Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

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

<table>
<thead>
<tr>
<th>AIX</th>
<th>DB2</th>
<th>DB2 Universal Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSeries</td>
<td>OS/2</td>
<td>OS/390</td>
</tr>
<tr>
<td>RACF</td>
<td>SP</td>
<td>SQL/DS</td>
</tr>
<tr>
<td>System/36</td>
<td>VisualAge</td>
<td>z/OS</td>
</tr>
</tbody>
</table>

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

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

Intel is a trademark 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 in the United States and other countries.

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

---

May 2003 Edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 1996, 2003. All rights reserved.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Trademarks ........................................................................ vii

Course Description ................................................................ ix

Agenda .............................................................................. xi

Unit 1. DB2 Stored Procedures and Platform Considerations ........ 1-1
  Unit Objectives .............................................................. 1-3
  1.1 Stored Procedures .................................................. 1-5
    What Is a Stored Procedure? ....................................... 1-6
    When to Use a Stored Procedure ................................. 1-7
    How Applications Utilize Stored Procedures ............... 1-9
    Application Flow ....................................................... 1-10
    Client/Server: The Big Picture .................................... 1-11
    Server Procedures - Limitations/General ...................... 1-12
  1.2 DB2 Platform Considerations .................................. 1-15
    Platforms ................................................................ 1-16
    DB2 UDB for Linux/UNIX/Windows .............................. 1-17
    DB2 UDB for iSeries ................................................ 1-19
    DB2 UDB for OS/390 and z/OS ................................... 1-21
    DB2 for OS/390 and z/OS and z/OS Workload Manager .. 1-22
    DB2 UDB for VM/VSE ............................................. 1-24
    DB2 Catalog Storage ................................................. 1-25
    Unit Summary ........................................................... 1-28

Unit 2. DB2 Development Center ................................... 2-1
  Unit Objectives .............................................................. 2-2
  2.1 DB2 Development Center ....................................... 2-3
    Before IBM DB2 Development Center? ....................... 2-4
    DB2 Development Center ........................................... 2-5
    Launching the Development Center .............................. 2-6
    Development Center Launchpad .................................. 2-7
    Development Center Desktop ....................................... 2-8
    Development Center Definitions (1 of 2) ....................... 2-10
    Development Center Definitions (2 of 2) ....................... 2-11
    New Project .............................................................. 2-13
    Connections .............................................................. 2-14
    Defining a Stored Procedure ...................................... 2-15
    Stored Procedure Wizard ........................................... 2-16
    Working with Stored Procedures ................................. 2-18
    Development Center Demo .......................................... 2-20
    Unit Summary ........................................................... 2-22
## Unit 7. DB2 Development Center Debug

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Objectives</td>
<td>7-2</td>
</tr>
<tr>
<td>7.1 Development Center Debug</td>
<td></td>
</tr>
<tr>
<td>Platforms Supported</td>
<td>7-4</td>
</tr>
<tr>
<td>Build for Debug</td>
<td>7-5</td>
</tr>
<tr>
<td>Debug Connect</td>
<td>7-6</td>
</tr>
<tr>
<td>Debug Session</td>
<td>7-7</td>
</tr>
<tr>
<td>Debug Support</td>
<td>7-8</td>
</tr>
<tr>
<td>Troubleshooting Checklist</td>
<td>7-9</td>
</tr>
<tr>
<td>Debugging Stored Procedures</td>
<td>7-10</td>
</tr>
<tr>
<td>Unit Summary</td>
<td>7-12</td>
</tr>
</tbody>
</table>

## Appendix A. Checkpoint Solutions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A</td>
<td>A-1</td>
</tr>
</tbody>
</table>

## Bibliography

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>X-1</td>
</tr>
</tbody>
</table>
Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

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

<table>
<thead>
<tr>
<th>AIX®</th>
<th>DB2®</th>
<th>DB2 Universal Database™</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSeries™</td>
<td>OS/2®</td>
<td>OS/390®</td>
</tr>
<tr>
<td>RACF®</td>
<td>SP™</td>
<td>SQL/DS™</td>
</tr>
<tr>
<td>System/36™</td>
<td>VisualAge®</td>
<td>z/OS™</td>
</tr>
</tbody>
</table>

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

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

Intel is a trademark 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 in the United States and other countries.

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

DB2 Stored Procedures Programming Workshop

Duration: 2 days

Purpose

This course is designed to provide the database application designer or application programmer with the knowledge to recognize when an IBM stored procedure is the correct solution for an application. It also covers how to set up the environment to support stored procedures, how to use the DB2 Development Center, how to CALL a stored procedure, how to create a stored procedure using DDL, and how to troubleshoot a stored procedure.

This course will be taught using classroom lectures and machine lab exercises. Student workstations will be on a Windows/2000 platform. The database server will be a DB2 for UDB system on Windows/2000.

Students will utilize the facilities of the DB2 Development Center, create SQL/PL stored procedures, Java stored procedures, and be able to troubleshoot stored procedures using the DB2 Development Center.

Audience

Database application designers and database application programmers supporting DB2 stored procedures would all benefit from attending this course.

Prerequisites

Database application designers and database application programmers supporting DB2 stored procedures with the SQL database language. This prerequisite knowledge may be from prior experience or attendance of the following IBM Learning Services courses:

- **CF03** - DB2 Family Fundamentals
- **CF12** - DB2 SQL Workshop
Objectives

After completing this course, you should be able to:

- Describe DB2 Stored Procedures
- List platform requirements for supporting DB2 Stored Procedures
- Discuss and demonstrate capabilities of the DB2 Development Center
- Describe and use DB2 SQL Procedure Language (SQL PL) statements
- Create and execute, using the DB2 Development Center, SQL PL stored procedures
- Create and execute, using the DB2 Development Center, Java stored procedures
- Describe and use the CALL SQL statement to invoke a DB2 stored procedure
- Describe and use the CREATE PROCEDURE SQL DDL statement
- Describe and execute DB2 Development Center's debugging capabilities

Curriculum Relationship

- CF03 - DB2 Family Fundamentals
- CF10 - DB2 Universal Database Programming Fundamentals
- CF11 - DB2 Universal Database Advanced Programming
- CF12 - DB2 SQL Workshop
- CF13 - DB2 Advanced SQL Workshop
- CF14 - DB2 Call Level Interface Programming
- CF16 - DB2 for OS/390 Application Program Design
- CF82 - DB2 for OS/390 Application Programming Workshop
Agenda

Day 1

Welcome
Unit 1 - DB2 Stored Procedures and Platform Considerations
   Lab - DB2 Stored Procedures and Platform Considerations
Unit 2 - DB2 Development Center
   Lab - DB2 Development Center
Unit 3 - DB2 SQL Procedure Language (SQL PL)
   Lab - DB2 Development Center and SQL PL
Unit 4 - DB2 Development Center and Java

Day 2

Unit 5 - The CALL SQL Statement
   Lab - The CALL SQL Statement
Unit 6 - The CREATE PROCEDURE SQL Statement
   Lab - The CREATE PROCEDURE SQL Statement
Unit 7 - DB2 Development Center and Debugging
   Lab - DB2 Development Center and Debugging
Unit 1. DB2 Stored Procedures and Platform Considerations

What This Unit Is About

This unit describes the execution time environment necessary to support stored procedures. It will highlight the catalog entries that are relevant to stored procedures and address the client and server system requirements for supporting stored procedures.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Describe what a stored procedure is
- Determine when a stored procedure should be used
- Describe execution flow using stored procedures
- Describe application flow using stored procedures
- Identify correct CALL syntax
- List DB2 procedure restrictions

How You Will Check Your Progress

Accountability:

- Checkpoint
- Machine exercises

References

SG24-5485  Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826  IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827  IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844  IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845  IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932  DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
<table>
<thead>
<tr>
<th>SC26-9933</th>
<th>DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC26-9944</td>
<td>DB2 UDB for OS/390 and z/OS V7 SQL Reference</td>
</tr>
</tbody>
</table>
Unit Objectives

After completing this unit, you should be able to:

- Describe what a stored procedure is
- Determine when a stored procedure should be used
- Describe execution flow using stored procedures
- Describe application flow using stored procedures
- Identify correct CALL syntax
- List DB2 procedure restrictions

Notes:
1.1 Stored Procedures
What Is a Stored Procedure?

A Stored Procedure:

- Is an executable program under control of the DB2 server
- Is invoked via the SQL CALL statement
- May contain business logic
- May be invoked locally or remotely
- May receive or pass parameters
- May produce result sets

Notes:

A stored procedure is a compiled program, stored at a DB2 local or remote server, that can execute SQL statements. A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a host language. A client application program uses the SQL statement CALL to invoke the stored procedure.
When to Use a Stored Procedure

**Stored procedures may be justified when:**

- The application's performance expectations are not being met.
- There are a large number of clients to which application code is distributed.
- The client application generates relatively heavy database activity, but involves little user interaction.
- The client application code changes frequently.
- The access to the client application code needs to be controlled.
- The client application executes many SQL statements.

**Notes:**

Consider using stored procedures for a client/server application that does at least one of the following things:

- Executes many remote SQL statements.
  
  Remote SQL statements can create many network send and receive operations, which results in increased processor costs.

  Stored procedures can encapsulate many of your application's SQL statements into a single message to the DB2 server, reducing network traffic to a single send and receive operation for a series of SQL statements.

- Accesses host variables for which you want to guarantee security and integrity.
  
  Stored procedures remove SQL applications from the workstation, which prevents workstation users from manipulating the contents of sensitive SQL statements and host variables.

- Application code has high maintenance potential.
The overhead of maintaining a stored procedure can be substantially less than a conventional application program.
How Applications Utilize Stored Procedures

**Stored procedures are incorporated by:**

- Determining client portion of application
- Thin client?
- Selecting stored procedure server
- Developing stored procedure
- Determining calling parameters
- Selecting programming language / DB2 Development Center
- Defining SP to server
- Coding and testing

**Notes:**

This is a list of the considerations for including stored procedures in an application solution.
Notes:
A client application that does use stored procedures will see the following statement flow:

- The client application program will connect to the server, if necessary.
- The client program will issue the **CALL** SQL statement to start the process of executing the stored procedure.
- The client program will check the return code from the **CALL** statement and process any resulting rows, if required.

By placing the database requests and as much business logic as appropriate to the application in the stored procedure, the client application program is streamlined. This may result in a significant savings in execution performance.
Client/Server: The Big Picture

Notes:

1. Program initialization and housekeeping sets up host variables that may be used on stored procedure calls.
2. A connection is made to the server that will execute the stored procedure.
3. The call is made to the stored procedure passing any parameters to be used by or returned by the stored procedure.
4. End the unit of work. This may not have to be done explicitly.
5. Sever the connection to the server when all work is complete. This also may not have to be done explicitly.
Server Procedure - Limitations/General

• Stored procedures run in background.
  – Do not use the standard I/O streams

• No commands to terminate current process.

• Overloading stored procedures restricted to parameter subsets.

• SQL limited by stored procedure CREATE parameters.

Notes:

• There is not a one-to-one relationship between the calling program and the stored procedure. This means that resources that may rely on such a relationship, such as a flat file, must be handled very carefully. If such a resource is needed, then the stored procedure logic has to be able to distinguish an instance of the resource for every call to the stored procedure. There is no built-in serialization of such resources so use them with caution.

• Do not terminate the process that called the stored procedure from within the stored procedure even if allowed by the execution environment. This decision should be left to the calling program.

