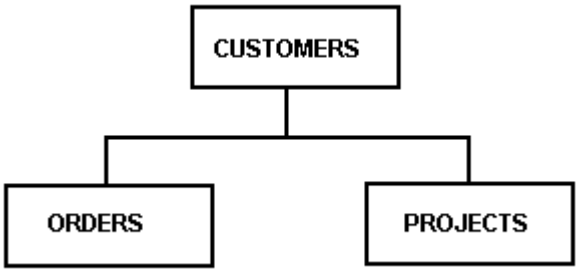
1. Reads the next CUSTOMERS record.
2. Repeats the whole process.

Parallel Linkage

The first record-structure in a parallel linkage is linked using a standard hierarchical linkage, as in

```
> ACCESS CUSTOMERS LINK TO ORDERS
```

A parallel linkage is a branch:



A parallel relationship defines a linkage branch in which more than one record-structure is linked to a common record-structure. Record-structures that are joined by the AND TO option aren't directly related to one another. The record-structures are instead directly related to a common record-structure. A single item in the common record-structure is used to link related data records in the other two record-structures.

Defining Parallel Linkage

Define a parallel linkage with the AND TO option. When you define this type of linkage, more than one record-structure is linked to a common record-structure to form a branch. The relationships that exist aren't directly related to one another.

For example,

```
> ACCESS CUSTOMERS LINK TO ORDERS AND TO PROJECTS
```

This defines a linkage between CUSTOMERS and ORDERS and between CUSTOMERS and PROJECTS.

CUSTOMERS

ACCOUNTNUMBER	CUSTOMER
1010	Kane Contractors

ORDERS

ORDERNUMBER	**ACCOUNTNUMBER**	PARTNUMBER
5001	**1010**	1001
5002	**1010**	1002

PROJECTS

PROJECT	CUSTOMER	TITLE	**ACCOUNTNUMBER**
6001	Kane	MIS	**1010**
6002	Kane	Support	**1010**

How Parallel Linkage Works

When you execute the request, QTP

1. Reads the first data record from CUSTOMERS.
2. Reads the first data record from ORDERS that has a value for ACCOUNTNUMBER equal to the value of ACCOUNTNUMBER in CUSTOMERS.
3. Reads the first data record from PROJECTS that has a value for ACCOUNTNUMBER equal to the value of ACCOUNTNUMBER in CUSTOMERS.
4. Processes the transaction.

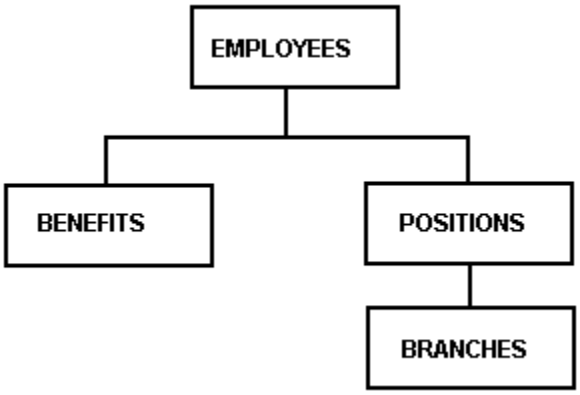
After processing the transaction, QTP

1. Reads the next record from both ORDERS and PROJECTS when the value for ACCOUNTNUMBER equals that of the CUSTOMERS record-structure.
2. Processes the new transaction. QTP continues, reading from both ORDERS and PROJECTS for each new transaction.

Mixed Hierarchical and Parallel Linkage

The following example shows a mixed hierarchical and parallel linkage:

```
> ACCESS EMPLOYEES LINK TO BENEFITS AND TO POSITIONS &
>   LINK TO BRANCHES
```



Depending on the requirements of your application and the nature of the data records that are stored in your files, you might choose either hierarchical or parallel linkage, or a combination.

How QTP Builds Transactions

If the number of related data records for the record-structures in the two parallel files isn't the same, QTP substitutes dummy data records for the record-structure that has run out of related records. The content of this dummy record depends on whether the record is from a relational database table, whether null value support is enabled in the PowerHouse dictionary, and whether the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified.

If the record is from a relational database table and null value support is enabled, and the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified, then the data record is initialized to null values. If null value support is disabled or the record is not from a relational database table, then the data record is initialized to zeroes, spaces, and dictionary initial values. Also, if the **noinitnulls** program parameter or the INITIALIZE NULLS OFF resource file statement is specified explicitly or by default, then the data record is initialized to zeroes, spaces, and dictionary initial values. Regardless, the linkitem is initialized to the value used for linkage.

For example:

```
> ACCESS CUSTOMERS LINK TO ORDERS AND TO PROJECTS
```

CUSTOMERS

ACCOUNTNUMBER	CUSTOMER
1010	Kane Contractors

ORDERS

ORDERNUMBER	**ACCOUNTNUMBER**	PARTNUMBER
5001	**1010**	1001

PROJECTS

PROJECT	CUSTOMER	TITLE	**ACCOUNTNUMBER**
6001	Kane	MIS	**1010**
6002	Kane	Support	**1010**

In this example, QTP builds two transactions for ACCOUNTNUMBER 1010 using the following steps:

To build transaction 1, QTP

1. Reads the CUSTOMERS record for ACCOUNTNUMBER 1010.
2. Reads the ORDERS record for ORDERNUMBER 5001.
3. Reads the PROJECTS record for PROJECT 6001.

To build transaction 2, QTP

1. Copies the CUSTOMERS record for ACCOUNTNUMBER 1010.
2. Adds a dummy data record for ORDERS (since there are no more records in ORDERS with ACCOUNTNUMBER 1010, but there are in PROJECTS).
3. Reads the PROJECTS record for PROJECT 6002.

OPTIONAL Linkage

In hierarchical linkage, QTP doesn't build a transaction if a data record in the primary record-structure has no related data records in a record-structure to which it is being linked. As a result, the primary record isn't processed. This prevents incomplete data from being processed.

The OPTIONAL keyword in the ACCESS statement defines a linkage that is processed even if no related data records are found.

For example, the following statement defines an optional linkage between EMPLOYEES and BILLINGS:

```
> ACCESS EMPLOYEES LINK TO BILLINGS OPTIONAL
```

When this statement is executed, data for employees with or without billings is processed. Without the OPTIONAL keyword, the data for employees without billings isn't processed at all.

In an optional linkage, when no related data records are found, QTP constructs a transaction using a dummy data record for the related record-structure. The content of this dummy record depends on whether the record is from a relational database table, whether null value support is enabled in the PowerHouse dictionary, and whether the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified.

If the record is from a relational database table and null value support is enabled, and the **initnulls** program parameter or INITIALIZE NULLS ON resource file statement is specified, then the data record is initialized to null values. If null value support is disabled or the record is not from a relational database table, then the data record is initialized to zeroes, spaces, and dictionary initial values. Also, if the **noinitnulls** program parameter or the INITIALIZE NULLS OFF resource file statement is specified explicitly or by default, then the data record is initialized to zeroes, spaces, and dictionary initial values. Regardless, the linkitem is initialized to the value used for linkage.

In a parallel linkage, use the OPTIONAL keyword when you want QTP to process data records from the primary record-structure even if neither parallel record-structure has any related data records. QTP automatically handles the situation in which related data records for one parallel record-structure run out before related data records for another parallel record-structure. If the OPTIONAL statement is declared against either parallel record-structure in the ACCESS statement, the primary record-structure alone can form a valid transaction.

How to Specify Linkage Explicitly

In many requests, getting the exact linkage you want is easier when you define your linkage explicitly. By using an explicit linkage, you identify the item or expression that QTP uses to link each pair of record structures.

You can specify linkage to record-structures by using

- the VIAINDEX option to specify linkage via a particular index
- an initial subset of the index segments
- names of items and segments
- record numbers

Specifying Linkage Using the VIAINDEX Option

You can specify linkage using the viaindex option by

- a single-segment index
- multiple-segment indexes

Single Segment Index

You can specify linkage using the VIAINDEX option of the ACCESS statement to explicitly control linkage to a record-structure via a single segment index.

For example,

```
> ACCESS EMPLOYEES LINK VIAINDEX EMPINDEX TO STOCKS
```

links the EMPLOYEES record-structure to the STOCKS record-structure via the index EMPINDEX of STOCKS.

Multiple-Segment Index

A record-structure and an index are defined in the data dictionary as follows:

```
Record BRANCHES
   Item SEGMENTX
   Item SEGMENTY
   Item SEGMENTZ
Index INDEX1 of BRANCHES
   Segment SEGMENTX
   Segment SEGMENTY
   Segment SEGMENTZ
```

You can specify linkage using the VIAINDEX option of the ACCESS statement to explicitly control linkage to a record-structure via a multiple-segment index.

```
> ACCESS DIVISIONS LINK VIAINDEX INDEX1 TO BRANCHES
```

Linkage between DIVISIONS and BRANCHES is based on a multiple-segment index. The linkage can be based on one, two, or three segments, depending on the name matches that QTP finds when establishing linkage.

If, in the preceding example, all three segments are used in an index to BRANCHES and DIVISIONS, then linkage is established by SEGMENTX, SEGMENTY, and SEGMENTZ.

Cursor Retention

Cursor retention is applied for files in the ACCESS statement when no VIAINDEX option is specified and SET LOCK RECORD UPDATE is in effect. Other files do not need this option.

For SET LOCK FILE REQUEST and SET LOCK FILE RUN, cursor retention is not used.

Ascending/Descending Index Support

If ascending or descending indexes are supported in a database or indexed file, PowerHouse uses the declared order to generate retrieval requests when an explicit VIAINDEX option is used.

When PowerHouse generates database retrieval requests as the result of an explicit VIAINDEX index, it generates the sorting specification in the request to match the order declared when the index was defined.

For example, if an index called PAYMENTS_CUSTOMER_DATE consists of two segments, CUSTOMER_NUM (ascending) and PAYMENT_DATE (descending), the PowerHouse database request generated for

```
ACCESS VIAINDEX PAYMENTS_CUSTOMER_DATE
```

contains the equivalent of

```
SORT ON CUSTOMER_NUM A, PAYMENT_DATE D
```

This would produce data in order of CUSTOMER_NUM with the most recent payments first for each customer.

Similarly, if an index called DATE_KEY consists of three segments, YEAR, MONTH and DAY, and that index is a descending index, the database request generated for ACCESS VIAINDEX DATE_KEY contains the equivalent of:

```
SORT ON YEAR D, MONTH D, DAY D
```

This would produce data with the most recent date values first.

QSHOW reports ordering information for relational indexes and indexes segments. For more information, see Chapter 4, "QSHOW Statements", in the *PDL and Utilities Reference* book.

Specifying Linkage Using an Initial Subset

You can specify linkage using an initial subset by using

- segments and items
- expressions
- lists of segments, items, or expressions

Segments and Items

You can specify linkage using an initial subset of the index segments. For example, you can specify linkage via the first two segments of INDEX1, as in

```
> ACCESS DIVISIONS LINK VIAINDEX INDEX1 &
>   TO SEGMENTX, SEGMENTY OF BRANCHES
```

Items in the DIVISIONS record are linked to segments of BRANCHES.

```
Record DIVISIONS
    Item ITEMA
    Item ITEMB
    Item ITEMC
```

You can also specify a list of items and/or segments from previously-declared record-structures. For example,

```
> ACCESS DIVISIONS LINK ITEMA, ITEMB &
>   VIAINDEX INDEX1 TO SEGMENTX, SEGMENTY OF BRANCHES
```

If you specify a list of items and/or segments, the number must equal the number of specified segments of the "link to" record structure.

Expressions

You can specify a list of expressions to be linked to the specified segments. In the following example, SEGMENTZ isn't specified:

```
> ACCESS DIVISIONS LINK "USA", "WEST" &
>   VIAINDEX INDEX1 TO SEGMENTX, SEGMENTY OF BRANCHES
```

If you specify a list of expressions, the number of expressions must be equal to the number of specified segments of the "link to" record-structure.

Lists of Segments, Items, or Expressions

You can specify a "from" list without a "to" list when specifying linkage. For example,

```
> ACCESS DIVISIONS LINK ITEMA, ITEMB, ITEMC &
>   VIAINDEX INDEX1 TO BRANCHES
```

Limit (**MPE/iX**)**:** IMAGE does not support linkage via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. All segments of the index must be specified.

Specifying Linkage Using Names

You can specify a linkage even if the linkage items and segments don't have the same names. To do this, specify the items and segments to be matched. QTP matches specified items and segments according to how they're declared in the ACCESS statement.

For example, suppose you want to link the DIVISIONS data records to the BRANCHES data records, but the names of the linkage items in these record-structures don't match.

You can specify explicit linkage using

- ITEMA and ITEMB of DIVISIONS
- SEGMENTX and SEGMENTY of BRANCHES

as in

```
> ACCESS DIVISIONS LINK ITEMA, ITEMB &
>   VIAINDEX index1 to SEGMENTX, SEGMENTY of branches
```

Specifying Linkage Using Record Numbers

You can specify linkage using record numbers (a numeric item, a number, or an expression) to link to record-structures that are stored in direct or relative files, or in subfiles.

For example,

```
> ACCESS EMPLOYEES LINK TO RECORD 2 OF BILLINGS
```

BUILD

Compiles and saves a QTP run.

Syntax

BUILD [filespec]

filespec

Specifies the file in which the compiled QTP run is saved.

Default: If a filespec is not specified on the BUILD statement, the filespec from the RUN statement is used. If there is no filespec on the RUN statement or if there is no RUN statement, the file is saved with the default name **QTPOBJ**.

Discussion

The BUILD statement compiles the current QTP run and saves it in a file in compiled form.

Compiled runs execute more quickly than source statements. To execute a compiled run, use the EXECUTE statement rather than the USE statement.

The GO statement can't be used to run a compiled run that has already been created using the BUILD statement.

Saving Source Statements

A compiled run is based on the current release of QTP. If you are using a release of QTP that differs from the one used to compile the run, you may have to recompile the run from the source statements.

The SAVE statement, and the following options of the SET statement aren't saved in a compiled run:

COMPILE|SYNTAX	DICTIONARY	JOB|NOJOB
LIST|NOLIST	PRINT|NOPRINT	VERIFY|NOVERIFY

When you execute a compiled run with the **auto** program parameter, the QTP session ends on completion of the run. This happens whether or not an EXIT statement was included in the run.

If SET SYNTAX is in effect, the statements are parsed, but not compiled.

The BUILD statement is disabled when the SYNTAX option is specified.

Limit

A compiled run is based on the current definition of files and elements in the data dictionary. Compiled runs must be recompiled when you make changes to the dictionary definitions used by that run.

For more information about how QTP creates compiled runs, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

Permanent Compiled Sections in an ALLBASE/SQL DBEnvironment (MPE/iX)

When you access tables in an ALLBASE/SQL DBEnvironment, QTP creates permanent sections when compiling with the BUILD command. QTP uses dynamic SQL statements when a program is invoked with GO.

Permanent sections are stored as modules in an ALLBASE/SQL database.

Module Names

Module names are derived from compiled file names, which come from the name on the BUILD statement. Periods in the compiled file name are replaced by underscores. The module name is always in the form:

FILENAME_GROUPNAME.

For example, if the screen name is MAINMENU and the current group is PUB, the module name is MAINMENU_PUB. For the screen name, MAINMENU.DEVELOP.TEST, the module name is MAINMENU_DEVELOP.

Because the owner name and the module name are stored in the PowerHouse object file, renaming or moving the object file does not invalidate access to the module. However, transferring ownership of the module to another owner makes it impossible for PowerHouse to locate the module at run-time, and an execution error results.

Creating Module Files

Modules can be transferred from one DBEnvironment to another using the **moduleloc** program parameter and the install process in ISQL. If you need a module in more than one DBEnvironment, you must compile the PowerHouse program using the **moduleloc** program parameter. This creates an installable module which can be copied to a second environment. The module naming rules are such that the owner of the compiled section in the new environment is the original creator.

moduleloc lets you create module files similar to the preprocessors. The general syntax for the **moduleloc** program parameter is:

MODULELOC=group

A file is created in the specified group in the filename specified on the QTP BUILD statement.

For example, to start QTP so that a module is created in the PUB group, enter

```
:QTP INFO="MODULELOC=PUB"
```

For more information about the **moduleloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Limit: The location specified by **moduleloc** cannot be the same as the location of the file named in the BUILD statement.

Example

The BUILD statement compiles the QTP run and saves it in the file TRANSFER. This run adds records to two files and edits the values from an old employee file.

```
> RUN TRANSFER
> REQUEST TRANSFER-EMPLOYEE-DATA ON EDIT ERRORS REPORT
>
> ACCESS OLDEMPL
>
> EDIT EMPLOYNO LOOKUP NOTON EMPLOY1 VIAINDEX &
>   EMPLOYEE, NOTON PAYROLLDATA VIAINDEX EMPLOYEE
> EDIT POSITIONCODE LOOKUP ON POSITION &
>   VIAINDEX POSITION
> EDIT MAILCODE PATTERN "((#####(-#####)<)|(^#^#^#))"
> EDIT SEX VALUES "M", "F"
> EDIT EMPLSALARY VALUES 10000 TO 75000
>
> OUTPUT EMPLOY1 ADD ALIAS EMPLOY1_OUT
> ITEM EMPLOYEE INITIAL EMPLOYNO OF OLDEMPL
> ITEM DATEAPPOINTED INITIAL STARTDATE
> ITEM POSITION INITIAL POSITIONCODE
> ITEM POSTALCODE INITIAL MAILCODE
>
> OUTPUT PAYROLLDATA ADD ALIAS PAYROLL_OUT
> ITEM EMPLOYEE INITIAL EMPLOYNO OF OLDEMPL
> ITEM SALARY INITIAL EMPLSALARY
>
```

```
> BUILD TRANSFER
```

To execute the compiled run, enter

```
> EXECUTE TRANSFER
```

[SQL] CALL

Calls a stored procedure or stored function from the specified database in the Sort phase, or more typically, in the Output phase.

Syntax

```
[SQL [IN database] [sql-options] [IF condition]
[ON ERRORS {BYPASS UNTIL sort-item|FINAL}|REPORT|
    {TERMINATE [REQUEST|RUN]}]]
CALL stored-procedure|stored-function
[([ITEM] item [IN [OUT]]|[OUT]
    [,[ITEM] item [IN [OUT]]|[OUT]]...)]
[ON ERROR CONTINUE|TERMINATE]
[RETURNING return-parameter]
```

IN database

Specifies against which database the stored procedure or function is executed.

Limit: Stored procedure calls are valid for DB2, ODBC, Oracle, Oracle Rdb (declared as TYPE RDB in the dictionary), and Sybase databases. Stored function calls are valid only for Oracle databases.

sql-options

Specifies the timing of the SQL CALL. The procedure call is performed at the specified break.

The sql-options are AT FINAL, AT INITIAL, and AT [START[OF]].

AT FINAL

Calls the stored procedure at the end of the transaction set after processing all transactions. This option has no effect unless there is an input phase.

AT INITIAL

Calls the stored procedure at the beginning of the transaction set before processing any transactions. This option has no effect unless there is an input phase.

AT [START[OF]]sort-item

The sort-item option indicates that the stored procedure is performed at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group. This option is not available unless a request has a sort phase.

IF condition

Calls the stored procedure only if the condition is satisfied. This option affects the processing sequence.

ON ERRORS

States what action to take if errors are encountered while performing the stored procedure.

BYPASS UNTIL sort-item|FINAL

Skips processing of all transactions until it reaches either the control break or the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

REPORT

Reports the error and skips processing of the transaction. Processing continues as if the error had not occurred.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run and rolls back any uncommitted updates.

Default: TERMINATE RUN

CALL stored-procedure

The name of a stored procedure or stored function in the database.

The syntax for a procedure name varies with the RDBMS. For information on a specific database system, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

([ITEM] item [IN[OUT]]|[OUT][, [ITEM] item [IN [OUT]]|[OUT]]...)

Items which are passed to the stored procedure or Oracle stored function, or received from the stored procedure. Input parameters can be temporary, defined or record items. Output parameters can be temporary or record items.

Blob items may also be used for both input and output parameters when calling an Oracle stored procedure or stored function.

IN

Specifies that the item is an input parameter.

IN OUT

Specifies that the item is both an input and output parameter. The changed values of the input/output parameters are available to PowerHouse when stored procedure execution is complete.

OUT

Specifies that the item is an output parameter. The changed values of the output parameters are available to PowerHouse when stored procedure execution is complete.

Default: IN

RETURNING return-parameter

The return-parameter must be defined as a temporary or record item.

For Sybase, identifies the item that contains the return status from a stored procedure upon completion of the Sybase stored procedure.

For Oracle, identifies the item that contains the value returned by a stored function upon completion of the Oracle stored function.

Limit: Valid for Oracle stored functions but not valid for Oracle stored procedures. For Sybase, the return-parameter must be defined as a 32-bit (4-byte) integer.

Discussion

Using the CALL statement, the developer can call local or remote stored procedures and pass input and output parameters and receive execution status. The stored procedures that fall into this category are ones that return output parameters, status or values to the calling application. Stored procedures that return result sets must be called as part of the DECLARE CURSOR statement.

CANCEL

Cancels the current QTP run specifications.

Syntax

CANCEL [CLEAR]

CLEAR

Removes any source statements in the temporary save file, QTPSAVE, once the run specifications are canceled.

Discussion

The CANCEL statement cancels the specifications of the current QTP run. The CANCEL statement does not cancel SET statements. The CANCEL statement does not clear the contents of the source statement save file, QTPSAVE, unless the CLEAR option is included.

All QTP statements that you enter are automatically stored in a temporary save file.

Example

The SAVE statement saves the statements following the CANCEL CLEAR in the file MONTHEND only.

```
> RUN MONTHEND
>      REQUEST ADD_INTEREST
>      ACESS      CUSTOMERACCNTS LINK TO CUSTOMERDETAIL
>      ^^^^^
*E* Expected: EDIT DEFINE DISPLAY ACCESS USE EXECUTE REVISE SET SHOW CANCEL
EXIT QUIT SAVE QSHOW <eol>

> CANCEL CLEAR
> RUN MONTHEND
> REQUEST ADD_INTEREST
> ACCESS CUSTOMERACCNTS LINK TO CUSTOEMRDETAIL
.
.
.
> SAVE MONTHEND
```

QTP cancels the RUN and REQUEST statements plus the invalid statement when the CANCEL command is entered. QTP writes all statements, including invalid ones, to the temporary save file. The CLEAR option ensures that all statements prior to CANCEL are removed from the temporary files so that when the SAVE statement is used, the file, MONTHEND, has the valid statements for that run.

CHOOSE

Extracts data from an indexed file, relational table or view, or SQL cursor by item value.

Limit: The CHOOSE statement can only be applied to indexes or relational items that are defined for the primary record-structure being accessed. You can use only one CHOOSE statement in a single request.

Syntax

CHOOSE sql-substitution...
CHOOSE VIAINDEX indexname
CHOOSE [sql-substitution...|VIAINDEX indexname] linkitem
[generic|nogeneric] [choose-options]
[,linkitem [generic|nogeneric] [choose-options]]...]

Options

sql-substitution

An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default sql-substitutions, WHERE and ORDERBY, will be inserted in generated SQL statements even if the corresponding substitution-variables do not exist on a DECLARE CURSOR statement.

The syntax for an sql-substitution is:

substitution-variable (text)

For more information about sql-substitutions and substitution-variables, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

Limit: An sql-substitution is valid only if the primary data structure is a cursor. The VIAINDEX and sql-substitution options are mutually exclusive.

VIAINDEX indexname

The name of an index in the primary record-structure of an indexed file or relational table. Using indexname alone forces data records to be retrieved in index order; otherwise, the order is determined by the file system. If you use indexname alone, without linkitems, you choose all the data records indexed by that index. If indexname and linkitems are used together, the linkitems must exist in the index and be an initial subset of that index. However, you don't have to specify all the segments that make up the index.

Limit: This option is not valid when the CHOOSE statement applies to a DECLARE CURSOR. The VIAINDEX and sql-substitution options are mutually exclusive. The VIAINDEX option is required when the CHOOSE statement is used with a B-Tree or OMNIDEX index on an IMAGE dataset.

linkitem

For a table or view in a relational database (if VIAINDEX is not specified), any column can be a linkitem. For a record-structure in an indexed file, the first linkitem must either be the segment of a single segment index or the first segment of a multiple-segment index. In the case of a multiple-segment index, the subsequent linkitems from the same index may be specified, but the linkitems must be specified in the same order in which the segments are defined in the dictionary.

Limit: 255 linkitems. All segments must be specified for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

GENERIC|NOGENERIC

GENERIC allows users to choose data records using partial index values of character items. NOGENERIC prevents users from choosing using partial values of character items. For more information about generic retrieval, see (p. 57).

Limit: Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Default: GENERIC

Choose Options

choose-options		
case-expression-set	conditional-expression-set	PARM
SYSTEMVALUE	value-set	

case-expression-set

Compares the value of an item against a value or range of values and selects one expression-set to be used to determine the values for the linkitem.

The general form is:

CASE [OF] item
WHEN value-set|EXISTS|NULL|MISSING
{THEN|:} expression-set|NULL|MISSING
[WHEN value-set|EXISTS|NULL|MISSING
{THEN|:} expression-set|NULL|MISSING]...
[DEFAULT expression-set|NULL|MISSING]

If there is no match, the specified default is used. If no default is specified, no records are chosen.

The general form of value-set used within the case-expression-set is the same as that used for the value-set choose-option. For more information, see (p. 57).

expression-set

The general form of an expression-set is:

expression|(conditional-expression)
[TO expression|(conditional-expression)]
[,expression|(conditional-expression)
[TO expression|(conditional-expression)]]...

A conditional-expression differs from a conditional-expression-set described below. For more information about conditional-expressions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

If specific values are used exclusively in the expression-set, the form is the same as that used for the value-set choose-option on (p. 57). If a defined item used in an expression-set contains a generic retrieval character (by default, @ or @@), the character is interpreted as data.

The conditional-expression must be enclosed within parentheses to avoid ambiguity between the linkitem and defined items within the expressions. These parentheses are required. At run-time, each form of expression evaluates to a value-set.

Limits:

- The resulting value of an expression must be compatible in type and size to that of the linkitem.
- Record items, table columns, and cursor columns may not be used in expressions as they are available only after the record complex is built (the CHOOSE is used to build the record complex).
- If an expression contains defined items, the defined items must not contain record items. Defined items used in expressions must be coded before the CHOOSE statement.

conditional-expression-set

A conditional-expression-set, when evaluated, selects one expression-set to be used to determine the values for the linkitem.

The general form is:

(expression-set [IF condition
[ELSE expression-set IF condition]...
[ELSE expression-set])

When a conditional-expression-set doesn't end with an unqualified ELSE option and none of the conditions have been satisfied, no records are chosen.

PARM [parm-option]...

Prompts for linkitem values at execution-time. Partial linkitem values using the generic retrieval character (by default, @ and @@) are allowed in response to a PARM prompt.

Limit: Only one value per input line can be entered. The PARM option cannot be used with any other CHOOSE option.

parm-options		
ALL|NOTALL	DOWNSHIFT|UPSHIFT	FORCE|NOFORCE CENTURY
FORMAT	ON ERRORS	PROMPT
RANGE|NORANGE	SEPARATOR	

ALL|NOTALL

ALL allows users to choose all the data records in a record-structure either by entering the generic retrieval character (by default, @) or by making a null entry in response to the first prompt. NOTALL prevents users from choosing all values using the generic retrieval character or by making a null entry.

Default: ALL

DOWNSHIFT|UPSHIFT

Shifts the entered value of a character item to either lowercase or uppercase. Stored values aren't changed.

Limit: Valid only for character items. Non-alphabetic characters within a character item aren't affected.

FORCE|NOFORCE CENTURY

FORCE CENTURY specifies that the user must enter a century on all century-included date fields. This option applies to century-included dates with two or four-digit year formats.

Default: To find out the active value of the option, you must look at the ELEMENT, the USAGE, and the SYSTEM OPTIONS statements in PDL. If the option is unspecified on the CHOOSE statement, the active value is taken from the ELEMENT. If the option is unspecified on the ELEMENT statement or a related USAGE, the active value is taken from the SYSTEM OPTIONS.

Limit: Valid only for century-included dates.

FORMAT date-format

Specifies the format for entering and displaying date item values. Date values can be entered either with or without separator characters. A date-format can be one of the following:

Date-format	Example	Date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMD D	2001/MAY/23
YYMM	01/05	YYMMM	01/MAY
YYYYMM	2001/05	YYYYMMM	2001/MAY
YYDDD	01/125	YYYYDDD	2001/125
MMDDYY	05/23/01	MMMDDYY	MAY/23/01

Date-format	Example	Date-format	Example
MMDDYYYY	05/23/2001	MMMDDYYYY	MAY/23/2001
MMYY	05/01	MMMYY	MAY/01
MMYYYY	05/2001	MMMYYYY	MAY/2001
MMDD	05/23	MMMDD	MAY/23
DDMMYY	23/05/01	DDMMMYY	23/MAY/01
DDMMYYYY	23/05/2001	DDMMMYYYY	23/MAY/2001
DDMM	23/05	DDMMM	23/MAY
DDDYY	125/01	DDDYYYY	125/2001

YYYY - four digit year (e.g., 2001)

MM - two digit month (e.g., 05)

MMM - three character month name (e.g., MAY)

DD - two digit day for a month (e.g., 23)

DDD - three digit day for a year (e.g., 365)

Regardless of the output order of the date, the internal working format is YYMMDD (for dates without centuries), YYYYMMDD (for dates with centuries)

The FORMAT option governs data entry by determining the way you can enter date values. Dates can always be entered in the format specified in the FORMAT option, with or without the established separator character and with either the MM or MMM month format.

If the FORMAT option is used but the SEPARATOR option isn't, the only separator character that QUIZ accepts is the separator character specified by System Options, or if it isn't specified, a slash (/).

If a two-digit year is specified in the date format, applications won't accept a four-digit year. A two-digit year is represented by YY (for example, 01).

If a four-digit year is specified in the date format, you can only enter a two-digit year if you enter a separator character between the year and any adjacent numeric component of the date. The default century is added automatically.

Single-digit day and month entries are accepted if the user enters the separator character, as in 4/8/2001. An entry of 4AUG2001 is also allowed, because PowerHouse accepts a single-digit day entry if the middle value is a three-character month.

A three-digit day of the year from 1 to 366 is represented by DDD.

Although values for date items can be entered in a variety of formats, the values are always stored in either YYMMDD or YYYYMMDD form.

Limit: Valid only for date items. This option only affects the entry format; the display format isn't affected.

Default: YYYYMMDD for eight-digit dates; YYMMDD for six-digit dates.

ON ERRORS [REPROMPT [n [TIMES]]] [TERMINATE [REQUEST|RUN]]

Specifies which action to take if entered values fail editing. If both REPROMPT and TERMINATE are used, REPROMPT must appear first.

REPROMPT [N [TIMES]]

Specifies how many times QTP reprompts the user if entered values fail editing. The run or request terminates if edit errors aren't resolved within the specified number (n) of reprompts.

When operating QTP in batch mode, REPROMPT is ignored because QTP prompts only once.

TIMES is used only for documentation.

Limit: 99 reprompts for interactive sessions; 1 prompt for batch jobs.

Default: Without a specification, QTP reprompts indefinitely.

TERMINATE [REQUEST|RUN]

Declares whether the request or run terminates if edit errors aren't resolved within the specified number (n) of reprompts.

When operating QTP interactively, the TERMINATE option without the REPROMPT option has no effect; QTP prompts indefinitely.

Default: TERMINATE RUN for batch and interactive sessions.

PROMPT [string] [n TIMES]

Displays a prompting message at execution-time.

string

Specifies the prompt message displayed at execution-time. If no prompt string is used, QTP uses the label assigned to the element in the data dictionary as a prompt string. If no label exists, QTP uses the name of the segment.

TIMES

Limits the number of accepted index values that users can enter. Invalid entries aren't counted. If omitted, QUIZ prompts until a null entry.

Limit: The number specified must be between 1 and 500.

RANGE [TOPROMPT string]|NORANGE

RANGE specifies that the user can enter a range of PARM values. NORANGE prevents the entry of range values.

Limit (**MPE/iX**): Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Default: NORANGE

TOPROMPT string

Specifies the string used to prompt for the ending or "to" value of the corresponding index at execution-time.

Limit: Valid when both PARM and RANGE have been specified.

Default string: If RANGE is specified and TOPROMPT is omitted, the upper limit of the range is prompted for with the string "Up to:". If no upper limit is entered, the CHOOSE statement is executed using an exact match of the value entered for the lower limit.

SEPARATOR char

Specifies the character that overrides the default separator character for dates, by default, the separator character specified by System Options, or if it isn't specified, a slash (/). The separator character separates the day, month, and year portions of a date element when it is entered.

If SEPARATOR is used and FORMAT isn't used, the specified separator character is used with the default date format. Date values can be entered either with or without separator characters.

Limit: Valid only for date items.

Default: A slash (/), unless otherwise specified by the SEPARATOR option of the SYSTEM OPTIONS statement in PDL.

SYSTEMVALUE [LOGICAL|SYMBOL]string-expression [RANGE|NORANGE]

Gives the ability to extract the values defined at the operating system level. Permits the use of values set at the operating system level.

Limit: The SYSTEMVALUE option cannot be used with any other CHOOSE option. If the SYSTEMVALUE cannot be translated, no records will be retrieved.

string-expression

Specifies the name of the logical name or symbol (**OpenVMS**) or system variable (**MPE/iX, UNIX, Windows**).

LOGICAL|SYMBOL (OpenVMS)

LOGICAL specifies the retrieval of an OpenVMS logical name. If the keyword LOGICAL is not explicitly used, a logical name is assumed.

SYMBOL specifies retrieval of a DCL symbol. Local symbols are retrieved before global symbols.

The standard OpenVMS logical name table search order, set by LNM$FILE_DEV, is used. Typically, the process table (LNM$PROCESS) is searched first, followed by the job table (LNM$JOB), the group table (LNM$GROUP) and finally the system table (LNM$SYSTEM).

RANGE|NORANGE

Specifies whether a sequence of values is interpreted as a range or a list.

Defaults: NORANGE (**MPE/iX, UNIX, Windows**) or LOGICAL NORANGE (**OpenVMS**).

value-set

Specifies a value, a series of values or a range of values. A value-set takes the general form:

value [TO value][[,] value [TO value]]...

If there are multiple records associated with a particular value, all such records are retrieved for processing.

Limit: The maximum number of values or value ranges is 500. The value-set option cannot be used with any other CHOOSE option. **MPE/iX:** Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Discussion

When the CHOOSE statement is used, QTP retrieves primary record-structure data records for the declared linkitem.

For a given linkitem, the CHOOSE options (case-expression-set, conditional-expression-set, PARM, SYSTEMVALUE, and value-set) are mutually exclusive.

The Difference Between the CHOOSE and SELECT Statements

The SELECT statement can also be used to retrieve a set of data records. However, the CHOOSE and SELECT statements aren't mutually exclusive. The CHOOSE statement always retrieves records by index value; the SELECT statement always reads records sequentially.

For more efficient performance, use the CHOOSE statement instead of the SELECT statement when possible. Instead of reading the primary record-structure sequentially, QTP reads records only by index value. This reduces processing time and increases efficiency.

