This presentation discusses the enhancements in ESQL exception handling and propagation in WebSphere Message Broker Version 6.

Agenda

- Exception Handling Enhancements
  - EXIT HANDLER
  - CONTINUE HANDLER
  - LOG STATEMENT
- Propagation

The agenda includes a discussion of Exception Handling Enhancements, including the new EXIT HANDLER, CONTINUE HANDLER, and LOG statement, and the enhancements made to propagation.

Exception Handling Enhancements

Exception Handling Enhancements

Handlers

- A handler is a group of statements that are executed when a specific error or exceptional condition is encountered in a programming scope
- Handlers are identified by the DECLARE … HANDLER statement

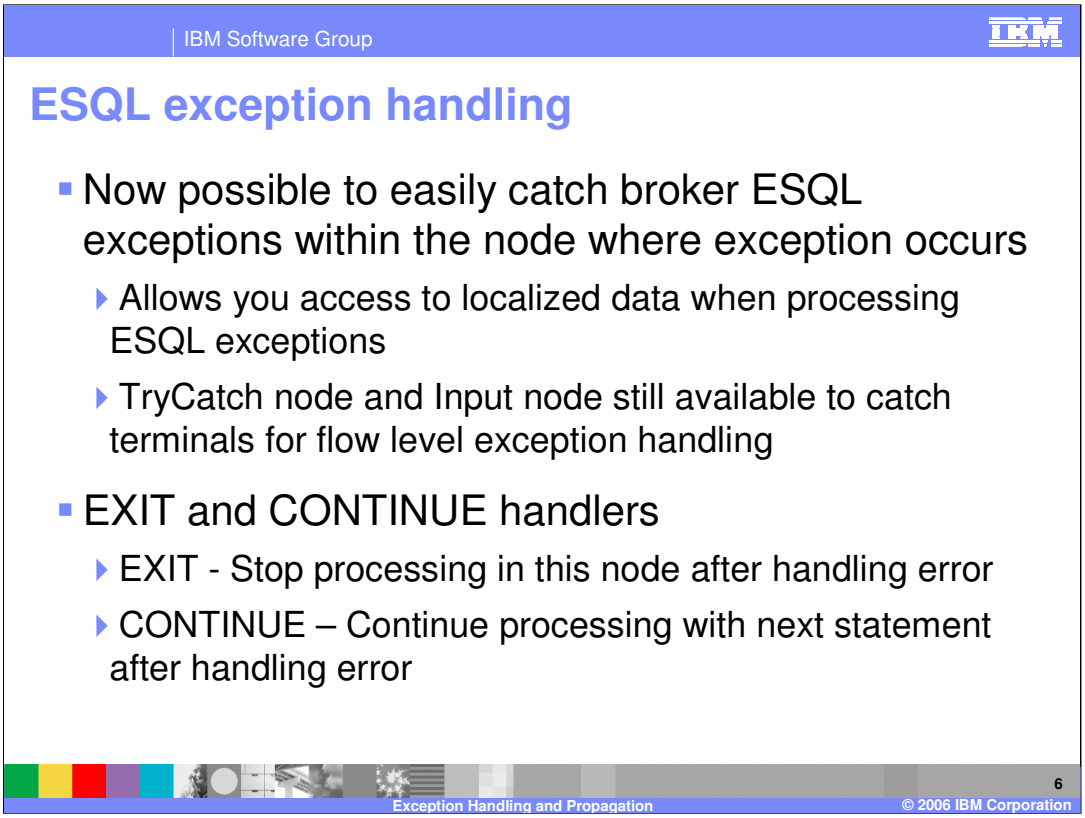
A handler is a group of statements that are invoked when a particular exception occurs within a programming scope. Handlers are identified by the DECLARE … HANDLER keywords.

EQSL handlers in previous releases

- Previously releases generally required a separate node for handling ESQL exceptions
- Much contextual data was lost because the handler was not executing within the same node
- V6 provides a solution for this problem

In previous releases of WebSphere Message Broker, a separate node was generally required for ESQL exception handling. A broker ESQL exception could not easily be handled within the node where the failure occurred. Processing in a separate node resulted in the loss of much contextual information related to the failure.

V6 provides a solution for this problem, eliminating the need for a separate node for handling ESQL exceptions.



In V6, using the new ESQL handlers, it is now possible to easily catch broker ESQL exceptions within the node where the exception occurs. This localized exception handling allows you to process the exception with access to specific contextual information within that node. You can still use the TryCatch node and the Input node catch terminals for flow level exception handling.