• More than one stored procedure can be registered with the same name as long as the number of parameters defined for each is different. For example:
  - Valid - CheckCNO (in CNO char(6)) and
    CheckCNO (in CNO char(6), in ZIP char (5))
  - Invalid - CheckCNO (in CNO char(6), in CRLIMIT dec(7,2)) and
CheckCNO (in CNO char(6), in ZIP char (5))

- The SQL executed by a stored procedure may be limited by the registration parameters of the CREATE PROCEDURE statement. This will be covered in more detail in the DDL discussion.
1.2 DB2 Platform Considerations
Platforms

**Stored Procedures supported in:**
- DB2 Universal Database for Linux, UNIX and Windows
- DB2 Universal Database for iSeries
- DB2 Universal Database for OS/390 and z/OS
- DB2 Server for VSE and VM

**Notes:**
- DB2 Universal Database Version 8
  - DB2 Universal Database for AIX
  - DB2 Universal Database for HP-UX
  - DB2 Universal Database for Linux
  - DB2 Universal Database for Sun Solaris
  - DB2 Universal Database for Windows
- DB2 Universal Database for iSeries
- DB2 Universal Database for OS/390 and z/OS
- DB2 Server for VSE and VM (replacing SQL/DS)

DB2 UDB for Linux/UNIX/Windows

Considerations:

- Do not use the standard I/O streams.
- Include only the SQL statements allowed by the CREATE PROCEDURE DDL.
- Do not use COMMIT statements if:
  - You catalog the stored procedure using the NO SQL clause.
  - The stored procedure is called from an application performing a multisite update.
- No connection-related statements or commands in stored procedures.
- On UNIX-based, NOT FENCED stored procedures run under the user ID of the DB2 Agent Process. FENCED stored procedures run under the user ID of the db2dari executable.
- No overloading of stored procedures that accept the same number of parameters.
- Do not terminate the current process.

Notes:

- Do not use the standard I/O streams, for example, calls to System.out.println() in Java, printf() in C/C++, or display in COBOL. Stored procedures run in the background, so you cannot write to the screen. However, you can write to a file.
- Include only the SQL statements allowed by the CREATE PROCEDURE statement with which you register the stored procedure. For information on using the NO SQL, READS SQL DATA, CONTAINS SQL, or MODIFIES SQL DATA clauses to catalog your stored procedure, refer to SQL Statements in Stored Procedures.
- You cannot use COMMIT statements in stored procedures when either or both of the following conditions is true: You catalog the stored procedure using the NO SQL clause, or the stored procedure is called from an application performing a multisite update.
- You cannot execute any connection-related statements or commands in stored procedures, including:
  - BACKUP
  - CONNECT
- CONNECT TO
- CONNECT RESET
- CREATE DATABASE
- DROP DATABASE
- FORWARD RECOVERY
- RESTORE

- On UNIX-based systems, NOT FENCED stored procedures run under the user ID of the DB2 Agent Process. FENCED stored procedures run under the user ID of the db2dari executable, which is set to the owner of the .fenced file in sqllib/adm. This user ID controls the system resources available to the stored procedure.

- You cannot overload stored procedures that accept the same number of parameters, even if the parameters are of different SQL data types.

- Stored procedures cannot contain commands that would terminate the current process. A stored procedure should always return control to the client without terminating the current process.
Considerations:

- DB2 SQL for iSeries stored procedure support provides a way for an SQL application to define and then invoke a procedure through SQL statements.
- You may define a procedure as either an SQL procedure or an external procedure.
- Coding stored procedures requires understanding the client/server needs.
- The CREATE PROCEDURE SQL statement registers the stored procedure.

Notes:

- DB2 SQL for iSeries stored procedure support provides a way for an SQL application to define and then invoke a procedure through SQL statements. Stored procedures can be used in both distributed and non-distributed DB2 SQL for iSeries applications. One of the big advantages in using stored procedures is that for distributed applications, the execution of one CALL statement on the application requester, or client, can perform any amount of work on the application server.
- You may define a procedure as either an SQL procedure or an external procedure. An external procedure can be any supported high-level language program (except System/36 programs and procedures) or a REXX procedure. The procedure does not need to contain SQL statements, but it may contain SQL statements. An SQL procedure is defined entirely in SQL, and can contain SQL statements that include SQL control statements.
- Coding stored procedures requires that the user understand the following:
  - Stored procedure definition through the CREATE PROCEDURE statement
- Stored procedure invocation through the CALL statement
- Parameter passing conventions
- Methods for returning a completion status to the program
- Invoking the procedure

• You may define stored procedures by using the CREATE PROCEDURE statement. The CREATE PROCEDURE statement adds procedure and parameter definitions to the catalog tables, SYSPROCS, and SYSPARMS. These definitions are then accessible by any SQL CALL statement on the system.
DB2 UDB for OS/390 and z/OS

Considerations:

- Do not include explicit attachment facility calls in a stored procedure.
- Do not include the SET CURRENT SQLID statement in your stored procedure.

Notes:

Restrictions on a stored procedure:

- Do not include explicit attachment facility calls in a stored procedure. Stored procedures running in a DB2-established address space use call attachment facility (CAF) calls implicitly. Stored procedures running in a WLM-established address space use Recoverable Resource Manager Services attachment facility (RRSAF) calls implicitly. If a stored procedure makes an explicit attachment facility call, DB2 rejects the call.
- Do not include the SET CURRENT SQLID statement in your stored procedure.

When DB2 encounters this statement, it places the DB2 thread in a must roll back state. When control returns to the calling program, the calling program must do one of the following things:

- Execute the ROLLBACK statement, so that it is free to execute other SQL statements after rollback is complete.
- Terminate, causing an automatic roll back of the unit of work.
DB2 for OS/390 and z/OS and z/OS Workload Manager

- Multiple stored procedure address spaces
- Access non-DB2 resources with two-phase commit
- More flexible prioritization of execution
- Workload balancing
- Flexible grouping or isolation
- RACF check based on client authid

Notes:

- DB2 for OS/390 and z/OS using the z/OS Workload Manager facility allows for the creation of multiple address spaces for DB2 stored procedures. Without z/OS Workload Manager, all stored procedures execute in a single z/OS address space. This limits flexibility in scheduling, prioritization, and testing.
- Non-DB2 resources can be registered with z/OS and participate in two-phase commit processing with DB2 for OS/390 and z/OS.
- Stored procedures could be categorized into low, medium, and high priority (for example) and isolated into different address spaces when using z/OS Workload Manager.
- Workload is balanced by the operating system in z/OS. By having stored procedures executing in more than one address space maximum resource utilization can be achieved.
• An example of isolation using z/OS Workload Manager is setting one WLM environment for critical applications. Another may be set up for debugging so that the failing stored procedure does not interfere with others.

• When a DB2 stored procedure executes in a WLM environment, RACF checking can be done at the primary authid level of the calling program. Otherwise, an ID assigned to the address space must be used by RACF for authorization checking.
DB2 Server for VM/VSE

**Considerations:**

- An SQL CALL statement is used to invoke a stored procedure.
- The SQL CALL statement is sent to the application server.
- The stored procedure server acts as an application requester.
- In VM, the stored procedure is a separate virtual machine.
- In VSE, the stored procedure is a separate static or dynamic partition.
- Stored procedure support on DB2 Server for VSE and VM is modeled after the DB2 for OS/390 and z/OS implementation as much as possible.

**Notes:**

- An SQL CALL statement is used to invoke a stored procedure. The SQL CALL statement is sent to the application server, which sends a request to the stored procedure server, causing it to invoke the stored procedure.
- The stored procedure server acts as an application requester, sending the SQL statements in the stored procedure to the application server.
- When the stored procedure completes, it returns to the database manager, which passes the results back to the original application requester.

The stored procedure server is the key to achieving a fenced implementation. A stored procedure that is fenced is separated from the database in terms of execution and memory usage. In VM, the stored procedure is a separate virtual machine, and in VSE, it is a separate static or dynamic partition.

- Stored procedure support on DB2 Server for VSE and VM is modeled after the DB2 for OS/390 and z/OS implementation as much as possible.
Notes:

There were some changes made to the DB2 V8 catalog that apply to stored procedures. Now the term Routine applies to stored procedures, functions and methods. Catalog views for these objects have been merged.

SYSCAT.FUNCTIONS and SYSCAT.PROCEDURES have been replaced by SYSCAT.ROUTINES.

SYSCAT.FUNCSELS and SYSCAT.PROCSELS have been replaced by SYSCAT.ROUTINESELS.

SYSCAT.FUNCDEP has been replaced by SYSCAT.ROUTINEDEP.

SYSCAT.ROUTINEAUTH contains one or more rows for each user or group who is granted EXECUTE privilege on a particular routine in the database.
Checkpoint

Exercise — Unit 1 Checkpoint

__ 1. A stored procedure is an executable program under control of
__________________________________________________________________________________________________________________________________________

__ 2. List three cases when a stored procedure may be justified.
__________________________________________________________________________________________________________________________________________
__________________________________________________________________________________________________________________________________________
__________________________________________________________________________________________________________________________________________
__________________________________________________________________________________________________________________________________________
__________________________________________________________________________________________________________________________________________

__ 3. The same procedure name appears to be registered more than once with the DB2 database manager. What does this imply about the parameters?
__________________________________________________________________________________________________________________________________________
Unit Summary

Since completing this unit, you should be able to:

- Describe what a stored procedure is
- Determine when a stored procedure should be used
- Describe execution flow using stored procedures
- Describe application flow using stored procedures
- Identify correct CALL syntax
- List DB2 procedure restrictions

Notes:
Unit 2. DB2 Development Center

What This Unit Is About

This unit describes the application program that calls a stored procedure.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Describe the DB2 Development Center
- Identify the capabilities of the DB2 Development Center
- Create and execute simple stored procedures using the Development Center

How You Will Check Your Progress

Accountability:

- Checkpoint
- Machine exercises

References

SG24-5485  Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826  IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827  IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844  IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845  IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932  DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933  DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
SC26-9944  DB2 UDB for OS/390 and z/OS V7 SQL Reference
Unit Objectives

After completing this unit, you should be able to:

- Describe the Development Center
- Identify the capabilities of the Development Center
- Create and execute simple stored procedures using the Development Center

Notes:
2.1 DB2 Development Center
Before IBM DB2 Development Center?

- Using the CREATE PROCEDURE DDL statement, register the stored procedure.
- Write, precompile, bind, compile, and link-edit the stored procedure.
- Write DEBUG application program.
- If Java, use JDBC calls instead of static SQL.
- Put stored procedure into production.

Notes:

‘Before IBM DB2 Development Center’ implies prior to the Development Center’s predecessor, DB2 Stored Procedure Builder, as well.

- The application developer or database administrator must register the new stored procedure with the database manager.
- As with other DB2 programs, other than calls to the database drivers, the program’s SQL must be processed by the DB2 precompiler. The output from that process would be used by conventional program compilers and the BIND process. A successful compile would be followed by a link-edit to create an executable module.
- Since the stored procedure would need to be tested, the calling application or a program that uses the same calling conventions must be written to call the new stored procedure.
- Execute the test program that calls the stored procedure.
- If all goes well, put the new procedure into production.
DB2 Development Center

Use the DB2 Development Center to:

- Create stored procedures
- Build on local and remote DB2 servers
- Modify and rebuild existing stored procedures
- Run stored procedures for testing and debugging the execution of installed stored procedures

Notes:

The DB2 Development Center allows application developers to build and maintain stored procedures with an easy-to-follow graphical user interface. It will provide a mechanism to organize these stored procedures into project folders for ease of maintenance.