If CHOOSE and SELECT are used in the same request, the CHOOSE statement is performed before the SELECT statement. Entering the CHOOSE statement by itself cancels any previous CHOOSE statements and associated EDIT statements.

Using Generic Retrieval

Include the partial segment value and the generic retrieval character (by default,@) in the statement within quotation marks. For example,

```
> CHOOSE LASTNAME "M@"
```

or

```
> CHOOSE LASTNAME "M@@"
```

The entry "M@" chooses all last names that start with the letter M. For example, M@ would match "Moffat" and "Morrissey" but not "Smith". The entry "M@@" chooses all last names beginning with the letter M through to the end of the file. M@@ would successfully match "Moffat", "Morrissey" and "Smith" and every other last name beginning with a letter greater than "M".

Retrieval by partial values isn't valid when you specify a series of values or a range of values using more than one generic retrieval character, as in

```
> CHOOSE LASTNAME "A@@" TO "B@@"
```

However, the entry

```
> CHOOSE LASTNAME "A" TO "B"
```

has a generic-like effect. This statement chooses names from the lowest value name beginning with A to the highest value name beginning with B.

Preventing Generic Retrieval

To prevent a user from selecting records using partial segment values and the generic retrieval character, include the NOGENERIC option. For example,

```
> CHOOSE LASTNAME NOGENERIC PARM
```

Now, an entry of "M@@" causes QTP to look for an index value of exactly M@@.

Responding to PARM Prompts

QTP waits three minutes for the user to respond to any prompt; otherwise, an exception error occurs and the run or request is terminated.

The user can enter only one value per input line.

A null entry (pressing [Enter]) in response to the first "from" prompt tells QTP to choose all the data records in a record-structure and continue processing, unless the NOTALL option is specified.

When the user runs QTP interactively, a null entry in response to the first CHOOSE prompt causes QTP to issue a message asking if the user wants to choose all the records.

If the user enters "y" or "yes", then QTP chooses all the records for that record-structure. If the user enters "n" or "no" or a null entry, then QTP doesn't choose any records.

No query is made when QTP is running in batch mode.

Interrupting Prompting

The user can interrupt prompting by pressing the following in response to a prompt: [Ctrl-Y] (**MPE/iX**) or [Ctrl-C] (**OpenVMS, UNIX, Windows**).

When the user does this, QTP issues a message asking if the user wants to continue processing:

- If the user responds "yes", QTP restarts processing of the request from the point of interruption.
- If the user responds "no", QTP terminates the execution of the current request.

Using Generic Retrieval in Response to PARM Prompts

For record-structures in indexed files and relational tables, the user can enter part of a value, along with the generic retrieval character (by default,@), in response to a PARM prompt. Generic retrieval is available only for character-type items.

Generic retrieval can be specified in two ways:

- a partial string followed by a single generic retrieval character selects all values beginning with that string
- a partial string followed by two generic retrieval characters selects from the string to high values

For example, if the user enters "M@" in response to the following prompt:

```
LASTNAME:
```

QTP retrieves all last names that begin with the letter M. The generic retrieval character instructs QTP to match the entry with all other values beginning with the specified letter (M). Likewise, the user could choose all last names beginning with MO, by entering MO@, or all last names beginning with the letter M through to the end of the file, by entering "M@@".

Sequence of Prompts

Prompting by the GLOBAL TEMPORARY statement occurs first. Prompts for CHOOSE and DEFINE statements occur in the sequence they are specified. Display strings appear next in the sequence in which they are specified in the request. For information about entering execution-time values for defined items, see (p. 73).

How QTP Selects Values

QTP selects values in one of two ways:

- If you specify a single value, QTP selects data records matching that exact value.
- If you specify a range, QTP selects any values within that range including the specified lower and upper range limits.

Editing Execution-Time Parameter Values

Use the EDIT statement to:

- validate entered values
- specify both a pattern and acceptable values for execution-time parameters.

When you use the EDIT statement with execution-time parameters, QTP checks values as they're entered:

- If an editing error occurs, QTP issues a message and repeats the prompt.
- If editing errors aren't resolved, then the QTP run or request is terminated. See the ON ERRORS option, on (p. 55).

Ascending/Descending Indexes

Although a record-structure may be indexed in ascending or descending order, you must specify the logical range from low values to high values. QTP retrieves the data records, processing them in ascending or descending order depending on how the record-structure is indexed.

If a record-structure is indexed in ascending order, the chosen records are returned from the lowest value to the highest value. If a record-structure is indexed in descending order, the same data is displayed in descending order.

A change in the collating sequence in the data dictionary doesn't affect the CHOOSE statement.

The CHOOSE statement can only be used for indexes; it uses the record-structure system's index information to determine which data records to process. If the primary record-structure is a relational table, any item in the record-structure can be used on the CHOOSE statement.

Retrieving Data Records

The CHOOSE statement searches for and retrieves stored values, not displayed values. If the item AMOUNT has an INPUT SCALE of 2, and you want to choose the data records with the value "$483.27", you must enter

```
> CHOOSE AMOUNT 48327
```

not

```
> CHOOSE AMOUNT 483.27
```

since the value is stored without any decimal point, leading sign, or trailing sign.

For more information about the INPUT SCALE option, see the ELEMENT statement, in Chapter 2, "PDL Statements", in the *PDL and Utilities Reference* book.

CHOOSE with Expressions

Expressions must be completely evaluated into a single value prior to choosing any records. Because the CHOOSE statement sets the retrieval criteria for the PRIMARY file and no records have been read when the CHOOSE is evaluated, record items cannot be used in the expressions.

At compile-time, PowerHouse creates all the combinations that could be used at run-time and the syntax is checked. You can see the text of the generated statements by using SET LIST SQL.

Examples

Using Ranges with the PARM option

This example demonstrates how to use ranges with the PARM option:

```
> ACCESS EMPLOYEES
> CHOOSE LASTNAME PARM RANGE &
>   PROMPT "ENTER THE STARTING VALUE FOR LASTNAME: " &
>   TOPROMPT "ENTER THE ENDING VALUE FOR LASTNAME: "
> GO
```

QTP displays these prompts at execution-time:

```
ENTER THE STARTING VALUE FOR LASTNAME:
ENTER THE ENDING VALUE FOR LASTNAME:
```

Entering a Sequence of Values

If you enter a sequence of values in response to the following prompts

```
ENTER THE STARTING VALUE FOR LASTNAME: WINDSOR
ENTER THE ENDING VALUE FOR LASTNAME: <CR>
ENTER THE STARTING VALUE FOR LASTNAME: BENNET
ENTER THE ENDING VALUE FOR LASTNAME: CUTHBERT
ENTER THE STARTING VALUE FOR LASTNAME: TRUMAN
ENTER THE ENDING VALUE FOR LASTNAME: <CR>
ENTER THE STARTING VALUE FOR LASTNAME: <CR>
```

QTP selects data records

- with only the last name "Windsor" or "Truman"
- with last names between "Bennet" and "Cuthbert" inclusive.

However, if there is a data record with a CUSTOMER_NAME of "Cuthbertson", QTP selects it as well. To avoid selecting "Cuthbertson" along with "Cuthbert", you must include a space at the end of "Cuthbert" when you respond to the "to" prompt.

Default Prompting by QTP

In the next example, only the PROMPT option is specified:

```
> ACCESS EMPLOYEES
> CHOOSE LASTNAME PARM RANGE &
>   PROMPT "ENTER LASTNAME: "
> GO
```

QTP must supply the prompt for the upper range limit at execution-time:

```
ENTER LASTNAME:
UP TO:
```

In the next example, neither the PROMPT nor the TOPROMPT option is specified:

```
> ACCESS EMPLOYEES
> CHOOSE LASTNAME PARM RANGE
> GO
```

QTP supplies the prompts at execution-time:

```
LASTNAME:
UP TO:
```

Using the TIMES Option

To indicate how many values can be entered, enter a number immediately following the prompt string (if you specified a string) or immediately following the prompt keyword (if you didn't specify a string). The number must be followed by the keyword TIMES.

This example

```
> CHOOSE LASTNAME &
>      PARM PROMPT "Enter Last Name: " &
>      3 TIMES
```

tells QTP to prompt for a last name a maximum of three times.

Using the EDIT Statement

To control the values that users can enter when responding to a CHOOSE statement, use the EDIT statement. This example

```
> CHOOSE EMPLOYEE &
>      PARM PROMPT "Enter Employee Number: "
> EDIT EMPLOYEE PATTERN "####"
```

restricts the user to entering employee numbers that contain four digits.

Using Expressions

Some examples of CHOOSE statements with expressions are as follows (Parentheses are required to ensure no ambiguity exists between the linkitem and defined items within the expression.)

```
> CHOOSE REPORTDATE (SYSDATE)
> CHOOSE REPORTDATE &
>   (STARTDATE TO ENDDATE &
>   IF STARTDATE <> 0 AND ENDDATE <> 0 &
>   ELSE SYSDATE)
> CHOOSE AREACODES (CASE OF PROVINCE &
>      WHEN "YK" : 403 &
>      WHEN "NT" : 403,604,709,819 &
>      WHEN "BC" : 604 &
>      WHEN "AL" : 403 &
>      WHEN "SA" : 306 &
>      WHEN "MA" : 204 &
>      WHEN "ON" : 807,705,613,416,519 &
>      WHEN "PQ" : 819,514,418 &
>      WHEN "NB" : 506 &
>      WHEN "NS" : 902 &
>      WHEN "PE" : 902 &
>      WHEN "NF" : 709 &
>          DEFAULT 0 TO 999)
```

Using the SYSTEMVALUE Option

One or more values may be specified using the SYSTEMVALUE option. Values are separated by commas. If you need to include a comma as part of the value, prefix that comma with a backslash (\). For example, if you want to choose the value "SMITH, JOHN", define the system value, CHOOSEVALUES, as follows:

MPE/iX:	`setvar CHOOSEVALUES "SMITH\,JOHN"`
OpenVMS:	`define CHOOSEVALUES "SMITH\,JOHN"`
UNIX:	`setenv CHOOSEVALUES "SMITH\,JOHN"`
Windows:	`set CHOOSEVALUES="SMITH\,JOHN"`

To define a single value, you can use a statement such as

MPE/iX:	`setvar LOG2 17`

OpenVMS:	`define LOG2 17`
UNIX:	`setenv LOG2 17`
Windows:	`set LOG2=17`

To define a list of values, the values must be expressed as a string, with commas separating values:

MPE/iX:	`setvar LOG3 "1,3,4"`
OpenVMS:	`define LOG3 "1,3,4"`
UNIX:	`setenv LOG3 "1,3,4"`
Windows:	`set LOG3="1,3,4"`

These values would be used as:

```
> CHOOSE PROJECT-NO SYSTEMVALUE "LOG3" NORANGE
```

This is equivalent to:

```
> CHOOSE PROJECT-NO PARM
> GO
Project-no: 1
Project-no: 3
Project-no: 4
Project-no: <CR>
```

Using Ranges with the SYSTEMVALUE Option

When using the RANGE option, both single values and ranges may be used. To indicate a single value, specify the value followed by two commas. The following specifies:

- the single value 1
- the range 3 to 5
- the single value 7
- the range 9 to 15

MPE/iX:	`setvar log4 "1,,3,5,7,,9,15"`
OpenVMS:	`define log4 "1,,3,5,7,,9,15"`
UNIX:	`setenv log4 "1,,3,5,7,,9,15"`
Windows:	`set log4="1,,3,5,7,,9,15"`

The SYSTEMVALUE option:

```
> CHOOSE PROJECT-NO SYSTEMVALUE "LOG4" RANGE
```

is equivalent to:

```
> CHOOSE PROJECT-NO PARM RANGE
> GO
PROJECT-NO: 1
UP TO: <cr>
PROJECT-NO 3
UP TO: 5
PROJECT-NO: 7
UP TO: <cr>
PROJECT-NO 9
UP TO: 15
PROJECT-NO <cr>
```

To choose all values for a segment, enter the value "@" or "@@".

COMMIT AT

Specifies when commits are done for transactions involving relational databases.

Default: COMMIT AT RUN

Syntax

COMMIT AT option

Options

COMMIT AT options		
FINAL	INITIAL	n TRANSACTIONS
REQUEST	RUN	sort-item
START OF	UPDATE	

FINAL

Within a request, the Update transaction is committed at the end of the last QTP transaction of the transaction set. At the end of the request, any Query and Update transactions that are still active are committed.

Limit: Only valid within a request.

INITIAL

Within a request, the Update transaction is committed before the start of the first QTP transaction of the transaction set. At the end of the request, any Query and Update transactions that are still active are committed.

Limit: Only valid within a request.

n TRANSACTIONS

Within a request, the Update transaction is committed at the end of the specified number of QTP transactions. One QTP transaction is defined as the completion of all the output actions in the transaction set. For example, if there are three output statements in a request, then a QTP transaction is considered complete when the output actions for all three of these records have been completed for a given transaction cycle. At the end of the request, any Query and Update transactions that are still active are committed.

REQUEST

Active Consistency transactions are committed at the end of each request. If the request doesn't finish successfully, relational transactions are rolled back; relational transactions in earlier requests are still committed.

RUN

Active Consistency transactions are committed at the end of the QTP run allowing an entire run to be treated as a whole unit.

If a later request isn't completed, relational transactions in earlier requests are rolled back. This option creates more locks for a longer period and requires more memory to maintain relational rollback information.

sort-item

Within a request, the Update transaction is committed at the end of the control break. At the end of the request, any Query and Update transactions that are still active are committed. A sort-item specifies a control break which occurs when the value of the sort-item changes. The sort-item is an item named in a SORT or SORTED statement in QTP.

Limit: Only valid within a request.

START OF sort-item

Within a request, the Update transaction is committed before the start of the following transaction group. At the end of the request, any Query and Update transactions that are still active are committed. A sort-item specifies a control break which occurs when the value of the sort-item changes. The sort-item is an item named in a SORT or SORTED statement in QTP.

Limit: Only valid within a request.

UPDATE

Within a request, the Update transaction is committed after every output action. At the end of the request, any Query and Update transactions that are still active are committed. The UPDATE option provides maximum concurrency, but only individual updates can be rolled back after failure.

COMMIT AT UPDATE and COMMIT AT 1 TRANSACTION are equivalent if there is only one OUTPUT statement in a request. If there is more than one OUTPUT statement, COMMIT AT UPDATE brackets each output with a single start and commit operation. COMMIT AT 1 TRANSACTION brackets all of the outputs with a single start and commit operation.

Discussion

The COMMIT AT statement specifies the timing of the commit to QTP.

A COMMIT AT statement prior to the first request statement sets the default commit timing for the complete run. If COMMIT AT RUN is specified, no other COMMIT AT statement may be issued.

When a shorter transaction duration is specified, commits are still performed at the higher-level commit breaks. With any commit option, a commit is performed at the end of the run. With all but COMMIT AT RUN, the transactions are committed at the end of each request.

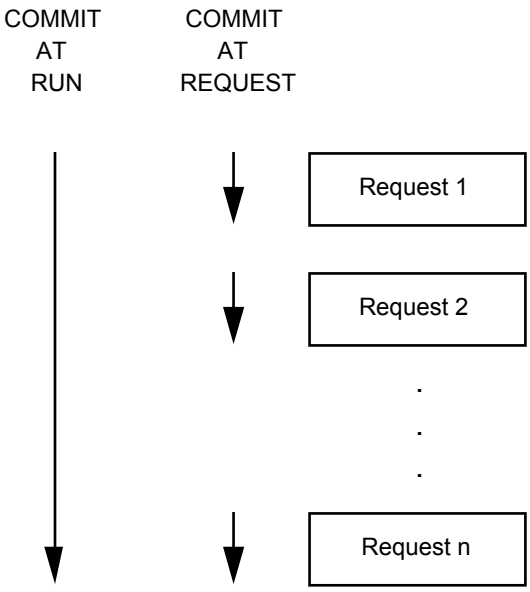
Transaction Models and Commit Frequency

The options on the COMMIT AT statement determine the transaction model and frequency of commits.

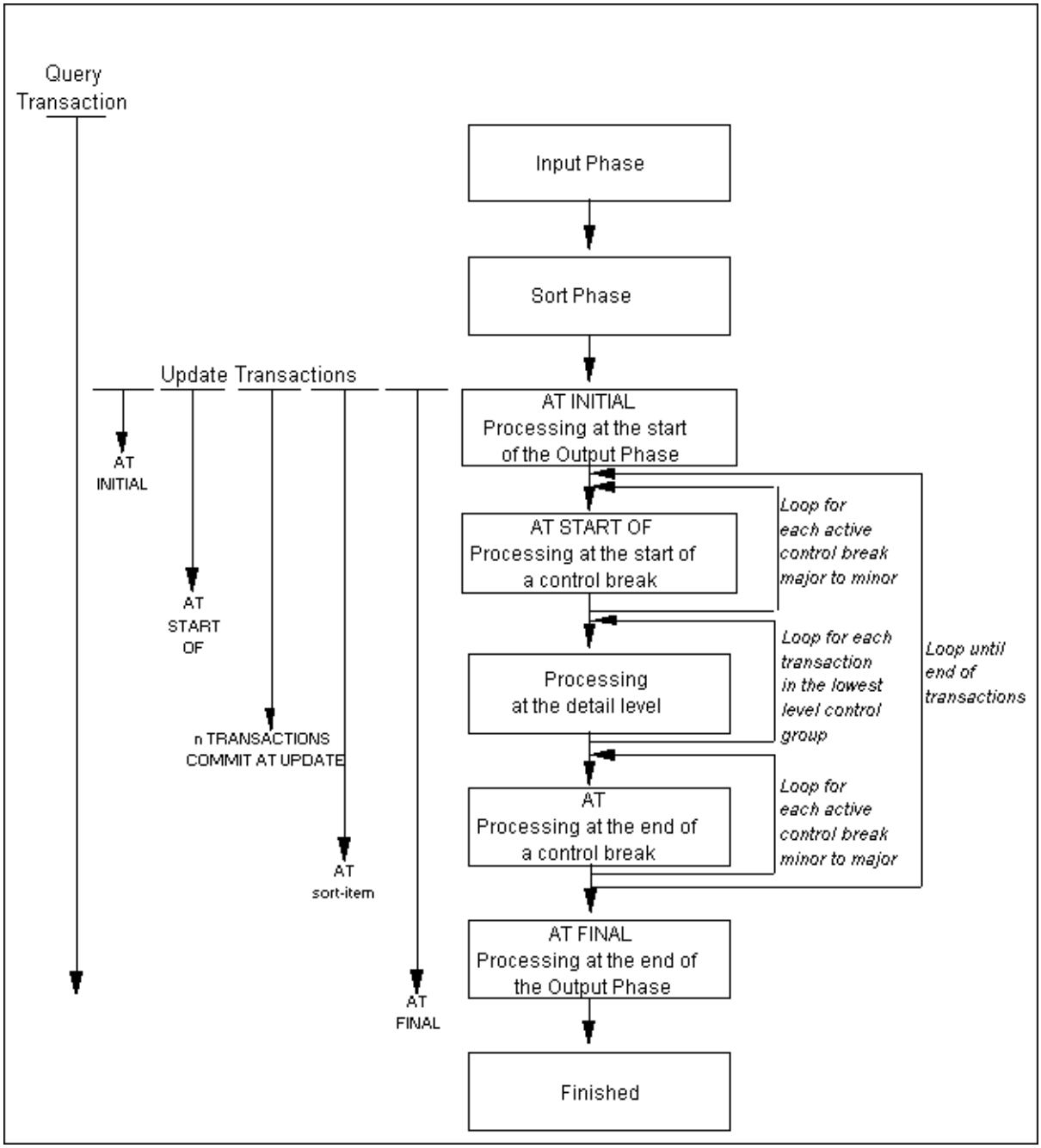
	Consistency Model	Concurrency Model
Commit Frequency	REQUEST	FINAL
	RUN	INITIAL
		TRANSACTIONS
		sort-item
		START OF sort-item
		UPDATE

The COMMIT AT options, REQUEST and RUN, use the Consistency model. All other options use the Concurrency model.

This diagram illustrates the commit frequency of the Consistency transaction.



Similarly, the following diagram illustrates the commit frequency of the Concurrency Transaction.



The Consistency Model

By default, QTP uses a read/write Consistency transaction for all database activity. The Consistency model provides high data consistency, at the cost of reduced concurrency.

In this model, there is a one predefined transaction, the Consistency transaction. The Consistency transaction is a read/write transaction with a high isolation level and is used for all application activities. The database performs checking for conflicts among concurrent users' updates.

As a result of the high isolation level used in this model, each user's data is protected from being changed by other users (though not necessarily guaranteeing that a user will be able to update). As a potential side effect of enforcing this level of isolation, database products may protect (lock) more than just the data that has been "touched" by the user, which may diminish other users' ability to access data concurrently.

The Concurrency Model

The Concurrency model provides multi-user access to data and full functionality, yet still enforces a fairly high level of consistency. In this model, multiple users can read data, but only one user can update the same record at a time. When a user updates, PowerHouse verifies that there is no conflict among concurrent updates by re-reading the record and comparing the checksum of the record to the checksum of the original record. The records are updated if the checksums are equal. This approach generally results in high concurrency since data is not locked until a user updates.

Calculated columns in relational tables are not included in the checksum calculation. If they were, the values could have been changed by the database, resulting in a checksum mismatch. Removing calculated columns from the checksum calculation eliminates these false errors.

In the Concurrency model, PowerHouse defines two predefined transactions:

- Query transaction
- Update transaction

QTP uses the Query transaction for retrieving and accessing records in the input phase, and the Update transaction to update records in the output phase.

During the output phase, each record that is to be updated is re-read on the Update transaction and compared to the record originally read on the Query transaction. If no change has occurred, the record is updated. Otherwise, an error message is issued and the update fails.

The Concurrency Model and ALLBASE/SQL

All ALLBASE/SQL activities are associated with a single Update transaction.

The Update transaction starts as soon as access to the database is required, and ends when data is committed.

For more information, see the sections, "The Concurrency Model in QUICK", in Chapter 2, "Relational Support in QDESIGN", and "The Concurrency Model in QTP", in Chapter 3, "Relational Support in QTP", in the *PowerHouse and Relational Databases* book.

The COMMIT AT Statement and SET LOCK

Use the COMMIT AT statement rather than the SET LOCK statement to control transaction duration.

Note: It is recommended that you do not use both statements in the same run.

If you use a SET LOCK statement before a COMMIT AT statement, the COMMIT AT statement takes precedence. QTP issues the following message:

```
*W* PREVIOUS COMMIT OPTION/SET LOCK OPTION OVERRIRDDEN FOR THIS RUN/REQUEST.
```

If you use a SET LOCK statement after a COMMIT AT statement, the SET LOCK statement only controls non-relational file locking.

The General Form of the COMMIT Option

Here is the general form that the various commit options take when used in a QTP run:

```
> RUN
>
> COMMIT AT 1 TRANSACTION
>
> REQUEST ONE
>      ACCESS RECORD_A
>      OUTPUT RECORD_A ADD UPDATE ALIAS RECORD_A_UPDATE
>      OUTPUT RECORD_B ADD
>      OUTPUT RECORD_C UPDATE
>
> REQUEST TWO
>      ACCESS RECORD_X
>      SORT ON ITEM_1 OF RECORD_X
>      COMMIT AT item_1
>      OUTPUT RECORD_Y DELETE
>
> REQUEST THREE
```

```
>       COMMIT AT REQUEST
>       .
>       .
>       .
>
> REQUEST FOUR
>       .
>       .
>       .
>
> GO
```

The first COMMIT AT statement sets the default transaction duration to one QTP transaction and the default transaction type to concurrency model.

In the first request, the Query transaction is used to read the records in RECORD_A. The Update transaction is started to re-read RECORD_A in the output phase to verify that no changes have occurred. After a record is written to each of RECORD_A, RECORD_B, and RECORD_C, the Update transaction is committed and QTP moves on to the next QTP transaction. At the end of the request, the Query transaction and, if still active, the Update transaction are committed.

In second REQUEST, the default commit timing is overridden for the duration of this request. The Query transaction is started at the first retrieval of RECORD_X. During the output phase, the Update transaction is started at the beginning of a control break and committed at the end of the control break.

In third REQUEST, a higher level of consistency is used specifically for that request. For this request, a Consistency transaction is used for all database operations. The transaction is started when the first access is required and committed at the end of the request.

In the last request, QTP reverts to Concurrency mode and uses a commit timing of COMMIT AT 1 TRANSACTION. When this request is complete, if either the Query or Update transactions are still active, they are committed.

[SQL] DECLARE CURSOR (query-specification)

Defines a set of data as a run-time view.

Syntax

[SQL[IN database]]
DECLARE name CURSOR FOR
query-specification [UNION [ALL] query-specification...]
[ORDER BY sort-specification]

IN database

Specifies the name PowerHouse uses to attach to the database. This is the name used to declare the database in PDL. For more information, see the section, "Setting the Database", in Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

name

Defines a logical name used to identify the set of data resulting from the query.

Limit: The name must be unique within the scope of the cursor. For a description of a cursor's scope, see the discussion section on (p. 69).

query-specification [UNION [ALL] query-specification...]

The query-specification defines a collection of rows that will be accessible when the cursor is opened.

The ALL option on the UNION option indicates that redundant duplicate rows are retained; otherwise, they are eliminated.

Parentheses are used in a union of three or more query specifications to enforce precedence in eliminating duplicate rows in the unioned sets. For example, a union of the three query-specifications X, Y, and Z, must be written as

(X UNION Y) UNION Z

or

X UNION (Y UNION Z).

For more information, see (p. 110).

[ORDER BY sort-specification]

The sort-specification syntax is:

{columnspec|n} [ASC|DESC][,{columnspec|n} [ASC|DESC]]...]

The columnspec must identify a column of the project-list. The default sort order is ascending.

The integer refers to the position of the column in the project-list. In the following example, the integer 2 refers to the derived column of averages.

```
> SQL DECLARE Y CURSOR &
> FOR SELECT SP.PNO, AVG(SP.QTY) &
>   FROM SP &
>   GROUP BY SP.PNO &
> ORDER BY 2
```

If the cursor definition involves a UNION, the sort specification may refer to column names if the corresponding column names of each query specification are identical; otherwise, the sort specification must reference an integer.

Discussion

Cursors declared before any RUN statement can be used throughout the remainder of the QTP session or until a CANCEL is encountered.

Cursors declared after a RUN, but before any REQUEST statement, can be used until the next RUN, CANCEL, or BUILD statement.

Cursors declared after a REQUEST statement can be used until the next REQUEST, RUN, CANCEL, or BUILD statement.

Example

The code from the initial SELECT up to and including the GROUP BY option is known as a query-specification.

```
> SQL IN EMPLOYEESDATABASE &
> DECLARE EMPSKILLS CURSOR FOR &
>   SELECT EMPLOYEES.ID, EMPLOYEES.FIRSTNAME, &
>   EMPLOYEES.LASTNAME, S.SKILL, &
>   FROM EMPLOYEES, SKILLS S &
>   WHERE EMPLOYEES.ID = S.ID &
>   AND EMPLOYEES.ID IN &
>    (SELECT ID FROM SELECTEDEMPLOYEES)&
>   GROUP BY EMPLOYEES ID
```

For more information about using cursors, see the section, "SQL Overview", in Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

[SQL] DECLARE CURSOR (stored procedure)

Calls a stored procedure or stored function from the specified database in the Input phase.

Syntax

```
[SQL [IN database]]
    DECLARE name CURSOR FOR
    CALL stored-procedure|stored-function
    [([ITEM] item [IN [OUT]]|[OUT]
        [,[ITEM] item [IN [OUT]]|[OUT]]...)]
    [ON ERROR CONTINUE|TERMINATE]
    [RETURNING return-parameter]
    [[RESULT] SET item [,item]...]
```

IN database

Specifies against which database the stored procedure is executed.

Limit: Stored procedure calls are valid for DB2, ODBC, Oracle, Oracle Rdb (declared as TYPE RDB in the dictionary), and Sybase databases. Stored function calls are valid only for Oracle databases.

DECLARE name

Defines a logical name used to identify the set of data resulting from the stored procedure.

CALL stored-procedure|stored-function

The name of a stored procedure or stored function in the database.

The syntax for a procedure name varies with the RDBMS. For information on a specific database system, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

([ITEM] item [IN [OUT]]|[OUT] [,[ITEM] item [IN [OUT]]|[OUT]]...)

Items which are passed to the stored procedure or Oracle stored function, or received from the stored procedure. Input parameters can be temporary, defined, or record items. Output parameters can be temporary or record items.

Blob items may also be used for both input and output parameters when calling an Oracle stored procedure or stored function.

Default: IN

IN

Specifies that the item is an input parameter.

IN OUT

Specifies that the item is both an input and output parameter. The changed values of the input/output parameters are available to PowerHouse when stored procedure execution is complete.

OUT

Specifies that the item is an output parameter. The changed values of the output parameters are available to PowerHouse when stored procedure execution is complete.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an SQL statement fails. If the TERMINATE option is in effect, processing ends. If CONTINUE is specified, the SQL error is ignored and the processing continues as if the error had not occurred.

Default: TERMINATE

RETURNING return-parameter

The return-parameter must be defined as a temporary or record item.

For Sybase, identifies the item that contains the return status from a stored procedure upon completion of the Sybase stored procedure.

For Oracle, identifies the item that contains the value returned by a stored function upon completion of the Oracle stored function.

Limit: Not valid for Oracle stored procedures. For Oracle stored functions, the return-parameter can be an integer, CHAR, or VARCHAR. For DB2, Sybase, and Microsoft SQL Server, the return-parameter must be defined as a 32-bit (4-byte) integer.

[RESULT] SET item [,item]...

The description of the result set returned from the stored procedure. Each item is defined using a name, datatype, and, optionally, its size.

name sql-datatype [**(n)**]

To identify the item datatypes that match your relational database datatypes, see "Relational PowerHouse Datatypes" in the *PowerHouse Rules* book.

Limit: This option is valid only for DB2, ODBC, Oracle, and Sybase. Only one result set can be returned from a stored procedure.

Discussion

Stored procedures and stored functions are collections of SQL statements and logic that are stored in a database. Calls to stored procedures can take input parameters from a calling program, and return values for output parameters to a calling program. A stored procedure in DB2, ODBC, Oracle, or Sybase may also return result sets. PowerHouse supports a single result set per execution of a stored procedure.

For information on stored procedures of specific database systems, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

Examples

In the following example, the DECLARE CURSOR statement declares a cursor, EMPSKILLS, for the stored procedure, SPEMPLOYEESKILLS, that returns a result set consisting of five items (ID, FIRSTNAME, LASTNAME, SKILL, and SKILLLEVEL).

```
> SQL IN EMPLOYEESDATABASE &
> DECLARE EMPSKILLS CURSOR FOR &
>  CALL SPEMPLOYEESKILLS(EMPLOYEEID IN, EMPCOUNT OUT) &
>  RESULT SET ID DECIMAL, &
>      FIRSTNAME VARCHAR(20), &
>      LASTNAME VARCHAR(20), &
>      SKILL CHARACTER(10), &
>      SKILLLEVEL FLOAT
```

This example declares a cursor for the stored procedure, spCheckPrice, in an Oracle database.

```
> SQL IN PartsDb_ORCL DECLARE Part_Price CURSOR FOR &
>   CALL spCheckPrice (PartNo int IN, Price int OUT)
```

DEFINE

Assigns a name to an expression or prompts for values at execution-time.

Syntax

DEFINE name [type[*n] [type-option]] =
{conditional-expression|case-processing|parm-processing}

name

Names the defined item.

Limit: 64 characters; must begin with a letter.

type *n

Establishes the physical format of the defined item.

type

Specifies the datatype for the defined item.

For more information about items, datatypes, and sizes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

*n

Specifies the size of the defined item.

For character items, *n specifies the maximum number of alphanumeric characters that the item can contain. For all datatypes that store numeric values, *n specifies the maximum number of digits that the numeric item can contain.

Limit: You can't specify a size (*n) when you define a date datatype (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, or ZDATE). The size of a DATE item is controlled strictly by the CENTURY INCLUDED | EXCLUDED option.

type-option

Specifies the type-options for the defined item.

The type-options are CENTURY, SIGNED, UNSIGNED, NUMERIC, and SIZE.

CENTURY INCLUDED|EXCLUDED

Specifies whether the year component of a date item includes a century prefix.

Limit: Valid only for DATE, JDATE, and PHDATE datatypes; must immediately follow the type in the statement.

Default: Determined by the dictionary.

NUMERIC

When following the ZONED datatype, indicates that the datatype is to have a type of ZONED NUMERIC rather than RIGHT OVERPUNCHED NUMERIC.

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED [WHEN POSITIVE]

Sets the range for INTEGER, PACKED, and ZONED. For datatypes PACKED and ZONED, unsigned items can store both positive and negative values. For datatype INTEGER, an unsigned item can store only positive values.

The following is a list of qualifications and exceptions for this option:

- This option is valid only for types INTEGER, PACKED, and ZONED and must immediately follow the type specification.

- For INTEGER, this option specifies whether PowerHouse interprets the number as a two's complement binary number (SIGNED) or as an absolute binary number (UNSIGNED).
- For PACKED and ZONED, this option specifies whether the item includes a sign (SIGNED) if positive or negative, or only when negative (UNSIGNED).
- The WHEN POSITIVE option is valid only for PACKED UNSIGNED or ZONED UNSIGNED, and is used only for documentation.

Default: UNSIGNED for ZONED; SIGNED for INTEGER and PACKED.

SIZE m [BYTES]

Specifies a storage size in bytes. Use SIZE m BYTES if specifying *n leads to an undesired default storage size.

If you use both *n and SIZE m, you must ensure that they don't conflict with each other.

BYTES is used only for documentation.

Limit: SIZE isn't valid for date datatypes (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE) or the item types NUMERIC and INTERVAL.

conditional-expression

A means of evaluating a series of expressions based on conditions. A means of evaluating a series of expressions based on conditions. A conditional-expression, when evaluated, results in the value of the defined item. The expression is calculated every time the defined item name is referenced during execution.

The general form is:

expression1 [IF condition1
[ELSE expression2 IF condition2]...
[ELSE expression3]]

When the IF option is used alone without ELSE and the condition isn't met, then numeric and date defined items are set to zero and character defined items are set to spaces.

For more information about conditional-expressions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Limit: Conditional-expressions, case-processing, and parm-processing can't be used together in any combination.

case-processing

Compares the value of an item against a known value or series of values. The comparison is calculated once for every transaction when the data to be evaluated is available.

The general form is:

CASE [OF] item
WHEN value-set|EXISTS|NULL|MISSING
{THEN|:} value|NULL|MISSING
[WHEN value-set|EXISTS|NULL|MISSING
{THEN|:} value|NULL|MISSING]...
[DEFAULT value|NULL|MISSING]

If there is a match, then the value specified after the THEN keyword is assigned to the defined item.

If there is no match, then the specified default is assigned. If no default is specified, then zeros or spaces are assigned.

When the defined item value is calculated based on the value of only one item, and those values are known, case-processing is more efficient than a conditional expression.