Two types of handlers are available:

- a DECLARE EXIT HANDLER, which exits processing in the node when the handler completes, and

- A DECLARE CONTINUE HANDLER, which, when the handler completes, allows processing in the node to continue with the next statement following the failure

Processing the exception

- Processing proceeds as follows
  - Exception is thrown
  - Handlers searched from innermost to outermost until match
  - Set **SQLSTATE, SQLCODE, SQLERRORTEXT, SQLNATIVEERROR**
  - Handler is executed (may **THROW** or **RESIGNAL**)
  - Normal statement of execution is according to handler type
    - **EXIT**: exit processing until higher level TryCatch or Input node is reached
    - **CONTINUE**: after statement which caused exception

Processing the exception proceeds in this manner:

-The exception is thrown

-The handlers are searched from the innermost to the outermost until a match on the conditions occurs

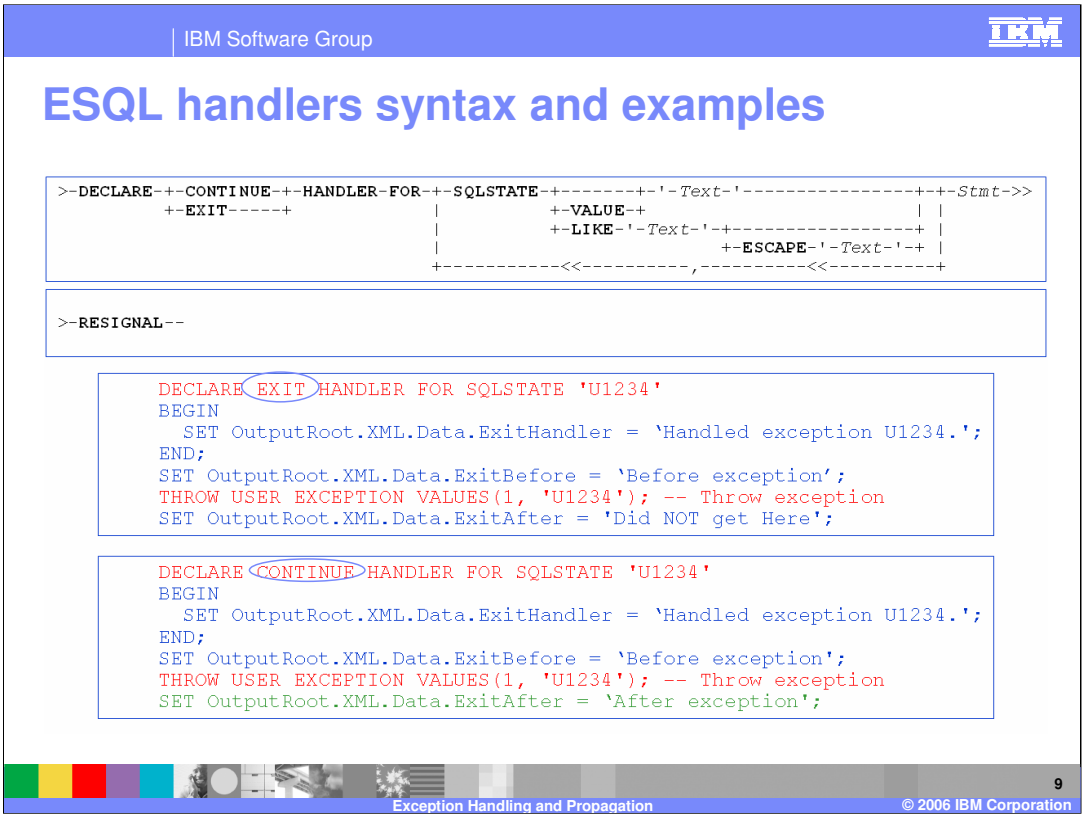
-The SQLSTATE, SQLCODE, SQLERRORTEXT and SQLNATIVEERROR variables are set

-The handler is invoked

-Assuming the handler itself does not issue a throw an exception or issue a resignal, the processing proceeds depending on the type of handler. After an EXIT HANDLER completes, the processing exits until an enclosing TryCatch is invoked or until an Input node is reached. After a CONTINUE HANDLER completes, processing resumes with the next statement following the failing statement.

SQLSTATE

A comprehensive set of SQLSTATEs are provided, enabling you to determine the cause of the exception within your handler code. The slide shows you a summary of these state values. Notice that certain character values within the SQLSTATEs denote a general classification of exception. The WebSphere Message Broker Information Center contains a complete list of the possible SQLSTATE values that WebSphere Message Broker can set.