The DB2 Development Center will aid the application developer in:

- Creating new stored procedures
- Creating and executing stored procedures on local or remote servers
- Modifying and rebuilding existing stored procedures
- Debugging and testing stored procedures

Note: The DB2 Development Center can also be used to develop and maintain User-Defined Functions (UDFs). However, this course focuses only on the stored procedure capabilities of the DB2 Development Center.
Launching the Development Center

Launching the Development Center

- IBM DB2 UDB Program Group
- A Universal Database center
- Microsoft Visual Studio application development environments
  - Visual Basic
  - Visual C++
  - Visual InterDev

Notes:

These are the current platforms from which an application developer may launch the DB2 Development Center.
Development Center Launchpad

Notes:

The DB2 Development Center has a launchpad that can lead you through the steps necessary to create a stored procedure. The actual steps will be looked at in greater detail throughout this unit.

By default, the launchpad will be displayed when the Development Center is started. You can select the Do not show this again option in order to not have the launchpad displayed. The launchpad can also be invoked from the Project menu option on the DB2 Development Center main window or desktop.
Development Center Desktop

Notes:
From this main window or desktop, you will be invoking all of the processes to create, modify, and run stored procedures.

- The object tree area displays your project at the highest level, all databases for the project, and all procedures for the databases.

- The detail contents pane contains different information based on which object in the object tree area is selected.

- The output view is divided into a status area and a message area and displays the following:
  - The status area displays the status of some action that was performed on a particular object.
  - After running the stored procedure, the Result tab will contain output from the stored procedure.
  - The Messages tab will contain progress messages from DB2 Development Center and any error messages that might have been issued.
- After running the stored procedure, the Parameters tab will contain a list of the stored procedure’s parameter and the values of any input parameters.
Notes:

From the Development Center desktop, you can manually invoke the same steps that are presented when using the Launchpad.

Right-clicking the Projects folder displays the menu used to create a new project.

Right-clicking an actual project displays a menu that allows you:

- To save the project
- To save the project under a different name
- To remove the project
- To change the properties of the project
Notes:

After having defined a project, connections to one or more databases can be defined. Right-clicking the Connections folder displays a menu to add a new database connection. The actual steps to do this will be shown later.

Right-clicking the Stored Procedure folder under a database allows you to work with stored procedures that will access that database. You can:

- Create a new stored procedure. This can either be an SQL stored procedure that is created using the editor or can be an SQL or Java stored procedure that is created using the wizard.
- Import an existing stored procedure from either a database, from a source file or another Development Center project.
- Export a stored procedure. You will be given the choice to export as a project, which means that you can use the Deployment Wizard to deploy the stored procedure. Or you can create an export script, which allows you to deploy the stored procedure from a command line.
• The Deployment Wizard allows you to deploy stored procedures to a production target database.
• Remove all stored procedures associated with the parent database.
Notes:

When invoking DB2 Development Center for the first time, you will set up a project. Subsequently, when you invoke the DB2 Development Center, you will see a window that will ask if you want to work with an existing project or create a new one. You can also access recent projects from the Projects menu item on the Development Center desktop. The information needed to create a new project is as follows:

- Project name — Type a name for the new project. The project name can be up to 255 characters, including the project path and filename extension.
- Project path — Enter a path for the subdirectory to contain the project.
Connections

There are several steps in defining a connection. First, you will define the connection type as being either online or offline.

Next, you define the connection.

- **Driver** — Choose the driver, a program (and possibly data files) that contains information needed to run a particular unit, for this project.

- **Database Alias** — Enter the alias name for a database used by this project.

- If the connection type is offline, then you will choose the type of DB2 and its version.

- If the connection type is online, then you will need to specify a userid and password. You have two options. You can specify a particular userid and its password or you can indicated that you want to use your current userid and password, that is the userid and password that you used to log onto your Windows workstation.

- You can test your online connection as well.

Notes:

This is an example of an online connection type
Defining a Stored Procedure

![DB2 Development Center](image)

**Notes:**

If you specify on the Development Center desktop that you want to create a stored procedure using the wizard, then the New Object dialog is displayed with a subtle difference. The left hand pane would only present a choice of stored procedure.

As stated before, the focus of this class is not User-Defined Functions. But it should be noted that if you choose to define a UDF from the Development Center desktop using the wizard, then the New Object dialog is displayed but only presenting a choice of User-Defined Function.

The third way to display the New Object dialog is via the Launchpad. When invoked this way, you have not yet specified if you are creating a stored procedure or a user-defined function. So you are presented with a choice of both.

Having made your choice on the type of object, you then choose the desired language.
Notes:

When you choose to create a stored procedure using the wizard, then you are led through this process, step by step.

First, you will specify the name of the stored procedure.

Next is the Definition stage.

- Here you can add code fragments to your stored procedure. There are different types of fragments, like header, variable and error. These fragments could be pieces of code that is common to several stored procedures.

- You can specify how to treat errors. Should you throw an exception or use return codes.

- How many results sets are to be returned, none, one or many.

- The statement setting will invoke an SQL Assistant. You specify how many SQL statements, none, one or multiple and then you will be led through the process of creating the SQL statements.
The third step is to define your parameters. You will need to define each parameter and specify if it is an input, output or inout parameter.

The fourth step is to define additional options. This window will differ depending on whether you are creating an SQL or a Java stored procedure. You can specify:

- The specific name of the stored procedure
- Whether to build the stored procedure and if it should be enabled for debugging.
- If it is a Java stored procedure, you will also specify:
  - The Jar ID
  - The Java class name
  - And whether you want dynamic SQL using JDBC or static SQL using SQLJ
Notes:

Once a stored procedure has been created, you can right-click it in order to do the following:

- **Edit** will bring up the editor so that you can manually modify the code.
- **Build** the stored procedure, with or without debug capabilities.
- **Run** the stored procedure.
- **Specify Run Settings**. This allows you to execute SQL statements before and/or after the running of the stored procedure.
- **Invoke the debugger**.
- **Deploy** the stored procedure to some production database.
- **Copy** the stored procedure so that it can later be pasted elsewhere.
- **Remove** from the project and drop from the database.
- **Show the Create Procedure SQL** that was used to define the stored procedure.
• Print the stored procedure.
• Modify the properties. For an SQL stored procedure, this only means that the comment field can be changed. For a Java stored procedure, the name, comment, parameters and a variety of other options can be changed.
Notes:

A short demonstration of the DB2 Development Center will prepare you for your first lab exercise dealing with the DB2 Development Center. You will see how the icon selections discussed preceding the demonstration relate to the creation and management of stored procedures.
Checkpoint

Exercise — Unit 2 Checkpoint

__ 1. List three DB2 Development Center functions that would aid application developers.

________________________________________________________________
________________________________________________________________
________________________________________________________________

__ 2. List three ways of launching the DB2 Development Center.

________________________________________________________________
________________________________________________________________
________________________________________________________________
________________________________________________________________

__ 3. IBM DB2 Development Center may be used to create stored procedures in which two languages?
Unit Summary

Since completing this unit, you should be able to:

- Describe the Development Center
- Identify the capabilities of the Development Center
- Create and execute simple stored procedures using the Development Center

Notes:
Unit 3. DB2 SQL Procedure Language

What This Unit Is About

This unit describes the DB2 SQL Procedure Language statements and how to use them to control SQL statement execution within a DB2 stored procedure.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Describe the DB2 SQL Procedure Language
• Identify valid SQL statements for a procedure
• Code process control statements for a procedure

How You Will Check Your Progress

Accountability:

• Checkpoint
• Machine exercises

References

SG24-5485 Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826 IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827 IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844 IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845 IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932 DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933 DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
SC26-9944 DB2 UDB for OS/390 and z/OS V7 SQL Reference
Unit Objectives

After completing this unit, you should be able to:

- Describe the DB2 SQL Procedure Language
- Identify valid SQL statements for a procedure
- Code process control statements for a procedure
3.1 DB2 SQL Procedure Language
SQL Stored Procedures

- Based on ANSI/ISO standard language SQL/PSM
- Simple language which includes:
  - Features from block-structured languages
  - Exception handling
  - Familiar to Sybase, Oracle, Informix, Microsoft SQL Server programmers

Notes:
- SQL/PSM, persistent stored modules, are incorporated into the ANSI/ISO standard to allow developers to provide solutions for their application requirements that can be reused.
- The language constructs will be familiar to programmers in conventional languages.
- The syntax allows for conditional processing, error handling, and setting of variables within the stored procedure.
- The IBM DB2 SQL Procedures language statements will allow easy migration from other database management systems.
SQL Procedure Language (1 of 3)

- **SQL Procedures support:**
  - Multiple parameters: input, output, input/output
  - Returning multiple output result sets to client

- **SQL Procedures are defined in DB2 catalog.**

- **SQL Procedure source is stored in DB2 catalog.**

- **SQL Procedure language is folded to upper case.**
  - *Exception: delimited values* 

**Notes:**

- DB2 stored procedures are implemented to handle the passing of data from a client application to the stored procedure, the passing of data from the stored procedure back to the client application, and the passing of data used as input to the stored procedure and as output from the stored procedure.

- Procedures written in SQL Procedure language are registered in the database manager that executes the stored procedure.

- The SQL Procedure Language source statements are stored in the DB2 catalog and can be selected by an authorized user from the catalog.
Figure 3-4. SQL Procedure Language (2 of 2)

```
CREATE PROCEDURE DB2ADMIN.Sample1 ( IN in_Dept INT )
   RESULT SETS 1
   LANGUAGE SQL
   -----------------------------------------------
   -- SQL Stored Procedure
   -----------------------------------------------
P1: BEGIN
   DECLARE r_error int default 0;
   DECLARE SQLCODE int default 0;
   DECLARE CONTINUE HANDLER FOR SQLWARNING, SQLEXCEPTION, NOT FOUND
   BEGIN
      SET r_error = SQLCODE;
   END;
   BEGIN
   DECLARE cursor1 CURSOR WITH RETURN FOR
      SELECT DEPTNAME, MANAGER, LOCATION
      FROM ORG
      WHERE
         DEPTNUMB = in_Dept;
   -- Cursor left open for client application
   OPEN cursor1;
   END;
   END P1
```

Notes:

This listing shows an SQL Procedure Language stored procedure. As of V7 for DB2 UDB for OS/390, nested compound statements are not supported. Remember that the SQL Procedure Language statements are part of the DDL for the CREATE PROCEDURE statement.

The procedure listed above has only one parameter which is defined within parentheses following the procedure name. This procedure is producing a result set for the calling program, as indicated by the RESULT SETS setting.

The label P1 starts the block of SQL Procedure Language code for this procedure. The BEGIN / END pairs signal the start and end of SQL Procedure Language blocks of code.

All DECLARE statements in the listing are needed for one of the following reasons:

- Local variables
- Condition processing
- The result set cursor
SQL Procedure Language (3 of 3)

• An SQL Procedure consists of:
  — A CREATE PROCEDURE statement
    • LANGUAGE = SQL
  — A procedure body which may include:
    • Compound statement(s): BEGIN ... END
    • Declaration statements
    • Assignment statements
    • Conditional statements
    • Iterative control structure: LOOPS, and so forth
    • Exception Handling
    • CALL another stored procedure

Notes:
• The LANGUAGE parameter of the CREATE statement has to be set to SQL for you to use SQL Procedure Language statements.
• The procedure itself will consist of the elements listed above.
• Within the limit of the SQL constructs allowed, you may put much of the application's business logic within the SQL Procedure Language procedure. This will allow you to create a much thinner client for your application solution.
Notes:

This is the infamous foo stored procedure. Keep in mind that the entire procedure code is part of the DDL statement - CREATE PROCEDURE.

The first part of the definition is the CREATE PROCEDURE clause that identifies the procedure by name, parameter definitions, and other parameters that apply to the stored procedure itself. This includes the LANGUAGE parameter which must be set to SQL.