OF is used only for documentation. A colon (:) can be substituted for the THEN keyword.

value-set

Specifies a value, a series of values, or a range of values. A value-set takes the general form:

value [TO value][[,] value[TO value]]...

The values assigned to the defined item by CASE must be of the same type as the defined item. For example, if you create the defined item PROJECTNAME and specify that it is a character-type item, you must assign a string to the item PROJECTNAME:

```
> DEFINE PROJECTNAME CHARACTER*20 = &
>      CASE OF EMPLOYEES &
>         WHEN 1001 THEN "PRODUCTION" &
>         WHEN 1002 THEN "PROMOTIONS"
```

Limit: Case-processing, conditional-expressions, and parm-processing can't be used together in any combination.

parm-processing

Prompts for defined item values at execution-time. The defined item is evaluated only once, when the prompt is done. It takes the following general form:

PARM [parm-option]...

Limit: Only one value per input line can be entered. Parm-processing, case-processing, and conditional-expressions can't be used together in any combination.

The parm-processing options are:

parm-processing options		
DOWNSHIFT|UPSHIFT	FORCE|NOFORCE CENTURY	FORMAT
ON ERRORS	PROMPT	SEPARATOR

DOWNSHIFT|UPSHIFT

Shifts alphabetic letters to either lowercase or uppercase.

Limit: Valid only for datatype CHARACTER.

FORCE|NOFORCE CENTURY

FORCE CENTURY specifies that the user must enter a century on all century-included date fields. The option applies to century-included dates with four-digit year formats.

Default: The default depends on what is specified for the same option of the SYSTEM OPTIONS statement in PDL.

Limit: Valid only for century-included dates.

FORMAT date-format

Specifies the format for entering date values in response to PARM prompts.

You can specify alternative formats for date items using the DEFINE statement:

```
> DEFINE DATEJOINED DATE = PARM &
>    FORMAT MMMDDYY SEPARATOR "_"
```

Date values can be entered either with or without separator characters. A date-format can be one of the following:

Date-format	Example	Date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMDD	2001/MAY/23
YYMM	01/05	YYMMM	01/MAY
YYYYMM	2001/05	YYYYMMM	2001/MAY
YYDDD	01/125	YYYYDDD	2001/125
MMDDYY	05/23/01	MMMDDYY	MAY/23/01

Date-format	Example	Date-format	Example
MMDDYYYY	05/23/2001	MMMDDYYYY	MAY/23/2001
MMYY	05/01	MMMYY	MAY/01
MMYYYY	05/2001	MMMYYYY	MAY/2001
MMDD	05/23	MMMDD	MAY/23
DDMMYY	23/05/01	DDMMMYY	23/MAY/01
DDMMYYYY	23/05/2001	DDMMMYYYY	23/MAY/2001
DDMM	23/05	DDMMM	23/MAY
DDDYY	125/01	DDDYYYY	125/2001

YYYY - four digit year (e.g., 2001)

MM - two digit month (e.g., 05)

MMM - three character month name (e.g., MAY)

DD - two digit day for a month (e.g., 23)

DDD - three digit day for a year (e.g., 365)

Regardless of the output order of the date, the internal working format is YYMMDD (for dates without centuries), YYYYMMDD (for dates with centuries)

The FORMAT option governs data entry by determining the way you can enter date values. Dates can always be entered in the format specified in the FORMAT option, with or without the established separator character and with either the MM or MMM month format.

If the FORMAT option is used but the SEPARATOR option isn't, the only separator character that QUIZ accepts is the separator character specified by System Options, or if it isn't specified, a slash (/).

If a two-digit year is specified in the date format, applications won't accept a four-digit year. A two-digit year is represented by YY (for example, 01).

If a four-digit year is specified in the date format, you can only enter a two-digit year if you enter a separator character between the year and any adjacent numeric component of the date. The default century is added automatically.

Single-digit day and month entries are accepted if the user enters the separator character, as in 4/8/2001. An entry of 4AUG2001 is also allowed, because PowerHouse accepts a single-digit day entry if the middle value is a three-character month.

A three-digit day of the year from 1 to 366 is represented by DDD.

Although values for date items can be entered in a variety of formats, the values are always stored in either YYMMDD or YYYYMMDD form.

Limit: Valid only for date items. This option only affects the entry format; the display format isn't affected.

Default: YYYYMMDD for eight-digit dates; YYMMDD for six-digit dates.

ON ERRORS [REPROMPT [n [TIMES]]] [TERMINATE [REQUEST|RUN]]

States what action to take if errors are encountered. If both REPROMPT and TERMINATE are used, REPROMPT must appear first. When operating QTP interactively, TERMINATE without REPROMPT has no effect and QTP prompts indefinitely.

REPROMPT [n [TIMES]]

Specifies how many times QTP reprompts if the entered values fail editing. The run or request terminates if edit errors aren't resolved within the specified number (n) of reprompts. When operating QTP in batch mode, REPROMPT is ignored because QTP prompts only once.

TIMES is used only for documentation.

Limit: 99 reprompts for interactive sessions; 1 prompt for batch jobs.

Default: Without a specification, QTP reprompts indefinitely.

TERMINATE [REQUEST|RUN]

Declares whether the request or run terminates if edit errors aren't resolved within the specified number (n) of reprompts.

Default: TERMINATE RUN

PROMPT string

Displays a prompting message at execution-time.

Default: The name of the defined item.

SEPARATOR char

Specifies the character that overrides the default separator character (a slash, /) for dates. The separator character separates the day, month, and year portions of a date item when it is displayed.

If the SEPARATOR option is used but the FORMAT option isn't used, then the specified separator character is used with the default date format. Date values can be entered either with or without separator characters.

Limit: Valid only for date items.

Default: A slash (/), unless otherwise specified by the SEPARATOR option of the SYSTEM OPTIONS statement in the dictionary.

Discussion

The DEFINE statement calculates expressions or conditions that are calculated every time the defined item name is referenced during execution, except when it gets its value from a prompt, in which case, it is only evaluated once.

Limit: A combined maximum of 1023 defined and temporary items can be declared in a request. The maximum size of all input records plus all defined items used in the input phase is 32,767 bytes. The maximum size of all input and output records, and defined, temporary, and global temporary items in a request is 65,535 bytes.

Performance Implications of Using Defined Items

Defined items are calculated every time they are referenced, either directly or indirectly. For example, if a defined item which is referenced in turn references another defined item then both defined items will be re-evaluated. Be careful of cascading re-evaluation of defined items as this could have serious performance repercussions.

Responding to Prompts

The user has three minutes to respond to any prompt; otherwise, an exception error occurs and the run or request is terminated.

A null entry (pressing [Return]) in response to a prompt from a DEFINE statement results in zeros or spaces being assigned to the defined item.

Interrupting Prompting

The user can interrupt prompting by pressing the following in response to a prompt: [Ctrl-Y] (**MPE/iX**) or [Ctrl-C] (**OpenVMS, UNIX, Windows**).

When the user does this, QTP issues a message asking if the user wants to continue processing:

- If the user responds "yes", QTP restarts processing of the request from the point of interruption.

- If the user responds "no", QTP terminates the execution of the current request.

Sequence of Prompts

Prompting by the GLOBAL TEMPORARY statement occurs first, followed by prompting by the CHOOSE statement, and finally prompting by the DEFINE statement. Display strings appear next in the sequence in which they are specified in the request. For information about prompting for CHOOSE statement values, see (p. 52).

Editing Execution-Time Parameters

You can use the EDIT statement to specify both a pattern and acceptable values for execution-time parameters. When you use the EDIT statement with execution-time parameters, QTP checks values as they're entered. If a numeric or date item contains characters, QTP issues a message and repeats the prompt.

Default: If an editing error occurs, QTP reprompts users indefinitely unless the ERRORS REPROMPT option is specified. If editing errors aren't resolved within the specified number of reprompts, the QTP run is terminated.

Example

The following QTP run is designed to calculate the payment history of a client. The TEMPORARY statements are used so that RESET options can be used on the corresponding ITEM statements to reset the invoice and account values.

The first DEFINE statement creates an item named DAYSOUTSTANDING. This item's value is equal to the current date minus the date of the invoice. The DAYS function converts the date to a single numeric value.

The second DEFINE statement creates an item named AMOUNTDAYS. This defined item determines the value for the temporary item TOTALAMOUNTDAYS.

```
> RUN PAYHIST
> ACCESS ACCOUNTMASTER
> SORT ON ACCOUNTNO
>
> DEFINE DAYSOUTSTANDING = &
>   DAYS (SYSDATE) - DAYS (INVOICEDATE)
>
> DEFINE AMOUNTDAYS = DAYSOUTSTANDING * INVOICEAMOUNT
>
> TEMPORARY TOTALAMOUNT NUMERIC
> ITEM TOTALAMOUNT SUBTOTAL INVOICEAMOUNT &
>      RESET AT ACCOUNTNO
> TEMPORARY TOTALDAYS NUMERIC
> ITEM TOTALDAYS SUBTOTAL DAYSOUTSTANDING &
>      RESET AT ACCOUNTNO
> TEMPORARY TOTALAMOUNTDAYS NUMERIC
> ITEM TOTALAMOUNTDAYS SUBTOTAL AMOUNTDAYS &
>      RESET AT ACCOUNTNO
>
> OUTPUT ACCOUNTMASTER UPDATE AT ACCOUNTNO
>
> ITEM AVERAGEVALUE FINAL &
>      TOTALAMOUNTDAYS / TOTALDAYS
> ITEM AVERAGEDAYS FINAL &
>      TOTALAMOUNTDAYS / TOTALAMOUNT
>
> BUILD
```

[SQL] DELETE

Removes rows in a table in a database.

Syntax

[SQL [IN database] [sql-options] [IF condition]
[ON ERRORS {BYPASS UNTIL sort-item|FINAL}|REPORT|
{TERMINATE [REQUEST|RUN]}]]
DELETE FROM tablespec
[WHERE sql-condition|{DBKEY = :expression}]

SQL [IN database]

The name of the database. The database must be attached to the current data dictionary.

sql-options

Specifies the timing of SQL DELETE. DELETE is performed at the specified break.

The sql-options are AT FINAL, AT INITIAL, and AT [START [OF]].

AT FINAL

Performs the delete at the end of the transaction set after processing all transactions. This option has no effect unless there is an input phase.

AT INITIAL

Performs the delete at the beginning of the transaction set before processing any transactions. This option has no effect unless there is an input phase.

AT [START[OF]]sort-item

The sort-item option indicates that the delete is to take place at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group. This option is not available unless a request has a sort phase.

IF condition

Performs the delete only if the condition is satisfied. This option affects the processing sequence.

ON ERRORS

States what action to take if errors are encountered while performing the delete.

Default: TERMINATE RUN

BYPASS UNTIL sort-item|FINAL

Skips processing of all transactions until it reaches either the control break or the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

REPORT

Reports the error and skips processing of the transaction. Processing continues as if the error had not occurred.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run and rolls back any uncommitted updates.

DELETE FROM tablespec

The name of a table in a relational database from which rows are to be removed. The syntax for tablespec is:

[[server-name.]database-name.][owner-name.]table-name

If server-name is included in a Sybase tablespec, double quotes are required for the server-name and database-name. For example,

```
"DBSRV01.ACCNT".MANAGER.BILLINGS_TB1
```

For Oracle, the syntax is:

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
MANAGER."BILLINGS_TB1@DBLNK01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

WHERE sql-condition|DBKEY = :expression

The sql condition is a condition which is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book or refer to an SQL reference manual. This option provides a way to determine which rows will be deleted. Without it, all rows in the table are deleted. DBKEY is available only if the underlying database supports it.

Limit: DBKEY can't be used with Sybase.

Discussion

The DELETE statement acts directly on a table or view in the database and is never generated by PowerHouse. It can be used in a request or run, or entered directly at the QTP prompt.

DISPLAY

Displays a message.

Syntax

DISPLAY string

string

Specifies the message to be displayed.

Limits: 256 characters per string.

Discussion

The DISPLAY statement creates messages to be displayed at execution-time. Several DISPLAY statements can be included and can precede the ACCESS statement or be entered between a RUN statement and a REQUEST statement.

Example

In the following example:

- A $10.00 late charge is applied for each day the invoice is overdue.
- When the INVOICENO changes, an ACCOUNTDETAIL record is added which will include the amount of LATECHARGES for each INVOICE.
- The LATECHARGES for each INVOICE (within the ACCOUNT) are added to the BALANCE. This new BALANCE is updated when there is a change in the ACCOUNTNO.
- The DISPLAY statement reminds QTP users that adding these charges is the purpose of the run.

```
> RUN MONTHEND
> DISPLAY "********************************"
> DISPLAY "*     LATE PAYMENT CHARGES     *"
> DISPLAY "********************************"
> REQUEST LATE-PAYMENT-CHARGES
> ACCESS INVOICEMASTER
> SELECT INVOICEMASTER IF INVOICEBALANCE > 0 &
>  AND DUEDATE < SYSDATE
>
> SORT ON ACCOUNTNO ON INVOICENO
> TEMPORARY DAYSOVERDUE NUMERIC*3
>      ITEM DAYSOVERDUE = DAYS(SYSDATE) - DAYS(DUEDATE)
>
> TEMPORARY LATECHARGES
>      ITEM LATECHARGES = DAYSOVERDUE * 1000
>
> OUTPUT ACCOUNTDETAIL ADD AT INVOICENO
>   ITEM LATEAMT FINAL LATECHARGES
> OUTPUT ACCOUNTMASTER UPDATE AT ACCOUNTNO
>      ITEM BALANCE SUBTOTAL LATECHARGES  AT INVOICENO
>
> GO
Item ACCOUNTNO OF ACCOUNTDETAIL initial ACCOUNTNO OF INVOICEMASTER
Item INVOICENO OF ACCOUNTDETAIL initial INVOICENO OF INVOICENUMBER



**************************
*   LATE PAYMENT CHARGES   *
**************************
```

EDIT

Validates record items or entries made in response to a PARM prompt.

Syntax

EDIT {ALL|[FILE] {cursor|record-structure|
table} |[ITEM] data-item [options]...}

ALL

Specifies that all the items of all the records in the ACCESS statement list must pass the editing specifications declared in the data dictionary.

[FILE] cursor|record-structure|table

Specifies that all the items of the named cursor, record-structure, relational table or view must pass the editing specifications declared in the data dictionary. The record-structure must be part of the transaction as declared in the ACCESS statement.

[ITEM] data-item [options]

Specifies that the named data-item must pass the editing specifications declared in the data dictionary or in the EDIT statement itself. A data-item can be one of the following:

- a record item in a record-structure declared in the ACCESS statement.
- a column in a table or view or cursor declared in the ACCESS statement.
- a global temporary item that is prompted for in a GLOBAL TEMPORARY statement.
- a defined item that is prompted for in a DEFINE statement.
- an item that is prompted for in a CHOOSE statement.

Limit: If a subscript is used, only that occurrence can be edited in a given request; only one EDIT statement is allowed per item. If an array is used in an EDIT statement without a subscript, each occurrence in the array is edited and all occurrences must pass the edit.

Options

date-item options		
CENTURY INCLUDE|EXCLUDE	DATE|NUMERIC	DOWNSHIFT|UPSHIFT
INPUT	LOOKUP	PATTERN
VALUES		

CENTURY INCLUDE|EXCLUDE

Specifies whether the year component of a date item includes a century prefix.

Limit: Valid only for date items.

Default: Determined by the dictionary.

DATE|NUMERIC

Edits the item as if it were a date or numeric item.

If you specify a format that doesn't include the days (for example, YYMM) when data is being entered, the days are set to 00 rather than being ignored. The same holds true for date options that do not include the year (for example, DDMM). Internally, the year is set to 00. Therefore, date editing fails if the month portion of the date is 0, but passes if either the days or years portion is 0.

DOWNSHIFT|UPSHIFT

Shifts character items to either lowercase or uppercase. This option is applied before any other editing.

Limit: Valid only for character items.

INPUT [SCALE] n

States that a numeric-type item is multiplied by 10 raised to the negative power of the input scale value (n) for editing purposes only. This option is applied prior to any other editing.

SCALE is used only for documentation.

Limit: Valid only for numeric items.

LOOKUP [ON|NOTON] {cursor [sql-substitution...] | record-structure [lookup-option]...} [,[ON|NOTON] {cursor [sql-substitution...] | record-structure [lookup-option]...}]...

The LOOKUP option is used for editing. Each LOOKUP option is related to a specific source item to be edited. When a lookup is specified, QTP assumes that a particular item is to be used as the value of the lookup, whether or not the item matches the name of an index of the lookup record-structure.

cursor [sql-substitution...]

A cursor-name is the name of a cursor defined by the PowerHouse SQL DECLARE CURSOR statement.

An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default substitutions, WHERE and ORDERBY, will be inserted in generated SQL statements even if the corresponding substitution-variables do not exist on a DECLARE CURSOR statement.

The syntax for an sql-substitution is:

substitution-variable (text)

For more information about sql-substitutions and substitution-variables, see the section, "SQL Overview", in Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

Limit: Any sql-substitutions must appear before any other options.

ON|NOTON

ON specifies that the value of the item must exist as a value in the named record-structure. The record-structure must be declared in the data dictionary or in an attached relational database; it does not have to be declared otherwise in the QTP request. No data transfer takes place. When there is more than one index in the lookup record-structure, the item name must match, or VIA must be specified.

NOTON specifies that the value of the item must not exist as a value in the named record-structure.

Limit: Not valid for global temporary items.

Default: ON

record-structure

Names a record-structure. There are two forms of records:

record-structure [in file]

The name of a record-structure, and optionally, the name of the file to which it belongs. Both the record-structure and the file must be declared in the data dictionary. Including the filename adds clarity if the filename differs from the record-structure name, as can be the case in files with more than one record-structure.

table [IN database]

The name of a table or view in a relational database, and the name of the database to which it belongs. The database must be attached to the current data dictionary.

Limit: 31 record-structures in a single request; 63 record-structures in a single run. Record-structures and subfiles named in the lookup option are counted against the total open files in a run or request.

lookup-option

The lookup-options are USING, VIA, and VIAINDEX.

USING expression [,expression]

Lists expressions that result in values that are used to perform the lookup.

If the lookup record-structure is in a direct or relative file, there can be only one value that QTP interprets as a record number of the lookup record-structure. If the lookup record-structure is in an indexed file, or if it is in an IMAGE or Eloquence dataset with a B-Tree, OMNIDEX, or TPI index, the series of expressions declared must define a series of contiguous segments contained within the index. In this case, the first item is the first segment within the index, the second item is the second segment within the index, and so on.

If the lookup record-structure is a relational table or view, there can be several values in the USING option, which QTP interprets as values of columns in the table. VIA or VIAINDEX must be used to indicate to which columns the values belong only if more than one index, or no index, is specified.

When VIA is used in combination with USING, there must be a one-to-one correspondence between the USING values and the VIA items.

When you use VIAINDEX with USING, there can be as many USING values as there are segments in the index, or there may be fewer values than segments. In the latter case, values are matched to segments in order, starting from the first segment. Any leftover or unmatched segments aren't used.

If USING is not specified and there is one single-segment index, QTP tries to match the value of the edited item to a value of that segment. If there is more than one index, and no VIA or VIAINDEX is specified, QTP tries to match the value of the edited item to the value of a like-named first segment in any index. If the lookup record-structure has more than one index and none of the first segment names match the name of the item being edited, then VIA or VIAINDEX must be used to indicate which index QTP should use.

Limit: This option is not valid if the LOOKUP refers to a cursor.

VIA linkitem [,linkitem]

Items to be used to specify retrieval criteria. When VIA is used in combination with USING, there must be a one-to-one correspondence between the USING values and the VIA items.

Limit: This option is valid only for record structures in indexed files, IMAGE or Eloquence datasets with a B-Tree, OMNIDEX, or TPI index, and relational tables and views; it is not valid if the LOOKUP refers to a cursor.

VIAINDEX indexname

If you are using an indexed file, it designates the index to be used to perform the lookup. If you are using a relational database, it designates the items to be used to perform the lookup.

The VIAINDEX option must name an index of a record-structure in an indexed file or an index of a table or view.

When the VIAINDEX option is used with the USING option, there can be as many USING values as there are segments in the index, or there may be fewer values than segments. In the latter case, values are matched to segments in order, starting from the first segment. Any leftover or unmatched segments are not used.

Limit: This option is valid only for record structures in indexed files, IMAGE or Eloquence datasets with a B-Tree, OMNIDEX, or TPI index, and relational tables and views; it is not valid if the LOOKUP refers to a cursor.

PATTERN string|=string-expression

Specifies a string of characters and metacharacters that provide a general description of values. To be valid, the entry must match the values specified in the pattern string.

VALUES value TO value [[,] value [TO value]]...

Specifies acceptable entry item values.

Limit: Date values must be in YYYYMMDD format (if CENTURY INCLUDED is in effect for the item) or YYMMDD format (if CENTURY EXCLUDED is in effect for the item). Commas must separate value sets. Valid only for record-structures in indexed files and relational tables and views.

Discussion

The EDIT statement validates values that have been entered in response to a DEFINE or CHOOSE statement prompt.

If an execution-time parameter in a CHOOSE statement is edited with an EDIT statement, a subsequent CHOOSE statement cancels both the EDIT statement and the previous CHOOSE statement, even if the item is the same in both CHOOSE statements.

If the lookup refers to a cursor, QTP performs an SQL open, fetch, and close to perform the edit check. The results of the query are discarded.

Linkage

EDIT statement linkage is constructed on a one-to-one basis but no data is retrieved; only the presence or absence of a data record is determined, as in

```
> ACCESS ORDERDETAIL
> EDIT PARTSUFFIX LOOKUP ON ORDERMASTER &
>   USING ("1000" + PARTSUFFIX)
```

A call to the file system is done to find the record match but a call to retrieve the record is not done.

Data Dictionary Editing Specifications

The ALL and FILE options make use of data dictionary editing specifications only. The ITEM option can include additional editing criteria.

An option specified in an EDIT statement overrides an identical type of edit from the data dictionary, although other editing from the data dictionary still applies. Only one EDIT statement is allowed for each item. Dictionary-based editing doesn't apply to defined items.

Validating Execution-Time Parameters

You can validate entered values with the EDIT statement. When you use the EDIT statement with execution-time parameters, QTP checks values as they are entered. If a numeric or date item contains invalid characters, QTP issues a message and repeats the prompt.

If an editing error occurs, QTP reprompts you indefinitely unless you specified the ON ERRORS REPROMPT option. If editing errors aren't resolved within the specified number of reprompts, then the QTP run is terminated.

Limits

If an array is used in an EDIT statement without a subscript, each occurrence in the array is edited and all occurrences must pass the edit.

Example

This QTP request splits an old employee record (OLDEMPL) into two new record-structures, PAYROLL and EMPLOYEE. To verify that no bad data is transferred, the data from OLDEMPL must pass a series of edits.

In the following example, "ON EDIT ERRORS REPORT", signifies that when an error is encountered on an EDIT statement, the error is reported and processing continues.

```
> RUN TRANSFER
> REQUEST TRANSFER-EMPLOYEE-DATA ON EDIT ERRORS REPORT
> ACCESS OLDEMPL
>
> CHOOSE BRANCH PARM PROMPT "Enter a BRANCH ---> "     &
    ON ERRORS REPROMPT 3 TIMES
```

```
>
> EDIT BRANCH LOOKUP ON BRANCHES
> EDIT EMPLOYNO LOOKUP NOTON EMPLOY1 &
>   VIAINDEX EMPLOYEE, NOTON PAYROLLDATA &
>   VIAINDEX EMPLOYEE
> EDIT POSITIONCODE LOOKUP ON POSITION &
>   VIAINDEX POSITION
> EDIT MAILCODE PATTERN "((#####(-#####)<)|(^#^#^#))"
> EDIT SEX VALUES "M", "F"
> EDIT EMPLSALARY VALUES 1000000 TO 7500000
>
> OUTPUT EMPLOY1 ADD
>   ITEM EMPLOYEE INITIAL EMPLOYNO
>   ITEM DATEAPPOINTED INITIAL STARTDATE
>   ITEM POSTALCODE INITIAL MAILCODE
>
> OUTPUT PAYROLLDATA ADD
>   ITEM EMPLOYEE INITIAL EMPLOYNO
>   ITEM SALARY INITIAL EMPLSALARY
```

The following could be used to verify that the branch number of the employees file exists on the BRANCHES table:

```
> SQL DECLARE BRANCHLOOKUP CURSOR FOR &
> SELECT * FROM BRANCHES

> SQL DECLARE EMPLOYEES CURSOR FOR &
> SELECT * FROM EMPLOYEES

> ACCESS EMPLOYEES
> EDIT BRANCH LOOKUP ON BRANCHLOOKUP &
> WHERE (BRANCH=:BRANCH)
```

EXECUTE

Executes a compiled QTP run.

Syntax

EXECUTE filespec [INPUT limit-option] [PROCESS limit-option]

filespec

Specifies the file that contains the compiled QTP run that you want to execute.

OpenVMS, UNIX, Windows: The default file extension is .qtc.

INPUT limit-option

Where limit-option is:

[LIMIT] n

LIMIT n specifies the maximum number of QTP transactions (n) selected for processing. The INPUT option is most often used to set transaction limits for testing purposes. If the specified input limit is exceeded, no further input data records are read, but the next processing phase continues normally.

If the INPUT LIMIT and PROCESS LIMIT options are used together on the same statement, the limit with the lower value is used and sets the type of limit.

LIMIT is used only for documentation.

Limit: 2,147,483,647 transactions

Default: 10,000

NOLIMIT (OpenVMS, UNIX, Windows)

NOLIMIT removes any limit on the number of transactions that are selected for processing for each request in the run.

PROCESS limit-option

Where limit-option is:

[LIMIT]n

LIMIT n specifies the maximum number of transactions (n) to be processed for each request in the run. If the process limit is exceeded during the course of a run or request, no further processing is performed and all active relational database transactions are rolled back. The entire run is stopped, potentially causing partial updates.

If the PROCESS LIMIT and INPUT LIMIT options are used together on the same statement, the limit with the lower value is used and sets the type of limit.

LIMIT is used only for documentation.

Limit: 2,147,483,647 transactions

Default: 10,000

NOLIMIT (OpenVMS, UNIX, Windows)

NOLIMIT removes any limit on the number of transactions that are processed for each request in the run.

Discussion

The EXECUTE statement executes a compiled QTP run.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the EXECUTE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Specifying the INPUT LIMIT and PROCESS LIMIT Options

You can specify the INPUT LIMIT or PROCESS LIMIT option in the EXECUTE statement, as in

```
> EXECUTE BACKUP PROCESS LIMIT 20000
```

Using the INPUT LIMIT or PROCESS LIMIT option in an EXECUTE statement affects the entire run. If the options are specified in a combination of EXECUTE, REQUEST, and SET statements, the order of precedence is

EXECUTE

REQUEST

SET

If you specify both INPUT LIMIT and PROCESS LIMIT in the same statement, QTP uses the lower value and corresponding limit type. If both limits are specified on separate SET statements, the last one encountered by QTP sets the limit.

For more information about how QTP accesses compiled runs, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

The EXECUTE statement in the following example tells QTP to initiate a compiled run. All phases of the run execute, but only 75 records are processed.

```
> EXECUTE DAYUP INPUT LIMIT 75
```

EXIT

Ends a QTP session.

Syntax

EXIT

Discussion

The EXIT statement ends the QTP session and returns control to the operating system or to the invoking program.

The EXIT statement can be abbreviated to E, EX, or EXI.

The EXIT statement and the QUIT statement perform in the exact same manner.

GLOBAL TEMPORARY

Creates a global temporary item for the duration of the run that does not exist in the data dictionary.

Limit: There can be a maximum of 1023 global temporary items. The maximum size of all input and output records, and defined, temporary, and global temporary items in a request is 65,535 bytes.

Syntax

GLOBAL TEMPORARY name [type [*n] [type-option]]
[INITIAL conditional-expression]
GLOBAL TEMPORARY name [type [*n] [type-option]]
PARM [parm-option]...

name

Names the global temporary item.

Limit: 64 characters; must begin with a letter.

type [*n]

Establishes the physical format of the global temporary item.

type

Specifies the type and datatype for the global temporary item.

For more information about items, datatypes, and sizes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Default: NUMERIC

*n

Specifies the number of display digits for numeric items.

For character items, *n specifies the maximum number of alphanumeric characters that the item can contain. For all datatypes that store numeric values, *n specifies the maximum number of digits that the numeric item can contain.

Limit: You can't specify a size (*n) when you define a date datatype (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE). The size of a DATE item is controlled strictly by the CENTURY INCLUDED|EXCLUDED option.

type-option

Specifies the type-options for the defined item.

The type-options are CENTURY, SIGNED, UNSIGNED, NUMERIC, and SIZE.

CENTURY INCLUDED|EXCLUDED

Specifies whether the year component of a date item includes a century prefix.

Limit: Valid only for PHDATE, DATE and JDATE datatypes; must immediately follow the type in the statement.

Default: Determined by the dictionary.

NUMERIC

Indicates that the datatype ZONED is to have a type of ZONED NUMERIC rather than RIGHT OVERPUNCHED NUMERIC.

Limit: Valid only for ZONED datatypes.

SIGNED|{UNSIGNED [WHEN POSITIVE]}

Sets the range for INTEGER, PACKED, and ZONED. For datatypes PACKED and ZONED, unsigned items can store both positive and negative values. For datatype INTEGER, an unsigned item can store positive values only. The following is a list of qualifications and exceptions for this option:

- This option is valid only for types INTEGER, PACKED, and ZONED and must immediately follow the type specification.
- For INTEGER, this option specifies whether PowerHouse interprets the number as a two's complement binary number (SIGNED) or as an absolute binary number (UNSIGNED).
- For PACKED and ZONED, this option specifies whether the item includes a sign (SIGNED) if positive or negative, or only when negative (UNSIGNED).
- The WHEN POSITIVE option is valid only for PACKED UNSIGNED or ZONED UNSIGNED, and is used only for documentation.

Default: UNSIGNED for ZONED; SIGNED for INTEGER and PACKED.

SIZE m [BYTES]

Specifies a storage size in bytes. Use SIZE m BYTES if specifying *n leads to an undesired default storage size.

If you use both *n and SIZE m, you must ensure that they don't conflict with each other.

BYTES is used only for documentation.

Limit: SIZE isn't valid for date datatypes (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE) or the item types NUMERIC and INTERVAL.

INITIAL conditional-expression

Assigns the value of the expression at the beginning of the run.

INITIAL can reference

- constants
- logical functions
- system functions
- previously declared global temporary items

INITIAL can't reference

- predefined conditions
- defined items
- temporary items
- record items

because they have not been calculated or initialized yet.

Limit: INITIAL and PARM options can't be used in the same GLOBAL TEMPORARY statement.

PARM [parm-option]...

Prompts for the value of the global temporary item at execution-time.

parm-options		
DOWNSHIFT|UPSHIFT	FORCE|NOFORCE CENTURY	FORMAT
ON ERRORS REPROMPT	PROMPT	SEPARATOR

Limit: PARM and INITIAL options can't be used in the same GLOBAL TEMPORARY statement. Only one value per input line can be entered.

DOWNSHIFT|UPSHIFT

Shifts alphabetic letters to lowercase or uppercase.

Limit: Valid only for alphabetic letters.

FORCE|NOFORCE CENTURY

FORCE CENTURY specifies that the user must enter a century on all century-included date fields. The option applies to century-included dates with two or four-digit year formats.

Limit: Valid only for century-included dates.

Default: The default depends on what is specified for the same option of the SYSTEM OPTIONS statement in PDL.

FORMAT date-format

Specifies the format for entering and displaying date item values. Date values can be entered either with or without separator characters. A date-format can be one of the following:

Date-format	Example	Date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMDD	2001/MAY/23
YYMM	01/05	YYMMM	01/MAY
YYYYMM	2001/05	YYYYMMM	2001/MAY
YYDDD	01/125	YYYYDDD	2001/125
MMDDYY	05/23/01	MMMDDYY	MAY/23/01
MMDDYYYY	05/23/2001	MMMDDYYYY	MAY/23/2001
MMYY	05/01	MMMYY	MAY/01
MMYYYY	05/2001	MMMYYYY	MAY/2001
MMDD	05/23	MMMDD	MAY/23
DDMMYY	23/05/01	DDMMMYY	23/MAY/01
DDMMYYYY	23/05/2001	DDMMMYYYY	23/MAY/2001
DDMM	23/05	DDMMM	23/MAY
DDDYY	125/01	DDDYYYY	125/2001

YYYY - four digit year (e.g., 2001)

MM - two digit month (e.g., 05)

MMM - three character month name (e.g., MAY)

DD - two digit day for a month (e.g., 23)

DDD - three digit day for a year (e.g., 365)

Regardless of the output order of the date, the internal working format is YYMMDD (for dates without centuries), YYYYMMDD (for dates with centuries)

If the FORMAT option is used but the SEPARATOR option isn't, the only separator character that QUIZ accepts is the separator character specified by System Options, or if it isn't specified, a slash (/).

Single-digit day and month entries are accepted if the user enters the separator character, as in 4/8/2001. An entry of 4AUG2001 is also allowed, because PowerHouse accepts a single-digit day entry if the middle value is a three-character month.

If a two-digit year is specified in the date format, applications won't accept a four-digit year. A two-digit year is represented by YY (for example, 01).

A three-digit day of the year from 1 to 366 is represented by DDD.

If a four-digit year is specified in the date format, you can only enter a two-digit year if you enter a separator character between the year and any adjacent numeric component of the date. The default century is added automatically.

Dates can always be entered with either the MM or MMM month format. Although values for date items can be entered in a variety of formats, the values are always stored in either YYMMDD or YYYYMMDD form.

Limit: Valid only for date items. This option only affects the entry format; the display format isn't affected.

Default: YYYYMMDD for eight-digit dates; YYMMDD for six-digit dates.

ON ERRORS REPROMPT [n [TIMES]]

Specifies how many times QTP reprompts if the entered values fail editing. The run terminates if edit errors aren't resolved within the specified number (n) of reprompts. When operating QTP in batch mode, the REPROMPT option is ignored because QTP prompts only once.

TIMES is used for documentation only.

Limit: 99 reprompts when specified

Default: Without a specification, QTP reprompts indefinitely.

PROMPT string

Displays a prompting message at execution-time. If you don't specify a string, QTP uses the item's name as the prompt.

Defaults: Item name.

SEPARATOR char

Specifies the character that overrides the default separator character (a slash, /) for dates.

The separator character separates the day, month, and year portions of a date element when it is displayed. For example, the separator character "-" produces dates that are displayed as "91-05-25".

If the separator option is used and FORMAT isn't, then the specified separator character is used with the default date format. Date values can be entered either with or without separator characters.

Limit: Valid only for date items.

Default: A slash (/), unless otherwise specified by the SEPARATOR option of the SYSTEM OPTIONS statement in the dictionary.

Discussion

The GLOBAL TEMPORARY statement creates and defines a global temporary item that does not exist in the data dictionary. The statement remains in effect for the duration of the run.

When to Use the GLOBAL TEMPORARY Statement

The GLOBAL TEMPORARY statement must be entered between a RUN statement and the first REQUEST statement. Both the RUN and REQUEST statements are mandatory in this case.