Here is a summary of the ESQL Handler Syntax. Notice immediately after the DECLARE the two types of HANDLERs – CONTINUE and EXIT.

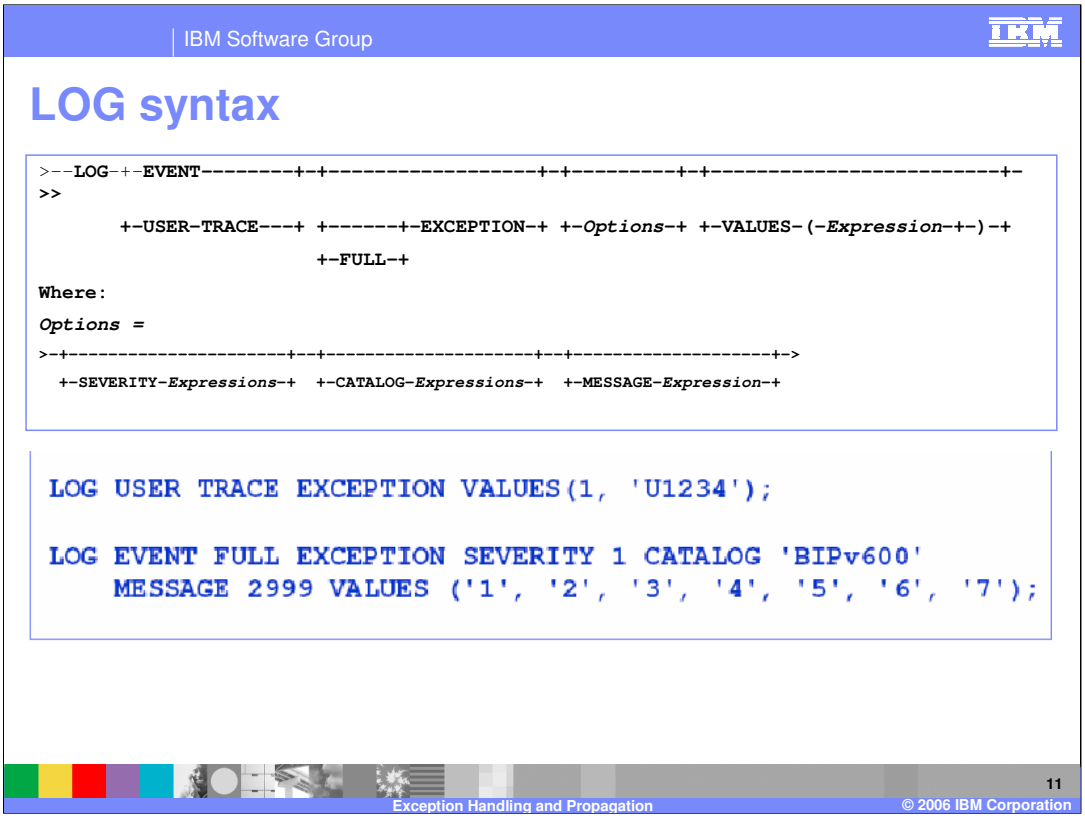
Also note the RESIGNAL statement. RESIGNAL re-throws the current exception. You can use it only in error handlers. Typically, RESIGNAL is used when an error handler catches an exception that it can't handle. The handler uses RESIGNAL to re-throw the original exception so that a handler in higher-level scope has the opportunity to handle it.

Below the syntax summary are two examples that show the flow difference between an EXIT HANDLER and a CONTINUE HANDLER. In the first example, when the EXIT HANDLER completes, processing will exit to a higher level programming scope; thus the last statement in the first example will NOT be executed. In the second example, when the CONTINUE HANDLER completes, processing will return to the statement immediately after the THROW statement.

Logging

- Allows ESQL authors to report message processing information
  - Progress
  - Unexpected conditions
- Variety of destinations supported
  - Windows® event log, UNIX® system logs, z/OS® job logs
  - User trace
- Useful for exception handler reporting

The LOG statement allows ESQL authors to report message processing information. You might want to report the progress of a certain process or report unexpected conditions. Logging supports a variety of destinations, including the Windows event log, UNIX system logs, and z/OS job logs. You can also record information in a user trace record as well. The LOG statement can be very useful in error handlers as well, to report on the invocation and progress of particular types of errors or exceptions.



Here is a summary of LOG syntax.

Below the syntax summary are two simple examples. The first example show logging an exception to a user trace entry. The second example shows logging the full exception information as an event.

Propagation

This section discusses the PROPAGATION statement.