All DECLARE statements must be found prior to the variables being used within the procedure.

Application logic and output parameter settings follow any DECLARE statements.

DB2 UDB for OS/390 and z/OS only allows one BEGIN/END block per stored procedure.
Structure (2 of 2)

• An SQL Procedure can be:
  
  — A single statement
    CREATE PROCEDURE Sample1 (OUT Parm1 CHAR(10))
    LANGUAGE SQL
    SET Parm1 = 'value1'

  — A compound statement
    CREATE PROCEDURE Sample2 (OUT Parm1 CHAR(10),
                                OUT Parm2 CHAR(10))
    LANGUAGE SQL
    BEGIN
      SET Parm1 = 'value1' ;
      SET Parm2 = 'value2';
    END

  — Or nested compound statements

Notes:

An SQL Procedure Language stored procedure may consist of:

• A single statement — Used infrequently, but it would allow you to do things like return special register information from the database server to a client.

• Compound statements — Used if you have more than one SQL Procedure Language statement required for your application. This is the more usual case and most SQL Procedure Language procedures will consist of at least one compound statement.

• Nested compound statements — Used if your SQL Procedure Language construct would normally allow only a single statement. By using another **BEGIN / END** block, you will overcome such restrictions.

However, DB2 UDB for OS/390 and z/OS only allow one BEGIN/END block per stored procedure.
SQL Procedure Language Statements

- Not limited to stored procedures
- Some platform differences
- Facilitate application solution
- Add business logic capability to SQL language

Notes:

- SQL Procedure Language statements may be valid in stored procedures, user-defined functions (UDFs), and triggers.
- Refer to *Cross-Platform DB2 Stored Procedures: Building and Debugging* for platform portability. Table 3 in Chapter 3 covers valid SQL Procedure Language statements for stored procedures.
- SQL Procedure Language allows for a business solution that does not require conventional programming if the SQL Procedure Language statements can satisfy the requirements of the application logic.
- The SQL language historically has provided only database access and some control over the database. With SQL Procedure Language, we now have a way of processing our data and including business logic without having to design and write conventional programs. The SQL Procedure Language is known to the database manager and the triggers, UDFs, and stored procedures are managed by the database manager along with the data.
Compound Statements

- Can have labels: \texttt{P1: BEGIN \ldots END P1}
- Can be atomic: \texttt{BEGIN ATOMIC}
- Can contain:
  - Declarations
  - Procedural statements
  - Other compound statements

Notes:

- Although not required, labels may be provided for the block containing multiple SQL Procedure Language statements. This is useful for recognition of blocks when the procedure consists of more than one compound statement.
- \texttt{BEGIN ATOMIC} — Indicates that if an error occurs in the compound statement, all SQL statements in the compound statement will be rolled back.
  
  \texttt{BEGIN ATOMIC} is not valid in DB2 UDB for OS/390 and z/OS.
- \texttt{BEGIN NOT ATOMIC} — Indicates that an error within the compound statement does not cause the compound statement to be rolled back.
- The compound statement may contain \texttt{DECLARE} statements, SQL Procedure Language, or other compound statements.
Example - Nested Compound Statements

```
CREATE PROCEDURE ADMIN.Proc1 (out p1_a integer, out p2_a integer)
    LANGUAGE SQL
P1: BEGIN
    declare a integer default 5;
    declare c1 cursor with return for select * from staff;
    P2: BEGIN
        declare a integer default 7;
        declare c1 cursor with return for select * from department;
        open c1;
        set p2_a = a;
        END P2;
        open c1;
        set p1_a = a;
    END P1
```

**Notes:**

The procedure above consists of two compound statements:

- **P1** — Works with cursor c1 and variables needed for working with cursor c1. Since this is the main statement for the procedure, there is no ; on the END.

- **P2** — Works with cursor c2 and variables needed for working with cursor c2. This is a nested compound statement within P1 and therefore needs the ending ;.

DB2 UDB for OS/390 and z/OS allow only one BEGIN/END block.
Compound Statements

- **Order of statements in compound statement**
  - SQL variables and condition declarations
  - Cursor declarations
  - Handler declarations
  - Procedure body statements

- **Terminating statements with ;**
  - Procedure body has no terminating character
  - Statement nested within other statements ends with ;

**Notes:**

These rules should be followed when constructing your SQL Procedure Language stored procedure. If you have multiple procedure blocks, follow the guidelines within each block.

Note that only the block associated with the main SQL Procedure Language compound statement has no ending semicolon (;). All others are considered normal SQL statements within the stored procedure and must end with a semicolon.
Example - ";"

```sql
CREATE PROCEDURE foo
  ( out day_of_Year int )

LANGUAGE SQL

-- SQL Stored Procedure

P1: BEGIN

  DECLARE c_Date DATE;
  SET c_Date = CURRENT_DATE;
  SET day_of_Year = dayofyear(c_Date);

END P1
```

**Notes:**

This example shows our very simple `foo` stored procedure. Note that the `;` terminates the `DECLARE` and `SET` SQL statements, but that the `END P1` does not need one because there is only one compound statement in this procedure.
Declarations (1 of 2)

• Local variables:

```sql
DECLARE var_name datatype [ DEFAULT value];
Ex: DECLARE my_var INTEGER DEFAULT 6;
```

– Default value is NULL
– Variable name is folded to upper case
– Rules for ambiguous names
  • Check if column name (table exists)
  • Check if SQL variable / parameter name
  • Assume to be a column name
  • **Note:** OS/390 and z/OS will check if variable / parameter name
    else assume a column
    ▶ Qualify with table name to force

**Notes:**

Local variables are those used by the SQL Procedure Language stored procedure. These variables may be used to save parameters that may be changed during the procedures execution or may be used for calculations performed by the stored procedure. The datatype specified will be one of the supported data types used in the column definitions of a DB2 table. A good programming technique would be to provide a name for stored procedure variables that would not be ambiguous to the database manager. For example, you might start all your local variable names with a `v_` if that name would not be found elsewhere for table or column names.
Declarations (2 of 2)

- **Condition declaration:**
  
  ```sql
  DECLARE not_found CONDITION FOR SQLSTATE '02000';
  ```

- **Local cursor declaration:**
  
  ```sql
  DECLARE c1 CURSOR FOR select * from staff;
  WITH RETURN TO CLIENT / WITH RETURN TO CALLER
  ```

- **Handler declaration:**
  
  ```sql
  DECLARE EXIT HANDLER FOR SQLEXCEPTION...;
  ```

**Notes:**

- The condition declaration is used in order to allow your SQL Procedure Language code to check for this condition in the logic portion of the stored procedure. With the example above, the stored procedure could contain an `IF` construct that tested for the `not_found` condition declared.

- A cursor declared within the stored procedure can be processed just like a cursor in a conventional program.
  - WITH RETURN — This clause indicates that the cursor declared will produce a result set.
  - TO CALLER (DB2 UDB only)— Specifies that the cursor can return a result set to the caller. For example, if the caller is another stored procedure, the result set is returned to that stored procedure. If the caller is a client application, the result set is returned to the client application.
  - TO CLIENT (DB2 UDB only) — Specifies that the cursor can return a result set to the client application. This cursor is invisible to any intermediate nested procedures.
• The handler declaration allows for execution of an SQL statement or a statement block for the declared condition.
Example "Cursors"

CREATE PROCEDURE Cur_Samp

  (IN v_name VARCHAR(254), OUT v_job VARCHAR(5))

LANGUAGE SQL

P1: BEGIN

  DECLARE c1 CURSOR FOR

       SELECT JOB FROM STAFF WHERE NAME = v_name;

  OPEN c1;

  FETCH c1 INTO v_job;

END P1

Notes:

This is an example of a stored procedure that DECLAREs a local cursor. Since it does not specify **WITH RETURN**, the cursor will be closed by the database manager when the stored procedure completes. Without any additional code in this stored procedure, it would not be very useful. Maybe you're just trying to do some syntax checking before you add the rest of the code.
Assignments

- **Syntax:**
  
  ```sql
  SET lv_name = expression;
  SET lv_name = NULL;
  ```

- **Example:**
  
  ```sql
  SET salary = salary + salary*0.1;
  SET init_salary = NULL;
  SET salary = (select salary from employee
  where empno = lv_emp_num);
  
  NOTE: SQLERROR if more than one row
  ```

**Notes:**

The **SET** statement is used to assign values to variables in an SQL Procedure Language stored procedure. The assignment can be a literal, a valid DB2 keyword, or an expression including an SQL statement such as the SELECT statement above.

DB2 UDB for OS/390 and z/OS does not allow the result of a SELECT statement to be assigned to a variable.
Control Flow Statements (1 of 2)

- **CASE statement**
  - Select execution path based on multiple conditions

- **IF statement**
  - Select execution path based on evaluation of conditions

- **LOOP statement**
  - Execute statements multiple times

- **REPEAT statement**
  - Execute statements until condition is true

**Notes:**

The control flow statements on this visual and the next allow the stored procedure developer to alter the logical processing path with the SQL Procedure Language blocks of code. These flow statements are similar to process control statements in conventional programming languages such as C++ or Java.
Control Flow Statements (2 of 2)

- **WHILE statement**
  - Execute statements while condition is true
- **FOR statement**
  - Execute statements for each row of a table
- **ITERATE**
  - Transfers flow on control to labeled block or loop
- **LEAVE statement**
  - Transfer control out of loop or block for FOR, LOOP, REPEAT or WHILE

**Notes:**

The FOR statement is not valid in DB2 UDB for OS/390 and z/OS.
The ITERATE statement is only valid in DB2 UDB Linux, UNIX and Windows.
The next few visuals will show examples of the control flow statements just shown with the exception of the **ITERATE**. This is how the **ITERATE** statement might be used by a developer:

```
ins_loop:
  LOOP
    FETCH c1 INTO v_dept, v_deptname, v_admdept;
    IF at_end = 1 THEN
      LEAVE ins_loop;
    ELSEIF v_dept = 'D11' THEN
      ITERATE ins_loop;
    END IF;
    INSERT INTO department (deptno, deptname, admrdept)
    VALUES ('NEW', v_deptname, v_admdept);
  END LOOP;
```
With the check for department, the **ITERATE** statement above would allow for ignoring all rows FETCHed that had a department value of **D11** by returning to the loop label.
Conditional Statements

- **Syntax:**

  ```sql
  IF cond1 THEN statement;
  ELSEIF cond2 THEN statement;
  ELSE statement;
  END IF;
  ```

- **Example:**

  ```sql
  IF rating = 1 THEN
      UPDATE EMPLOYEE SET salary = salary*1.10 WHERE empno = i_num;
  ELSEIF rating = 2 THEN
      UPDATE EMPLOYEE SET salary = salary*1.05 WHERE empno = i_num;
  ELSE
      UPDATE EMPLOYEE SET salary = salary*1.03 WHERE empno = i_num;
  END IF;
  ```

**Notes:**

The **IF** construct allows for conditional processing of statements within a block in the stored procedure. The condition checking can be more than one level by using the **ELSEIF** clause. You can also extend the capability by using **BEGIN / END** blocks within the **IF** statement unless your database server is DB2 UDB for OS/390 and z/OS since only one BEGIN/END block is allowed.
CASE Statement (1 of 2)

- Simple CASE statement:

```sql
CREATE PROCEDURE foo ( IN v_workdept CHAR(3))
LANGUAGE SQL
P1: BEGIN
    CASE v_workdept
        WHEN 'A00'
            THEN UPDATE department
                SET deptname = 'DATA ACCESS 1';
        WHEN 'B01'
            THEN UPDATE department
                SET deptname = 'DATA ACCESS 2';
        ELSE UPDATE department
            SET deptname = 'DATA ACCESS 3';
    END CASE
END P1
```