Using the PARM Option

You can use the PARM option to enter values at execution-time. You can edit entered parameters with the EDIT statement. The EDIT statement can be entered after the GLOBAL TEMPORARY statement and before the first REQUEST statement.

Sequence of Prompts

Prompting by the GLOBAL TEMPORARY statement occurs first. Prompts for CHOOSE and DEFINE statements occur in the sequence they are specified.

Editing Execution-Time Parameter Values

You can use the EDIT statement to validate entered values.

When you use the EDIT statement with execution-time parameters, QTP checks values as they're entered:

- If a numeric or date item contains characters, QTP issues a message and repeats the prompt indefinitely unless the ON ERRORS REPROMPT option is specified.
- If editing errors aren't resolved within the specified number of reprompts, then the QTP run is terminated.

You must enter all EDIT statements for the GLOBAL TEMPORARY statements before the first REQUEST statement.

How ITEM Statements are Processed

Any item statements with the final option specified for global temporary items are completed after all transactions have been processed, and after all at final processing. They aren't completed if either

- the REQUEST isn't executed under conditional execution
- no transactions are processed

Example

The following example shows the use of the GLOBAL TEMPORARY statement in a QTP run that performs a backup. In this example:

- The GLOBAL TEMPORARY named BACKUP determines whether or not a request is executed. CHARACTER indicates that the item is one character in length. PROMPT determines that this one character is entered in response to the question of whether or not a request should be backed up.
- Some REQUEST statements in this run aren't executed if the response to this prompt is "P".

```
> RUN BACKUPS
> DISPLAY"****************************"
> DISPLAY "        B A C K U P         "
> DISPLAY "****************************"
> GLOBAL TEMPORARY BACKUP CHARACTER*1 PARM PROMPT &
> "Enter P or F for a Partial/Full backup --->" &
> ON ERRORS REPROMPT 3 TIMES UPSHIFT
> EDIT BACKUP VALUES "P", "F"
>
> REQUEST PHASE1 EXECUTE IF BACKUP = "F"
> ACCESS EMPLOY1
> SUBFILE EMPL91 INCLUDE EMPLOY1
>
> REQUEST PHASE2 EXECUTE IF BACKUP = "P" OR BACKUP = "F"
> ACCESS BILLINGS
> SUBFILE BILL91 INCLUDE BILLINGS

> REQUEST PHASE3 EXECUTE IF BACKUP = "F"
> ACCESS SALES
> SUBFILE SALES91 INCLUDE SALES
>
> REQUEST PHASE4 EXECUTE IF BACKUP = "P" OR BACKUP = "F"
> ACCESS PROJECTS
> SUBFILE PROJ91 INCLUDE PROJECTS
>
> GO
 ************************
         B A C K U P
 ************************
Enter P or F for a Partial/Full backup --->
```

GO

Initiates execution of a QTP run.

Syntax

GO

Discussion

The GO statement signifies that the specifications for the current QTP run are complete and executes the run.

The GO statement can't be used to run a compiled run that has already been created using the BUILD statement.

Example

The following example demonstrates the use of the GO statement in a test run that is designed to reset customer records at year-end:

```
> RUN YEAREND
> REQUEST ZEROCUST INPUT 10
>
> ACCESS CUSTOMERMASTER
>
> OUTPUT CUSTOMERMASTER UPDATE
>      ITEM YEARTODATE FINAL 0
>
> GO
```

[SQL] INSERT

Adds new rows to a table.

Syntax

```
[SQL [IN database] [sql-options] [IF condition]
    [ON ERRORS {BYPASS UNTIL
        sort-item|FINAL}|REPORT|
        {TERMINATE [REQUEST|RUN]}]]
    INSERT INTO tablespec
    [(column-name [,column-name]...)]
    query-expression|
        {VALUES (sql-expression|NULL
        [,sql-expression|NULL]...)
            [RETURNING DBKEY INTO :item]}
```

SQL [IN database]

The name of the database. The database must be attached to the current data dictionary.

sql-options

Specifies the timing of the SQL INSERT. The INSERT is performed at the specified break.

The sql-options are AT FINAL, AT INITIAL, and AT [START [OF]].

AT FINAL

Performs the insert at the end of the transaction set after processing all transactions. This option has no effect unless there is an input phase.

AT INITIAL

Performs the insert at the beginning of the transaction set before processing any transactions. This option has no effect unless there is an input phase.

AT [START[OF]]sort-item

The sort-item option indicates that the insert is to take place at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group. This option is not available unless a request has a sort phase.

IF condition

Performs the insert only if the condition is satisfied. This option affects the processing sequence.

ON ERRORS

States what action to take if errors are encountered while performing the insert.

Default: TERMINATE RUN

BYPASS UNTIL sort-item|FINAL

Skips processing of all transactions until it reaches either the control break or the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

REPORT

Reports the error and skips processing of the transaction. Processing continues as if the error had not occurred.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run and rolls back any uncommitted updates.

INSERT INTO tablespec [(column-name [,column-name]...)]

The name of a table in a relational database where new rows are to be added. The syntax for tablespec is:

[[server-name.]database-name.][owner-name.]table-name

UNIX, Windows: If server-name is included in a Sybase tablespec, double quotes are required for the server-name and database-name. For example,

```
"DBSVR01.ACCNT".MANAGER.BILLINGS_TBl
```

For Oracle, the syntax is:

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
MANAGER."BILLINGS_TBl@DBLNK01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

query-expression

The query-expression is a query-specification or the union of two or more query-specifications.

If a query-expression is used, multiple rows may be inserted into the table.

VALUES ({sql-expression|NULL }[,{sql-expression|NULL}]...)

There must be the same number of values following the VALUES keyword as there are columns in the table or in the column list, if specified. The values must be in the correct order. The first value is inserted in the first column, the second in the second column, and so on. If a column in the table is not assigned a value, the incidence of that column in the row will contain a null value.

The values must be consistent with the column's datatype.

RETURNING DBKEY INTO :item

Allows you to get the DBKEY of the inserted record so you can use it in a subsequent SQL UPDATE statement. DBKEY is available only if the underlying database supports it.

Limit: DBKEY is not supported by Sybase.

Discussion

The INSERT statement acts directly on a table or view in the database and is never generated by PowerHouse. It can be used in a request or run, or entered directly at the QTP prompt.

ITEM

Assigns values to record items, global temporary items, or temporary items.

Syntax

ITEM item [option]...

item

Specifies one of the following:

- a record item or a column of a relational table in a record-structure previously declared in an OUTPUT or SUBFILE statement
- a global temporary item previously declared in a GLOBAL TEMPORARY statement
- a temporary item previously declared in a TEMPORARY statement

Limit: Not valid for defined items.

Options

ITEM options		
=expression	AVERAGE	COUNT
FINAL	INITIAL	MAXIMUM
MINIMUM	OMIT	RESET
RESET AT INITIAL	RESET AT	NORESET
SUBTOTAL		

= expression [AT sort-item]

Assigns a value of the expression to the item for every transaction. AT specifies that the action is performed at the control break set by the named sort-item.

Limit: Not valid for subfile items.

AVERAGE item2 [AT sort-item] [IF condition]

Specifies that the subtotal of the named item and a transaction count are to be kept. When the item is referenced on output, the value is calculated by dividing the subtotal by the count. Both the subtotal and count are reset to zero when the target item is reinitialized.

AT specifies that the action is performed at the control break. IF specifies that the actions are performed only if the condition is satisfied.

Limit: Not valid for subfile items.

COUNT [AT sort-item] [IF condition] [NEGATIVE]

Adds 1 to the item for each transaction.

AT specifies that the action is performed at the control break. IF specifies that the action is performed only if the condition is satisfied. NEGATIVE subtracts, rather than adds, 1.

Limit: Not valid for subfile items.

FINAL expression

Assigns a value to the named item that is calculated by this expression whenever the record is updated.

Limit: Not valid for temporary items; valid for global temporary items, record items, and subfile items.

INITIAL expression

Assigns a value to the item, with the following qualifications:

- If the item is a record item, the value is assigned to new data records only when the data record is initialized. For example, INITIAL is only effective if ADD or ADD UPDATE is the current output action.
- If the item is a temporary item, the value is assigned at the beginning of the output phase.
- INITIAL is ignored when used with RESET.
- NORESET is ignored when used with INITIAL.

Limit: Not valid for subfile items.

MAXIMUM item2 [AT sort-item] [IF condition]

Specifies that the maximum value of either the item named in the ITEM statement or item2 is placed in the item for each transaction.

AT specifies that the action is performed at the control break. IF specifies that the action is performed only if the condition is satisfied.

Limit: Not valid for subfile items.

MINIMUM item2 [AT sort-item] [IF condition]

Specifies that the minimum value of either the item named in the ITEM statement or item2 is placed in the item for each transaction.

AT specifies that the action is performed at the control break. IF specifies that the action is performed only if the condition is satisfied.

Limit: Not valid for subfile items.

OMIT

Indicates that the item is a read-only column and excludes it from relational database inserts and deletes, as well as any checksum calculations. The OMIT option can be used in cases where PowerHouse cannot determine that the column is read only. For example, ODBC may not return sufficient information to determine the read status of a column. Read-only columns are columns that are updated or controlled by the database, such as computed columns or columns whose value is calculated by a stored procedure. While PowerHouse can use the values from such columns, it should not attempt to update them.

Including read-only columns in inserts or deletes can result in database errors. Including such columns in checksum calculations can result in update errors due to a checksum mismatch. When PowerHouse calculates the initial checksum on retrieval, it includes all columns not identified as read only. If the database changes the value of the column, its changed value will cause the checksum calculated on re-retrieval to be different than the one originally calculated.

The OMIT option must be the only option on the ITEM statement and the only option specified for the item. An error is issued if there are any other ITEM statements for the item. Specifying the OMIT option suppresses the generation of automatic item initialization for that column.

Limit: Ignored for non-relational items.

RESET [TO expression]
RESET AT INITIAL [TO expression]
RESET AT sort-item [TO expression]
NORESET

RESET initializes the record item, global temporary item, or temporary item named in the ITEM statement.

For record items, if no AT option is specified, the timing from the corresponding OUTPUT statement is used by default.

For global temporary and temporary items, RESET without an AT option is the same as RESET AT INITIAL.

reset at initial applies to initialization at the start of a QTP request only. The reset at sort-item initializes the item at each control break for the sort-item. If to is missing, the following occurs:

- If the item is numeric or date, it is initialized to zero.
- If the item is character, it is initialized to spaces.
- If MINIMUM is specified, RESET sets the target item to "high-values".
- If MAXIMUM is specified, RESET sets the target item to "low-values".

NORESET doesn't reinitialize an item. NORESET is ignored when used with INITIAL.

SUBTOTAL item2 [IF condition] [NEGATIVE] [,item3 [IF condition] [NEGATIVE]]...[AT sort-item]

Specifies that, for each transaction, the value of each named item is added to the value of the item named in the ITEM statement.

AT specifies that the action is performed at the control break. IF specifies that the action is performed only if the condition is satisfied. NEGATIVE subtracts the item's value, rather than adding it.

Limit: The maximum number of items that can be subtotaled by an ITEM statement using subtotal is 18. Not valid for subfile items.

Discussion

The ITEM statement assigns values to record items, global temporary items, or temporary items.

The ITEM statement can't be used until the item is declared.

The =, AVERAGE, COUNT, MAXIMUM, MINIMUM, and SUBTOTAL options are mutually exclusive. Summary operations sometimes give unexpected results when referencing columns in relational tables that allow NULL values. For more information about NULL values, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

The FINAL option cannot be used with temporary items. However, it is the only option valid with subfile items.

The INITIAL option is only effective when ADD or ADD UPDATE is the current output action.

If a string that is being assigned to a temporary item is too long, it will be truncated and a warning will be issued.

Initialization of Items

QTP automatically initializes items in new data records to the values of identically-named items in the ACCESS statement list. QTP also displays a message or series of messages regarding its initialization assumptions. If there is an ITEM statement for an item, QTP doesn't generate ITEM statements for automatic item initialization based on name matching to other record-structures. In all cases for non-relational items, QTP initializes items to spaces, zeros, and data dictionary initial values if the record buffer is initialized, as when adding a data record.

During item initialization relational items are initialized according to the following precedence:

1. If there is an ITEM INITIAL clause for the ITEM statement, the item is initialized to this value.
2. If there is no ITEM INITIAL clause, and an element exists in the dictionary that corresponds to this item which has an initial value, the item is initialized to the element initial value.
3. If neither of the above occur, the item is initialized to null if null is allowed for the item, or it is initialized to default values (spaces for character items and zero for numeric).

When initializing items of one relational record-structure based on the items of another relational record, a null value is copied, provided the target item allows null values. Otherwise, the item is initialized to default values.

When a non-relational data structure is initialized from a relational data structure and the source item has a null value, the non-relational item is initialized to default values (spaces for character items and zero for numeric).

Processing

When ITEM statements for global temporary and temporary items are mixed with ITEM statements for record items, the processing sequence must be considered.

For more information about output phase processing, see (p. 16).

Operating on Null Values in PowerHouse

As a general rule, if any term of an expression or any parameter of a function has a null value, the value of the expression or the function result is set to null. This applies to numeric expressions, to string expressions including concatenation, and to most data manipulation functions.

There are certain exceptions to this rule. In QTP, the summary and statistical operations AVERAGE, MAXIMUM, MINIMUM, and SUBTOTAL do not return the value null unless all of the values used in the calculation are null. Null values are not included in the computations.

Example

The following example demonstrates the use of the ITEM statement in a typical QTP run. This run has been designed to add a monthly interest charge to a set of invoices and, based on this charge, generate a new set of invoices. In this example:

- The ITEM statement for INVOICEDATE assigns the values of TRANSDATE of ACCOUNTDETAIL to the ITEM INVOICEDATE based on the type of record
- The ITEM statement for INVOICEBALANCE controls subtotaling for invoices. Values are added into INVOICEBALANCE (for "Invoices" and "Interest") and are subtracted from INVOICEBALANCE (for "Payment").
- RESET resets the value of INVOICEBALANCE to zero at every control break (which is at every new value of INVOICENO).
- The OUTPUT statement determines whether or not a payment is late. A new invoice detail record is added if the invoice-balance is greater than 0 and the invoice is overdue.
- The ITEM statement for AMOUNT tells QTP to multiply the amount of the item INVOICEBALANCE by .02 at update. The result is assigned to AMOUNT.
- The "type" of the ACCOUNTDETAIL item is finalized to "Interest", since the only interest calculation is being made to each outstanding invoice.

```
> RUN MONTHEND
> REQUEST ADDINTEREST
> ACCESS ACCOUNTMASTER LINK TO ACCOUNTDETAIL
> SELECT IF BALANCE > 0
> SORT ON ACCOUNTNO ON INVOICENO
> TEMPORARY INVOICEDATE DATE
>   ITEM INVOICEDATE = TRANSDATE OF ACCOUNTDETAIL &
>      IF TYPE OF ACCOUNTDETAIL = "INVOICE" &
>      RESET TO SYSDATE AT INVOICENO
>
> TEMPORARY INVOICEBALANCE NUMERIC*10
>
>   ITEM INVOICEBALANCE &
>      SUBTOTAL AMOUNT OF ACCOUNTDETAIL &
>      IF TYPE OF ACCOUNTDETAIL = "INVOICE" &
>      OR TYPE OF ACCOUNTDETAIL = "INTEREST", &
>      AMOUNT OF ACCOUNTDETAIL NEGATIVE &
>      IF TYPE OF ACCOUNTDETAIL = "PAYMENT" &
>      RESET AT INVOICENO
>
> OUTPUT ACCOUNTDETAIL ALIAS INTERESTREC &
>   ADD AT INVOICENO IF INVOICEBALANCE > 0 &
>   AND INVOICEDATE < DATE(DAYS(SYSDATE) - 30)
>
>   ITEM AMOUNT FINAL INVOICEBALANCE * 0.02
>   ITEM TYPE FINAL "INTEREST"
>   ITEM TRANSDATE FINAL SYSDATE
>
> OUTPUT ACCOUNTMASTER UPDATE AT ACCOUNTNO
```

```
>   ITEM BALANCE SUBTOTAL AMOUNT OF INTERESTREC AT INVOICENO
>
> BUILD
```

The ITEM statement for BALANCE ensures that the calculation of interest that was applied to the invoices of the ACCOUNTDETAIL record-structure is also added to the ACCOUNTMASTER record-structure. This amount is calculated at the invoice number and then added to the master file at ACCOUNTNO, as specified in the OUTPUT statement.

Assigning Null Values

You can use the INITIAL and FINAL options of the ITEM statement to assign a null value to an item, as in

```
> ITEM RELITEM INITIAL NULL
```

and

```
> ITEM RELITEM FINAL NULL
```

OUTPUT

Defines output record-structures and output actions.

Limit: You can access a maximum of 31 record-structures (including subfiles) in a request; 63 record-structures in a run. This includes the following: all files declared in the ACCESS statement, OUTPUT statements, SUBFILE statements, and files declared with the LOOKUP option in EDIT statements. The maximum size of all input and output records, and defined, temporary, and global temporary items in a request is 65,535 bytes.

Syntax

OUTPUT record|[*]subfilespec [option]...

record

Names a record-structure to which data is to be written.

There are two forms of records:

record-structure [IN file]

Names a record-structure and, optionally, the file to which it belongs. Both the record-structure and the file must be defined in the data dictionary. Including the file name adds clarity when the file differs from the record-structure name, as is the case with coded record-structures.

table [IN database]

Names the table or view in a relational database. The database is the name of a relational database declared in the data dictionary. It must be attached to the current data dictionary.

If you run QTP using the SEARCH option of the **subdictionary** program parameter, you can specify the name of the table or view, without the IN database qualifier. However, using the qualifier is always a good idea because it tells QTP where to look for the specified record-structure and therefore reduces QTP's searching time.

For more information about the **subdictionary** program parameter, see Chapter 3, "Resource File Statements", in the *PowerHouse Rules* book.

[*]subfilespec

Names an existing subfile or portable subfile minidictionary.

If the subfile hasn't been previously declared in the request or the subfile is being updated, you must use the asterisk (*) prefix.

If the subfile has already been declared in the ACCESS list, and it is referenced in the OUTPUT statement, you must use the ALIAS option to assign a unique name to the subfile for the remainder of the run.

Options

Specifies the output action to be performed. The effect of each action depends on the organization of the record-structure of the file.

The action options are ADD, ADD UPDATE, UPDATE, UPDATE ADD, and DELETE.

ADD

For record-structures in indexed and relational files, IMAGE Master and Detail datasets, adds the data record, otherwise, an error occurs. If indexed, relational, IMAGE Masters or indexed subfiles files have unique keys defined in the data dictionary, adding a record with the same key value causes a file system error. Appends the data record to the end of the file for record-structures in sequential and direct files, and non-indexed subfiles. For relative files (**MPE/iX**, **OpenVMS**), ADD appends records to the end of the file, or adds records at a specific record location if the USING option is included.

UPDATE

For indexed, relational, direct and relative (**MPE/iX**, **OpenVMS**) files, IMAGE Master and Detail datasets, and indexed subfiles, replaces the record-structure if it already exists; otherwise, an error occurs. For direct and relative (**MPE/iX**, **OpenVMS**) files not declared in the ACCESS statement, UPDATE requires a record number specified with the USING option. If the resulting record number doesn't correspond to an existing record in the file, an error occurs.

Limit: Not valid for sequential files.

DELETE

For indexed and relational files, IMAGE Master and Detail datasets, relative files (**MPE/iX**, **OpenVMS**) deletes the record if it already exists; otherwise, an error occurs. For relative (**MPE/iX**, **OpenVMS**) files not declared in the ACCESS statement, DELETE requires a record number specified with the USING option. If the resulting record number does not correspond to an existing record in the file, an error occurs.

Limit: Not valid for sequential or direct files, or subfiles.

ADD UPDATE|UPDATE ADD

For indexed, direct, relative (**MPE/iX**, **OpenVMS**), and relational files, IMAGE Master and Detail datasets or indexed subfiles, adds the record if it doesn't already exist for record-structures; otherwise the data record is replaced.

Limit: Not valid for sequential files.

This table specifies the type of files that are valid for the ADD, UPDATE, DELETE, and ADD UPDATE options on **MPE/iX**, **OpenVMS**, **UNIX** and **Windows**.

	MPE/iX	**OpenVMS**	**UNIX, Windows**
ADD	IMAGE master IMAGE detail indexed relational Appends: direct relative sequential subfiles	indexed relational Appends: direct relative sequential subfiles	indexed relational Appends: direct sequential subfiles
UPDATE	IMAGE master IMAGE detail direct indexed relational relative subfiles **Invalid:** sequential	direct indexed relational relative subfiles **Invalid:** sequential	direct indexed relational subfiles **Invalid:** sequential
DELETE	IMAGE master IMAGE detail indexed relational relative **Invalid:** direct sequential subfiles	indexed relational relative **Invalid:** direct sequential subfiles	indexed relational **Invalid:** direct sequential subfiles

	MPE/iX	OpenVMS	UNIX, Windows
ADD UPDATE	IMAGE detail IMAGE master direct indexed relational relative subfiles **Invalid:** sequential	direct indexed relational relative subfiles **Invalid:** sequential	direct indexed relational subfiles **Invalid:** sequential

Options

OUTPUT options		
ALIAS	AT FINAL	AT INITIAL
AT [START[OF]]	IF	INITIALIZE FROM
NOCHECK	NOITEMS	ON ERRORS REPORT
ON ERRORS TERMINATE	ON ERRORS BYPASS UNTIL	USING
VIA	VIAINDEX	

ALIAS name

Assigns an alternative name for the record-structure. When a record-structure is declared more than once in a QTP request, this option allows a unique identifier for each declaration. Once the alias is assigned, subsequent references to the record-structure must use this name.

Limit: Required when a record-structure named in the ACCESS statement has new data records added to it in the OUTPUT statement.

AT FINAL
AT INITIAL
AT [START [OF]] sort-item

Specifies the timing of the output action. This option affects the processing sequence.

FINAL

Performs the output action at the end of the transaction set after processing all transactions.

INITIAL

Performs the output action at the beginning of the transaction set before processing any transactions.

[START [OF]] sort-item

The sort-item option indicates that the output action is to take place at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group.

IF condition

Performs the action only if the condition is satisfied. This option affects the processing sequence.

INITIALIZE FROM record2

Initializes items using values from identically-named items in the named record-structure first. Items not initialized from the named record-structures can be initialized by using automatic initialization.

NOCHECK

States that checksums aren't calculated and that the record is not to be checked for changes that may have occurred since the record was first read. The use of the NOCHECK option is not generally recommended as it could compromise data integrity.

NOITEMS

Instructs QTP not to perform automatic initialization for items for this record-structure.

ON ERRORS REPORT
ON ERRORS TERMINATE [REQUEST|RUN]
ON ERRORS BYPASS UNTIL sort-item|FINAL

States what action to take if errors are encountered while performing the output action.

Default: TERMINATE RUN

REPORT

Reports the error and skips processing of the transaction for this output file. Processing continues as if the error had not occurred.

For relational databases, this means changes are still committed to the database unless other errors cause termination.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run.

For relational databases, updates are rolled back. If the COMMIT AT RUN statement has been specified, earlier requests are also rolled back; otherwise, only the current request (for the COMMIT AT REQUEST statement) or the current update (for the COMMIT AT UPDATE statement) is rolled back. For non-relational files, there is no rollback.

If the TERMINATE REQUEST option is specified, processing continues under a new transaction with the next request in the run.

Default: TERMINATE RUN

BYPASS UNTIL sort-item|FINAL

Skips processing of all transactions until it reaches either the control break or the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

For relational databases, this means changes are still committed to the database unless other errors cause termination.

USING expression[, expression]

Lists expressions that result in values that are used to retrieve records to be updated. If the output record-structure is in a direct file, there can be only one value that QTP interprets as a record number of the output record-structure. If the output record-structure is in an indexed file or IMAGE database, the series of items declared must define a series of segments contained within the record-structure. In this case, the first item is the first segment within the index, the second item is the second segment within the index, and so on.

QTP will default to use the primary index of an indexed file if none is specified. If that is not what is desired then VIA or VIAINDEX must be used to indicate which index QTP should use.

If the lookup record-structure is a relational table or view, there can be several values in the USING option, which QTP interprets as values of columns in the table. VIA or VIAINDEX must be used to indicate to which columns the values belong only if more than one index, or no index, is specified.

When you use VIAINDEX with USING, there can be as many USING values as there are segments in the index, or there may be fewer values than segments. In the latter case, values are matched to segments in order, starting from the first segment. Any left-over or unmatched segments are not used.

Limit (**MPE/iX**): For IMAGE indexes, all segments of the index must be specified, except for B-Tree and OMNIDEX indexes (in which case you can use the initial subset of segments of the index).

VIA linkitem [,linkitem]...

Items to be used to specify retrieval criteria. When VIA is used in combination with USING, there must be a one-to-one correspondence between the USING values and the VIA items.

Limit: For IMAGE indexes, all segments of the index must be specified, except for B-Tree and OMNIDEX indexes.

VIAINDEX indexname

Specifies the index to use for retrieval.

When VIAINDEX is used with USING, there can be as many, or fewer, USING values as segments in the index. When there are fewer, the values are matched to the segments in order, starting from the first segment. Any leftover or unmatched segments are not used.

Limit: VIAINDEX must name the index of the record-structure in the indexed file or the index of the relational table.

MPE/iX: The VIAINDEX option is ignored for IMAGE databases, unless the index is a B-Tree or OMNIDEX index.

Discussion

The OUTPUT statement declares record-structures that are updated in a QTP request.

Establishing Retrieval Criteria

When an output record-structure isn't also an input record-structure, output data records are retrieved in the output phase. In each case, only one data record is retrieved. Just as linkage between record-structures must be established in the ACCESS statement, the retrieval criteria must be established for record-structures read during the output phase.

QTP usually establishes the retrieval criteria using the same rules as the ACCESS statement uses for default linkage. QTP matches the output record-structure linkitem with items in the transaction. This matching establishes the index used for retrieval.

QTP attempts to establish retrieval criteria automatically when you enter an OUTPUT statement. If successful, QTP displays a message that details the assumptions it has made. If QTP can't automatically determine how to retrieve existing data records, it issues an error message.

When the linkage can't be established automatically, or if QTP's assumptions aren't what you intended, you specify retrieval criteria in the OUTPUT statement by using the VIA and USING options.

When updating a record-structure that doesn't appear in the ACCESS statement, you can specify the record to be updated using the initial subset of segments of an index. The initial subset of segments is specified with the USING and VIA options. QTP issues a warning that the linkage should be via a unique index. If more than one record in the output file has the specified values for the segments, then only the first record will be updated.

Example

The following example demonstrates the use of the OUTPUT statement in deleting and updating sales data records at the end of the year. This run is used to locate and delete data records for sales representatives whose contracts have terminated, identified by the standing of "T". The run erases the data records associated with the former employees and, since it is the beginning of a new year, sets the current year sales totals to zero.

In this example:

- The first OUTPUT statement updates data records for sales representatives whose STANDING isn't "T" (Terminated). UPDATE changes these data records at the control-break SALESREPCODE. The ITEM statements that follow this OUTPUT statement tell QTP what specific changes are to be made to the data.
- The second OUTPUT statement uses the DELETE option to erase all data records of terminated sales representatives.
- The third OUTPUT statement deletes all the data records from the subordinate record-structure, SALESDETAIL.
- The final OUTPUT statement adds a SALESHISTORY record after all transactions have been processed.

```
> RUN YEAREND
>
> REQUEST YEAR-END-PROCESSING
> ACCESS SALESREPMASTER LINK TO SALESDETAIL
>
> CHOOSE SALESREPCODE
> SORTED ON SALESREPCODE
>
> TEMPORARY TOTALSALES NUMERIC*8
> ITEM TOTALSALES SUBTOTAL THISYEARSALES
>
> OUTPUT SALESREPMASTER UPDATE AT SALESREPCODE &
>     IF STANDING <> "T"
>   ITEM PREVYEARSALES FINAL THISYEARSALES
>   ITEM THISYEARSALES FINAL 0
>   ITEM DATELASTMOD FINAL SYSDATE
>
> OUTPUT SALESREPMASTER DELETE AT SALESREPCODE &
>   ALIAS SALESMAST IF STANDING = "T"
>
> OUTPUT SALESDETAIL DELETE
>
> OUTPUT SALESHISTORY ADD AT FINAL
>   ITEM SALESYEAR FINAL FLOOR(SYSDATE / 10000) - 1
>   ITEM YEARLYSALES FINAL TOTALSALES
```

QSHOW

Runs QSHOW from QTP.

Syntax

QSHOW

Discussion

The QSHOW statement initiates a QSHOW session. QSHOW enables you to make quick online inquiries about entities (such as elements, files, and record-structures) in the data dictionary.

QSHOW is ready when its prompt character (-) appears.

When you exit from QSHOW, your session resumes at the point at which it was interrupted.

For more information about QSHOW, see Chapter 4, "QSHOW Statements", in the *PDL and Utilities Reference* book.

Example

The following example, shows how to access QSHOW from a QTP session in order to see which record-structures are available.

```
> ACCESS EMPLOYEES
> QSHOW
>
- SHOW FILES
  .
  .
  .
```

query-specification (SELECT)

Defines a collection of rows that will be accessible when the cursor is opened.

Query-specification is a component of the [SQL] DECLARE CURSOR statement which is documented on (p. 69).

Syntax

SELECT [ALL|DISTINCT] {*|project-list}
FROM tablespec [,tablespec]...
[WHERE sql-condition]
[GROUP BY columnspec [,columnspec]...]
[HAVING sql-condition]

The syntax for a subquery is the same as for a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery includes enclosing brackets.

ALL|DISTINCT

ALL indicates that duplicate rows are included. DISTINCT indicates that duplicate rows are eliminated.

Default: ALL

*

Selects all the columns from the specified tables.

project-list

A columnspec or derived column, or a list of columnspecs and derived columns separated by commas. If names are ambiguous, they must be qualified to ensure they can be identified uniquely.

The syntax for a columnspec is:

[table-name.|correlation-name.]column-name

The syntax for a derived column is:

expression [AS name]

The AS option assigns an alias to a column. It can be used

- to save typing because it is usually shorter than the column-name
- to uniquely identify multiple references to the same column
- to give a name to a derived column so it can be referenced in a program

For more information about expressions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

FROM tablespec [,tablespec]...

The FROM option identifies the tables where data in the project-list is retrieved. Rows can be retrieved from simple tables, derived tables, or joined tables. A derived table is a full query-specification including an optional ORDER BY option.

Joins are used to combine data from two or more tables based on the relationships between data in those tables. The type of join affects the rows retrieved by the query-specification.

If a correlation name is defined for the tablespec, subsequent PowerHouse references to the table must use the correlation name.

The general form of the tablespec syntax used in the following options is:

[owner.]table-name [correlation-name]

In addition to the general form of the tablespec, the following forms are also valid for this option:

(derived table) correlation-name

The correlation name must be defined for a derived table, and subsequent PowerHouse references to the derived table must use the correlation name.

tablespec CROSS JOIN tablespec

In a cross join, every possible combination of rows from the two tables being joined is created, without regard for any matching.

tablespec [INNER] JOIN tablespec
ON columnspec=columnspec
[AND columnspec = columnspec]...

In an inner join, a row is included in the result-set only if it has a matching row in the other table. The INNER keyword is for documentation only.

The ON option specifies the condition of the join.

tablespec LEFT|RIGHT|FULL [OUTER] JOIN tablespec
ON columnspec = columnspec
[AND columnspec = columnspec]...

An outer join includes all rows in the tables whether or not there are matching rows.

The left outer join returns rows from the table listed before the JOIN keyword, even if they don't have a matching row in the second table listed.

The right outer join returns rows from the table listed after the JOIN keyword, even if they don't have a matching row in the first table listed.

The full outer join returns rows from both tables listed, even if they don't have a matching row in the other table listed.

The ON option specifies the condition of the join.

[WHERE sql-condition]

The sql-condition defines linkage between tables in the query, and search criteria for rows to be retrieved. Only data which meets the criteria is available for use by PowerHouse.

The sql-condition is a condition which is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book, or refer to an SQL reference manual.

[GROUP BY columnspec [,columnspec]...]

This option rearranges the result-set into the minimum number of groups such that all rows within any one group have the same value for the GROUP BY columns. Rows that do not satisfy the WHERE option are eliminated before any grouping is done. The result is known as a grouped table.

To use the GROUP BY option:

- the grouping columns need not appear in the project-list
- aggregates in the project-list cannot be used in the GROUP BY option

```
> SQL DECLARE X CURSOR FOR &
> SELECT SP.PNO, MAX(SP.QTY), MIN(SP.QTY) &
>  FROM SP &
>  WHERE SP.SNO <> 'S1' &
>  GROUP BY SP.PNO
```

For detailed information about the GROUP BY option, refer to an SQL reference manual.

[HAVING sql-condition]

Eliminates groups, just as the WHERE option is used to eliminate rows.

The sql-condition is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book, or refer to an SQL reference manual.

```
> SQL DECLARE X CURSOR FOR &
> SELECT SP.PNO &
>  FROM SP &
>  GROUP BY SP.PNO &
>  HAVING COUNT(*) > 1
```

Limit: The sql-condition must evaluate to a single value per group.

Discussion

Specifying Selection Criteria Using WHERE Option and Substitutions

The WHERE option is used to define selection criteria for rows to be retrieved. Only data which meets the criteria is available for use by PowerHouse. In addition to specifying selection criteria within the query-specification, you can specify selection criteria on the following statements:

- ACCESS
- LOOKUP option on the EDIT statement

Examples

The following example demonstrates the way PowerHouse creates a single SQL query combining multiple conditions from multiple statements:

```
> SET LIST SQL
> SQL IN EMPBASE DECLARE X CURSOR FOR &
> SELECT * FROM EMPLOYEES &
>  WHERE CITY = 'BOSTON'
> SCREEN EMPLOYC
> CURSOR X WHERE(EMPLOYEE BETWEEN 1000 AND 5000) &
>  PRIMARY KEY EMPLOYEE
> ACCESS WHERE(POSITION = 'PRG') &
>  VIA EMPLOYEE ORDERED
```

The final query includes all three conditions specified in the WHERE options.

```
__ Sql after substitutions are applied:
__ SELECT *
__    FROM EMPLOYEES
__    WHERE POSITION = 'PRG' and
__          EMPLOYEE BETWEEN 1000 AND
__          5000 and
__          CITY = 'BOSTON'
```

Inner Joins

The following inner join would report all customers with matching invoices.

```
> SQL DECLARE X CURSOR FOR &
> SELECT * FROM CUSTOMER C &
>  INNER JOIN INVOICES I &
>  ON C.CUSTOMER_NUM=I.CUSTOMER_NUM
```

Outer Joins

The following example would report all customers even if they didn't have any invoices. Invoices without matching customers would not be reported.

```
> SQL DECLARE X CURSOR FOR &
> SELECT * FROM CUSTOMER C &
>  LEFT OUTER JOIN INVOICES I &
>  ON C.CUSTOMER_NUM=I.CUSTOMER_NUM
```

The next example would report all invoices even if they didn't have any matching customer information. Customers without invoices would not be reported.

```
> SQL DECLARE X CURSOR FOR &
> SELECT * FROM CUSTOMER C &
>  RIGHT OUTER JOIN INVOICES I &
>  ON I.CUSTOMER_NUM=C.CUSTOMER_NUM
```

QUIT

Ends a QTP session.