Handling errors in other nodes

- Exceptions occurring in another node as a result of a PROPAGATE statement might be caught by current exception handler
- Current exception handler node might not be correct node to handle exception
- Changes to compute node, database node, and PROPAGATE statement help you route messages to correct node for handling the exception

Exceptions occurring in other nodes downstream of a PROPAGATE statement might be caught by exception handlers. Handling such errors intelligently, however, poses the special problem that, as another node was involved in the original exception or error, another node, and not necessarily the originator of the exception, is very likely to be involved in handling it.

Changes have been made in V6 to the Compute node, the Database node and to the PROPAGATE statement to help you route messages to the appropriate node for exception handling.

New terminals in compute, database nodes

- For V6, “out1”, “out2”, “out3”, and “out4” terminals added to Database node and Compute node
- PROPAGATE statement enhanced to allow it to take advantage of these new terminals

To help in these situations the Database and Compute nodes have four new terminals called out1, out2, out3, and out4. Error handlers can now propagate to these terminals for any required error handling situations. In addition, the syntax of the PROPAGATE statement has been extended to include target expression, message source and control clauses to give more control over these extra terminals.

PROPAGATE

- The syntax of the PROPAGATE statement has been extended to include:
  - Target expression
  - Message source
  - Control clauses

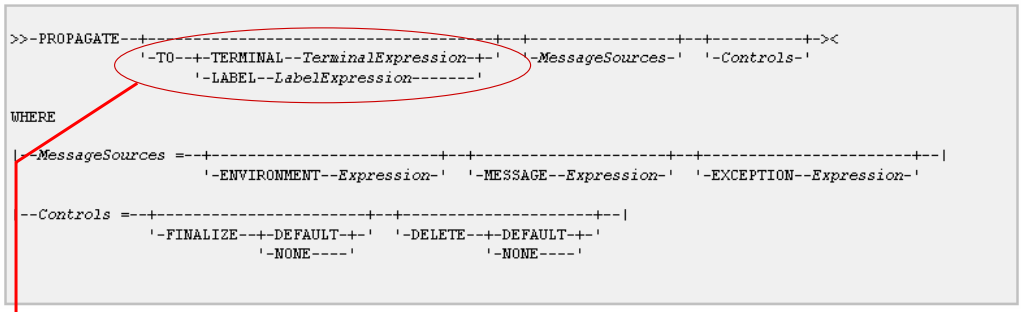
The syntax of the PROPAGATE statement has been extended to include target expression, message source and control clauses to give more control over these extra terminals.

PROPAGATE syntax

```
>>-PROPAGATE--+---------------------------------------+--+----------------+--+----------+-><
              '-TO--+-TERMINAL--TerminalExpression-+-'  '-MessageSources-'  '-Controls-'
                    '-LABEL--LabelExpression-------'

WHERE

|--MessageSources =--+-------------------------+--+---------------------+--+-----------------------+--|
                     '-ENVIRONMENT--Expression-'  '-MESSAGE--Expression-'  '-EXCEPTION--Expression-'

|--Controls =--+-----------------------+--+---------------------+--|
               '-FINALIZE--+-DEFAULT-+-'  '-DELETE--+-DEFAULT-+-'
                           '-NONE----'              '-NONE----'
```


Here is the syntax of the Propagate statement. The next slides will point out the new features.

PROPAGATE syntax



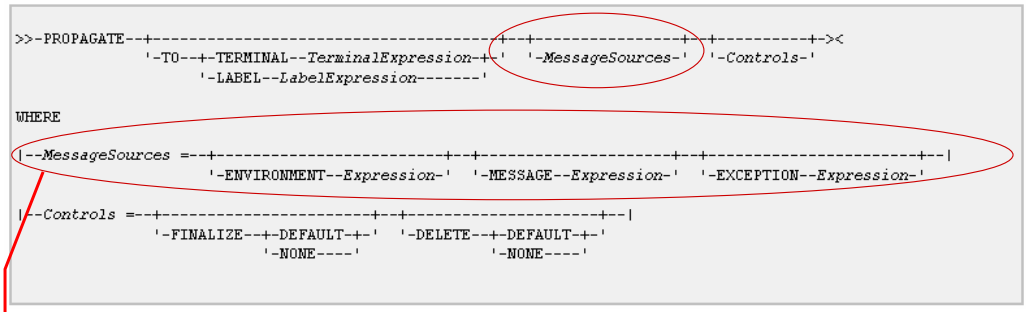
TO TERMINAL – allows you to identify desired output terminal as a target

TO LABEL – allows you to identify a Label node as a target

The TO TERMINAL allows you to identify the desired output terminal to which you wish to propagate.