Notes:
The simple CASE statement allows the developer to use a variable as an argument and test that variable within the CASE construct for as many possible values as the developer decides to code for. The ELSE clause would allow for statements to be coded for any conditions that are not met. This construct is relatively easy to follow for anyone having to maintain the code in the stored procedure.
CASE Statement (2 of 2)

- Searched CASE statement:

```sql
CREATE PROCEDURE foo ( IN v_workdept CHAR(3))
LANGUAGE SQL
P1: BEGIN
  CASE
    WHEN v_workdept = 'A00'
      THEN UPDATE department
          SET deptname = 'DATA ACCESS 1';
    WHEN v_workdept = 'B01'
      THEN UPDATE department
          SET deptname = 'DATA ACCESS 2';
    ELSE UPDATE department
          SET deptname = 'DATA ACCESS 3';
  END CASE;
END P1
```

Figure 3-21. CASE Statement (2 of 2)

Notes:

This variation of the `CASE` statement differs very little from the simple `CASE` statement. The argument to the `CASE` statement is checked on every `WHEN` clause. This would give the developer more flexibility in doing the checking, if needed.
LOOP Statement

- **Syntax:**
  
  \[
  \text{[label]} \text{ LOOP} \\
  \text{ SQL-procedure-statement(s);} \\
  \text{ END LOOP [label]}
  \]

- **Example:**
  
  ```sql
  fetch_loop:
  LOOP
    FETCH c1 INTO v_firstname, v_lastname;
    SET counter = counter + 1;
    IF counter = 51 THEN
      LEAVE fetch_loop;
    END IF;
  END LOOP fetch_loop;
  ```

**Notes:**

The **LOOP** statement allows the developer to code statements that will be re-executed within the stored procedure until a condition is met that causes an exit from the **LOOP**. This capability is found in conventional programming languages and should be very familiar to the developer of a stored procedure. The **LEAVE** statement is used to exit from the **LOOP** processing, so it is very important that the developer have a way of testing some condition that will always be met in order to terminate **LOOP** processing. Not being careful about this check will result in code that will continue to execute endlessly. This could make some people suspect your programming skills.
FOR Statement

- **Syntax:**

  
  [label] FOR for-loop-name AS [cursor-name CURSOR FOR]
  
  select-statement DO
  
  SQL-procedure-statement(s);
  
  END FOR [label]

- **Example:**

  DECLARE fullname CHAR(40);
  FOR v1 AS c1 CURSOR FOR
    select firstname, midinit, lastname FROM employee
  DO
    SET fullname = lastname || ‘,’ || firstname || ‘,’ midinit;
    INSERT INTO tname VALUE (fullname)
  END FOR;

**Notes:**

- The **FOR** statement executes a statement or group of statements for each row of a table.
- If the beginning label is specified, that label can be used in LEAVE and ITERATE statements. If the ending label is specified, it must be the same as the beginning label.
- The for-loop-name is used to qualify column names returned by the SELECT statement in the **FOR** block.
- The select list must consist of unique column names and the table specified in the select list must exist when the procedure is created, or it must be a table created in a previous SQL procedure statement.
- The cursor specified in a **FOR** statement cannot be referenced outside the for-statement and cannot be specified in an OPEN, FETCH, or CLOSE statement.

DB2 UDB for OS/390 and z/OS does not support the FOR statement.
Other Control Flow Statements

- **REPEAT Statement**

  ftch_loop2:
  ```sql
  REPEAT
    FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
  UNTIL SQLCODE <> 0 END REPEAT ftch_loop2;
  ```

- **WHILE Statement**

  ```sql
  WHILE at_end = 0 DO
    FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
    IF SQLCODE = 100 THEN SET at_end = 1;
    END IF;
  END WHILE;
  ```

**Notes:**

The **REPEAT** statement and the **WHILE** statement are very similar in function. When the test for the condition being checked is different for the two statements.

- Testing for termination of execution for the **REPEAT** statement is done at the end of the statement with the **UNTIL** clause. If the condition is met the **REPEAT** statement execution terminates, otherwise, the **REPEAT** statement is reexecuted. This means that the code within the **REPEAT** block will execute at least once.

- Testing for termination of execution for the **WHILE** statement is done at the start of the statement. If the condition is met the statement executes, otherwise, processing continues with the statement after the **WHILE** statement.
Error Handling

- **SQL procedure terminates if an SQL error occurs unless a handler is declared**
  - Warning: data truncation on a set is an SQL error

- **SQLSTATE and SQLCODE**
  - Access requires explicit declaration, for example:
    - `DECLARE SQLSTATE CHAR(5) DEFAULT '00000';`
    - `DECLARE SQLCODE INTEGER DEFAULT 0;`

**Notes:**

Since there is no SQLCA equivalent for the SQL Procedure Language stored procedure, use of SQLSTATE or SQLCODE within the SQL Procedure Language stored procedure requires that these variables must be explicitly defined in the stored procedure. The visual above shows the correct `DECLARE` statement for each.

Data truncation is treated as an error in an SQL procedure, so not having a declare handler for that condition would result in the SQL procedure abnormally terminating.
Condition Handlers

- **Condition Handling**
  - Compound statement contains any number of handlers
  - A condition handler must specify:
    - A set of conditions it is prepared to handle
    - Action to perform to handle the condition
    - Where to resume the execution
      - EXIT, CONTINUE, UNDO
  - Action can be any SQL statement or compound statement

**Notes:**

There are three types of condition handlers:

- **CONTINUE** — After the handler is invoked successfully, control is returned to the SQL statement that follows the statement that raised the exception.

- **EXIT** — After the handler is invoked successfully, control is returned to the end of the compound statement that declared the handler.

- **UNDO** — Before the handler is invoked, any SQL changes that were made in the compound statement are rolled back. After the handler is invoked successfully, control is returned to the end of the compound statement that declared the handler. If UNDO is specified, then ATOMIC must be specified.

DB2 UDB for OS/390 and z/OS do not support UNDO.
Condition Handlers - Example

BEGIN [ATOMIC]

DECLARE type HANDLER FOR conditions

handler-action

statement_1; raises exception

statement_2; CONTINUE point

statement_3;

END UNDO or EXIT point

Notes:

This is the general layout for an SQL Procedure Language stored procedure compound statement that is coded for handling an exceptional condition. It is important to remember that the condition handler is in effect for that compound statement only.

The type is one of the following:

- CONTINUE
- EXIT
- UNDO

DB2 UDB for OS/390 and z/OS do not support UNDO.

Reminder: DB2 for OS/390 and z/OS do not support ATOMIC.
Error Conditions

• Conditions raised
  
  – Implicitly by the system via error situation

  – Explicitly via SIGNAL or RESIGNAL

  • SIGNAL
    ➤ Signal an error or warning condition.
    ➤ It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text.

  • RESIGNAL
    ➤ Resignal an error or warning condition.
    ➤ It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text.

Notes:

Implicit error conditions are set by the database manager and the developer can then set condition handler for those.

The SQL Procedure Language stored procedure developers may choose to generate their own error conditions by using the SIGNAL and RESIGNAL statements.

The SIGNAL and RESIGNAL statements may set SQLSTATE or a condition previously declared in the compound statement.

DB2 UDB for OS/390 and z/OS do not support SIGNAL/RESIGNAL statements.
Example "SIGNAL/RESIGNAL"

CREATE PROCEDURE divide ( IN numerator INTEGER,
                        IN denominator INTEGER, OUT result INTEGER)
    LANGUAGE SQL
BEGIN
    DECLARE overflow CONDITION FOR SQLSTATE '22003';
    DECLARE CONTINUE HANDLER FOR overflow
        RESIGNAL SQLSTATE '22375' ;
    IF denominator = 0 THEN
        SIGNAL overflow;
    ELSE
        SET result = numerator / denominator;
    END IF;
END

Notes:
This example illustrates the use of both the SIGNAL and the RESIGNAL statements. The important points to this block of code are:

- The DECLARE for the overflow CONDITION variable if an SQLSTATE of '22003' is detected by the database manager.
- The DECLARE for the CONTINUE HANDLER if the overflow condition occurs. In this handler the developer wants to change the original SQLSTATE to return the SQLSTATE with a value of '22375'.
- Finally, the check in the application program to explicitly force the overflow condition if the denominator is zero.
Example "Exception Handling"

CREATE PROCEDURE ITERATOR() LANGUAGE SQL
BEGIN
  DECLARE at_end INTEGER DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';
  DECLARE c1 CURSOR FOR ....;
  DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
  OPEN c1;
  ftch_loop1: LOOP
    FETCH c1 INTO v_dept, v_deptname, v_admdept;
    IF at_end = 1 THEN
      LEAVE ftch_loop1;
    ELSEIF v_dept = 'D01' THEN
      ITERATE ftch_loop1;
    END IF;
    INSERT INTO department (deptno, deptname, admrdept)
      VALUES ( 'NEW', v_deptname, v_admdept);
  END LOOP;
  CLOSE c1;
END

Notes:

This example shows how a developer may code for dealing with the end of a set of qualifying rows or no qualifying rows found for an SQL statement.

- The FETCH statement will cause the database manager to move data from the table columns into the local variables until no more rows meet the search criteria. At that time, or if no rows qualify, the SQLSTATE code of '02000' is issued by the database manager.

- The DECLARE not_found CONDITION FOR SQLSTATE '02000' allows us to use the not_found variable in a condition handler.

- The DECLARE CONTINUE HANDLER instructs the database manager to execute the statement following the statement that caused the condition handler to be executed once the condition handler is processed.

- Because the condition handler set the variable, at_end = 1, the next time the loop is executed the LEAVE statement executes.
Nested SP (return to caller) Example

- Client A calls X
  - X opens C1 (return to caller)
  - X call Y
    - Y opens C2 (return to caller)
    - Y cannot access C1
    - Y calls Z
      - Z opens C3 (return to caller)
      - Z cannot access C1, C2
      - Z returns
    - Y can access C3
    - Y returns
  - X can access C2
  - X cannot access C3
  - X returns

- A can access C1
- A cannot access C2, C3

Notes:

This example illustrates how different levels of stored procedures and the original calling program would have access to result set data. Remember that the RETURN TO CALLER and RETURN TO CLIENT are specified in the DECLARE CURSOR for the cursor that is to produce a result set. In this case, result sets are only returned back one level, to the caller. DB2 UDB for OS/390 and z/OS do not support RETURN TO CALLER or RETURN TO CLIENT.
Nested SP (return to client) Example

- **Client A calls X**
  - X opens C1 (return to client)
  - X call Y
    - Y opens C2 (return to client)
    - Y cannot access C1
    - Y calls Z
      - Z opens C3 (return to client)
      - Z cannot access C1, C2
      - Z returns
    - Y cannot access C3
    - Y returns
  - X cannot access C2, C3
  - X returns

- **A can access C1, C2, C3**

**Notes:**

This example illustrates how different levels of stored procedures and the original calling program would have access to result set data. Remember that the RETURN TO CALLER and RETURN TO CLIENT are specified in the DECLARE CURSOR for the cursor that is to produce a result set. In this case, the original client can access all result sets produced by the nested CALLeD procedures.

DB2 UDB for OS/390 and z/OS do not support RETURN TO CLIENT or RETURN TO CALLER.
SQL Procedures - Under the Covers

Executable Stored Procedure is compiled C!