Syntax

QUIT

Discussion

The QUIT statement ends your QTP session and returns control to the operating system or the invoking program.

The QUIT statement can be abbreviated to Q, QU, or QUI.

The QUIT statement and the EXIT statement perform in the exact same manner.

REQUEST

Initiates a QTP request.

Syntax

REQUEST [name [option]...]

name

Names the QTP request.

Options

REQUEST options		
EXECUTE IF	SKIP IF	TERMINATE IF
INPUT	ON CALCULATION ERRORS	ON EDIT ERRORS
ON EXCEPTION ERRORS	PROCESS	

EXECUTE IF condition
SKIP IF condition
TERMINATE IF condition

Declares that the execution of a request is based on a condition. A condition is a logical test on a logical expression, logical-function, or predefined-condition that must be satisfied in order for a specified action to occur.

The condition can reference

- constants
- logical functions
- system functions
- previously declared global temporary items

The condition can't reference

- predefined conditions
- defined items
- temporary items
- record items

because they have not been calculated or initialized yet.

EXECUTE

Executes the request if the condition is satisfied.

SKIP

Bypasses the request if the condition is satisfied.

TERMINATE

Terminates the run if the condition is satisfied.

Limit: If prompting for execution-time parameters is also used, conditional execution can’t be used with the SET JOB statement.

INPUT limit-option

Where limit-option is

[LIMIT] n

LIMIT n specifies the maximum number of transactions (n) selected for processing for this request. The INPUT option is most often used to set transaction limits for testing purposes. If the specified input limit is exceeded during the course of the request, then no further input data records are read; the next processing phase continues normally.

If the PROCESS LIMIT and INPUT LIMIT options are used together in the same statement, the limit with the lower value is used and sets the type of limit. If both limits are specified on separate SET statements, the last one encountered by QTP sets the limit.

The INPUT LIMIT option is saved in a compiled report.

Limit: 2,147,483,647 transactions

Default: 10,000

NOLIMIT (OpenVMS, UNIX, Windows)

NOLIMIT removes any limit on the number of transactions that are selected for processing for this request.

The INPUT NOLIMIT option is saved in a compiled report.

ON CALCULATION [ERRORS REPORT]
ON CALCULATION [ERRORS TERMINATE [REQUEST|RUN]]

States what action to take when calculation errors are encountered. REPORT specifies that the error is reported, but processing is to continue as if the error had not occurred. TERMINATE specifies that the request or run is terminated.

A calculation error occurs when any of the following

- specified calculations
- summary operations
- evaluation of conditions
- evaluation of ITEM statements

results in

- invalid subscripts
- invalid data (such as an invalid number in a zoned item)
- integer overflow
- other numeric data exceptions

Default: The default error handling option in all cases is TERMINATE RUN.

ON EDIT [ERRORS REPORT]
ON EDIT [ERRORS TERMINATE [REQUEST|RUN]]

States what action to take when editing errors are encountered. REPORT specifies that the error is reported and the transaction bypassed after completion of editing, but otherwise processing is to continue as if the error had not occurred. TERMINATE specifies that the request or run is terminated at the end of the input phase of the request in which the editing error occurred.

Default: The default error handling option in all cases is TERMINATE RUN.

ON EXCEPTION [ERRORS TERMINATE [REQUEST|RUN]]

States what action to take when exception errors are encountered. An exception error is one that makes further processing of the run or request impossible. For those exception errors that occur at the run level, such as failure to open a file, the action is to report the error and terminate the run. For those exception errors that occur at the request level, such as exceeding the process limit, the action is to terminate either the run or the request.

Default: The default error handling option in all cases is TERMINATE RUN.

PROCESS limit-option

Where limit-option is

[LIMIT]n

Specifies the maximum number of transactions (n) selected for processing for this request. If the process limit is exceeded during the course of the request, no further processing is performed and all active relational database transactions are rolled back.

The ON EXCEPTION option determines whether or not later requests in the run are executed.

If the PROCESS LIMIT and INPUT LIMIT options are used together in the same statement, the limit with the lower value is used and sets the type of limit.

PROCESS LIMIT is saved in a compiled report.

LIMIT is used for documentation only.

Limit: 2,147,483,647 transactions

NOLIMIT (OpenVMS, UNIX, Windows)

The NOLIMIT option removes any limit on the number of transactions to be processed for this request.

PROCESS NOLIMIT is saved in a compiled report.

Discussion

The REQUEST statement starts a QTP request.

When to Use the REQUEST Statement

The REQUEST statement is optional on the first request of a QTP run unless a GLOBAL TEMPORARY statement precedes. The REQUEST statement is required for all requests except the first one.

Using INPUT LIMIT or PROCESS LIMIT on a REQUEST statement affects only that request. When used on a SET statement, they affect all requests from that point on until another SET statement specifies a new limit. When used on an EXECUTE or USE statement, they affect all requests in the compiled file being executed. If limits are specified on a combination of EXECUTE, USE, REQUEST, and SET statements, the order of precedence is as follows:

- EXECUTE or USE
- REQUEST
- SET

Example

```
> RUN TOTALS
> DISPLAY "************************"
> DISPLAY "*  BILLING   TOTALS     *"
> DISPLAY "************************"
> GLOBAL TEMPORARY TOTALTYPE CHARACTER*1 PARM PROMPT &
>  "Enter P or E for totals by Project or Employee: "
>
> REQUEST PROJECT-TOTALS EXECUTE IF TOTALTYPE = "P"  &
>   ON EDIT ERRORS REPORT &
>   PROCESS LIMIT 500
>
> ACCESS BILLINGS
> SORT ON PROJECT
> EDIT PROJECT LOOKUP ON PROJECTS
>
> TEMPORARY PROJECTTOTAL NUMERIC*8
> ITEM PROJECTTOTAL SUBTOTAL BILLINGAMOUNT RESET AT PROJECT
> SUBFILE PROJTOTS KEEP AT PROJECT &
>   INCLUDE PROJECT, PROJECTTOTAL
```

```
>
> REQUEST EMPLOYEE-TOTALS EXECUTE IF TOTALTYPE = "E" &
>   ON EDIT ERRORS TERMINATE RUN &
>   INPUT LIMIT 100
>
> ACCESS BILLINGS
> SORT ON EMPLOYEE
> EDIT EMPLOYEE LOOKUP ON EMPLOY1
>
> TEMPORARY EMPLOYEETOTAL NUMERIC*8
> ITEM EMPLOYEETOTAL SUBTOTAL BILLINGAMOUNT RESET AT EMPLOYEE
> SUBFILE EMPTOTS KEEP AT EMPLOYEE &
>   INCLUDE EMPLOYEE, EMPLOYEETOTAL
> BUILD
```

REVISE

Edits the current temporary save file or a specified file.

Syntax

REVISE [*|filespec [option]...]

*****

Signifies that the current temporary save file, QTPSAVE, is to be edited.

The source statement save file is a temporary file that PowerHouse opens at the beginning of a session.

All QTP statements you've entered since the last CANCEL CLEAR, SAVE CLEAR, or SET SAVE CLEAR statement are recorded in QTPSAVE as you enter them. However, CANCEL CLEAR, SAVE CLEAR, SET SAVE CLEAR, SAVE, and EXIT are not recorded in QTPSAVE.

The asterisk isn't required if you are editing the QTPSAVE file without changing any default options. However, the asterisk is required if you're overriding any of the default options, as in

```
> REVISE * NOLIST
```

filespec

The name of an existing permanent file. If this file does not contain QTP statements, use the NOUSE option so that QTP doesn't try to process the file when you exit from the system editor.

Options

The options are DETAIL, NODETAIL, LIST, NOLIST, USE, and NOUSE.

DETAIL|NODETAIL

DETAIL copies the contents of the revised file into the QTPSAVE file when you exit from the system editor; NODETAIL does not.

If you're revising a permanent file and the USE and NODETAIL options are in effect, then a USE statement is written to the current QTPSAVE file and invoked when you exit from the system editor.

Limit: NODETAIL is not valid with QTPSAVE, the temporary source statement save file.

Default: DETAIL

LIST|NOLIST

LIST displays the statements in the revised file as QTP processes them; NOLIST does not.

Default: LIST

USE|NOUSE

USE processes the revised statements when you exit from the system editor. NOUSE returns you to QTP at the point from which you left it without processing the revised statements.

Default: USE

Discussion

The REVISE statement indicates which file is to be edited, and, optionally, how the revised file is to act upon reentering QTP.

When you enter the REVISE statement without a filename, QTP automatically performs a CANCEL CLEAR prior to processing the statements. When you enter the REVISE statement with a filename, the automatic CANCEL CLEAR isn't performed.

The REVISE statement lets you use the system editor to edit either the QTPSAVE file or a permanent file from within QTP. The QTPSAVE file is edited by default.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the REVISE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Choosing a Different Editor

MPE/iX

By default, the REVISE statement uses EDIT/3000 as the system editor.

By default, the PowerHouse UDC uses the file equation:

```
:  FILE COGEDITR=EDITOR.PUB.SYS
```

If you want to use an editor other than EDIT/3000 when you invoke REVISE, change this file equation. For example, if you want to designate MYEDITOR as your editor, enter this:

```
:  FILE COGEDITR=MYEDITOR
```

The REVISE statement also uses a file equation for the file EDITTEXT, if it exists. For example, if you want to designate MYFILE as the input file to the editor, enter this:

```
:  FILE EDTTEXT=MYFILE
```

If you elect to use other editors, they must comply with the HP standard regarding the entry point BASICENTRY and the input file specification EDTTEXT.

OpenVMS

The REVISE statement invokes the DCL command assigned to the global symbol PHEDIT (usually used to designate an editor). By default, the SET POWERHOUSE command sets PHEDIT to

```
$PHEDIT :==EDIT/EDT
```

causing the REVISE statement to invoke the EDT editor.

You can change the default editor by changing the setting of the PHEDIT symbol. For example, to use the special interface to EDT called UTILITIES:EDT.COM, change the setting to

```
$PHEDIT :==@UTILITIES:EDT.COM
```

We recommend that you use either EDIT/EDT or EDIT/TPU as the setting for PHEDIT. In either of these cases, the editor can be called directly; otherwise, a subprocess is spawned.

If you intend to use the **nodcl** program parameter to restrict user access to the operating system, we further recommend that you do not select editors (such as TPU) that provide operating system access. When **nodcl** is in effect, users will continue to be able to access the system editor through the REVISE statement.

UNIX

By default, the REVISE statement uses the editor defined by the environment variable PHEDIT. If PHEDIT is not defined, the system checks the environment variable EDITOR. If you have not defined either of these variables, the REVISE statement fails.

Windows

By default, the REVISE statement uses the editor defined by the environment variable PHEDIT. If PHEDIT is not defined, the system checks the environment variable, EDITOR. The PowerHouse 4GL installation procedure sets PHEDIT to specify the Windows Notepad as the editor unless the PHEDIT environment variable is already set, in which case the setting is left as is.

RUN

Initiates a QTP run.

Syntax

RUN [filespec [USERS INCLUDE ALL|class [,class]...]]

filespec

Names the file in which the QTP run is stored when compiled.

USERS INCLUDE ALL|class [,class]...

Restricts the run to the application security classes listed. The application security class UNKNOWN can be listed as a class. The ALL security class includes all application security classes declared in the data dictionary, including UNKNOWN.

Limit: 1 to 64 classes. Application security class names must be declared in the data dictionary.

Discussion

The RUN statement starts a QTP run. A run is composed of one or more QTP requests.

The RUN statement is optional unless a GLOBAL TEMPORARY statement is included in the run.

Example

The following example demonstrates the use of the RUN statement in initiating a run that resets all expenditure data records to a value of zero. This run is limited to the application security class MANAGER.

```
> RUN YEAREND USERS INCLUDE MANAGER
>
> REQUEST ZEROCUST
> ACCESS CUSTOMERMASTER
> OUTPUT CUSTOMERMASTER UPDATE
> ITEM EXPENDRECORD FINAL 0
> GO
```

SAVE

Saves current QTP source statements in a file.

Syntax

SAVE filespec [CLEAR]

filespec

Names a file that will contain the QTP statements.

If QTP finds an existing file with the same name, it prompts for confirmation before creating a new version or overwriting the existing file. If SET NOVERIFY DELETE is in effect, no prompting takes place.

CLEAR

Removes any source statements in the temporary save file, QTPSAVE, once the contents are copied to a permanent file.

Discussion

The SAVE statement relates to QTP's temporary source statement save file, QTPSAVE. Statements are written to this file as you enter them. The SAVE statement itself is not included in the file.

The SAVE statement creates a permanent copy of QTPSAVE at the point where the SAVE statement was entered. You can use the saved contents as a source file for documentation and future changes, or as a working file for modification using the system editor. The saved statements can also be processed by QTP with the USE statement.

The CLEAR option clears the temporary save file after its contents have been saved so that you can enter and then save a new set of QTP statements in the same session. To clear the temporary save file without saving its contents, use the SET SAVE CLEAR statement.

Example

The following example demonstrates the proper use of the SAVE statement with the CANCEL statement. It is important to clear QTP's temporary save file when errors are made, using the CANCEL statement with the CLEAR option. A common error made when using the SAVE statement is neglecting to clear the temporary save file. The SAVE statement saves everything, including invalid statements.

If you have not cleared the file before you save statements to a permanent file, you might inadvertently include statements that you don't want. In the following example, only the statements following the CANCEL CLEAR statement will be saved in the file YEAREND:

```
> RUN YEAREND
> REQUESTCUST_ADVANCE
> ACCESS BRANCHES LINK TO BILLINGS AND TO EXPENSES
> CANCEL CLEAR
>
> RUN YEAREND
> REQUEST CUST_ADVANCE
> ACCESS BRANCHES LINK TO BILLINGS AND TO SALESREPS
>
> SAVE YEAREND
```

SELECT

Applies selection conditions to records and transactions.

Syntax

SELECT IF condition

SELECT [FILE] record-structure IF condition

SELECT IF condition

Establishes a condition that must be satisfied for the transaction to be processed.

SELECT [FILE] record-structure IF condition

Applies a condition to a data record as it is read. Specifies that if the selection condition is satisfied, the data record is included in the transaction. If the condition is not satisfied, the data record, not the transaction, is bypassed, and the next data record is read before the transaction is constructed.

Only one SELECT record-structure statement can be in effect for each record-structure named in the ACCESS or OUTPUT statement. The condition specified in the SELECT record-structure statement may be based only on the items in the files specified in the ACCESS statement list prior to and including the file specified in the SELECT statement. For instance, only items from files A and B may be specified for the following example:

```
ACCESS A LINK TO B LINK TO C
SELECT B IF condition...
```

It is recommended that items occurring in more than one file be qualified with the OF clause. This is because the record structure in the SELECT file IF is searched first, followed by the record structures in the ACCESS statement in first to last order.

Discussion

The SELECT statement defines selection conditions that are applied against the record-structures in the ACCESS statement.

Only one SELECT statement without a record qualifier can be used in a request and its selection criteria are applied against all record-structures in the ACCESS statement. QTP attempts to evaluate the condition as the transaction is built. As soon as the condition fails, building of the transaction stops and the next transaction begins.

Use caution when using the SELECT IF statement with a condition involving items from record-structures in a parallel relationship. Since the transaction is abandoned if data records cannot be retrieved, potentially significant data can be lost.

The SELECT record-structure IF applies to both the input and the output phases of QTP. The input phase can use both forms of the SELECT statement (SELECT IF and SELECT record-structure IF). The output phase uses only SELECT record-structure IF, which must be positioned after the OUTPUT statement.

In every request, one SELECT statement with a record qualifier can be included for each record-structure named in an ACCESS or OUTPUT statement. The condition is applied to each data record of the named record-structure as it is read. If the condition is not satisfied, the data record, not the transaction, is bypassed and the next data record is read.

Both SELECT statements ignore trailing blanks in any selection values. This means that the SELECT statement values "1000 " and "1000" are treated as the same for retrieval. This is consistent with the way in which relational systems return data.

If the SELECT statement references a record-structure whose name is included in both the ACCESS and OUTPUT statements, it is applied to the file as read by the ACCESS statement.

Speed and Efficiency of Execution

The order of the record-structures in the ACCESS statement affects the speed of execution. Conditions applied to record-structures that appear earlier in the ACCESS statement result in lower execution-times than those applied to record-structures that appear later in the ACCESS statement.

The form of the SELECT statement used can affect processing efficiency. In this example

```
> ACCESS EMPLOYEES LINK TO SKILLS
> SELECT IF PROVSTATE="ON" AND SKILL="ACCOUNTING"
```

QTP makes more evaluations than in this example:

```
> ACCESS EMPLOYEES LINK TO SKILLS
> SELECT EMPLOYEES IF PROVSTATE = "ON"
> SELECT SKILLS IF SKILL = "ACCOUNTING"
```

When possible, use the SELECT statement with a record qualifier. The SELECT statement without the record qualifier is not recommended for subordinate record items in parallel detail relationships. This is because the entire record complex is lost along with possibly significant data from other subordinate data records in the parallel relationship. You can avoid this situation by using the SELECT record IF statement instead.

In general, record-structures upon which selection conditions are based should be specified as early as possible in the ACCESS statement. This permits QTP to apply selection conditions to records in input record-structures as the transaction is built, often eliminating unwanted transactions before completion.

Differences Between the CHOOSE and SELECT Statements

The CHOOSE statement can also be used to retrieve specific data records. The SELECT statement always reads records sequentially; the CHOOSE statement retrieves records by indexes. The CHOOSE statement restricts a QTP request to primary data records with certain index values. However, the SELECT and CHOOSE statements are not mutually exclusive. If the SELECT and CHOOSE statements are used in the same request, the CHOOSE statement is performed before the SELECT statement. If selection conditions are based on an index of a primary record-structure, use the CHOOSE statement to eliminate unnecessary reads.

When a new collating sequence is defined in the data dictionary, the CHOOSE and SELECT statements perform differently from one another. Although the CHOOSE and SELECT statements seem to be asking for the same thing, the results may actually be different. The changed collating sequence has no impact on the records retrieved with the CHOOSE statement, but does affect the data records you can retrieve with SELECT.

For example, if the order of the standard English alphabet is reversed in the data dictionary, and you enter the following syntax:

```
> SELECT IF LASTNAME > "A"
```

no data records are selected.

However, if you enter this syntax

```
> CHOOSE LASTNAME "A@@"
```

all data records are selected. For more information, see (p. 52).

Example

The following example extracts information from a large inventory file. The output is sent to a subfile for later reporting in QUIZ.

```
> RUN QUIZREC
> ACCESS INVENTORY LINK TO INVOICE-NO OF BILLINGS
> DEFINE BRANCH CHARACTER*2= PARM PROMPT "BRANCH NO: "
> DEFINE DATE1 DATE= PARM PROMPT "START DATE: "
> DEFINE DATE2 DATE= PARM PROMPT "END DATE: "
> DEFINE CUSTOMNO CHARACTER*9=PARM PROMPT "CUSTOMER NO: "
>
> SELECT IF BRANCH = BRANCHNAME OF BILLINGS AND &
>    CUSTOMNO = CUSTOMER OF BILLINGS AND &
```

```
>    BILLINGDATE >= DATE1 AND &
>    BILLINGDATE <= DATE2
>
> SORT ON BRANCH
> SUBFILE INVEN2 INCLUDE INVENTORY
> BUILD
```

The SELECT statement names the items that QTP uses as selection criteria when retrieving data records. In this instance, the value of the item BRANCH that was prompted for in the DEFINE statement must be a value that can be found as a valid branch name in the BILLINGS record-structure. The same is true for the item CUSTOMNO. The QTP user's responses to the PARM prompts for the items DATE1 and DATE2 are used in the SELECT statement to determine the selection of date parameters as part of each retrieval.

SET

Overrides default QTP settings.

Syntax

SET DEFAULT|options...

DEFAULT

Resets all the set options, except SET DICTIONARY and SET FILE OPEN, to the following default values:

COMPILE	LIST
LOCK BASE RUN (**MPE/iX**)	LOCK FILE RUN (**OpenVMS, UNIX, Windows**)
LOGGING (**MPE/iX**)	NOJOB
NOLIST SQL	NOPRINT
NOVERIFY ERRORS	PROCESS LIMIT 10000
REPORT TERMINAL ERRORS	SCRATCH AUTO
STACKSIZE 400	STATISTIC
VERIFY DELETE	WARNINGS

The DATABASE and DOWNSHIFT|UPSHIFT|NOSHIFT options are reset to what is specified in your current dictionary.

If SET DEFAULT is entered within a run, SET STACKSIZE isn't reset to the default.

Options

SET options		
COMPILE|SYNTAX	DATABASE	DICTIONARY
DOWNSHIFT|UPSHIFT|NOSHIFT	FILE	INPUT
JOB|NOJOB	LIST|NOLIST	{LIST|NOLIST} SQL
LOCK	LOGGING|NOLOGGING	PRINT|NOPRINT
PROCESS	REPORT	SAVE CLEAR
SCRATCH	STACKSIZE	STATISTICS|NOSTATISTICS
TRACE|NOTRACE	VERIFY|NOVERIFY	WARNINGS|NOWARNINGS

COMPILE|SYNTAX

COMPILE allows QTP to compile and execute runs. SYNTAX allows QTP to parse statements, but does not allow QTP to execute a run or compile it into a permanent file. The BUILD, EXECUTE, and GO statements are disabled when SYNTAX is specified.

Limit: COMPILE or SYNTAX options aren't saved in compiled runs.

Default: COMPILE

DATABASE database-name

For SQL support, each SQL statement requires a name to attach to the database. The database name must exist as a logical name in the current dictionary.

The database name can also be set when loading the dictionary, or by using the IN database option of the DECLARE CURSOR statement, or in the PowerHouse resource file.

For more information about setting the database, see the section, "SQL Overview", in Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

DICTIONARY filespec [TYPE PHD|PDC]

Changes the data dictionary used for the current QTP session. The SET DICTIONARY statement can be used any number of times in a single session, and is helpful when more than one dictionary is being referenced. This option overrides any other dictionary setting method.

Limit: The dictionary option is not saved in compiled runs.

Default: PHD (**MPE/iX, OpenVMS**) or phd.pdc (**UNIX, Windows**)

[TYPE PHD|PDC] (OpenVMS)

Specifies the default dictionary type. When the TYPE option is specified in a PowerHouse component, it applies to subsequent SET DICTIONARY statements in the component.

When searching for a dictionary, PowerHouse limits searches to the dictionary type specified by the TYPE option. If the TYPE option is not specified, PowerHouse searches first for a PHD dictionary, then a PDC dictionary.

Default: PHD

DOWNSHIFT|UPSHIFT|NOSHIFT

Specifies that the names of entered identifier names be shifted to lowercase, uppercase, or left as entered. This option overrides the dictionary or the program parameter setting.

These options allow dictionaries to be created with case-sensitive entity names. For system-wide access to mixed, lowercase, or uppercase identifiers, you can specify the SHIFT option in the SYSTEM OPTION statement, or with the **upshift|downshift|noshift** program parameter.

FILE

MPE/iX:	**FILE record-structure OPEN [n] [LOCK|NOLOCK]** **[DBMODE n [DEFERRED]]|** **[access-type] [exclusivity]**
OpenVMS, UNIX, Windows:	**FILE record-structure OPEN [n]** **[access-type] [exclusivity]**

Limits: The APPEND, CLEAR, WRITE, and EXCLUSIVE options of OPEN are not valid with relational databases.

OPEN

Controls how data records are opened during a run. The record-structure named here must have been previously named in the ACCESS or OUTPUT statement.

FILE OPEN options are saved in compiled runs and can be used to force a separate file open.

The following table lists the defaults. Defaults for the access-type depend on how the file is used and on the file's type.

DBMODES and Access-types

FILE Use	OPEN Number	IMAGE	Eloquence	Sequential	Direct	Relative	Indexed
Input or lookup only	0	DBMODE 5	DBMODE 9	READ	READ	READ	READ
Adding	0	DBMODE 1	DBMODE 1	APPEND	APPEND	APPEND[1]	UPDATE
Updating	0	DBMODE 1	DBMODE 1	n/a	UPDATE	UPDATE	UPDATE
Deleting	0	DBMODE 1	DBMODE 1	n/a	n/a	UPDATE	UPDATE

[1]*If the USING option of the OUTPUT statement is used, the default is UPDATE.*

[n]

Defines the transaction number for relational databases and the open number for non-relational databases. All tables given the same number are updated and committed in one transaction.This option forces a separate file or database open, even though the open mode is otherwise compatible with an existing open. Forced opens may be necessary if a file is accessed by different paths and/or keys simultaneously.

Limit: n can be any value between 0 to 30.

Default: 0

LOCK|NOLOCK (MPE/iX)

The lock option indicates whether the file is opened with locking enabled or locking disabled. If the first user to open a file does so with locking enabled (or disabled), then all other users must open the file with locking enabled (or disabled), or else the open fails. Opening with locking enabled does not mean that locking is performed.

For KSAM files, locking must be enabled in order to open the file for anything other than read unless the file is opened for exclusive access, in which case locking is not performed.

If the LOCK|NOLOCK option is specified, it must precede the access-type.

Limit: Not valid for relational or IMAGE databases.

DBMODE n [DEFERRED] (MPE/iX, UNIX, Windows)

Used to specify the open mode for IMAGE and Eloquence databases. IMAGE supports open modes from 1 to 8. Eloquence accepts open modes from 1 to 9. The new mode, DBMODE 9, only allows PowerHouse to read the database, but allows other concurrent users of the database to read and update data. Eloquence only fully supports modes 1, 3, 8 and 9. All other modes are mapped to one of these supported modes. This means that if a PowerHouse application uses modes 2, 4, 5, 6 or 7, it might not give the same results with Eloquence as it does with IMAGE. For more information about open modes, refer to your IMAGE or Eloquence documentation.

DEFERRED enables deferred output and is only applicable to DBMODE 3.

Limit: Valid only for IMAGE and Eloquence database.

access-type

Describes how a file may be accessed.

The access-type options are APPEND, CLEAR, READ, UPDATE, and WRITE.

APPEND	Opens a record-structure's associated file for write access and adds data records at the end of the file. Creates a read-write transaction for relational tables. Limit: Not valid for indexed files and mailboxes. This option does not permit update access.
CLEAR	Opens a record-structure's associated file for write access and deletes all previous data. CLEAR deletes all records in the file when processing begins. Limit: Valid only for direct and sequential files. Not valid for mailboxes.
READ	Opens a record-structure associated file for read access only. Creates a read-only transaction for relational tables.
UPDATE	Opens a record-structure's associated file for read and write access. This option allows existing data records to be updated in place and deleted. Creates a read-write transaction for relational tables. Limit: Not valid for sequential files except Mailboxes.
WRITE	Opens a record-structure's associated file for write access only. **MPE/iX, UNIX, Windows**: For direct, relative (MPE/iX), and sequential files, the WRITE option deletes all data records in the record-structure when processing begins. The file can be read and written to by others unless the exclusivity option specifies otherwise. **OpenVMS**: For direct, relative, and sequential files, records added overwrite existing records starting at the beginning of the file. Any data not overwritten is retained. The file cannot be read or written to by others until it is closed. **MPE/iX, OpenVMS, UNIX, Windows**: For indexed files, adding a record that has the same unique key value as an existing record causes the existing record to be replaced. Adding a record with a new key value causes that record to be added. Existing data is retained. The file cannot be read or written to by others until it is closed. Limit: Not valid for relational tables.

exclusivity

Describes how other users can access a file.

Limit: Not valid for IMAGE databases and some relational databases (**Microsoft SQL Server, ODBC, DB2,** and **Oracle Rdb**).

The exclusivity options are EXCLUSIVE, SEMIEXCLUSIVE, and SHARE.

EXCLUSIVE	Prevents other users and processes from opening the record-structure's associated file. Limit: UPDATE EXCLUSIVE is not valid for **DISAM** files.
SEMIEXCLUSIVE (**MPE/iX, OpenVMS**)	Specifies that other users and processes can open the record-structure's associated file for read access only.
SHARE	Specifies that other users or processes can open the record-structure's associated file for read or write access.

SIGNAL|NOSIGNAL [ON CLOSE] (OpenVMS)

SIGNAL ON CLOSE sends an End-Of-Data signal to the mailbox when the process closes the file; NOSIGNAL ON CLOSE does not.

Default: NOSIGNAL

Limit: Valid only for mailboxes.

WAIT|NOWAIT [ON SEND|RECEIVE|FULL] (OpenVMS)

WAIT specifies that the process waits for the message. If either WAIT or NOWAIT is specified without options, it is set for all subsequent options.

Limit: Valid only for mailboxes.

Default: NOWAIT

ON SEND

WAIT ON SEND waits for the message sent to a mailbox to be received before returning control to the process; NOWAIT ON SEND does not.

ON RECEIVE

WAIT ON RECEIVE indicates that when reading messages the process will wait for a message to be posted if one is not already there. When using this option, an empty mailbox is not treated as an End-of-Data condition; instead, an End-of-File condition is always processed as an End-of-Data condition and could be used to control processing. NOWAIT ON RECEIVE specifies that a message should be returned if one exists, but if the mailbox is empty, control should be returned to the process immediately.

ON FULL

WAIT ON FULL specifies that if a message is sent to a full mailbox, the process waits until the message can be added before returning control to the process. NOWAIT ON FULL specifies that if a message is sent to a full mailbox, that control be returned to the process immediately and an error status is issued.

INPUT limit-option

Where limit-option is one of the following:

[LIMIT] n

LIMIT n specifies the maximum number of transactions selected for processing. The INPUT option is most often used to set reasonable transaction limits for testing purposes. If the specified input limit is exceeded, no further input data records are read, but the next processing phase continues normally.

If the INPUT LIMIT and PROCESS LIMIT options are used together on the same statement, the limit with the lower value is used and sets the type of limit.

If both limits are specified on separate SET statements, the last one encountered by QTP sets the limit.

The INPUT LIMIT option is saved in compiled runs.

LIMIT is used for documentation only.

Limit: 2,147,483,647 transactions

Default: 10,000

NOLIMIT (OpenVMS, UNIX, Windows)

NOLIMIT removes any limit on the number of transactions that are selected for processing.

The INPUT NOLIMIT option is saved in compiled runs.

JOB|NOJOB (MPE/iX, UNIX, Windows)

JOB submits the QTP run as a batch job; NOJOB processes the run interactively.

For more information about how to run batch jobs, see (p. 136).

Limit: The JOB and NOJOB options aren't saved in compiled runs. They are invalid if QTP statements are being processed in batch.

Default: NOJOB

JOB [option] (OpenVMS)

For information on submitting reports as a batch job, see (p. 136).

JOB options		
AFTER	CHARACTERISTICS|NOCHARACTERISTICS	CPUTIME
HOLD|NOHOLD	IDENTIFY|NOIDENTIFY	KEEP|NOKEEP
LOGFILE|NOLOGFILE	NAME	NOTIFY|NONOTIFY
PARAMETERS	PRINTER|NOPRINTER	PRIORITY
QUEUE	RESTART|NORESTART	USER
WSDEFAULT	WSEXTENT	WSQUOTA

AFTER [absolute-time][+|- delta-time] (OpenVMS)

Specifies the time at which the job is to start executing. The default is the time that the job reaches the top of the queue. An absolute-time is a specific date and/or time of day. The general form is:

{dd-mmm[-yyyy][:[hours][:[minutes][:[seconds][.[hundredths]]]]]|{TODAY|TOMORROW|YESTERDAY}

A delta-time is an offset from the current time to a time in the future. The general form is:

[days][-[hours]][:[minutes][:[seconds][.[hundredths]]]]

You must indicate an absolute time, a delta time, a combination of both absolute and delta time, or the time indicators TODAY, TOMORROW, and YESTERDAY. For example,

```
> SET JOB AFTER 15-MAY-2001:12
```

executes at noon on May 15, 2001;

```
> SET JOB AFTER +3
```

executes three hours from the current time;

```
> SET JOB AFTER TOMORROW -1
```

executes at 11:00 p.m. on the current date.

For more information about time representation, see the *OpenVMS Users Manual*.

CHARACTERISTICS number|string[[,]number|string]... NOCHARACTERISTICS (OpenVMS)

CHARACTERISTICS specifies one or more of the characteristics, to a maximum of 127, that you can use in defining the job queue for executing a job. A string is a series of displayable characters (letters, numbers, or special characters) enclosed in double or single quotation marks.

NOCHARACTERISTICS specifies that any previously set characteristics are to be canceled.

Limit: Specified numbers can range from 0 to 127. If you specify a string, there is a maximum of 31 characters for a physical characteristic and a maximum of 255 characters for a logical characteristic. These characteristics are installation specific. The DCL command $ SHOW QUEUE /CHAR shows you the characteristics that are in effect for your system.

CPUTIME n (OpenVMS)

Specifies the maximum CPU time for the job. For more information, see the *OpenVMS Users Manual*.

Limit: You cannot request more time than permitted by either the base queue limit or your own UAF record-structure.

HOLD|NOHOLD (OpenVMS)

HOLD stipulates that the job is to be held in a queue until specifically released; NOHOLD does not.

Default: NOHOLD

IDENTIFY|NOIDENTIFY (OpenVMS)

IDENTIFY indicates that a message containing the job number and queue message is to be displayed when the job is sent to the batch queue; NOIDENTIFY does not.

Default: IDENTIFY

KEEP|NOKEEP (OpenVMS)

KEEP indicates that the job log file is to be kept after printing; NOKEEP does not.

Default: NOKEEP

LOGFILE [name]|NOLOGFILE (OpenVMS)

LOGFILE specifies the name to be given to the log file.

NOLOGFILE specifies that no log file is to be kept for the job. This also implies NOKEEP. In the following example, the job log information is written to the LOGTEMP file:

```
> SET JOB LOGFILE LOGTEMP
```

Default: The job name with an extension of .LOG.

NAME filespec (OpenVMS)

Used to give the job a name. A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name.

Limit: The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters semi-colon (;), dollar sign ($), and leading question mark (?) have special meanings in PowerHouse and are prohibited.

A file specification takes the general form:

[NODE::][DEVICE:][[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

In the following example, the name EMPLIST is assigned to the QTP report:

```
> SET JOB NAME EMPLIST
```

Default: The name of the temporary job file created by QTP, which has the file extension .JOB.

NOTIFY|NONOTIFY (OpenVMS)

NOTIFY indicates that the user should be notified when the job is completed; NONOTIFY does not.

Default: NONOTIFY

PARAMETERS string [,string]... (OpenVMS)

Specifies a list of parameters, to a maximum of eight, to be passed to the job. Parameters are typically used when you are running a command file that requires them.

Limit: 255 characters per string

PRINTER name|NOPRINTER (OpenVMS)

PRINTER name specifies the name of the print queue to which the log file is to be directed when the job is completed; NOPRINTER does not.

Default: SYS$PRINT

PRIORITY n (OpenVMS)

Sets an output sequence for the job in the batch queue. The range is from 0 through 255.

QUEUE queuename (OpenVMS)

Specifies the name of the queue where the job is waiting for execution.

Limit: The maximum size for a queuename is 31 characters; for a logical queuename, the maximum size is 255 characters.