The TO LABEL allows you to identify a Label node as a target of the propagate.

PROPAGATE syntax



***MessageSources* -** This clause select the message trees to be propagated; applies only to the Compute node (it has no effect in the Database node).

The values that you can specify in MessageSources clauses are:

ENVIRONMENT :	InputLocalEnvironment	OutputLocalEnvironment
MESSAGE :	InputRoot	OutputRoot
EXCEPTION :	InputExceptionList	OutputExceptionList

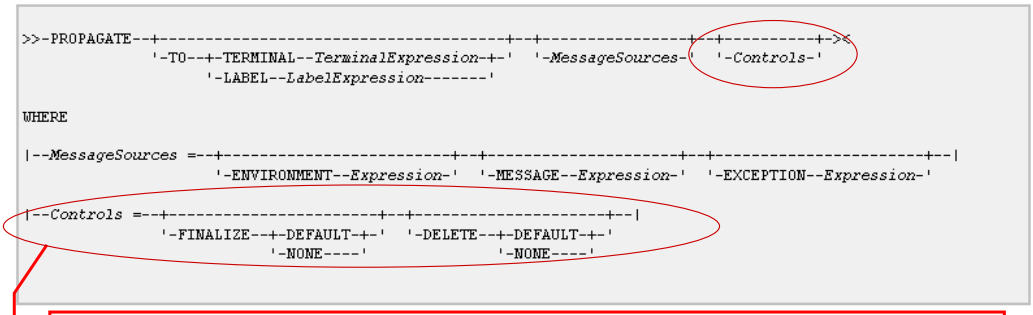
The MessageSources clause allows you to select the message trees to be propagated. It applies to the Compute node only.

ENVIRONMENT allows you to select either InputLocalEnvironment or OutputLocalEnvironment.

MESSAGE allows you to select either the InputRoot or the OutputRoot.

EXCEPTION allows you to select either InputExceptionList or OutputExceptionList.

PROPAGATE syntax



***Controls* -** controls for Finalize and Delete; applies only to Compute node (it has no effect in a Database node)

FINALIZE DEFAULT - fixes header changes, makes Properties folder match headers. The output message (but not the Environment, Local Environment or Exception List) is finalized before propagation.
FINALIZE NONE - no finalization takes place

DELETE DEFAULT – the output local environment, message, and exception list are all cleared and their memory is recovered immediately after propagation.
DELETE NONE – nothing is cleared.

The Controls clause allows you to specify finalization settings and deletion settings. These controls apply only to the Compute node.

FINALIZE DEFAULT fixes the header changes and makes the Property folder match the headers. The output message is finalized before propagation.

FINALIZE NONE indicates that no finalization should take place.

DELETE DEFAULT indicates that the output local environment, the message, and the exception list are all cleared and their memory is recovered immediately after propagation.

DELETE NONE indicates that nothing is cleared.

PROPAGATE examples

```
PROPAGATE TO TERMINAL 'out1'
        ENVIRONMENT      OutputLocalEnvironment
        MESSAGE          OutputRoot
        EXCEPTION        OutputExceptionList
        FINALIZE         NONE
        DELETE           NONE;
```

```
PROPAGATE TO LABEL 'Out1Label'
        ENVIRONMENT      OutputLocalEnvironment
        MESSAGE          OutputRoot
        EXCEPTION        OutputExceptionList
        FINALIZE         DEFAULT
        DELETE           DEFAULT;
```


Here are two simple examples of propagation. The first example is propagating to a terminal called 'out1'. The second example is propagating to a label node called 'Out1Label'.

Section

Summary and references

The last portion of the presentation contains a summary and references.

Summary

- Two new ESQL exception handlers are provided
  - DECLARE EXIT HANDLER
  - DECLARE CONTINUE HANDLER
- A comprehensive set of SQLSTATEs are provided
- The LOG statement can be used to record information about message processing or the invocation of an exception handler
- Consider using PROPAGATE statement if the handler can receive errors from a downstream node

In summary, two new ESQL exception handlers are provided in V6: the EXIT HANDLER and the CONTINUE HANDLER. The handlers can be invoked within the same node as the statement causing the exception, allowing you access to local environmental data. You can use the LOG statement to record information about message processing or the invocation of an exception handler. If your handler can be invoked by an ESQL exception from a downstream node, consider using the enhanced PROPAGATE statement within your handler to redirect the exception to the appropriate node for additional processing or recovery.

References

- WebSphere Message Broker library:

http://www-306.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

References

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.