Notes:
This visual illustrates the processes that the DB2 UDB database manager goes through on behalf of the developer when an SQL Procedure Language stored procedure is built. After the SQL parsing and translation of the SQL Procedure Language statements to C, the processes are the same as if the program were a conventional C program with SQL statements in the source. The executable module is a compiled C program.
Creating OS/390 and z/OS SQL Procedures

Notes:
This visual illustrates the creation of an SQL Procedure Language stored procedure for the DB2 UDB for OS/390 and z/OS environment. There are three different approaches that can be used:

1. Use the DB2 Development Center. After Build is requested, the SQL Procedure Language source is passed to DB2 UDB for OS/390 and z/OS and a stored procedure, DSNTPSMP, is invoked to create the stored procedure.

2. Use a provided sample program to invoke the stored procedure, DSNTPSMP. The parameters to DSNTPSMP provide all the information needed to do the create of the new stored procedure.

3. Submit a batch z/OS process to execute the steps to create the new stored procedure.
Using the Development Center

Notes:
This is a quick overview of how the DB2 Development Center could make life easier for a
developer. In these four easy steps, a developer could define a stored procedure, write or
have the SQL assist function produce the SQL Procedure Language statements that make
up the code in the stored procedure, build the stored procedure, and execute the stored
procedure without leaving the comfort of your favorite rolling office chair. The DB2
Development Center provides a terrific development platform that will make you more
productive.
Checkpoint

Exercise — Unit 3 Checkpoint

__ 1.  T/F  The SQL Procedure Language is part of the DDF CREATE statement.

__ 2.  What would you find in a DECLARE statement within a stored procedure?

__ 3.  How is an assignment made to a parameter or a local variable?
Unit Summary

Since completing this unit, you should be able to:

- Describe the DB2 SQL Procedure Language
- Identify valid SQL statements for a procedure
- Code process control statements for a procedure

Notes:
Unit 4. DB2 Development Center andJava

What This Unit Is About

This unit describes the DB2 Development Center’s support of Java stored procedures.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Describe how the DB2 Development Center is used with Java procedures
• Identify similarities and differences between JDBC and SQLJ
• Create and build a Java stored procedure with the DB2 Development Center
• Recognize the Java code generated by the DB2 Development Center

How You Will Check Your Progress

Accountability:

• Checkpoint
• Machine exercises

References

SG24-5485 Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826 IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827 IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844 IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845 IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932 DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933 DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
Unit Objectives

After completing this unit, you should be able to:

- Describe how the Development Center is used with Java procedures
- Identify similarities and differences between JDBC and SQLJ
- Create and build a Java stored procedure with the Development Center
- Recognize the Java code generated by the Development Center

Figure 4-1. Unit Objectives

Notes:
4.1 DB2 Development Center and Java
DB2 Family Java Support

- **Java is a key language for e-business**
  - Language for both Web applications and database logic.

- **DB2 Java support including Java SPs first introduced in 1996 for Intel and UNIX**
  - Support for JDBC, SQLJ Part 0 and Part 1 specifications

**Notes:**

Stored procedures can provide major benefits in the areas of application performance, code reuse, security and integrity. Java’s inherent portability and openness, combined with the availability of skilled programming resource, has made it an increasingly attractive choice as the central plank in the e-business strategy of many organizations.

Java is designed to be portable, and Java programs developed on one platform can often run unchanged on many computer systems. The most important characteristics that make Java portable are:

- Java compiles to machine-independent byte code.

  Java is typically compiled and executed in a two-step process. First, the Java code is compiled to byte code, which means an assembly language for an idealized Java virtual machine (JVM). Second, this byte code is executed by a run-time system, which can either be an interpreter, that is, an emulator for the JVS, or a Just-In-Time (JIT) compiler that first compiles the byte code to native code, then executes the result. Summarizing, the two steps of this process can be done on totally separate platforms.
• Java has a portable graphic library.

In many software systems, the biggest hindrance to portability is the user interface. Interfaces have typically been developed using a native windowing system rather than a cross platform graphics toolkit, because, until now, this was the most convenient and widely available option. However, this often meant that distribution on a different operating system required a complete rewrite of the GUI. The Java developers realized that a truly portable language would need a standard graphics library, and included one from the beginning of the language’s development.

• Java avoids hard-to-port constructs.

The Java specification defines the size of primitive data structures such as booleans, doubles, unlike some languages that allow this to vary among implementations. For objects, Java programs cannot inadvertently depend on implementation-specific details such as the amount of memory an object consumes, or the internals of how fields or functions are laid out within an object. Java even avoids reference to the local file system when specifying which classes your program needs, using operating system neutral class and package name instead.
Notes:

There might be cases where you need an application that can access DB2 databases across the Internet. Using the Java programming language, you can develop applications and applets that access and manipulate data in the DB2 database. DB2 provides support for Sun Microsystem’s Java Database connectivity (JDBC) API. The support is provided through a DB2 JDBC driver that comes with DB2. The JDBC API, which is similar to ODBC APIs, provides a standard way to access databases from JAVA code. The Java code passes SQL statements as function arguments to the DB2 JDBC driver. The driver handles the JDBC API calls from the client Java code. DB2’s Java enablement has three independent components:

- Support for client applications and applets written in Java using JDBC to access DB2.
- Precompile and binding support for client applications and applets written in Java using SQLJ to access DB2.
- Support for Java UDFs and stored procedures on the database server.
The graphic illustrates how a DB2 JDBC application works. You can think of a DB2 JDBC application as a DB2 CLI application, only you write it using the Java language. Calls to JDBC are translated to calls to DB2 CLI through Java native methods. JDBC requests flow from the DB2 client through DB2 CLI to the DB2 Server. SQLJ applications use this JDBC support, and in addition require the SQLJ run-time classes to authenticate and execute any SQL packages that were bound to the database at the precompiling and binding stages.
**DB2 Java Applets**

**Notes:**

This visual illustrates how the JDBC applet works. The driver consists of a JDBC client and a JDBC server. The JDBC client driver is loaded on the Web browser along with the applet. When the applet requests a connection to the database, the client opens a TCP/IP socket to the JDBC server on the machine where the Web server is running. After a connection is set up, the client sends each of the subsequent database access requests from the applet to the JDBC server through the TCP/IP connection. The JDBC server then makes corresponding CLI (ODBC) calls to perform the task. Upon completion, the JDBC server sends the results back to the client through the connection. SQLJ applets add the SQLJ client driver on top of the JDBC client driver, but otherwise work the same as JDBC applets.
Java Options

- **DB2 provides support for:**
  - Java applications
    - Require DB2 Run-time Client
  - Java applets
    - No DB2 component code on the client
  - Java servlets
  - UDFs and stored procedures

- **DB2 access via:**
  - JDBC
  - SQLJ

**Notes:**

You can use DB2’s Java support to build:

- Java applications, which rely on the DB2 UDB run-time client to connect to DB2.
- Java applets, that do not require any DB2 component code on the client.
- Java servlets, that run on the server.
- Java UDFs and stored procedures.

Java database connectivity has matured to the point that the developer now has two alternative APIs for accessing a database from within a Java application: SQLJ and JDBC. JDBC is a component of the core Java API standard as defined by Sun, as it is an integral part of the Java Development Kit (JDK). JDBC does not directly embed SQL in the Java code, but implements a set of methods that allow SQL statements to be passed to the database, and results to be accessed and manipulated.
SQLJ is an ANSI standard set of extensions to the core Java classes that allow SQL to be directly embedded within Java applications, in much the same way as it can be embedded in other languages such as C, C++, COBOL and PL/I.
JDBC Support

- Standard Java interface for connecting to relational databases
- Supports use of dynamic SQL
- No precompile
- No BIND

Notes:
The JDBC API is a standard Java interface for connecting to relational databases from Java. JDBC consists of two parts: the high-level API and multiple low-level drivers for connecting to different databases. It specifies Java interface, classes, and exceptions to support connections, SQL data manipulation language (DML), SQL data definition language (DDL), processing of data result sets, as well as other functions.

JDBC supports the use of dynamic SQL so it can deal with situations where you do not know the table and column names at the time the application is written. Unlike static embedded SQL programs, dynamic SQL programs involve the execution of at least some SQL statements that are not completely known until runtime. Dynamic SQL allows you to create general and flexible applications because the full text of the SQL statement does not have to be known at compilation time.

After coding your program, compile it as you would any other Java program. You don’t need to perform any special precompile or bind steps.
SQLJ Support

- **SQLJ application, servlet, and applet support for Java.**

- **Static SQL**
  - INSERT, UPDATE, DELETE, CREATE, GRANT and so forth
  - Singleton and cursor-based SELECT
  - Calls to stored procedures (including result sets)
  - COMMIT, ROLLBACK
  - Methods for CONNECT and DISCONNECT

- **Static SQL can be faster.**

- **JDBC must be used for dynamic SQL.**

---

Notes:

IBM’s DB2 SQL driver complies with the ANSI X.3.1.135 standard, and allows you to create, build, and run Java applications containing embedded static SQL statements that are bound to a DB2 database.

SQLJ can provide better performance and security than JDBC, based on the program author’s SQL privileges. SQLJ is preprocessed by the DB2 SQLJ Translator. Coding SQLJ is much simpler, is more efficient, and can perform better at run-time. This can lead to more programmer productivity and better system utilization.

SQLJ consists of a set of clauses that extend Java programs. The language specification is a joint one, supported by leading database vendors. It is supported by IBM’s VisualAge for Java tools.

When using SQLJ, you directly embed SQL statements within your Java programs following certain standard syntax rules.
Comparing JDBC and SQLJ

- **SQLJ:**
  - Extends Java to support static SQL constructs.
  - Complements JDBC dynamic SQL API model.

- **Similar considerations and tradeoffs as in other languages:**
  - **Static versus dynamic**
    - Dynamic is flexible at run time.
    - Building SQL on the fly requires dynamic access.

- **Pre-compilation of SQLJ based code**
  - Type-checking performed against the database during development.
  - Simplifies task for programmer:
    - Less source code.
    - Translator handles most of the work for the programmer.
    - Simplified rules for calling Stored Procedures and functions.

**Notes:**

The JDBC and SQLJ APIs are both widely accepted open industry standards. However, each has its own set of advantages and disadvantages:

- **Application coding considerations**
  
  If you come from a database background and have SQL skills, SQLJ is usually quicker and more intuitive than JDBC, as it allows you to directly embed SQL within your application code. In contrast, JDBC uses a series of generic methods to perform the data access, which is almost always more verbose than the SQLJ equivalent.

- **Performance: Static versus Dynamic SQL**
  
  Another major differentiator between JDBC and SQLJ can be the way in which your programs perform. JDBC presents a string containing an SQL statement to the database at run time. The first time DB2 gets to see the SQL is when it is actually executed, so all JDBC calls, by their very nature, consist of dynamic SQL. When DB2 is passed the SQL statement, it must perform a number of steps to prepare the statement before it is able to execute it (including syntax checking, authorization checking, and
access path selection). SQLJ is able to use static SQL by actually embedding the SQL statements within the application code. The SQLJ program preparation process extracts this SQL and binds it against the database, allowing DB2 to perform all of the checks and access path selection at program preparation time. At run time, DB2 uses the pre-prepared access plan and is able to immediately execute the SQL.