Default: SYS$BATCH

RESTART|NORESTART(OpenVMS)

RESTART indicates that the job is to restart after a system crash or after a $ STOP /QUEUE /REQUEUE command is issued; NORESTART does not.

Default: RESTART

USER username (OpenVMS)

Specifies the name of the user for whom you're submitting the job. Privileges are necessary to use this option.

WSDEFAULT n (OpenVMS)

Defines a working set default for the batch job. For more information, see the *OpenVMS Users Manual*.

WSEXTENT n (OpenVMS)

Defines a working set extent for the batch job. For more information, see the *OpenVMS Users Manual*.

WSQUOTA n (OpenVMS)

Defines the maximum working set size for the batch job. For more information, see the *OpenVMS Users Manual*.

LIST|NOLIST

LIST displays the contents of a QTP source statement file referenced by a USE statement; NOLIST does not.

Limit: The LIST and NOLIST options are not saved in compiled runs.

Default: LIST

{LIST|NOLIST} SQL

The SQL option controls the listing of SQL statements. It shows the SQL requests sent from PowerHouse to the database, including the effects of any substitutions.

Default: SET NOLIST SQL

LOCK

MPE/iX:	**LOCK [BASE|FILE] {RUN|REQUEST|UPDATE}** **LOCK RECORD UPDATE**
OpenVMS, UNIX, Windows:	**LOCK [FILE] {RUN|REQUEST|UPDATE}** **LOCK RECORD UPDATE**

Controls both the level and the duration of locking.

For more information, see (p. 140).

Default: LOCK BASE RUN (**MPE/iX**); LOCK FILE RUN (**OpenVMS, UNIX, Windows**)

LOGGING|NOLOGGING (MPE/iX)

SET LOGGING causes QTP to mark the beginning and ending of logical transactions in the IMAGE logfile. This facilitates recovery of the database's contents in the event of a system failure.

SET NOLOGGING does not cause the beginning and ending of logical transactions to be marked. It does not prevent changes to the database from being posted to the logfile.

Limit: These statements have no effect on non-IMAGE files. They have no effect if logging is not enabled for a database.

Default: LOGGING

PRINT|NOPRINT

The PRINT option sends the source listing to:

MPE/iX:	the designated file STDPRINT.
OpenVMS:	the designated file, SYSPRINT, or SYS$PRINT if SYSPRINT is not defined.
UNIX, Windows:	the default printer. The default printer is obtained from the value of the environment variable, PH_PRINTER. If this variable is not set, the system default printer is used.

Limit: The PRINT and NOPRINT options are ignored in batch jobs and are not saved in compiled runs.

Default: NOPRINT

PROCESS limit-option

Where limit-option is:

[LIMIT] n

LIMIT n specifies the maximum number of transactions (n) processed for each request in the run. If the process limit is exceeded during the course of a run or request, no further processing is performed and all active relational database transactions are rolled back. The entire run is stopped.

If PROCESS LIMIT and INPUT LIMIT are used together on the same statement, the limit with the lower value is used and sets the type of limit. If both limits are specified on separate SET statements, the last one encountered by QTP sets the limit.

LIMIT is optional and used for documentation only.

The PROCESS LIMIT option is saved in compiled runs.

Limit: 2,147,483,647 transactions

Default: 10,000

NOLIMIT (OpenVMS, UNIX, Windows)

NOLIMIT removes any limit on the number of transactions that are processed in the run.

The PROCESS NOLIMIT option is saved in compiled runs.

REPORT options

SET REPORT specifies the destination and type of the QTP output error report.

MPE/iX, OpenVMS:	**[TERMINAL|PRINTER|BOTH] [ERRORS|DETAIL]**
UNIX, Windows:	**[TERMINAL|PRINTER[string]|BOTH] [ERRORS|DETAIL]**

Specifies that the QTP error report is output to the terminal, or to the printer, or to both.

The ERRORS option instructs QTP to report only errors; while the DETAIL option causes QTP to report all output actions as well as errors.

The REPORT option is saved in compiled runs.

Default: REPORT TERMINAL ERRORS

DESTINATION printername (UNIX, Windows)

Specifies the printer where the report is to be sent.

Use the DESTINATION option instead of the PRINTER string option to identify a specific printer if operating system access is not allowed.

The DESTINATION option is ignored if a string is specified on the SET REPORT PRINTER statement.

Limit: Valid for SET REPORT PRINTER only.

In PowerHouse for **UNIX**, an optional string may be specified. This string is the UNIX Shell command used to send the report to the printer.

A string is not valid if the **noosaccess** program parameter is specified or the OSACCESS resource file option equals OFF. A syntax error will result.

If the string is omitted, PowerHouse follows these steps to determine the appropriate values:

1. If a string was used in the last SET REPORT PRINTER [string] statement, PowerHouse uses that value. The DESTINATION option is ignored.
2. If the value of the environment variable PH_PRINTER is defined, PowerHouse uses this value. The DESTINATION option is ignored. QTP assumes this argument is available in the PH_PRINTER environment variable.
3. The system default string is used. If specified, PowerHouse uses the DESTINATION option; otherwise, the default for this option is used.

If the string is null, steps 2 and 3 are used to determine the string value.

For the SET REPORT BOTH statement, the value of the printer string is determined as above.

SAVE CLEAR

Clears QTP's source statement save file, QTPSAVE, at the point that the SET SAVE CLEAR statement is entered.

Limit: The SAVE CLEAR option is not saved in compiled runs.

SCRATCH [AUTO]

SCRATCH AUTO allows QTP to determine if an intermediate file is needed to process a request. SET SCRATCH forces QTP to use an intermediate file. For more information, see (p. 14).

The SCRATCH option is saved in compiled runs.

Default: SCRATCH AUTO

STACKSIZE n

Changes the area used for expression processing. The stack is an internal buffer where expressions are stored. This option is needed only when QTP issues a message that the stacksize is too small.

This option is saved in compiled runs.

Default: 400

STATISTICS|NOSTATISTICS

STATISTICS issues statistics at the end of each request regarding the number of records read or changed; NOSTATISTICS does not.

The STATISTICS and NOSTATISTICS options are saved in compiled runs.

Default: STATISTICS

TRACE|NOTRACE

See Chapter 4, "QTP Tracer".

VERIFY|NOVERIFY [DELETE] [ERRORS]

VERIFY enables requests for authorization to proceed with processing. NOVERIFY disables requests for authorization to proceed with processing.

VERIFY and NOVERIFY aren't mutually exclusive unless they include identical options. SET VERIFY without any options specifies VERIFY DELETE ERRORS. SET NOVERIFY without any options specifies NOVERIFY DELETE ERRORS.

Limit: The VERIFY and NOVERIFY options are not saved in compiled runs.

Default: VERIFY DELETE and NOVERIFY ERRORS

DELETE

If VERIFY is specified, QTP requests authorization to proceed with processing when creating a new version or replacing an existing QTP file with a new file of the same name when using the SAVE or BUILD statements. If NOVERIFY is specified, QTP does not request authorization.

ERRORS

If VERIFY is specified, pauses when errors are encountered in a file processed by a USE statement. QTP must wait either for a carriage return to continue processing or a user break to end processing.

If NOVERIFY is specified, QTP doesn't pause if errors are encountered.

WARNINGS|NOWARNINGS

WARNINGS issues warning messages as required; NOWARNINGS doesn't.

The WARNINGS and NOWARNINGS options are saved in compiled runs.

Default: WARNINGS

Discussion

The SET statement specifies the values of parameters that control the functioning of a QTP run.

Individual SET statements can be listed in any sequence. To see which SET statements are in effect, enter

```
> SHOW STATUS
```

Using SET Statement Options

Any number of options can be specified in a single SET statement. Options are in effect for the duration of a QTP process or until changed by another SET statement. The SET statement entered without an option does not change any settings.

Although the statements are easier to read if each SET statement is on a separate line, the different options of the statement can be combined into one statement.

For your convenience, the following table shows you the SET statement options which are saved in compiled runs, and those which are not:

Saved in Compiled Runs	Not Saved in Compiled Runs
FILE OPEN	COMPILE|SYNTAX
INPUT LIMIT	DICTIONARY
PROCESS LIMIT	JOB|NOJOB
REPORT	LIST|NOLIST
SCRATCH	PRINT|NOPRINT
STACKSIZE	SAVE CLEAR

Saved in Compiled Runs	Not Saved in Compiled Runs
STATISTICS|NOSTATISTICS	TRACE|NOTRACE
WARNINGS|NOWARNINGS	VERIFY|NOVERIFY [DELETE] [ERRORS]

Specifying the INPUT LIMIT and PROCESS LIMIT Options

Using the INPUT LIMIT or PROCESS LIMIT option in a SET statement affects all requests from that point on until another SET statement specifies a new limit. If the INPUT LIMIT and PROCESS LIMIT options are used in an EXECUTE or USE statement, they affect all requests in the compiled file being executed. If they're used in a REQUEST statement, they affect only that request.

If limits are specified in a combination of EXECUTE, REQUEST, USE and SET statements, the order of precedence is

- EXECUTE or USE
- REQUEST
- SET

Combining SET Statements

You can enter multiple SET statements. Each statement is effective unless mutually exclusive options are entered. If mutually exclusive options are entered, the last option that is entered is effective.

Ways to Run a Batch Job

There are two ways to run a QTP run as a batch job:

- use the SET JOB statement and enter the necessary statements
- construct a text file containing operating system commands, QTP statements, and execution-time parameters, and stream (**MPE/iX**) or submit this file

Generating a Batch Job Using SET JOB

Using the SET JOB statement allows online prompting for execution-time parameters while the QTP run is executed in batch. It works as follows:

1. The SET JOB statement suppresses the interactive execution of any subsequent QTP run.
2. When the SET JOB statement is entered in QTP, the QTPSAVE file is cleared.
3. If QTP encounters the designated file QTPJOB, the contents are written to the QTPSAVE file.

MPE/iX:	Operating system commands prefixed with a colon (:) execute immediately. Operating system commands prefixed with an exclamation mark (!) are not executed immediately but are written to QTPSAVE.
OpenVMS:	Operating system commands prefixed by two operating system prompts ($$) write to the currently active save file (prefixed with a single prompt). Operating system commands prefixed with a single prompt ($) execute immediately.
UNIX:	Operating system commands prefixed by two exclamation marks (!!) write the Shell command to the currently active save file with one exclamation mark. Operating system commands prefixed with a single exclamation mark (!) suspend the current process, and execute the Shell command.
Windows:	Operating system commands prefixed by two exclamation marks (!!) write the command to the currently active save file with one exclamation mark. Operating system commands prefixed with a single exclamation mark (!) execute immediately.

4. All other statements are parsed and written to QTPSAVE.
5. When an EXECUTE or GO statement is encountered (or a USE statement that refers to a compiled QTP run), QTP prompts for any execution-time parameters. When the parameters are entered and accepted, they're written to QTPSAVE. If a REQUEST is conditional, the condition is evaluated before submission. If the condition is false, parameters are not prompted for.
6. When the SET JOB statement is in effect, and the request specifies the creation of a subfile, QTP creates an interim minidictionary. Succeeding requests that reference the subfile in the same job can determine formats from this minidictionary. Once the job is submitted and QTP encounters an EXIT statement, the interim minidictionary is deleted.
7. When an EXIT or QUIT statement is encountered, QTPSAVE is submitted immediately for execution.

   OpenVMS: Job files are created in the directory, SYS$SCRATCH (and given the extension, .JOB), where they are held until the job is complete, and then deleted. Job files may not be deleted after a system crash.

   Windows: The job is submitted as a separate spawned process just before the product exits. By default, the spawned process window is hidden. To show the window, use the **setjobshow** program parameter. The default is **nosetjobshow**. There is also a resource file statement SETJOBSHOW.

The REVISE statement cannot be used when the SET JOB statement is in effect.

The **noosaccess** and **nodcl** (**OpenVMS**) program parameters cause QTP to reject any command that's preceded by an operating system prompt. If QTP is invoked with the **noosaccess** and **nodcl** (**OpenVMS**) program parameters, then the SET JOB feature is also rejected.

Any operating system commands necessary to execute QTP as a batch job must either be entered after the SET JOB statement or be included in the QTPJOB file. Consider the following examples.

Examples

These examples show you how to generate a batch job using the SET JOB statement.

UNIX, Windows: It is not necessary to include a command to run QTP. QTP automatically generates the appropriate command after the SET JOB statement. Since the QTP command is generated automatically, use a SET DICTIONARY statement to specify a non-default dictionary.

MPE/iX

```
> SET JOB
> !JOB USERNAME/PASSWORD.ACCOUNT/PASSWORD
> !QTP INFO="DICTIONARY=MPEIXDIC"
> REQUEST NAMES
> ACCESS EMPLOY1
> CHOOSE EMPLOYEE PARM
> DEFINE STARTDATE = PARM
> SELECT IF JOINED DATE GE STARTDATE
> SUBFILE SF1 INCLUDE LASTNAME,FIRSTNAME
> GO
> EXIT
```

OpenVMS

```
> SET JOB
> $$QTP DICT=PERSONNEL
> REQUEST NAMES
> ACCESS EMPLOY1
> CHOOSE EMPLOYEE PARM
> DEFINE STARTDATE = PARM
> SELECT IF JOINED DATE GE STARTDATE
> SUBFILE SF1 INCLUDE LASTNAME,FIRSTNAME
> GO
> EXIT
```

UNIX, Windows

```
> SET JOB
```

```
> SET DICTIONARY personnel
> REQUEST NAMES
> ACCESS EMPLOY1
> CHOOSE EMPLOYEE PARM
> DEFINE STARTDATE = PARM
> SELECT IF JOINED DATE GE STARTDATE
> SUBFILE SF1 INCLUDE LASTNAME,FIRSTNAME
> GO
> EXIT
```

Submitting Jobs as Text Files

The text file that you create for your batch job must include execution-time values (responses) for all CHOOSE, DEFINE, and GLOBAL TEMPORARY statement prompts. These values are placed in the file starting on the line below the GO statement (or the EXECUTE statement, if compiled reports or runs are used).

You must place each response to the CHOOSE statement prompt on its own line; multiple responses on a single line are not allowed.

You must include a blank line in the file after all of the values for each linkitem that is prompted for by a CHOOSE. This blank line tells QTP to stop prompting for values of the linkitem and move on to prompting for values of the next linkitem or DEFINE.

For each DEFINE or GLOBAL TEMPORARY statement prompt, you must include a single response in the job file. For example, if your QTP request contains ten DEFINE statements with the PARM option, you must include ten values (each on its own line) in your file.

All values must be listed in an order that corresponds to the appearance of the CHOOSE, DEFINE, and GLOBAL TEMPORARY statements in the run.

When submitting batch jobs, the general format of the text file is as follows:

MPE/iX

```
!JOB  USERNAME/PASSWORD.ACCOUNT/PASSWORD
!QTP DICTIONARY=MYDICT
ACCESS FILEA
CHOOSE FIELDA PARM
SUBFILE FILEB KEEP INCLUDE FILEA
GO
10

EXIT
!EOJ
```

OpenVMS

```
$QTP
ACCESS EMPLOYEES
DEFINE STARTDATE DATE = PARM
CHOOSE EMPLOYEE PARM
SELECT IF DATEJOINED GE STARTDATE
REPORT LASTNAME BRANCH POSITION DATEJOINED
GO
1
4
9
11
22
900101
EXIT
```

UNIX

```
QTP << EOF
SET  DICTIONARY MYDICT
ACCESS FILEA
CHOOSE FIELDA PARM
SUBFILE FILEB KEEP INCLUDE FILEA
GO
```

```
10

EXIT
EOF
```

Windows

```
QTP
SET  DICTIONARY MYDICT
ACCESS FILEA
CHOOSE FIELDA PARM
SUBFILE FILEB KEEP INCLUDE FILEA
GO
10

EXIT
```

Example

This next example demonstrates the use of the SET statement in a run that is designed to create a series of subfiles. The subfiles are then referenced by QUIZ at a later date for reports about budget and project forecasts.

```
> SET REPORT PRINTER DETAIL &
>   INPUT LIMIT 150 NOWARNINGS
> RUN MONTHEND
> REQUEST BUDG ON EXCEPTION ERRORS TERMINATE REQUEST
```

The SET statement uses several options to set certain parameters around this run. The REPORT option specifies that a detailed report of the run be routed to the default printer. The INPUT LIMIT option indicates that only 150 data records can be processed. This is useful for testing the run. The NOWARNINGS option suppresses any warning messages.

```
> ACCESS BUDGETS
> CHOOSE PROJECT CODE PARM &
>   PROMPT "ENTER PROJECT NUMBER: "
>
> EDIT PROJECTCODE OF BUDGETS LOOKUP ON BUDGETS
> SUBFILE PROJBUD KEEP INCLUDE BUDGETS
> GO
```

The following example sets the printer destination and then directs the detailed output to the both the printer and the terminal.

```
SET REPORT DESTINATION PRINTERNAME BOTH DETAIL
```

For clarity, the options can be specified on two SET REPORT statements as follows:

```
SET REPORT DESTINATION PRINTERNAME
SET REPORT BOTH DETAIL
```

In the next example, the DESTINATION is set but is not used, because the REPORT device is the terminal.

```
SET REPORT DESTINATION PRINTERNAME
SET REPORT TERMINAL
```

SET LOCK

Overrides default QTP locking settings.

Syntax

MPE/iX:	**SET LOCK [BASE|FILE] [RUN|REQUEST|UPDATE]** or **SET LOCK RECORD UPDATE** Default: SET LOCK BASE RUN
OpenVMS, UNIX, Windows:	**SET LOCK [FILE] [RUN|REQUEST|UPDATE]** or **SET LOCK RECORD UPDATE** Default: SET LOCK FILE RUN

BASE (MPE/iX)

Controls both the level and the duration of file locking. SET LOCK BASE issues one lock for each IMAGE database, KSAM, or MPE file involved.

FILE

Controls both the level and the duration of file locking.

MPE/iX: Differs from the BASE option only for the IMAGE databases. Individual locks are applied for each dataset involved. Datasets not mentioned are not locked. This will also lock each table involved in the QTP request that is in a relational database.

OpenVMS: Lock Management Services (LMS) controls file locking. The optional keyword FILE is for documentation only.

UNIX, Windows: The optional keyword FILE is for documentation only.

Options

The SET LOCK [FILE] options are RUN, REQUEST, and UPDATE. These options control the duration of locking.

RUN

For non-relational files, QTP locks all files associated with record-structures involved in a QTP run for the duration of the run. When locking multiple files, QTP uses special logic to automatically avoid deadlocks.

For relational databases, SET LOCK FILE RUN is equivalent to the COMMIT AT RUN statement. For more information, see (p. 63).

For relational databases, QTP commits transactions once a complete run is finished, allowing an entire run to be treated as a whole. If a later request isn't completed, earlier requests are rolled back. This option creates more locks for a longer period and requires more room to maintain rollback information.

REQUEST

For non-relational files, REQUEST specifies that all the files in a request are to be locked for the duration of each request (using an exclusive LMS lock on **OpenVMS**). The set lock request statement applies locking for the duration of each request. Only files involved in the request are locked at the start and unlocked at the end of that request. (Except for subfiles, all files and databases are opened at the beginning of the run and closed at the end, regardless of the set lock statement.) RMS record-level locking is applied on update (**OpenVMS**).

For relational databases, SET LOCK FILE REQUEST is equivalent to the COMMIT AT REQUEST statement. For more information, see (p. 63).

UPDATE

For non-relational files, specifies that locking is applied only for the duration of each individual update. The UPDATE option provides maximum concurrency, but only individual updates can be rolled back after a failure.

OpenVMS: An exclusive file lock using LMS, and an RMS record-level lock is applied.

The SET LOCK UPDATE statement specifies that locking is applied only for the duration of each individual update. Since the SET LOCK UPDATE statement lets other users change the data between the time the data record is read and the time it is updated by QTP, a checksum is calculated for each data record read. When updating, the data record is re-read, and another checksum calculated. These two checksums are compared, and the update is performed only if they match.

If a change to a data record is detected, a file update error occurs. This check can be bypassed by using the NOCHECK option of the OUTPUT statement. However, the use of the SET LOCK UPDATE statement and the NOCHECK option is not generally recommended, as this could compromise data integrity.

For relational databases, SET LOCK FILE UPDATE is equivalent to the COMMIT AT UPDATE statement. For more information, see (p. 63).

RECORD UPDATE

MPE/iX, UNIX, Windows:	For non-relational files, RECORD UPDATE causes individual records to be locked during each update.
OpenVMS:	For non-relational files, RECORD UPDATE locks a file for concurrent write using LMS and records using RMS.

If an intermediate file is not used and the file is also specified in the ACCESS statement, the locks are applied when the record is read in the input phase and held until the next record in that file is read. If an intermediate file is used or the file is not specified in the ACCESS statement, the locks are applied when the record is retrieved in the output phase.

MPE/iX:	Affects only IMAGE databases. If the RECORD option is specified and no IMAGE database files are associated with the run or request, QTP ignores the SET statement. There is no way to unlock any one record. Any condition that causes a record to be unlocked will also cause all records locked in the same file by the calling process to be unlocked.
UNIX:	Record-level locking is supported for C-ISAM files on all UNIX platforms and Net-ISAM files on Sun SPARC. There is no way to unlock any one record. Any condition that causes a record to be unlocked will also cause all records locked in the same file by the calling process to be unlocked.
Windows:	Record-level locking is supported for DISAM files. There is no way to unlock any one record. Any condition that causes a record to be unlocked will also cause all records locked in the same file by the calling process to be unlocked.

Discussion

The SET LOCK statement overrides default QTP settings. Only one SET LOCK statement should be specified per run. The last SET LOCK statement that is specified before the run is completed (with a BUILD or GO statement) is in effect for the entire run.

Locking RMS Files (OpenVMS)

Unless otherwise instructed, QTP opens all files in shared mode and applies an exclusive lock for output files using the Lock Management Services (LMS) for the duration of the run. RMS record-level locking is applied on update to provide additional protection. When locking multiple files, QTP works with OpenVMS to automatically detect and avoid deadlocks.

Cursor Retention

Cursor retention is applied for files in the ACCESS statement when no VIAINDEX option is specified and SET LOCK RECORD UPDATE is in effect.

For SET LOCK FILE REQUEST and SET LOCK FILE RUN, cursor retention is not used.

Interactive Locking

When running an interactive session, QTP issues conditional locks. If another process has already locked the first file that QTP attempts to lock, QTP tries the lock again every two seconds until the lock is successful.

While trying to lock, QTP issues a message indicating that it's waiting for a lock. This message is followed by a series of periods (one for each lock attempt). If necessary, the user can terminate the run by entering a user break: [Ctrl-Y] (**MPE/iX**) or [Ctrl-C] (**OpenVMS, UNIX, Windows**).

Once the first file is locked, QTP doesn't continue to attempt locks for other files. If a subsequent lock is not immediately successful, all files locked to that point are unlocked, and QTP starts the locking process again from the first file. When this process is complete, QTP issues a message to that effect.

Batch-Oriented Locking

When running a batch job, QTP issues an unconditional lock for the first file. Once the file is locked, QTP issues conditional locks for the rest of the files to be locked, and uses the same logic as interactive sessions to avoid deadlocks. QTP doesn't issue messages about waiting for locks when running a batch job.

Effects of OPEN Options on Locking

The discussions in previous sections assume that the default modes for opening files are in effect. The SET LOCK statement is unnecessary if you use the SET FILE statement to override the default open mode.

MPE/iX:	When IMAGE databases are opened in DBMODE 3 or DBMODE 7 (exclusive access), or DBMODE 8 (read-only access), or when KSAM or MPE files are opened for exclusive or semi-exclusive access, no locking is performed. QTP does lock databases opened in other modes, including those opened for read access only.
OpenVMS:	When RMS files are opened for exclusive access, no further locking strategy is required. QTP locks RMS files opened in other modes. Mailbox files and files opened for read-only access are not locked.
UNIX, Windows:	When indexed, sequential, and direct files are opened for exclusive access, no further locking strategy is required. QTP locks files opened in other modes. Files opened for read-only access are not locked. UPDATE EXCLUSIVE opens are not permitted on DISAM files. When Eloquence databases are opened in DBMODE 3, DBMODE 7 or DBMODE 8, no locking is performed. QTP does lock databases opened in other modes, including those opened for read access only.

In a QTP run, if you do not want another user to update a read-only file, for example, PROJECT_TOTALS, you can enter:

```
> SET FILE PROJECT_TOTALS OPEN UPDATE
```

The COMMIT AT Statement and SET LOCK

Use the COMMIT AT statement rather than the SET LOCK statement to control transaction duration.

If you use a SET LOCK statement before a COMMIT AT statement, the COMMIT AT statement takes precedence. QTP issues the following message:

```
*W* Previous COMMIT option/SET LOCK option overridden
for this RUN/REQUEST.
```

It is recommended that you do not use both statements in the same run.

If you use a SET LOCK statement after a COMMIT AT statement, the SET LOCK statement only controls non-relational file locking.

Record Level Locking (MPE/iX)

Specifying Lock Items

PowerHouse determines internally which item to use as the lock item, using the following rules:

- If a primary index exists, the item associated to that index is used.
- If no primary index exists but there is an alternating/repeating index, the first alternating/repeating index found is used.

As a result, you can determine which item is to be used as the lock item only through the data dictionary specification for the database. This makes it particularly important to determine an effective locking strategy when designing an application, in order to minimize potential problems. For example, it is possible to lock nonexistent records by specifying a value that does not exist in the dataset. If that value is subsequently added to the dataset, the lock is automatically applied, resulting in confusion for other users.

Read chains can be handled by record level locking, but you must take care to lock the entire chain, otherwise, a broken chain can occur.

Record level locking gives you exclusive write access to records in the dataset that are associated with the locked item. The value for the item in the record that is open at the time of the lock request is the value used.

If there are multiple occurrences of the same item value in a dataset, all records containing that value are locked. For example, if the lock item value is the lastname "Smith", the records for all the Smiths in that dataset are locked. Records in other datasets are not affected. Other users can access the locked records, but cannot update them until the lock is removed.

Limits

Some database operations that involve master set records and critical items require dataset locking as the minimum level. As a result, record level locking cannot be used with the following operations:

- addition or deletion of master set records
- updates to critical items associated with a master set record (key items)
- updates to critical items associated with a detail set record (key items)

Example

This example shows the use of the SET LOCK FILE REQUEST statement in a run that is designed to retrieve data records for subfiles that are used in a series of project reports. In this example, the locks are released at the end of each request:

```
> RUN TOTALS
> SET LOCK FILE REQUEST
> DISPLAY "************************"
> DISPLAY "   BILLING   TOTALS      "
> DISPLAY "************************"
> GLOBAL TEMPORARY TOTAL_TYPE CHARACTER*1 PARM PROMPT  &
>      "Enter P or E for totals by Project or Employee: "
>
> REQUEST PROJECT-TOTALS EXECUTE IF TOTAL_TYPE = "P"   &
```

```
>         ON EDIT ERRORS REPORT &
>         PROCESS LIMIT 500
>
> ACCESS BILLINGS
> SORT ON PROJECT
> EDIT PROJECT LOOKUP ON PROJECTS
>
> TEMPORARY PROJECT_TOTAL NUMERIC*8
> ITEM PROJECT_TOTAL SUBTOTAL BILLING_AMOUNT &
>            RESET AT PROJECT
> SUBFILE PROJTOTS KEEP AT PROJECT  &
>            INCLUDE PROJECT, &
>            PROJECT_TOTAL
>
> REQUEST EMPLOYEE-TOTALS EXECUTE IF TOTAL_TYPE = "E" &
>            ON EDIT ERRORS TERMINATE RUN &
>            INPUT LIMIT 100
>
> ACCESS BILLINGS
> SORT ON EMPLOYEE
> EDIT EMPLOYEE LOOKUP ON EMPLOY1
>
> TEMPORARY EMPLOYEE_TOTAL NUMERIC*8
> ITEM EMPLOYEE_TOTAL SUBTOTAL BILLING_AMOUNT   &
>            RESET AT EMPLOYEE
> SUBFILE EMPTOTS KEEP AT EMPLOYEE &
>            INCLUDE EMPLOYEE, &
>            EMPLOYEE_TOTAL
```

SHOW

Displays all data structures or items for the run.

Syntax

SHOW option

Options

The options are DATABASE, FILES, and ITEMS.

DATABASES

Lists the tables or views from databases that are declared with either FILE or DATABASE statements in PDL.

FILES

Lists the names of record structures, tables, or views from files and databases that are declared with FILE or DATABASE statements in PDL.

ITEMS

Displays the names of all items that are part of the record-structures named in the current ACCESS statement.

Index items are identified by asterisks (*). If there are multiple-segment indexes, only the first segment is identified by an asterisk. Substructured items are identified by periods up to the fourth substructured level; for items substructured from levels 5 to 15 (the maximum), the display format is .5 to .15 respectively. In this way, items substructured from the fifth to the fifteenth levels are identified explicitly by number rather than by nested format.

Picture overflow is identified by an ellipsis (...). The maximum picture size allowed in the SHOW ITEMS display is fifteen characters long. All characters for a picture that is exactly fifteen characters long are displayed. Redefinitions are identified by underscores (_).

Discussion

The SHOW statement shows you all the files and items regardless of access.

SHOW ITEMS shows the items for the current request.

SHOW FILES or SHOW DATABASES shows everything from the dictionary, whether or not it was used in the current run or request.

SORT

Sorts transactions in a desired sequence and defines control breaks.

Syntax

MPE/iX, UNIX, Windows:	**SORT [ON sort-item [A|D]] [[,] ON sort-item [A|D]]...**
OpenVMS:	**SORT [ON sort-item [A|D]] [[,] ON sort-item [A|D]]... [RETAIN ORDER]**

ON sort-item

Specifies a record item or defined item on which to sequence the transactions. The item then becomes a sort-item.

If more than one item is listed in the SORT statement, a comma or the keyword ON must precede each additional item.

Limit: Items can't be subscripted. You cannot sort on a temporary or global temporary item.

A|D

Specifies that sorting is done in ascending (A) or descending (D) order.

Default: Ascending (A)

RETAIN ORDER (OpenVMS)

Stipulates that records with duplicate sort-item values must be returned in the order in which they were read from the record-structure.

Discussion

The SORT statement states which transactions are to be sorted and the sequence in which they are to be sorted.

The last item named in the SORT statement becomes the item associated with the lowest-level control break. A new SORT statement cancels any previous SORT or SORTED statement. The SORT statement used alone declares that no sort is to occur.

Control Breaks

Each sort-item establishes a control break (the point in a request where the value of a sort-item changes or "breaks") where various actions can be performed. The transactions within a control break are collectively called a transaction group.

Example

The SORT statement sorts transactions according to the value of the customer account number and the individual invoice numbers within that account. This also establishes the control breaks that occur with each sort-item. Since the item ACCOUNTNO precedes the item INVOICENO, the transactions are sorted by account number first.

```
> ACCESS INVOICEMASTER
> SELECT INVOICEMASTER IF INVOICEBALANCE > 0 AND &
>   DUEDATE < SYSDATE
>
> SORT ON ACCOUNTNO ON INVOICENO
>
> TEMPORARY DAYSOVERDUE NUMERIC*3
> ITEM DAYSOVERDUE = DAYS(SYSDATE) - DAYS(DUEDATE)
>
```

```
> TEMPORARY LATECHARGES
> ITEM LATECHARGES = DAYSOVERDUE * 1000
>
> OUTPUT ACCOUNTDETAIL ADD AT INVOICENO
> ITEM LATEAMT FINAL LATECHARGES
>
> OUTPUT ACCOUNTMASTER UPDATE AT ACCOUNTNO
> ITEM BALANCE SUBTOTAL LATECHARGES AT INVOICENO
```

When the INVOICENO changes, an ACCOUNTDETAIL record is added which will include the amount of LATECHARGES for each invoice. The LATECHARGES for each invoice (within the ACCOUNT) are added to the BALANCE. This new BALANCE is updated when there is a change in the ACCOUNTNO.

SORTED

Defines control breaks for records known to be in sorted order.

Syntax

SORTED ON sort-item [A|D] [ON|, sort-item [A|D]]...

sort-item

Specifies a record item or defined item that establishes a control break.

Limit: Items can't be subscripted. You cannot sort on a temporary item or a global temporary item.

A|D

Specifies that sorting is done in ascending (A) or descending (D) order.

This option is used only for documentation; the actual sequence is neither checked nor altered.

Default: Ascending (A)

Discussion

The SORTED statement declares items that will be used as control breaks. It indicates that the transactions are already in the desired sequence, so QTP won't sort them.

A new SORTED statement cancels all previous SORT or SORTED statements.

Use the SORTED statement to eliminate unnecessary sorting whenever records are already in the desired sequence.

Example

The following example demonstrates the use of the SORTED statement in a QTP run that performs a year-end reset of sales figures. This includes the deletion of data records of all employees who are no longer with the company. These former employees are identified by the standing of "T".

```
> RUN YEAREND
> REQUEST YEAR-END PROCESSING
> ACCESS SALESREPMASTER LINK TO SALESDETAIL
> CHOOSE SALESREPCODE
>
> SORTED ON SALESREPCODE
>
> TEMPORARY TOTALSALES NUMERIC*8
>      ITEM TOTALSALES SUBTOTAL THISYEARSALES
> OUTPUT SALESREPMASTER UPDATE AT SALESREPCODE &
>          IF STANDING <> "T"
>          ITEM PREVYEARSALES FINAL THISYEARSALES
>          ITEM THISYEARSALES FINAL 0
>          ITEM DATELASTMOD FINAL SYSDATE
> OUTPUT SALESREPMASTER DELETE AT SALESREPCODE &
>          ALIAS SALESMAST IF STANDING = "T"
ACCESS VIA SALESREPCODE USING SALESREPCODE OF
SALESREPMASTER.
> OUTPUT SALESDETAIL DELETE
> OUTPUT SALESHISTORY ADD AT FINAL
>             ITEM SALESYEAR FINAL FLOOR(SYSDATE / 10000) - 1
>             ITEM YEARLYSALES FINAL TOTALSALES
> GO
```

The SORTED statement tells QTP that control breaks occur at the item SALESREPCODE. As a result, a control break is performed when the value for the item SALESREPCODE changes.

SUBFILE

Directs output to a subfile.

Syntax

SUBFILE filespec [options]...
INCLUDE entity [, entity]...
[INDEX name [index-option]
[SEGMENT item [ASCENDING|DESCENDING]
[,item [ASCENDING|DESCENDING]...]...

filespec

Names the subfile to be created.

OpenVMS, UNIX, Windows: Do not specify the file suffix for a subfile, since the subfile must consist of two files, a datafile and a minidictionary file. If the subfile has no minidictionary, there is only one file. For a regular subfile, QTP appends the file suffix .sfd to the minidictionary portion of a subfile while the data portion is assigned the file suffix .sf. Both filenames must be identical. A subfile without a minidictionary is assigned the suffix .dat.

For information about portable subfiles, see the PORTABLE option on (p. 153).

INCLUDE [FILE|ITEM] entity [,entity]...

Lists the record-structures, subfiles, or items that QTP copies to the subfile. If you specify a record-structure, PowerHouse copies all the items of that record-structure to the subfile. An item can be a record item, temporary item, or defined item, but not a subscripted item.