- Authorization

The way in which authorization is handled can also differ, due to the way in which dynamic and static SQL authority checking is managed by DB2. As JDBC uses dynamic SQL, all authorization checking is performed at execution time. By default, DB2 will use the authorization ID of the application process for authorization of the dynamic SQL statements. This requires granting sufficient authority directly to that authorization ID to be able to execute any of the SQL statements the JDBC program is likely to submit. SQLJ uses static SQL, so all authority checking is performed at BIND time. The BIND authorization ID is used in the authorization checks, and all that is required at run time is authority to execute the DB2 package associated with the stored procedure.
Java Stored Procedures

- Java code which executes at the DB2 server.
- Java Stored Procedure maps to static void method.
  - Out and InOut parameters are single-element Java arrays.
  - Input parameters are host variables.
- Both static (SQLJ) and dynamic (JDBC) SQL allowed in the body of the Stored Procedure.
- Java routines support recursion.

Notes:

DB2 stored procedures can be written in a number of languages, including SQL Procedure Language which is discussed in other topics in this course, and Java, which is the subject of this topic.

A Java stored procedure must be coded as a `public static` method in a `public` class. Parameters which are used to return data (OUT or INOUT) are single-element Java arrays. The stored procedure must receive input parameters as host variables.

Both SQLJ and JDBC SQL are allowed in the body of the stored procedure.

Java routines now support recursion. There are no restrictions on the types of routines that can be nested. Java routines can invoke routines written in other languages and can be invoked by routines written in other languages.
Using the Development Center with Java

- Create stored procedure
  - Using Wizard
  - Without SQL
- Modify existing Java stored procedures
  - Edit the code or SQL, use SQL Assistant
  - Change procedure properties
- Build the Java stored procedure
  - SQLJ
  - JDBC

Notes:
The DB2 Development Center greatly simplifies the process of creating and installing stored procedures on a DB2 database server. With the DB2 Development Center, you can create stored procedures in Java using a Wizard. You can generate stored procedures that do not contain SQL statements.

If you open a project in the Development Center that contains existing Java stored procedures, you can modify the source code, including the SQL statements. The DB2 Development Center drops the old stored procedure from the database and creates a stored procedure that reflects the changes that you made.

Java stored procedures that are built by the DB2 Development Center conform to the SQLJ Routines specification. Java stored procedures are defined in the catalog with LANGUAGE JAVA and PARAMETER STYLE JAVA.
What Gets Built - SQLJ (1 of 2)

```java
import java.sql.*; // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*/
#sql context SPContext;

#sql iterator PROC3_Cursor1 ( int );
#sql iterator PROC3_Cursor2 ( int );

public class PROC3
{
    public static void PROC3 ( int whichQuery, int[] count ) throws SQLException, Exception
    {
        PROC3_Cursor1 cursor1 = null;
        PROC3_Cursor2 cursor2 = null;
        SPContext ctx = null;
        try
        {
            ctx = new SPContext( "jdbc:default:connection", false );
            switch (whichQuery)
            {
                case 0:
                    #sql [ctx] cursor1 = {SELECT COUNT(*) FROM ADMIN.EMPLOYEE};
                    break;
                case 1:
                    #sql [ctx] cursor2 = {SELECT COUNT(*) FROM ADMIN.DEPARTMENT};
                    break;
                default:
                    break;
            }
        }
    }
}
```

Figure 4-11. What Gets Built - SQLJ (1 of 2)

Notes:

The SQL server procedure that is generated by the DB2 Development Center includes the code necessary for setting up the SQLJ stored procedure and the SQL statements that you specify.

1. It imports the Java packages for SQLJ run-time support and the JDBC interfaces that are used by SQLJ.
2. It codes the Java stored procedure as a `public static method` in a `public class`.
3. It declares any local variables that you have specified in the stored procedure definition.
4. If you indicated that multiple SQL: statements were to be executed, it creates the switch statement.
5. It sets the returned variable value.
6. You will have to uncomment the ResultSet statements and add assigning values to your return parameters.

While it is possible, using SQLJ, to code an SQL statement with an INTO clause, we decided to work with the code that was generated by the Development Center.
//Sample code to access the results
ResultSet rs1 = null;  //uncommented the following
switch (whichQuery)
{
    case 0:
        rs1 = cursor1.getResultSet();
        break;
    case 1:
        rs1 = cursor2.getResultSet();
        break;
    default:
        break;
}
while (rs1.next())
{
    count[0] = rs1.getInt(1);  //added
}

// Close open resources
if (rs1 != null) rs1.close();  //might want to add this
if (cursor1 != null) cursor1.close();
if (cursor2 != null) cursor2.close();
if (ctx != null) ctx.close();  //might want to add this

// Set return parameters
cursor[0] = cursor[0];  //could be deleted
}
catch (SQLException e)
{
    ...
}

Notes:
This is a continuation of the code from the previous visual.
The code that is generated will not be complete. You will have to specify any additional logic required. As noted in the visuals, you may have to code some additional SQL statements. But it does give you a good starting point for development effort.
What Gets Built - JDBC (1 of 2)

import java.sql.*;       // JDBC classes

public class PROC4
{
    public static void PROC4( int whichQuery,int[] count )
       throws SQLException, Exception
    {
        // Get connection to the database
        Connection con = DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement stmt = null;
        ResultSet rs1 = null;
        Boolean bFlag;
        String sql;

        switch (whichQuery)
        {
        case 0:
            sql = "SELECT COUNT(*) FROM ADMIN.EMPLOYEE";
            stmt = con.prepareStatement( sql );
            break;
        case 1:
            sql = "SELECT COUNT(*) FROM ADMIN.DEPARTMENT";
            stmt = con.prepareStatement( sql );
            break;
        default:
            ...
        }

Figure 4-13. What Gets Built - JDBC (1 of 2)

Notes:

For a JDBC program, the DB2 Development Center creates the statements needed to set
up the JDBC stored procedure.

1. It imports the JDBC interfaces that are used by the procedure.
2. It codes the Java stored procedure as a public static method in a public class.
3. Since the stored procedure runs under the same connection and transaction as the
   client application, the JDBC program must obtain access to the Connection object of
   the client application. The way that a JDBC stored procedure gets the caller's
   connection to the database is to use the getConnection() method with a default
   argument. This method is inherited from the StoredProc class.
4. It declares local variables for the dynamic SQL statement and result set.
5. If you indicated that one of multiple statements were to be executed, it creates the
   switch statement and sets up the text for the statements you specified.
6. The rest of the code is continued on the next visual.
What Gets Built - JDBC (2 of 2)

bFlag = stmt.execute;
rs1 = stmt.getResultSet();

// Access query results
while (rs.next()) (uncommented the following)
{
    count[0] = rs1.getInt(1); (added)
}

// Close open resources
if (rs1 != null) rs1.close();
if (stmt != null) stmt.close();
if (con != null) con.close();

// Set return parameter
count[0] = count[0]; (could be deleted)

Notes:
For a JDBC program, the DB2 Development Center creates the statements needed to set up the JDBC stored procedure. This is the continuation of the code that we saw illustrated on the previous graphic.

1. It executes the statement.
2. Unlike SQLJ, you cannot indicate an INTO clause on the SQL statement. So, to acquire the result from the query, you must uncomment the code provided by the DB2 Development Center to return the desired value.
3. It closes resources which are open.
4.2 Building a Java Stored Procedure Manually - Reference Only
Server Procedure - Method Definition

```java
public class MyClass
{
    public static void MyProc
    {
        String [] var1,
        double var2,
        int [] var3) throws Exception
    {
        int counter;
        /* receives IN parameters as host variables */
        /* receives OUT and INOUT parameters */
        /* as single entry arrays */
        var1[0] = "new value";
        #sql { SELECT COUNT (*) INTO :counter
            FROM EMP
            WHERE SALARY > :var2 };
        var3[0] = counter;
    }
}
```

**Notes:**

- Server procedure is another term for the stored procedure code that is stored on the server machine.
- A Java stored procedure must be coded as a `public static method` in a `public` class.
- No connection-related statements are permitted in the server procedure.
- Server procedure runs in the background, so screen writes `System.out.println()` are not allowed. File writes are permitted.
- The server procedure should always return control to the client without terminating the current process.
- COMMIT statements are not allowed in any registered stored procedure. A COMMIT statement can only be issued by the client application.

**Note:** On UNIX-based systems, your stored procedure runs under the UID of the DB2 Agent Process (NOT FENCED), or the UID which owns the `db2dari` executable (FENCED). It is the userid specified when the instance is created for running fenced...
user-defined functions and stored procedures. This UID controls the system resources available to the stored procedure.
CREATE PROCEDURE

CREATE PROCEDURE MYPROC (INOUT HOST1 CHAR(15),
                        IN HOST2 DOUBLE, OUT HOST3 INTEGER)
EXTERNAL NAME 'MyClass.MyProc'
LANGUAGE JAVA
PARAMETER STYLE JAVA
READS SQL DATA

#sql { CALL MYPROC
       (host_var1, host_var2,
        host_var3);}

Notes:

- A stored procedure can be registered with the application server with the CREATE PROCEDURE statement.
- MYPROC represents the name that designates the stored procedure. This is the name that can be used on the SQL CALL statement issued from the client application.
- INOUT HOST1 CHAR(15) represents a parameter being used by the stored procedure. INOUT indicates that this parameter will be used both to provide information to the stored procedure, and to pass information from the stored procedure back to the client application. A value of IN indicates that the parameter is input only. A value of OUT indicates that the parameter is output only.
- No two identically named procedures within the same schema are permitted to have exactly the same number and types of parameters.
- EXTERNAL NAME indicates the name of the user-written code which implements the procedure being defined.
- **LANGUAGE** is a mandatory clause that indicates the language interface conventions to which the stored procedure body is written. **LANGUAGE C** or **LANGUAGE JAVA** can be specified.

- **PARAMETER STYLE** specifies the conventions used for passing parameters to and returning the values from the stored procedure.
  
  **DB2DARI** means that the stored procedure will use a parameter passing convention that conforms to C language calling and linkage convention; it must be used with **LANGUAGE C**.
  
  **DB2GENERAL** means that the stored procedure will use a parameter passing convention that is defined for use with Java methods; it can only be used with **LANGUAGE JAVA**.
  
  **JAVA** means that the stored procedure will use a parameter passing convention that conforms to the Java language and SQLJ Routines specification. IN/OUT and OUT parameters will be passed as single entry arrays to facilitate returning values. This can only be specified when **LANGUAGE JAVA** is used.

- **READS SQL DATA** indicates that the stored procedure will be issuing SELECT statements. Other options include **NO SQL**, **CONTAINS SQL**, and **MODIFIES SQL DATA**.
Specifying EXTERNAL NAME

CREATE PROCEDURE MYPROC (...)  

EXTERNAL NAME 'MyClass.MyProc2'

EXTERNAL NAME 'MyJar:MyClass.MyProc'

Notes:

- The EXTERNAL clause of the CREATE PROCEDURE statement tells the database manager the location of the library that contains the stored procedure. If you do not specify a jar name, the database manager searches the function directory. The function directory is a directory defined for your operating system as follows:

  **Linux/UNIX operating systems**: `sqlib/function` or `sqlib/function/unfenced`

  **Windows 32-bit operating systems**: `instance_name\function` or `instance_name\function\unfenced` where `instance_name` represents the value of the `DB2INSTPROF` instance-specific registry setting. If DB2 does not find the stored procedure in `instance_name\function`, DB2 searches the directories defined by the `PATH` and `LIBPATH` environment variables.

  For example, the function directory for a Windows NT server with DB2 installed in the `C:\sqlib` directory, and with no specified `DB2INSTPROF` registry setting, is: `C:\sqlib\function`. If DB2 is installed in `C:\Program Files\sqlib`, with
no specified value for DB2INSTPROF, the function directory will be C:\Program Files\sql\lib.