The maximum number of items that can be written to a subfile depends on the format that is used. For more information, see the FORMAT option on (p. 152). The maximum record size is 32, 767 bytes. The maximum size of all input and output records, and defined, temporary, and global temporary items in a request is 65,535 bytes.

When a record-structure or subfile is specified, the full syntax is:

[FILE] record-structure|subfilespec

When an item is specified, the full syntax is:

[ITEM] item [type[*size] [type-option]] [OF filespec]
[ALIAS name]

[type[*size] [type-option]]

Defines the physical format if different from the existing format. If the item type is DATE, the size specification is not valid.

For more information about items, datatypes, and sizes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

The type-options are CENTURY, NUMERIC, SIGNED, UNSIGNED, and SIZE.

CENTURY INCLUDED|EXCLUDED

Use CENTURY INCLUDED for a four-digit year (2001); use CENTURY EXCLUDED for a two-digit year (99).

Limit: Valid only for date values; must immediately follow the DATE keyword.

Default: CENTURY EXCLUDED, unless otherwise specified in the CENTURY INCLUDED|EXCLUDED option of the SYSTEM OPTIONS statement in the data dictionary.

NUMERIC

Indicates that the datatype ZONED is to have a type of ZONED NUMERIC rather than RIGHT OVERPUNCHED NUMERIC.

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED [WHEN POSITIVE]

Specifies that the defined item can store negative as well as positive values. The WHEN POSITIVE option is valid only for unsigned defined items of datatype PACKED or ZONED, and it suppresses the printing of a sign for positive values.

Limit: Valid only for INTEGER, PACKED, and ZONED defined items; must be specified immediately following the INTEGER, PACKED, or ZONED keyword.

SIZE m [BYTES]

Defines the storage size in bytes for the item. Ensure the specified size accommodates the largest possible value assigned to the item.

Limit: Size isn't valid for date datatypes (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE) or the item types NUMERIC and INTERVAL.

OF filespec

If several files are accessed, and item names are duplicated in those files, use OF filespec to clarify which item is to be written to the subfile.

ALIAS name

Assigns an alternative name to the item. This option allows QTP to distinguish between two identically-named items that are written to the same subfile.

INDEX name [index-option]...

Indicates that the subfile is an indexed subfile and names the index to be referenced. If the name is an item from the INCLUDE list or file, the item is considered an index segment. If the name is not an item, a segment must be specified using the SEGMENT option.

Limit: You can't use the INDEX option with the PORTABLE subfile-option.

The index-options are ALTERNATE, PRIMARY, ASCENDING, DESCENDING, ORDERED, UNORDERED, REPEATING, and UNIQUE.

ALTERNATE|PRIMARY

Specifies whether the index is an alternate index or a primary index. An indexed subfile always contains one primary index, and can also contain alternate indexes.

Default: The first index defined is the primary index by default. All subsequent indexes are alternate indexes.

ASCENDING|DESCENDING

Specifies the default sort order of the index.

Default: ASCENDING if no sort order is specified on segments.

Limit: MPE/iX only accepts ASCENDING.

ORDERED|UNORDERED (MPE/iX)

Specifies whether the index records for duplicate keys are stored in chronological (ORDERED) or random (UNORDERED) order.

REPEATING|UNIQUE

REPEATING specifies that records may have the same index values. UNIQUE specifies that every record in the subfile must have a unique index value.

Default: REPEATING

SEGMENT item [ASCENDING|DESCENDING]

Specifies the name of an item in the record structure. The item must be in the INCLUDE list.

ASCENDING (MPE/iX, UNIX, Windows)|DESCENDING (UNIX, Windows)

Specifies the default sort order of the segment.

Limit: A sort order cannot be specified for a segment if a sort order is included on the INDEX option.

Default: ASCENDING

Options

SUBFILE options		
ALIAS	APPEND|NOAPPEND	AT FINAL
AT INITIAL	AT [START[OF]]	DICTIONARY|NODICTIONARY
FORMAT	IF	KEEP|TEMPORARY
ON ERRORS	PORTABLE	SIZE

ALIAS name

Assigns an alternative logical name to the subfile. Once assigned, subsequent references to the subfile must use the alias.

APPEND|NOAPPEND

APPEND adds data records to the end of the existing subfile. The minidictionary describing the contents of the original subfile is not altered.

If the subfile has a different format from the information to be appended and the record sizes are the same, QTP appends to the subfile. However, problems arise when you try to access the subfile because the minidictionary contains only the original record layout.

If the subfile has a different record size from the information to be appended, QTP issues an error.

NOAPPEND creates a new subfile. If the subfile already exists, QTP erases existing subfile data records before adding the new data records.

Default: NOAPPEND

AT FINAL
AT INITIAL
AT [START [OF]] sort-item

Controls the timing of the output to the subfile.

AT FINAL

Indicates an output action at the end of the transaction set after processing all transactions.

AT INITIAL

Indicates an output action at the beginning of the transaction set before processing any transactions.

AT [START [OF]] sort-item

AT sort-item indicates the output action is to take place at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group.

DICTIONARY|NODICTIONARY

DICTIONARY writes subfile format information to a subfile minidictionary. A subfile created with NODICTIONARY does not have a related subfile dictionary and cannot be accessed by PowerHouse components. The NODICTIONARY option is ignored when overwriting an existing permanent subfile.

MPE/iX: A nonportable subfile created with the DICTIONARY option has a file code of 644. A nonportable subfile created with NODICTIONARY has a file code of 0, and is simply a file full of data.

Default: DICTIONARY

FORMAT 0|3|5|6|7|8 (MPE/iX)
FORMAT 4|7|8 (OpenVMS)
FORMAT 3|5|6|7|8 (UNIX)
FORMAT 8 (Windows)

The format dictates how much information is kept about items in the subfile.

Attention must be given to format when creating subfiles that must be compatible with an earlier version of PowerHouse. However, compatibility is not an issue when using later versions of PowerHouse. A format introduced with one version of PowerHouse is supported in later versions of PowerHouse.

The following table lists compatible PowerHouse versions for the various formats:

Platform	Format Option	Compatible PowerHouse Version
MPE/iX:	0 and 3	all
	5	6.09 and higher
	6	7.09 and higher
	7	7.29 and higher
	8	8.09 and higher
OpenVMS:	4	5.04 and higher
	7	7.x0 and higher
	8	8.x0 and higher
UNIX:	3	all
	5 and 6	6.03 and higher
	7	7.23 and higher
	8	8.x
Windows:	8	8.01 and higher

Limits: FORMAT 0 subfiles do not accept dates with four-digit years. FORMAT cannot be used with portable subfiles.

Default: 8

IF condition

Writes subfile data records only if this condition is satisfied.

KEEP|TEMPORARY

KEEP creates a permanent subfile that may be used in future sessions. TEMPORARY creates a temporary subfile that the system automatically deletes when the user leaves QTP (**OpenVMS, UNIX, Windows**) or logs off (**MPE/iX**). Use temporary subfiles when the information is needed only for the current session.

Default: Portable subfiles default to KEEP. Nonportable subfiles default to TEMPORARY.

ON ERRORS option

Determines the action taken if errors are encountered while the subfile data record is being written.

The ON ERRORS options are BYPASS UNTIL, REPORT and TERMINATE.

BYPASS UNTIL sort-item|FINAL

Skips processing of all transactions until the control break or until the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

REPORT

Reports the error and skips processing of this transaction for this subfile. Processing continues, however, as if the error had not occurred.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run.

Default: TERMINATE RUN

PORTABLE [DATAFILE] filespec

Creates a portable subfile in ASCII format to simplify intermachine transfer.

The PORTABLE option is saved in compiled runs.

MPE/iX:	Portable subfiles in MPE/iX have a "Q" appended to the subfile name indicating the data portion. For example, the following subfile: `SUBONE PORTABLE INCLUDE FILEA` produces the following subfiles: SUBONE as the Portable data dictionary and SUBONEQ as the Portable data portion.
OpenVMS, UNIX, Windows:	For a portable subfile, QTP appends the file suffix .psd to the minidictionary portion and the file suffix .ps to the data portion.

For information about how regular subfiles are named, see (p. 149).

DATAFILE filespec

The file specification for the data file of a PowerHouse portable subfile. The DATAFILE option lets you specify a file specification for a data file that is different from the dictionary portion of the portable subfile. For example,

```
SET SUBFILE abc PORTABLE DATAFILE xyz
```

When referencing the subfile, use the open name for the dictionary file. The open name for the data file is contained in the minidictionary and is used by the operating system when creating the physical file.

Although you can qualify both the dictionary file name and the data file name with a location, the data file qualification is not be saved in the dictionary. So although the files is written to the locations specified, the data file cannot be located because the qualification is lost. Portable subfiles are intended to be used to transfer data from one platform to another. Including the qualification in the data file name might render the data file unusable on a platform with a different filespec format.

Limit (**OpenVMS**): If you specify a logical name for the datafile name, then you must specify a logical name for the dictionary name. The logical paths you specify for the datafile name and the dictionary must be identical.

SIZE n (MPE/iX)

Sets the maximum number of records (n) that can be written to the subfile. This option is ignored when overwriting an existing permanent subfile.

Default: The lower of the PROCESS LIMIT or INPUT LIMIT.

TEMPORARY

See KEEP|TEMPORARY on (p. 153).

Discussion

The SUBFILE statement either creates a self-describing direct or indexed file and adds new data records, or adds new data records to an existing subfile.

Include all options for each subfile in one SUBFILE statement:

```
> SUBFILE CUSTOMER KEEP ON ERRORS TERMINATE RUN &
>   INCLUDE ORDERS
```

PowerHouse stores subfiles as externally-described files. These files contain both the data and the information that describes the data layout or data record of the subfile. The exception to this rule is interim subfiles, which do not contain data.

Using Subfiles

You can use subfiles to

- store and access data produced by QTP runs
- extract and manipulate data from existing files for use in other runs or programs
- temporarily hold data
- pass data between QTP and QUIZ or QUICK
- transfer data between requests
- process runs in two or three passes (one pass for data manipulation and another for request results)
- extract and build data files from large databases
- process data from more than one file with no common index
- create files for use by other systems and languages

Types of Subfiles

There are three types of subfiles:

- interim
- temporary
- permanent

QTP creates subfiles at the following stages:

BUILD	creates an interim subfile
EXECUTE	creates permanent and/or temporary subfiles, overwrites any existing interim subfiles.
GO	creates permanent and/or temporary subfiles, overwrites any existing interim subfiles

Interim Subfiles

Interim subfiles are externally described subfiles that do not contain data.

QTP uses interim subfiles when it requires subfile metadata at parse-time, which is before QTP has actually created either a temporary or permanent subfile. QTP does not use interim subfiles at execution time.

QTP deletes interim subfiles without warning when it creates the corresponding temporary or permanent subfile. This is true even if you specify the APPEND option, or if record-structure differences are present.

QTP overwrites interim subfiles when it creates another interim subfile with the same name.

When you try to create an interim subfile when a temporary subfile with the same name already exists, QTP issues a warning asking if you want to delete the existing temporary subfile.

Temporary Subfiles

The TEMPORARY option creates a temporary subfile that the system automatically deletes when you exit QTP (**OpenVMS, UNIX, Windows**) or log off (**MPE/iX**), unless you use the **phtempkeep** program parameter (**UNIX, Windows**). When you create a temporary subfile in a batch job, the system deletes it when the batch job ends.

Permanent Subfiles

The KEEP and PORTABLE options create permanent subfiles that can be used in other sessions or batch jobs.

Indexed Subfiles

PowerHouse applications can use subfiles to pass information between QUICK, QUIZ, QTP, and other applications. Subfiles may be either direct or indexed. Indexed subfiles have the following advantages:

- They may reduce the number of passes required by QTP and QUIZ for data manipulation.
- They may allow you to expand dictionary functionality without recompiling QUICK screens.
- They don't need to be converted for use by other applications that require indexed subfiles.

Indexed subfiles can have single or multiple indexes with single or multiple segments.

Recreating Regular Subfiles and Adding Indexes (MPE/iX)

Subfiles can be recreated and/or overwritten by using the same subfile name. If you attempt to recreate and/or overwrite a regular subfile, and you add an index to the new subfile, you get an error message and the new subfile is not created:

```
EXECUTING REQUEST <NAME> ...
*E* SUBFILE ALREADY EXISTS AS A DIRECT FILE
RECORDS READ:
  <DATA-STRUCTURE>                0
TRANSACTIONS PROCESSED:           0
RECORDS PROCESSED:    ADDED    UPDATED    UNCHANGED   DELETE
   <SUBFILE NAME>        0          0           0         0
```

The error occurs because you are trying to go from a direct format to an indexed format. When this situation occurs, simply delete or rename the existing subfile before you attempt to recreate or overwrite it.

Accessing Subfiles

Use the ACCESS statement to access subfiles by prefixing the physical file name with an asterisk (*). You may specify an alias that can be used for subsequent subfile references. If you do not specify an alias, PowerHouse will assign a logical name that must be used in subsequent references to the to the subfile, for example in OUTPUT statements. The asterisk indicates that the file is a subfile rather than a file declared in the data dictionary. A subfile that is referenced for the first time in an OUTPUT statement must use the physical file name and the asterisk (*), as in the ACCESS statement. Also, if you reference a subfile from the ACCESS statement in an OUTPUT ADD statement, you must use the physical file name, an asterisk (*) and specify an alias.

Subfiles created by QTP can also be used as input files to QUIZ and QUICK. Likewise, QUIZ subfiles can be used as input to QTP and QUICK.

Regular subfiles must be the primary file unless you link to the subfile by record number. The first record of the file is 0 (**MPE/iX, UNIX, Windows**) or 1 (**OpenVMS**). You may link to indexed subfiles in the same manner as any indexed file.

You can see a subfile record layout by using the SHOW SUBFILE statement of QSHOW.

Locking Subfiles (UNIX, Windows)

By default, PowerHouse uses a file lock when writing to subfiles on UNIX and Windows. To specify record locking when writing to subfiles, use the syntax:

SET FILE <subfilename> OPEN SHARE

For more information about the SHARE option of the SET FILE OPEN statement, see (p. 128).

Updating Subfiles

Include the name of the subfile in the OUTPUT statement to update a subfile.

ITEM statements with the FINAL option can be used to change the value of items included in the subfile.

Actions QTP Performs on Subfiles

If a SUBFILE statement specifies the name of an existing subfile, QTP does one of the following at parse time:

- purges the existing subfile and creates an interim minidictionary
- uses the existing subfile's minidictionary
- reports an error

Once the run is executed, if a subfile other than the minidictionary created at parse time exists, QTP does one of the following:

- retains the existing file and modifies it. The APPEND option affects the processing in the following ways:
  - if the APPEND option is specified, new records are added to the existing subfile
  - if the APPEND option is not specified, the data portion of the subfile is overwritten by new records
- purges the existing subfile and creates a new subfile
- reports an error

For more information about how QTP locates subfiles, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

How QTP Writes Items to Subfiles

QTP writes data record items to the subfile with the same format as the original object.

QTP writes defined, temporary, and global temporary items to the subfile with the same format as the corresponding DEFINE, TEMPORARY, and GLOBAL TEMPORARY statements.

If a character item is null, QTP substitutes blanks when writing the item to the subfile. If a numeric or date item is null, QTP writes zeroes to the subfile.

An entire array can be written to a subfile if it is specified as the unsubscripted item name in the INCLUDE option. To include a single occurrence of an array in a subfile, create a defined item set to the subscripted item using the DEFINE statement, and specify the defined item in the INCLUDE option.

How QTP Writes Substructured Items to Subfiles

If a record-structure containing an item substructure is included in the INCLUDE list, the substructure is written to the subfile. Otherwise, QTP only writes the specified items of the substructure.

Assume the following structure:

```
> RECORD EMPLOYEES
> ITEM FULLNAME
```

```
> BEGIN STRUCTURE
>   ITEM LASTNAME
>   ITEM FIRSTNAME
>   END STRUCTURE
> ITEM POSITIONCODE
```

The statement

```
> SUBFILE NEWSUB KEEP INCLUDE FULLNAME
```

produces a subfile with the item:

```
FULLNAME
```

However, the statement

```
> SUBFILE NEWSUB KEEP INCLUDE EMPLOYEES
```

produces a subfile with the item substructure:

```
FULLNAME
•LASTNAME
•FIRSTNAME
POSITIONCODE
```

Limits

You can reference a maximum of 31 record-structures (including subfiles) per request.

You can reference a maximum of 63 record-structures (including subfiles) per run.

You can write a maximum of 1023 items to a subfile.

Examples

Using Multiple Subfiles

The following example demonstrates a request that builds four subfiles to extract and manipulate data from one input record-structure. The subfile TAPEDATA contains summary information about each salesrep's performance for the period. The other subfiles are used to provide sales information by amount of the item TOTALSALES.

```
> RUN MONTHEND
> REQUEST SALESTOT
>
> ACCESS SALES &
>   LINK TO EMPLOYEES &
>   AND TO BRANCHES
>
> SUBFILE TAPEDATA KEEP &
>   INCLUDE EMPLOYEENO, TOTALSALES, &
>   BRANCH, SALESMONTH
>
> SUBFILE DIST KEEP IF TOTALSALES < 1000000 &
>   INCLUDE EMPLOYEENO, &
>   TOTALSALES, SALARY
>
> SUBFILE MERIT KEEP IF TOTALSALES > 500000 AND &
>   TOTALSALES < 1000000  &
>   INCLUDE EMPLOYEENO, &
>   TOTALSALES, SALARY
>
> SUBFILE IMPROVE KEEP IF TOTALSALES < 500000 &
>   INCLUDE EMPLOYEENO, &
>   TOTALSALES, SALARY
> GO
```

Extracting Data Records for Storage in a Subfile

This example shows how the SUBFILE statement is used to extract data records from files and store them in a subfile for later use. The selection criteria ensures that only salesreps whose names match the pattern specified are selected for retrieval. The SUBFILE statement uses the INCLUDE option to identify items that are stored in the TYPESET subfile.

```
> RUN CAMPAIGN
> REQUEST EXTRACT
> ACCESS CUSTOMERMASTER LINK TO SALESREPMASTER
> SELECT IF &
>   MATCHPATTERN (CHARACTER(SALESREPCODE),"(K|L)@")
> SUBFILE TYPESET KEEP INCLUDE &
>   CUSTOMERNAME OF CUSTOMERMASTER, &
>   STREET OF CUSTOMERMASTER, &
>   CITY OF CUSTOMERMASTER, &
>   STATE OF CUSTOMERMASTER, &
>   ZIPCODE OF CUSTOMERMASTER, &
>   CONTACT OF CUSTOMERMASTER, &
>   SALESREPCODE OF CUSTOMERMASTER, &
>   SALESREPNAME OF CUSTOMERMASTER
> BUILD
```

Indexed Subfile example

In the following example, the subfile TOTALS is created. The subfile has a unique index, SALES-IDX, consisting of two segments, EMPLOYNO and MTHYEAR:

```
> SUBFILE TOTALS &
>   INCLUDE EMPLYNO, MTHYEAR, TOTALSALES &
>   INDEX SALES-IDX UNIQUE &
>   SEGMENT EMPLOYNO, MTHYEAR
```

If the subfile were not indexed, a sort would be required to get the records in the correct sequence:

```
> ACCESS *TOTALS
> SORT ON EMPLOYNO ON MTHYEAR
> OUTPUT SALES ADD AT EMPLOYNO
> GO
```

With indexed subfiles, however, you can use the CHOOSE statement, provided the subfile is the primary file and the desired order is that of the index. This avoids the extra overhead of a SORT statement:

```
> ACCESS *TOTALS
> CHOOSE VIAINDEX SALES-IDX
> SORTED ON EMPLOYNO ON MTHYEAR
> OUTPUT SALES ADD AT EMPLOYNO
> GO
```

TEMPORARY

Creates a temporary item for the duration of the request that does not exist in the data dictionary.

Limit: There can be a combined maximum of 1023 defined and temporary items in a request. The maximum size of all input and output records, and defined, temporary, and global temporary items in a request is 65,535 bytes.

Syntax

TEMPORARY name [type[*n] [type-option]]

name

Names the temporary item.

Limit: The name must begin with a letter and can be up to 64 characters in length.

type[*n]

Establishes the physical format of the temporary item.

type

Specifies the datatype for the temporary item.

For more information about items, datatypes, and sizes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Default: NUMERIC*6

*n

Specifies the size of the defined item.

For character items, *n specifies the maximum number of alphanumeric characters that the item can contain.

For all datatypes that store numeric values, *n specifies the maximum number of digits that the numeric item can contain.

Limit: You can't specify a size (*n) when you define a date datatype (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE). The size of a DATE item is controlled strictly by the CENTURY INCLUDED|EXCLUDED option.

type-option

Specifies the type-options for the temporary item.

The type-options are CENTURY, SIGNED, UNSIGNED, and SIZE.

CENTURY INCLUDED|EXCLUDED

Specifies whether the year component of a date item includes a century prefix.

Limit: Valid only for the JDATE, PHDATE and DATE datatypes; must immediately follow the type in the statement.

Default: Determined by the dictionary.

NUMERIC

Indicates that the datatype is to have a type of ZONED NUMERIC rather than RIGHT OVERPUNCHED NUMERIC.

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED [WHEN POSITIVE]

Sets the range for INTEGER, PACKED, and ZONED. For datatypes PACKED and ZONED, unsigned items can store both positive and negative values. For datatype INTEGER, an unsigned item can store positive values only.

The following is a list of qualifications and exceptions for this option:

- This option is valid only for data types INTEGER, PACKED, and ZONED and must immediately follow the type specification.
- For INTEGER, this option specifies whether PowerHouse interprets the number as a two's complement binary number (SIGNED) or as an absolute binary number (UNSIGNED).
- For PACKED and ZONED, this option specifies whether the item includes a sign (SIGNED) if positive or negative, or only when negative (UNSIGNED).
- The WHEN POSITIVE option is valid only for PACKED UNSIGNED or ZONED UNSIGNED, and is used only for documentation.

Default: UNSIGNED for ZONED; SIGNED for INTEGER and PACKED.

SIZE m [BYTES]

Specifies a storage size in bytes. Use SIZE m BYTES if specifying *n leads to an undesired default storage size.

If you use both *n and SIZE m, you must ensure that they don't conflict with each other.

BYTES is used for documentation only.

Limit: SIZE isn't valid for date datatypes (DATE, DATETIME, INTERVAL, PHDATE, JDATE, VMSDATE, ZDATE) or the item types NUMERIC and INTERVAL.

Discussion

The TEMPORARY statement creates and defines a temporary item that does not exist in the data dictionary. The statement remains in effect for the request in which it was declared.

When to Use the TEMPORARY Statement

Temporary items can be used in calculations and summary operations in the output phase.

If an item doesn't change for the duration of a run or is required in different requests, use a global temporary item. The ITEM statement can reference temporary items.

Limit: The FINAL option of the ITEM statement can't be used with temporary items.

Advantages

Temporary items are more efficient than defined items when the value of the item does not change during the request.

Defined items are re-evaluated each time they are referenced unless the defined item got its value from a parm, in which case, it is evaluated only once. This re-evaluation increases processing time.

Disadvantages

If you reference a temporary item in input or sort-phase statements, then it may lead to unpredictable results since the items aren't initialized until the output phase.

[SQL] UPDATE

Updates rows in a table.

Syntax

```
    [SQL [IN database] [sql-options] [IF condition]
    [ON ERRORS {BYPASS UNTIL sort-item|FINAL}|REPORT|
        {TERMINATE [REQUEST|RUN]}]]
UPDATE tablespec SET column-name = {sql-expression|NULL}
[,column-name = {sql-expression|NULL}]...
    [WHERE sql-condition|{DBKEY = :expression}]
```

sql-options

Specifies the timing of the SQL UPDATE. The update is performed at the specified sort-break.

The sql-options are AT FINAL, AT INITIAL, and AT [START [OF]].

AT FINAL

Performs the update at the end of the transaction set after processing all transactions. This option has no effect unless there is an input phase.

AT INITIAL

Performs the update at the beginning of the transaction set before processing any transactions. This option has no effect unless there is an input phase.

AT [START[OF]]sort-item

The sort-item option indicates that the update is performed at the control break. With the START OF option, the action is performed at the beginning of the transaction group before processing any transactions in the group. Without the START OF option, the action is performed at the end of the transaction group after processing all transactions in the group. This option is not available unless a request has a sort phase.

IF condition

Performs the update only if the condition is satisfied. This option affects the processing sequence.

ON ERRORS

States what action to take if errors are encountered while performing the update.

Default: TERMINATE RUN

BYPASS UNTIL sort-item|FINAL

Skips processing all transactions until it reaches either the control break or the end of the current QTP request. Final operations at the control break or at the end of the request are still performed.

REPORT

Reports the error and skips processing of the transaction. Processing continues as if the error had not occurred.

TERMINATE [REQUEST|RUN]

Terminates the current QTP request or run, and rolls back any uncommitted updates.

UPDATE tablespec

The name of a table or view in a relational database to be updated. The syntax for tablespec is:

[[server-name.]database-name.][owner-name.]table-name

If server-name is included in a Sybase tablespec, double quotes are required for the server-name and database-name. For example,

```
"DBSRV01.ACCNT".MANAGER.BILLINGS_TBl
```

For Oracle, the syntax is:

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
MANAGER."BILLINGS_TBl@DBLNK01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

WHERE sql-condition|DBKEY = :expression

Sql-condition identifies the rows of the table to be updated. If you don't specify a WHERE clause, all the rows in the relational table are changed. DBKEY is available only if the underlying database supports it.

The sql condition is a condition which is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book, or refer to an SQL reference manual.

Limit: DBKEY can't be used with Sybase.

Discussion

The UPDATE option acts directly on a table or view in the database and is never generated by PowerHouse.

USE

Processes QTP source statements that are contained in a file.

Syntax

USE filespec [option]

filespec

Names the file that contains the QTP source statements or a compiled QTP run you want to use.

If the file doesn't exist, QTP issues an error message.

Options

The options are DETAIL, NODETAIL, INPUT, LIST, NOLIST, and PROCESS.

DETAIL|NODETAIL

Writes the contents of the file to QTP's source statement save file, QTPSAVE, rather than just the USE statement itself. NODETAIL writes just the USE statement to the temporary save file rather than the contents of the file being used.

Default: DETAIL

INPUT [LIMIT] n

Specifies the maximum number of transactions (n) to be selected for processing.

Limit: Valid for compiled run files only.

LIST|NOLIST

Displays the statements as they are processed; NOLIST does not.

Default: LIST

PROCESS [LIMIT] n

Specifies the maximum number of transactions (n) to be processed.

Limit: Valid only for compiled run files.

Discussion

The USE statement instructs QTP to process the named file for statement input.

If the file contains source statements, QTP reads and interprets each statement as if it had been entered from the terminal.

If the file is a compiled QTP run, QTP loads the file and immediately begins execution.

Nesting USE Statements

A file referenced in a USE statement can itself contain other USE statements. USE files can be nested to a maximum of 20 levels. Permanent files containing valid source code can be included at any time provided they are consistent with QTP syntax and structure.

If limits are specified on a combination of EXECUTE, USE, REQUEST, and SET statements, the order of precedence is as follows:

- EXECUTE or USE
- REQUEST
- SET

For information about how QTP locates USE files, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the USE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Examples

The following example demonstrates the use of the USE statement in processing QTP source statement files. The NODETAIL option tells QTP not to write the statements to the temporary save file, QTPSAVE. The second USE statement gives an example of a nested USE statement that introduces a permanent file of QTP statements.

```
> USE MONTHEND NODETAIL
```

This USE statement initiates the processing of the following statements which are included in the file MONTHEND:

```
> RUN MONTHEND
> REQUEST MONTHLY-BILLING-TOTALS
>
> ACCESS BILLINGS LINK TO EMPLOY1
>
> CHOOSE EMPLOYEE
>
> SORTED ON EMPLOYEE
>
> TEMPORARY EMPLBILLINGS NUMERIC*8
>      ITEM EMPLBILLINGS SUBTOTAL BILLINGAMOUNT
>
> SUBFILE TOTBILLS KEEP AT EMPLOYEE &
>         INCLUDE EMPLOYEE OF EMPLOY1, &
>         BRANCH, &
>         EMPLBILLINGS
>
> USE SETSTMTS NOLIST
```

The preceding USE statement shows one method of using a nested USE statement. The permanent file SETSTMTS contains the following statements:

```
> SET INPUT LIMIT 500
> SET NOSTATISTICS
> SET NOWARNINGS
```

Smaller USE files within larger ones is a good way to maintain a flexible and complex run file.

Chapter 4: QTP Tracer

Overview

This chapter provides information about QTP's debug tracing features. You'll find information about

- QTP Tracer syntax
- QTP Tracer options
- debugging multiple request runs
- QTP Tracer output
- summary operations
- RESET and INITIAL actions
- data conversion errors
- debugging examples

QTP Tracer allows you to display the details of QTP's processing. Tracing is enabled using SET syntax.

QTP is driven by an internal action table, which it creates at parse time. The entries in this table contain information about what action is to take place, on what, from what, when, and under what conditions. QTP Tracer takes this information when an action is performed and tries to relate it back to the original QTP syntax wherever possible. It adds internal information such as timing and condition success when possible, and displays a formatted output trace message for the user. QTP Tracer also gives as much information as it can about data conversion errors, and during which activity they occurred.

QTP Tracer is built in a way that should have minimal impact on production runs. Trace structures are allocated only when tracing is initiated, and all trace code checks for the existence of this structure before performing any tracing. So when tracing is not enabled, there is only the additional overhead of checks for a NULL pointer where tracing would be done.

To use the tracing feature, you can specify the tracing options you want anywhere before a GO and the current run will be traced. Alternatively, you can specify generic tracing options and then EXECUTE a compiled run.

Tracer syntax is listed in two sections: "Generic Syntax" and "Selective Syntax". Generic Syntax is available any time during QTP. Selective Syntax, an enhancement to the Generic Syntax, extends some of the options and is only available during the parsing of a request.

Tracing options are not saved in compiled runs.

QTP Tracer allows you to trace various actions. Due to the nature of QTP, turning off too much detail may remove the context of the action, and the trace may not make sense. If you want to select only some options, then it is best to just SET TRACE first to set all defaults on and then use SET TRACE with specific options to turn off selected details, or turn on those that are not on by default. Doing a SET TRACE with a list of options first will only turn on/off those options that you specify. CANCEL and SET DEFAULT do not affect trace settings. A SET NOTRACE will clear trace settings.

QTP Tracer Syntax

Generic Syntax

SET TRACE|NOTRACE [option]

No options specified sets all trace options on, with the exception of DEFINE and m to n (record range tracing).

Options

Tracer Syntax options		
DEFINE|NODEFINE	DETAIL|NODETAIL	{FILE[ALL|INDEX]}|NOFILE
ITEM|NOITEM	SORT|NOSORT	WAIT|NOWAIT
m [TO n]		

DEFINE|NODEFINE

Traces defined items when they are evaluated and are listed interspersed with other trace statements.

DEFINE statements will occur primarily in two formats:

- in between ITEM BEFORE and AFTER if the evaluation is during ITEM processing
- before an OUTPUT or SUBFILE statement where they are required for condition processing.

NODEFINE does not trace defined items.

Default: NODEFINE

DETAIL|NODETAIL

Traces detail level processing. NODETAIL suppresses tracing of records that do not cause a control break. Tracing actually starts with the detail output for the record before the control break and includes all control break output. This option is used to isolate activity around control breaks and limit the trace volume of output.

{FILE [ALL|INDEX]}|NOFILE

Traces all file activity (OUTPUT and SUBFILE statements). NOFILE suppresses output and subfile tracing.

ALL

Provides full record item details. ALL is the default and is assumed if nothing is specified. For relational tables, only items that are referenced or updated are reported.

INDEX

Displays only index information. For relational tables, only those indexes with segments that are referenced or updated are reported, and only segments referred to are reported.

ITEM|NOITEM

Traces all item level actions (record items, temporary items, and global temporary items). NOITEM does not trace item level actions.

ITEM INITIAL statements for record items are not traced individually, but are shown in the FILE trace for an INITIALIZE RECORD action. ITEM FINAL statements are traced individually under a FINAL VALUES heading, in a BEFORE and AFTER format.

ITEM record item = statements may be grouped with ITEM FINALs depending on the AT options specified.

The ITEM FINAL trace may not include all items in the list, or may list none due to the optimization that QTP does by performing block moves, where possible. QTP Tracer does not trace block moves.

SORT|NOSORT

Displays the sort or intermediate file general contents and size, and notes if an intermediate file is used when there is no sort. This trace shows at parse time only after a GO or a BUILD.

This does not list the data that goes into the sort or intermediate file, just the general contents and size.

NOSORT does not provide sort or intermediate file information.

WAIT|NOWAIT

Displays the following prompt at the bottom of each full screen of trace information.

```
Press Return to continue.
```

NOWAIT displays continuous tracing without stopping at each full screen.

The WAIT page size is set so that the Continue prompt will occur on the last line of a 24-line terminal. However, there are times when this will not happen. QTP Tracer only counts trace output lines, not normal QTP output, so the Continue prompt may be after more than 24 lines. If your data causes a trace message to wrap over one line, these extra lines are not counted by the trace facility.

m [TO n]

Starts to trace a range of records at record complex m and optionally turns it off after record complex n.

Selective Syntax

In order to decrease the amount of output from QTP Tracer, you can also apply Selective Tracing to parsed code.

From the previous discussion on tracing, we know that non–selective trace statements can be specified at any time in the parsing cycle. However, selective trace statements can only be set after the associated file/items have been defined.

For example, SET TRACE ITEM X must follow an ITEM X statement. (The TEMP X statement is not enough.) A DEFINE X must follow the corresponding DEFINE statement. A SET TRACE FILE X statement can follow any statement defining access to the file OUTPUT, SUBFILE).

Selective Tracing is enabled by using some additional qualifiers on the SET TRACE ITEM, FILE, and DEFINE options.

Selective Tracing Options

The options are:

- ITEM <itemname>[,<itemname>…]
- DEFINE <itemname>[,<itemname>…]
- FILE <filename> [ALL|INDEX] [,<filename> [ALL|INDEX]…]
- FILE <filename> INCLUDE <itemname> [,<itemname>…] [FILE…]

ITEM <itemname>[,<itemname>...]

Traces ITEM actions only for items on the list. This activity does not affect OUTPUT tracing.

DEFINE <itemname>[,<itemname>...]

Traces all actions that evaluate these defined items.

Note: These items may be harder to interpret if the related ITEMS are not also being traced.

FILE <filename> [ALL|INDEX] [,<filename>... [ALL|INDEX]...]

Traces just the files listed for either ALL or INDEX as noted for each file. No other files are traced.

Note: For multiple occurrences of the same FILE, QTP will use the last definition.

FILE <filename> INCLUDE <itemname> [,<itemname>...] [FILE...]

Traces selected items for the files listed. Once an INCLUDE is used, the next file must be signaled by a FILE (with no comma in front of it). The INCLUDE is also terminated by any other keyword (INDEX, DEFINE, and so on) which has no comma in front of it. For example:

Valid syntax:

SET TRACE FILE X ALL, Y INDEX, P INCLUDE X, Y, FD FILE Q ALL

Invalid syntax:

SET TRACE FILE P INCLUDE X, Y, FD, X ALL, Y INDEX, Q ALL

QTP Tracer Options

Multiple Request Runs

Multiple request runs that use different files cannot be traced as a run using Selective Tracing. They must be broken up into separate requests, each with a GO and a new SET TRACE statement(s) in between. All Selective Tracing information will be cleared between REQUESTS. You will receive a warning when this occurs. Tracing for the portion that had Selective Tracing (ITEM|DEFINE|FILE) will be turned off. The appropriate trace information must be set up for each request. Selective tracing cannot be used with compiled files because the selection cannot be parsed prior to the request being loaded.

Clearing Messages

I Warning—Selective Item tracing has been cleared.

I Warning—Selective File tracing has been cleared.

I Warning—Selective Define tracing has been cleared.

These messages will be displayed when trace information is cleared.

Precedence Rules

Multiple SET TRACE FILE X statements whether on one line or separate lines will replace the options for file X, the last one being the one used. A SET TRACE FILE will remove all selective tracing selections for the FILE option and return to a simple trace (for either ALL or INDEX as specified). A SET TRACE Y following a SET TRACE X will just add Y to the list. Previously defined files are not removed except by SET TRACE FILE with no files listed, SET TRACE NOFILE, or the end of the request.

A SET TRACE INDEX or DEFINE of any form replaces previously defined criteria specified for that kind of tracing.

SHOW TRACE

SHOW TRACE lists the positive or negative for each item indicated. If an option is on or off and if record range tracing is on, it includes the range.

Example

```
SHOW TRACE
TRACE SORT NOITEM FILE ALL NODEFINE DETAIL WAIT
```

For Selective Tracing:

```
TRACE SORT NOITEM FILE SELECTIVE NODEFINE DETAIL WAIT
```

Trace Format

QTP Tracer can create a large amount of output, but it is broken up by action blocks such as:

AT INITIAL	NEW RECORD COMPLEX	AT START OF x value
DETAIL	AT END OF x value	AT FINAL

These groupings help to provide the context of when an action is being performed so that the timing can be related to the code that initiated the action. The use of SET REPORT LIMIT n can be used in a test situation to limit the amount of output.

NULL values will be displayed as a blank space, by default, or whatever is set as your dictionary null value character if nulls are allowed in an item.

Trace Output

One of the best uses for QTP Tracer is to help you learn how QTP timing works without using numerous subfiles to try to figure out what's happening. You can focus on items only and turn off file tracing by using the NOFILE keyword, or you can enable just files and use NOITEM, or enable both ITEM and FILE. QTP Tracer outputs its actions as it does them. Each action trace contains timing information and is output in the order processed, so you can see which timing caused an item to be updated, or a file OUTPUT to be done. You'll see the ITEM statements being processed in the order they happen so you'll be able to see the effects of your timing specifications. You can then change the timing, redo the trace, and see the altered behavior caused by the changes.

Before using QTP Tracer, it is important to understand the trace output. QTP Tracer can generate a large amount of output with everything turned on. To cut down on the output, use selective tracing. You can also redirect standard output to a file. (Tracer output goes to standard output.) As for the general format of QTP Tracer output, the output comes in blocks of information for each record complex and for each sort break action level within each record complex. Also, there are blocks of output from ITEM traces and OUTPUT traces which follow a consistent format (discussed below). The first thing you will see is the SORT trace block, output at parse time after a GO or BUILD. This is followed by the following:

```
*
TRACING RUN START ; indicates that the OUTPUT phase tracing is about to begin
*
```

The Global Temp initialization trace output comes next if there is any. (See the section, "Anatomy of an ITEM Trace Output", on (p. 170).) Next, the request message comes out from standard QTP and the first record complex trace statement:

```
Executing request 1 ... ; standard QTP output
# NEW RECORD COMPLEX -> 1 ; first record complex being traced
```

The trace outputs that were requested follow, and are described in greater detail on (p. 170) through to (p. 171). The example below shows the basic form of QTP Tracer outputs; the indentation structure indicates which sort level you are at. The ellipse represents trace output.

```
AT INITIAL PROCESSING ; occurs within first record block
...
...
* PROCESSING AT START OF xxx <sort value>
*...
*...
** PROCESSING AT START OF yy <sort value>
**...
**...
*** DETAIL PROCESSING
***...
***...
** PROCESSING AT END OF yy <sort value>
**...
**...
* PROCESSING AT END OF xxx  <sort value>
```

```
*...
*...
AT FINAL PROCESSING ; occurs at end of last record block
...
...
# NEW RECORD COMPLEX -> 2 ; next record
...
Executing request 2 ...
```

SORT trace output will only exist if there is a SORT being done or a scratch file being used. Here is an example of SORT output:

```
Scratch File Being Used ; this trace output will exist if no sort is being
                          used but a scratch file is. The following information
                          will relate to the contents of the scratch file
                          rather than the sort file.
Sort/Scratch File Size

    Record EMPLOYEES  - Record Size 112  - Total Size 188

    Record SKILLS  - Record Size 14  - Total Size 90

    Sort Key Area Size 4

  Sort/Scratch File Size (in bytes) 282
```

Each record in the ACCESS statement will have a single line showing the record size and the total space required by QTP to process this record, including internal overhead. Following this will be a line indicating the amount of space in the sort record containing the sort key information. This line is only reported for sort file usage, not scratch files. The last line is the total size of the record for the sort or scratch file.

Suppressing Repeating Details

If you choose the NODETAIL option to suppress tracing of records that do not cause a sort break, you will see QTP Tracer messages as follows:

```
 == Details Suppressed ==
```

This indicates that record tracing has been suppressed for one or more records.

Anatomy of an ITEM Trace Output

Here is an example of an ITEM trace output.

```
* ITEM XXY = : DETAIL
    BEFORE        1234567890123
    AFTER       1234567890123456
```

or

```
* ITEM itemname action : timing if condition note
     BEFORE         beforevalue
     AFTER          aftervalue
```

Following is a description of what each piece means and can contain:

*****

A level indicator of one or more asterisks showing the sort level of this trace statement. No asterisks means either AT INITIAL or AT FINAL.

ITEM

Indicates that this is an ITEM, TEMPORARY, or GLOBAL TEMPORARY trace statement.

itemname

Shows which item the trace statement refers to. (This is the alias if one is used.)

OF record

Appears after the item name if this is a record item. (Record may be an alias.)

action

Comes from the action on the ITEM statement and is one of the following:

- SUBTOTAL
- AVERAGE
- MAXIMUM
- MINIMUM
- COUNT
- =

Regardless of the actual function being traced, the action from the ITEM statement will be shown so that you can group trace outputs together and tie them back to the original ITEM statement that caused the trace. The single exception for this is the two-phase ITEM initialization where the first phase is always an "=", not the action on the ITEM statement. The two phases are often traced together in the AT INITIAL phase, so the proximity allows you to connect the two trace outputs. At other times, the "=" is in the AT INITIAL phase and the other will be later. The two phases only occur with summary action items, and are part of the accumulator processing.

: timing

One of the following:

INITIAL

Specifies initialization of item, either default or as specified on the ITEM statement.

DETAIL

Indicates that no timing is specified on the ITEM statement, and this is the actual execution of the ITEM statement function.

AT xxx

Sets the sort level of timing if there is any associated with the action. If there is no AT on the ITEM, this will not be on any trace statements for that ITEM.

AT FINAL

Appears if the ITEM statement had an AT FINAL clause.

RESET AT xxx

Resets at this level.

IF CONDITION FAILED

Appears if there was an IF condition on the ITEM statement and it evaluated to false. This will only show up for items using summary actions. For simple assignments (INITIAL and FINAL values), the IF is part of the expression, not the action, and therefore isn't traced.

Discussion

The ITEM trace output excluding the BEFORE and AFTER values will be on one line. Following that will either be a BEFORE and AFTER line with the value of the item before the action and as a result of the action, or just an AFTER line which is used only for item initialization where there is no BEFORE value.

The trace output for an item will usually contain an INITIAL trace, a DETAIL, AT xxx, or AT FINAL trace, and possibly a RESET AT trace output.

For example, the QTP source `ITEM XXY = ...` could have the following possible QTP Tracer output:

```
ITEM  XXY = : INITIAL
   AFTER 1234567890123456     ; value of XXY after initialization
* ITEM XXY = : DETAIL
*    BEFORE  1234567890123456 ; value of XXY before the action
*    AFTER   123456789012334  ; value of XXY after the action
```

The value of XXY both before and after assignment is shown, but just after for initialization. : DETAIL indicates that the timing is at detail level and the "*" indicates the number of the level (in this case there was no SORT or SORTED so detail level is level 1).

Anatomy of a DEFINE Trace Output

The trace contains the asterisks as in the ITEM statement followed by "DEFINE name" and the value. If define tracing is enabled, this trace line will show up when the DEFINE is evaluated, thus you need to trace sufficient information around the DEFINE so that you can tell when it is being processed.

For example:

```
QTP SOURCE:
 def xx int*10 = 19900504
 def xxd int*8 = days(xx)
 access skills
 temp yy date
 item yy subtotal employee of skills if 3 = xxd initial xx
```

Trace output:

```
DEFINE XX             19900504
 AT INITIAL PROCESSING
   ITEM  YY =   : INITIAL
     AFTER
    DEFINE XX             19900504
    DEFINE XXD            32996
   ITEM  YY SUBTOTAL   : INITIAL  IF CONDITION FAILED
     AFTER
```

xx is evaluated and traced for the first phase initial value (the first ITEM YY =… trace) and then xxd is evaluated when the IF condition is done for YY on the second phase of the initialization. xxd uses xx so it is evaluated again and traced, then xxd is evaluated and traced, followed by the ITEM YY SUBTOTAL.

Anatomy of a FILE Trace Output

The FILE trace has some of the same pieces as the ITEM trace as well as several of its own. Here is an example of a FILE trace:

```
* OUTPUT SKILLS UPDATE   : DETAIL
*   EMPLOYEE           00001
*   SKILL              COBOL
```

or

```
* OUTPUT filename action : timing   if condition note
*   itemname     itemvalue
*   itemname     itemvalue
```

*****

A level indicator of one or more asterisks showing the sort level of this trace statement. No asterisks means either AT INITIAL or AT FINAL.

SUBFILE

Indicates that this is a SUBFILE trace statement.

OUTPUT

Indicates that this is an OUTPUT trace statement.

filename

Shows which file the trace statement refers to. (This is the alias if one is used.)

action

Comes from the action on the FILE statement and is one of the following:

- ADD
- UPDATE ADD (always this, regardless of the OUTPUT statement order)
- UPDATE
- DELETE

: timing

One of the following:

INITIAL RECORD

Specifies initialization of a record, not in the ACCESS statement, that will be added.

READ RECORD

Specifies the record was not in the ACCESS statement and has been read for output.

FINAL VALUES

Reports the results of FINAL VALUES.

DETAIL

Specifies that no timing was specified on the FILE statement and this is the actual execution of the OUTPUT or SUBFILE statement.

AT xxx

Indicates the sort level or timing if there is any associated with the action. (If there is no AT on the FILE, this will not be on any trace statements for that FILE.)

AT START xxx

Indicates an output action at start of sort break rather than at end.

AT FINAL

Appears if the FILE statement had an AT FINAL clause.

IF CONDITION FAILED

Appears if there was an IF condition on the FILE statement and it evaluated to FALSE, no actual put will be done.

UNCHANGED

Appears if the record was unchanged and therefore will not actually be put to the file.

Discussion

The action information of the FILE trace output will be on one line. Following that will be a list of items in one of two formats. For FINAL VALUES, each file item will be in the form of ITEM xxx followed by a BEFORE and AFTER. For this form, there is no IF CONDITION clause on the ITEM line, as the IF in this case is part of the expression, not part of the action. For all other FILE traces there is just a list of items by name with their values. For INITIALIZE RECORD, the values are those after initialization. For READ RECORD, the values are those read in. For DETAIL, the values are those that will be put on file (if not UNCHANGED or IF CONDITION FAILED).

The trace output for a file will always contain one of DETAIL | AT xxx | AT START xxx | AT FINAL. For non-ACCESSed or for ADD files, there will be a READ RECORD or INITIALIZE RECORD trace as well. This trace will be at the start of the NEW RECORD COMPLEX loop. The other file traces will be further down. If the file items have final values or ='s with no timing, there will also be a FINAL VALUES trace.

For SET TRACE FILE INDEX, the above listings will include only index items. The output form will be:

```
index_name
   segment_name segment_value
   segment_name segment_value
```

For relational files, the items listed for files will include only those items referenced.

For record items, there are a few special cases that don't trace as ITEM traces, but which will be found in FILE traces. They are:

ITEM INITIAL statements are not traced individually, but are shown in the FILE DETAIL trace for an INITIALIZE RECORD action.

ITEM FINAL statements are traced individually under a FINAL VALUES heading, in a BEFORE and AFTER format.

ITEM = statements with no timing will be bundled with ITEM FINALs.

Summary Operations

Summary items usually create two actions, one to initialize the item and one to actually do the action requested. At initialization time, both actions are performed. At other times, only the summary action is performed. Summary processing is actually done in an accumulator or holding area, not in the item buffer itself. When the action is performed, the value in the holding area is moved to the item buffer for use by QTP.

RESET and INITIAL

A QTP Tracer RESET AT action actually only takes the value from the holding area and moves it to the item buffer. The timing of this action is at END of a control break. The only time when this will appear to change the contents of the buffer is when you use the ITEM for both a summary action and a non-summary action, or for multiple summary actions which will overwrite each other. In these cases, the action that is labeled RESET AT (because it is actually triggered by the RESET AT in your ITEM statement) will take the value in the holding area and move it to the item buffer. If you have changed the item buffer by some other action, it will be overwritten. Coding in this manner may not give you the desired results.

The actual RESET (TO) action takes place at the start of the control break and is labeled INITIAL. There are usually the two actions described previously in the summary operations, both with the INITIAL action. Most of the time these two actions appear to do the same thing.

Conditions

Conditional summary operations will have a failure of the IF condition noted in the trace because the IF is tied to the action. However, the ITEM = will not have the failure of an IF condition noted because the IF is just part of the conditional expression that results in a value, and is not tied to the action.

Data Conversion Errors

Where possible, additional information is given about the data conversion error, for example, whether it is in the IF condition or in a subscript expression. If DEFINE tracing is turned on, then a DEFINE resulting in a data conversion error will indicate "DATA CONVERSION ERROR" in place of the value. Not all data conversion errors can be traced directly to their source, but the trace provides as much information as is available.

Debugging with the QTP Tracer

Earlier we mentioned that when code is parsed, entries are generated into an internal action table. Each entry triggers actions based on settings of AT, RESET AT, source of data, IF conditions, and so on. QTP takes these tables and loops through them, performing the appropriate "action" at each point in the processing. A single action table entry can generate multiple traceable actions such as initializing a record, doing final values for a record, and outputting a record. QTP Tracer traces the actions as they are performed.

Sometimes data conversion errors (DCEs) can be difficult to pin down. With QTP Tracer you can narrow down what the problem is because when a DCE occurs, QTP Tracer attempts to tell you what kind of action it was trying to perform (for example, an ACCESS link, an IF statement, or an assignment). This information, combined with the tracing of DEFINEs, usually will tell you what part of the code is falling down. It won't pinpoint which function is the cause if multiple functions are used for a single define, but it will give more information than is presently provided.

The next section will take examples and discuss how QTP Tracer output ties back to the original code, and how it can help you debug your code by seeing what actions it really generates, and the timing of these actions.

Connecting OUTPUT Traces

There are three main types of file traces.

- The first possibility is a file that is in the ACCESS statement and in an OUTPUT UPDATE/DELETE statement. QTP Tracer will not have any tracing indicating the reading of the record, but will have an OUTPUT UPDATE or OUTPUT DELETE trace. All types of scenarios may have an OUTPUT : FINAL VALUES trace block.
- The second possibility is a file that is not in the ACCESS statement and which has an OUTPUT UPDATE or DELETE. QTP Tracer will produce an OUTPUT : READ RECORD for a record read for update/delete. This trace will come at the start of a record complex tracing block. The corresponding OUTPUT : FINAL VALUES and actual OUTPUT action trace will come in the section associated with the timing on the source OUTPUT statement.
- The third possibility is a new record for an OUTPUT ADD. An OUTPUT : INITIALIZE RECORD trace will come at the start of the record complex and the actual output traces will happen at their associated timing.

For example:

```
access skills
output skills update
```

This produces a trace with just the :DETAIL output block.

```
Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* DETAIL PROCESSING
*   OUTPUT SKILLS UPDATE   : DETAIL
*     EMPLOYEE            00001
*     SKILL               COBOL
```

Here's another example:

```
access employees
output skills add update
item skill = "new skill"
```

This produces a trace with either a READ RECORD for records that are to be updated, or an INITIALIZE RECORD for those new SKILLS records, as well as an UPDATE ADD: DETAIL. Notice that the trace does not tell you specifically whether the actual OUTPUT was an update or an add. The previous INITIALIZE RECORD, READ RECORD or, in the case of an update from an ACCESSed file, the lack of either tells you which situation is the correct one.

```
# NEW RECORD COMPLEX ->  1
* DETAIL PROCESSING
*   OUTPUT SKILLS    : READ RECORD
*     EMPLOYEE            00001
*     SKILL               COBOL
*   ITEM  SKILL OF SKILLS =   : DETAIL
*     BEFORE              COBOL
*     AFTER               new skill
*   OUTPUT SKILLS UPDATE ADD   : DETAIL
*     EMPLOYEE            00001
*     SKILL               new skill
...
# NEW RECORD COMPLEX ->  13
* DETAIL PROCESSING
*   OUTPUT SKILLS    : INITIALIZE RECORD
*     EMPLOYEE            00013
*     SKILL
*  ITEM  SKILL OF SKILLS =   : DETAIL
*     BEFORE
*     AFTER               new skill
*   OUTPUT SKILLS UPDATE ADD   : DETAIL
*     EMPLOYEE            00013
*     SKILL                new skill
```

Examples

The following table of data will be used in some of the examples:

PROJECT	EMPLOYEE	BILLING MONTH	BILLING	EXPENSE
100	11	1992/06/01	30	15
		1993/03/01	50	5
		1994/02/01	10	-
	25	1993/06/01	20	25
		1993/12/01	60	-
200	11	1991/11/01	40	-
	33	1992/01/01	70	15
		1992/03/01	10	-

Example 1: Summary Item Accumulator

The first example demonstrates why the SUBTOTAL doesn't quite do what you expect in this case. We expect it to add all the billings together and put the total into PROJECTAMOUNT. Notice that in the trace output you can see that PROJECTAMOUNT is initialized each time a record is read, with the result that the SUBTOTAL is done only on the value of the last record read.

```
access projects link to billings
sorted on project of projects
output projects update
  ITEM projectamount SUBTOTAL billing

Executing request 1 ...
# NEW RECORD COMPLEX ->  1
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
**     AFTER ; resetting project amount
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   :
**     BEFORE ; resetting project amount
```

```
**     AFTER
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER              30.00
**   OUTPUT PROJECTS UPDATE   : DETAIL
**     PROJECT            P000100
**     PROJECTNAME        PAP 1
**     PROJECTAMOUNT      30.00
**     PROJECTMGR         Bob Deskin
**     PROJECTBALANCE
# NEW RECORD COMPLEX ->  2
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
**     AFTER ; resetting project amount
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   :
**     BEFORE ; resetting project amount
**     AFTER
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER              50.00
**   OUTPUT PROJECTS UPDATE   : DETAIL
**     PROJECT            P000100
**     PROJECTNAME        PAP 1
**     PROJECTAMOUNT      50.00
**     PROJECTMGR         Bob Deskin
**     PROJECTBALANCE
```

The following example demonstrates the solution and the trace output that it generates. The AT PROJECT forces the resetting of the accumulator to happen only at the sort break, allowing the accumulation to actually take place as you want. Note that the record item accumulator gets initialized to 0 at the start of the run and at each project break. Note also that the BEFORE value for the record item is also 0, but this comes from the record data itself. The AFTER value is not the amount of billing, but the subtotal of the accumulated billings stored in the accumulator, which is what you want. Note the RESET AT that occurs at the end of the sort break as previously discussed, and which doesn't appear to do anything. The INITIAL action at the start of the next record complex does the actual resetting.

```
access projects link to billings
sorted on project of projects
output projects update at project
  ITEM projectamount SUBTOTAL billing

Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF PROJECT P000001
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER ; resetting project amount only once
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE ; accumulating as desired
**     AFTER               30.00
# NEW RECORD COMPLEX ->  2
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               80.00
# NEW RECORD COMPLEX ->  3
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               90.00
# NEW RECORD COMPLEX ->  4
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER              110.00
# NEW RECORD COMPLEX ->  5
** DETAIL PROCESSING
```

```
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               170.00
* PROCESSING AT END OF PROJECT P000100
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : RESET AT PROJECT
*     BEFORE             170.00
*     AFTER              170.00
*   OUTPUT PROJECTS UPDATE   : AT PROJECT
*     PROJECT            P000100
*     PROJECTNAME        PAP 1
*     PROJECTAMOUNT      170.00
*     PROJECTMGR         Bob Deskin
*     PROJECTBALANCE
```

Example 2: Multiple Summary Accumulators and RESET AT

The next example shows an attempt to accumulate both billings and expenses into the same target item. Billings is subtotaled positively and expenses negatively. QTP Tracer output makes it look as if PROJECTAMOUNT is oscillating between positive and negative subtotals. In this case two accumulators are being used, one for each ITEM statement. As each record is processed, first the positive accumulator (BILLING) is moved into the record buffer, then the negative one (EXPENSE). The order is that of the original code. The result is that the final subtotal only includes the expenses, and is negative.

```
access projects link to billings and to expenses
sorted on project of projects
 output projects update at project
   ITEM projectamount SUBTOTAL billing
   ITEM projectamount SUBTOTAL expense NEGATIVE

Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF PROJECT P000100
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               30.00 ; positive accumulation
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE              30.00 ; negative accumulation different
**     AFTER               -15.00 ; accumulator
# NEW RECORD COMPLEX ->  2
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               80.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE              80.00
**     AFTER               -20.00
# NEW RECORD COMPLEX ->  3
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               90.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE              90.00
**     AFTER               -45.00
# NEW RECORD COMPLEX ->  4
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               110.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE              110.00
**     AFTER               -45.00
```

```
# NEW RECORD COMPLEX ->  5
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER               170.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE              170.00
**     AFTER               -45.00
* PROCESSING AT END OF PROJECT P000100
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : RESET AT PROJECT
*     BEFORE             -45.00 ; oscillating between accumulators
*     AFTER              170.00 ; pseudo RESET AT
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : RESET AT PROJECT
*     BEFORE             170.00 ; oscillating between accumulators
*     AFTER              -45.00 ; pseudo RESET AT
*   OUTPUT PROJECTS UPDATE   : AT PROJECT
*     PROJECT            P000100
*     PROJECTNAME        PAP 1
*     PROJECTAMOUNT      -45.00
*     PROJECTMGR         Bob Deskin
*     PROJECTBALANCE
*     PROJECTBILLINGS
*     PROJECTEXPENSES
*     EXPENSEFLAG
*     REDFLAG
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF PROJECT P000200
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER                     ; Actual RESET of item
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER                     ; Etc.
```

The solution is to change the code so that only one accumulator is being used so that both positive and negative values go against the same subtotal. Now the two subtotaling actions do what you desire: adding the billing and subtracting the expense from the final total that is put into PROJECTAMOUNT. Notice that PROJECTAMOUNT starts off as zero (blank) for each record because when the record is read, the PROJECTAMOUNT is zero until the record is updated AT PROJECT. The AFTER value confirms that the accumulator is being used even with the starting zero and the correct amount is being moved into the record buffer each time, even though it isn't actually output until the end of the project.

```
access projects link to billings and to expenses
sorted on project of projects
output projects update at project
  ITEM projectamount SUBTOTAL billing, expense NEGATIVE

Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF PROJECT P000100
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER       30.00 ; positive accumulation
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE      30.00 ; negative accumulation, same accumulator
**     AFTER           15.00
# NEW RECORD COMPLEX ->  2
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER       65.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE      65.00
**     AFTER       60.00
```

```
# NEW RECORD COMPLEX ->  3
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER       70.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE      70.00
**     AFTER       45.00
# NEW RECORD COMPLEX ->  4
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER       65.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE      65.00
**     AFTER       65.00
# NEW RECORD COMPLEX ->  5
** DETAIL PROCESSING
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE
**     AFTER        125.00
**   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : DETAIL
**     BEFORE       125.00
**     AFTER        125.00
* PROCESSING AT END OF PROJECT P000100
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : RESET AT PROJECT
*     BEFORE       125.00 ; pseudo RESET AT as mentioned
*     AFTER        125.00 ; in discussions above
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : RESET AT PROJECT
*     BEFORE       125.00
*     AFTER        125.00
*   OUTPUT PROJECTS UPDATE   : AT PROJECT
*     PROJECT      P000100
*     PROJECTNAME  PAP 1
*     PROJECTAMOUNT 125.00
*     PROJECTMGR   Bob Deskin
*     PROJECTBALANCE
*     PROJECTBILLINGS
*     PROJECTEXPENSES
*     EXPENSEFLAG
*   REDFLAG
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF PROJECT P000200
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER               ; Actual RESET of item
*   ITEM  PROJECTAMOUNT OF PROJECTS SUBTOTAL   : INITIAL
*     AFTER               ; Etc.
```

Example 3: Timing Issues

The following example demonstrates how timing problems can show up through the trace. The original code is trying to output a record with the total of all project billings by project and year. It doesn't quite work, as you can see by the trace output. Assume there is already one record on the project summary file for 1992.

```
access billings
choose project 100
select if employee = 11
TEMPORARY temp-year INTEGER SIZE 4
ITEM temp-year = FLOOR(month / 10000)
output project_summary ADD UPDATE &
  VIA project USING project of billings
  SELECT project_summary IF year = temp-year
  ITEM project FINAL project of billings
  ITEM project_billing FINAL project_billing + billing
  ITEM year FINAL temp-year
set trace
go
```

```
Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* DETAIL PROCESSING
*   OUTPUT PROJECT_SUMMARY   : INITIALIZE RECORD ; the first input record is
1992
*     PROJECT              P000100
*     YEAR                 0 ; we wanted to get the 1992 record but temp-year
hasn't been set yet so the SELECT failed to retrieve 1992.
*     PROJECT_BILLING
*   ITEM  TEMP-YEAR =    : DETAIL
*     BEFORE               0
*     AFTER                1992 ; now it's set! - too late
*   OUTPUT PROJECT_SUMMARY UPDATE ADD   : FINAL VALUES
*    ITEM PROJECT
*      BEFORE              P000100
*      AFTER               P000100
*    ITEM PROJECT_BILLING
*      BEFORE
*      AFTER               30.00
*    ITEM YEAR
*      BEFORE              0
*      AFTER               1992
*   OUTPUT PROJECT_SUMMARY UPDATE ADD   : DETAIL
*     PROJECT              P000100
*     YEAR                 1992
*     PROJECT_BILLING      30.00
# NEW RECORD COMPLEX ->  2
* DETAIL PROCESSING
*   OUTPUT PROJECT_SUMMARY    : READ RECORD
*     PROJECT              P000100
*     YEAR                1992 ; still set from previous record, not reset yet
by this output cycle, so retrieves 1992 record to update
*     PROJECT_BILLING      40.00
*   ITEM  TEMP-YEAR =    : DETAIL
*     BEFORE               1992
*     AFTER                1993 ; now set to current value
*   OUTPUT PROJECT_SUMMARY UPDATE ADD   : FINAL VALUES
*    ITEM PROJECT
*      BEFORE              P000100
*      AFTER               P000100
*    ITEM PROJECT_BILLING
*      BEFORE              40.00
*      AFTER               90.00
*    ITEM YEAR
*      BEFORE              1992
*      AFTER               1993
*   OUTPUT PROJECT_SUMMARY UPDATE ADD   : DETAIL
*     PROJECT              P000100
*     YEAR                 1993
*     PROJECT_BILLING      90.00
# NEW RECORD COMPLEX ->  3     ; Etc.
```

During initialization QTP attempts to retrieve a project summary record and uses the SELECT file IF. … Note that the value used for the TEMP-YEAR in the SELECT is whatever is in the item buffer at the time of retrieval. This is the root of our problem. The evaluation of TEMP-YEAR is not done until output processing because the = option is done at output processing time, not initialization. We actually wanted it done before retrieval.

So, how do we get the TEMP-YEAR to initialize at the right time? We need it to be set at the start of initialization. A RESET AT will do that, but for that we need a SORT break to set it at. Since we don't really want or need to do a sort, use SORTED ON. Here's the new code and its trace output.

```
access billings
choose project 100
select if employee = 11
SORTED on month
TEMPORARY temp-year INTEGER SIZE 4
ITEM temp-year RESET AT month TO FLOOR(month / 10000)
```

```
output project_summary ADD UPDATE &
  VIA project USING project of billings
  SELECT project_summary IF year = temp-year
  ITEM project FINAL project of billings
  ITEM project_billing FINAL project_billing + billing
  ITEM year FINAL temp-year
set trace
go

Executing request 1 ...
# NEW RECORD COMPLEX ->  1
* PROCESSING AT START OF MONTH 1992/06/01
*   ITEM  TEMP-YEAR =   : INITIAL ; now temp-year is being set before the
retrieval is attempted
*     AFTER               1992
** DETAIL PROCESSING
**   OUTPUT PROJECT_SUMMARY    : READ RECORD
**     PROJECT             P00010
**     YEAR                1992 ; the 1992 record is retrieved for update
**     PROJECT_BILLING     40.00
** DETAIL PROCESSING
**   OUTPUT PROJECT_SUMMARY UPDATE ADD   : FINAL VALUES
**    ITEM PROJECT
**      BEFORE              P000100
**      AFTER               P000100
**    ITEM PROJECT_BILLIN
**      BEFORE              40.00
**      AFTER               70.00
**    ITEM YEAR
**      BEFORE              1992
**      AFTER               1992
**   OUTPUT PROJECT_SUMMARY UPDATE ADD   : DETAIL
**     PROJECT             P000100
**     YEAR                1992
**     PROJECT_BILLING     70.00
# NEW RECORD COMPLEX ->  2
* PROCESSING AT START OF MONTH 1993/03/01
*   ITEM  TEMP-YEAR =   : INITIAL ; the next input value is set
*     AFTER               1993
** DETAIL PROCESSING
**   OUTPUT PROJECT_SUMMARY    : INITIALIZE RECORD
**     PROJECT             P000100 ; a new record is initialized
**     YEAR                0
**     PROJECT_BILLING
** DETAIL PROCESSING
**   OUTPUT PROJECT_SUMMARY UPDATE ADD   : FINAL VALUES
**    ITEM PROJECT
**      BEFORE              P000100
**      AFTER               P000100
**    ITEM PROJECT_BILLING
**      BEFOR
**      AFTER               50.00
**    ITEM YEAR
**      BEFORE              0
**      AFTER               1993
**   OUTPUT PROJECT_SUMMARY UPDATE ADD   : DETAIL
**     PROJECT             P000100
**     YEAR                1993
**     PROJECT_BILLING     50.00
# NEW RECORD COMPLEX ->  3         ; Etc.
```

Example 4: Data Conversion Errors and DEFINE Tracing

The QTP Tracer provides additional information about DCEs. The following are examples of the kind of trace statements you might see. Here are more examples of where define trace output shows the order of processing, and identifies where the DCE occurred. In this example, errors have been forced by doing things that should not be done.

```
run xdgs
```

```
global temp xxy int*16 initial 1234567890123456
global temp xxg date initial xxy
request xdb on calc errors report on edit errors report
def xx int*10 = 5599559879
def xxd int*8 = days(xx)
access skills
temp yy date
temp zzz date
item zzz = xx
;edit employee of skills lookup on billings via employee using xx
item yy subtotal employee of skills if 3 = xxd initial xx
output employees update ;using xxd via employee if 33 = xxd
item datejoined final xxd

   ITEM  XXY =   : INITIAL
     AFTER              1234567890123456
DCE in GLOBAL ITEM INITIAL of:       ; DCE in a global temp initialization
following item
-------------------------------------------------------------------
Data conversion error.               ; QTP output, not Tracer
   Item:  XXG
-------------------------------------------------------------------
   ITEM  XXG =   : INITIAL           ; item causing the error
     AFTER

Executing request XDB ...
# NEW RECORD COMPLEX ->  1
    DEFINE XX             5599559879 ; define processed to YY
DCE in SET to expression of:         ; result of moving XX to YY caused DCE
-------------------------------------------------------------------
Data conversion error.               ; QTP output, not Tracer
 [1]
   Item:  YY

Action Taken: Report and Continue.
-------------------------------------------------------------------
 AT INITIAL PROCESSING
   ITEM  YY =   : INITIAL
     AFTER
    DEFINE XX             5599559879  ; define processed for XXD for IF
    DEFINE XXD           Data Conversion Error ; DAYS fails in evaluating for
IF
DCE in ITEM IF Condition of:         ; DCE in IF (don't see one for XX because
fails here)
-------------------------------------------------------------------
Data conversion error.                  ; QTP output, not Tracer
 [1]
   Item:  YY

Action Taken: Report and Continue.
-------------------------------------------------------------------
   ITEM  YY SUBTOTAL   : INITIAL IF CONDITION FAILED ; notes that condition
failed
     AFTER
* DETAIL PROCESSING
...
*   ITEM  ZZZ =   : DETAIL
*     BEFORE
*       DEFINE XX            5599559879 ; define done for setting of ZZZ
DCE in SET to expression of:            ; DCE caused in setting AFTER value
-------------------------------------------------------------------
Data conversion error.                  ; QTP output, not Tracer
 [1]
   Item:  ZZZ

Action Taken: Report and Continue.
-------------------------------------------------------------------
*     AFTER
```

```
*       DEFINE XX             5599559879 ; same as init version
*       DEFINE XXD            Data Conversion Error
DCE in IF Condition of:
-------------------------------------------------------------------
Data conversion error.                  ; QTP output, not Tracer
 [1]
   Item:  YY

Action Taken: Report and Continue.
-------------------------------------------------------------------
*   ITEM  YY SUBTOTAL   : DETAIL IF CONDITION FAILED
*   OUTPUT EMPLOYEES UPDATE   : FINAL VALUES
*    ITEM DATEJOINED
*      BEFORE              81/04/23
*       DEFINE XX             5599559879 ; define done for XXD
*       DEFINE XXD            Data Conversion Error ; DAYS of XX fails
DCE in Final Value of:                 ; doing FINAL VALUE of DATEJOINED AFTER
value
-------------------------------------------------------------------
Data conversion error.                  ; QTP output, not Tracer
 [1]
   Item:  DATEJOINED
   File:  EMPLOYEES
     Linkitem:  EMPLOYEE                          00001
     Linkitem:  LASTNAME                          SMITH

Action Taken: Report and Continue.
-------------------------------------------------------------------
*      AFTER              81/04/23
```

You can also get trace output like this when there is a data conversion error in something in the ACCESS statement:

```
DCE in ACCESS linkage ; followed by the normal DCE QTP output block
```

There is an equivalent message for errors found in LOOKUP and OUTPUT linkage.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W