If you choose to use individual class files, you must store the class files in the appropriate directory for your operating system. If you declare a class to be part of a Java package, create the corresponding subdirectories in the function directory and place the files in the corresponding subdirectory. For example, if you create a class ibm.tests.test1 for a Linux system, store the corresponding Java bytecode file (named test1.class) in sql\lib\function\ibm\tests.

• For LANGUAGE JAVA stored procedures, use:
  - (Optional) The name you gave to the jar file when you installed it in the database, followed by a colon. You can qualify the jar name with a schema name. If you do not specify a jar file, the database manager looks for the class in the function directory.
  - The class name; if the class is part of a package, include the complete package prefix.
  - The method name.

• The Java interpreter that DB2 invokes uses the CLASSPATH environment variable to locate Java files. DB2 adds the function directory and sql\lib\java\db2java.zip to the front of your CLASSPATH setting.

• To set your environment so that the Java interpreter can find where you have stored the Java class files, you may need to set the jdk11_path configuration parameter, or else use the default value. Also, you may need to set the java_heap_sz configuration parameter to increase the heap size for your application.
Using JAR Files

1. Create a separate directory for the files.
2. Create the Java function (stored procedure) as a Java method. Compile the Java source code into a Java class file.
   sqlj..., db2profc..., javac MyProc.java
3. Collect the class files containing the Java function in a JAR file.
   jar cvf SPJar.jar *.class *.ser
4. Install the JAR file in the DB2 instance.
   db2 call sqlj.install_jar
      ('file:\javasp\SPJar.jar', 'MyJar')
5. Issue the appropriate CREATE PROCEDURE statement.
   db2 create procedure myproc (...) external name 'MyJar:MyClass.MyProc'

   Note: jar uvf SPjar.jar *.class *.ser
        jar tvf SPjar.jar

Notes:

• To register a Java stored procedure:
  1. Create a subdirectory to use for the files for this stored procedure.
  2. Create the Java function (stored procedure) as a Java method. This would involve the manual process of doing the SQLJ translation (sqlj) and customizing the profile (db2profc). Compile the Java source code into a Java class file.
  3. Collect the class file containing the Java function into a jar file. You can collect one or more class files into a single JAR file. You will also need to collect serialized profiles for SQLJ programs into the JAR file.
  4. Install the JAR file in the DB2 instance. This requires the use of a DB2 command:
     CALL SQLJ.INSTALL_JAR.
  5. Issue the appropriate CREATE PROCEDURE statement for the Java function.

• The syntax to install a JAR file in the DB2 instance is:
   CALL SQLJ.INSTALL_JAR ('jar-url','jar-id')
where

jar-url specifies the URL containing the JAR file to be installed. The only URL scheme supported is 'file:'.

jar-id specifies the JAR identifier in the database to be associated with the file specified by jar-url.

- In a similar fashion, the syntax to replace a JAR file in the DB2 instance is:

  CALL SQLJ.REPLACE_JAR ('jar-url','jar-id')

- Once the JAR file is installed, subsequent SQL commands that use the JAR file refer to it with the name specified in jar-id.

- To remove a JAR file from the database, use the following:

  CALL SQLJ.REMOVE_JAR ('jar-id')

- When you install a JAR file, DB2 extracts the Java class files from the JAR file and registers each procedure in the system catalog. DB2 copies the JAR file to a jar/schema subdirectory of the function directory. DB2 gives the new copy of the JAR file the name given in the jar-id clause. Do not directly modify a JAR file which has been installed in the DB2 instance. Instead, you can use the CALL SQLJ.REMOVE_JAR and CALL SQLJ.REPLACE_JAR commands to remove or replace an installed JAR file.

- Once a JAR file has been created, you can update it using:

  jar uvf SPJar.jar *.class *.ser

  or look at its contents using:

  jar tvf SPJar.jar
Creating Java SP Manually

- **(Step 1) Create Logic**
  - Write Code
  - SQLJ Translation
  - Java Compile
  - DB2Profc
  - Create Jar

- **(Step 2) Install Logic**
  - Install or Replace Jar

- **(Step 3) Register Procedure**
  - Create and Execute DDL

- **(Step 4) Test SP**
  - Write and Execute test program to call SP

---

**Notes:**
Checkpoint

Exercise — Unit 4 Checkpoint

__ 1. The DB2 Development Center only supports two languages within the DB2 Development Center: ___________ and ___________.

__ 2. The DB2 Development Center will allow you to build a stored procedure in Java that will access the DB2 database manager using either SQLJ or ______________.

__ 3. What is the defining difference between SQLJ and JDBC?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
Unit Summary

Since completing this unit, you should be able to:

- Describe how the Development Center is used with Java procedures
- Identify similarities and differences between JDBC and SQLJ
- Create and build a Java stored procedure with the Development Center
- Recognize the Java code generated by the Development Center

Notes:
Unit 5. Stored Procedure CALL

What This Unit Is About

This unit will describe the correct syntax of the CALL SQL statement and the calling program's pre and post CALL requirements.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Identify correct CALL syntax
• Set up for Client CALL to stored procedure
• Set up the Java stored procedure method

How You Will Check Your Progress

Accountability:

• Checkpoint
• Machine exercises

References

SG24-5485 Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826 IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827 IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844 IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845 IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932 DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933 DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
SC26-9944 DB2 UDB for OS/390 and z/OS V7 SQL Reference
Unit Objectives

After completing this unit, you should be able to:

- Identify correct CALL syntax
- Set up for Client CALL to stored procedure
- Process result sets

Notes:
5.1 Stored Procedure CALL
CALL Syntax

\[
\text{CALL} \quad \text{procedure-name} \\
\quad \text{host-variable}
\]

The client application must
- declare
- initialize
- pass data area

for each value to be passed either TO or FROM the stored procedure.

Notes:
- UDB for Linux, UNIX and Windows
  - NULL keyword not allowed
- DB2 Server for VSE and VM
  - Allows constant in parameter list
- UDB for iSeries
  - Allows constant in parameter list
  - Allows special register in parameter list
  - Allows DLVALUE (args) in parameter list
  - Allows cast function in parameter list
Server Procedure - Generic

A stored procedure, EMPCHG, with the following parameters will be called by various clients:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>EmpID</td>
<td>string</td>
</tr>
<tr>
<td>INOUT</td>
<td>NewSal</td>
<td>double</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>JobCode</td>
<td>int</td>
</tr>
</tbody>
</table>

The three parameters are expected in this order by the stored procedure. The first two should be initialized by the client. The third one would contain a value on return from the CALL.

Notes:
Assume that the stored procedure being called is documented as above. We will follow through calling this procedure using several different approaches. Note the parameters being used for input and output should be set by the calling program. The output parameter data type should match the data type that the procedure expects.
Client Application - SQLJ

```java
string EmpID = "01327";
double NewSal = 17600.00;
int JobCode;

// call the stored procedure
#sql { CALL EMPCHG( :EmpID,
    :NewSal,
    :JobCode) }
```

**Notes:**
The local variable definitions will be in native Java. Any local variables used in the SQL statement must be defined in the application program in order to be recognized.
Client Application - JDBC

```java
string EmpID = "01327";
double NewSal = 17600.00;
int JobCode;
String sql = "CALL EMPCHG(?,?,?)";

CallableStatement callStmt = con.prepareCall(sql);
callStmt.setString (1, EmpID);
callStmt.setDouble (2, NewSal);
callStmt.setInt (3, JobCode);

// register the parameters
callStmt.registerInParameter (1, Types.CHAR);
callStmt.registerInoutParameter (2, Types.DOUBLE);
callStmt.registerOutParameter (3, Types.INT);
// call the stored procedure
callStmt.execute();
```

**Notes:**

- Program must define host variable prior to use.
- The `?` marks represent the parameters needed by the stored procedure to be called.
- The `prepareCall(sql)` is a JDBC provided method to dynamically prepare an SQL statement.
- The JDBC driver will pass this statement to the DB2 server.
Client Application - C/C++

```c
char[6] EmpID = "01327";
double NewSal;
int JobCode;

NewSal = 17600.00;
EXEC SQL CALL EMPCHG (:
    :EmpID,
    :NewSal,
    :JobCode);
```

**Notes:**

- Note the host variables being defined to the program prior to being used for the CALL statement.
- Host variable definitions must match the data type of the parameters being defined to the CALLeed stored procedure.
- Data type mismatches are a common source of errors when using stored procedures.
Client Application - COBOL

01 EmpID pic x(10) value "01327".
01 NewSal pic s9(9) comp-3.
01 JobCode s9(5) comp-5.

NewSal = 17600.00;
EXEC SQL CALL EMPCHG (:EmpID,
                      :NewSal,
                      :JobCode)
END-EXEC.

Notes:

- Note the delimiter for COBOL is the EXEC SQL and END-EXEC combination of keywords.
- Program variable definitions correspond to the data types for the stored procedure parameters as with the other stored procedure calls previously shown.
CALL from Stored Procedure

- A procedure may call another procedure.
- An SQL procedure may only call another SQL procedure.
- Some restrictions in place for SQL statements.
- Result sets may be processed.

**Notes:**

Most SQL statements can be contained within an SQL procedure. Refer to *Cross-Platform DB2 Stored Procedures: Building and Debugging*, topic 3.6.1 for the tables on platform portability.
Locator Variable

CREATE PROCEDURE CALLING_PROC
   (IN inParm SMALLINT, OUT outParm DOUBLE)
LANGUAGE SQL
BEGIN
   DECLARE loc1, loc2 RESULT_SET_LOCATOR VARYING;

   • One locator variable for each result set

Notes:

• More than one locator can be assigned to a result set. You can issue the same
  ASSOCIATE LOCATORS statement more than once with different result set locator
  variables.

• If the number of result set locator variables that are listed in the ASSOCIATE
  LOCATORS statement is less than the number of locators returned by the stored
  procedure, all variables in the statement are assigned a value, and a warning is issued.

• If the number of result set locator variables that are listed in the ASSOCIATE
  LOCATORS statement is greater than the number of locators returned by the stored
  procedure, the extra variables are assigned a value of 0.

• If a stored procedure is called more than once from the same caller, only the most
  recent result sets are accessible.
ASSOCIATE

CALL CALLED_PROC;
ASSOCIATE RESULT SET LOCATORS (loc1, loc2)
WITH PROCEDURE CALLED_PROC;

• Locator variables are associated with the result set produced by the CALLeD procedure.

Notes:
The ASSOCIATE statement ties the locator variables to the DB2 stored procedure.
CALL CALLED_PROC;
ASSOCIATE RESULT SET LOCATORS (loc1, loc2) WITH PROCEDURE CALLED_PROC;
ALLOCATE C1 CURSOR FOR RESULT SET loc1;
ALLOCATE C2 CURSOR FOR RESULT SET loc2;

• One ALLOCATE statement for each result set to be processed.

Notes:
The following rules apply when using an allocated cursor:
• An allocated cursor cannot be opened with the OPEN statement (SQLSTATE 24502).
• An allocated cursor can be closed with the CLOSE statement. Closing an allocated cursor closes the associated cursor in the stored procedure.

Only one cursor can be allocated to each result set.
Allocated cursors last until a roll back operation, an implicit close, or an explicit close.
A commit operation destroys allocated cursors that are not defined WITH HOLD by the stored procedure.
Destroying an allocated cursor closes the associated cursor in the SQL procedure.
FETCH

```
FETCH C1 INTO :var1;
...
FETCH C2 INTO :var2;
```

- FETCHes processed same as if CURSOR was DECLAREd.
- CURSOR may be closed at any time.
- End of result set will set not found condition.

**Notes:**

- FETCH into variables previously defined to the program or SQL procedure acting as the client.
- If all rows of the result set are FETCHed, an SQLSTATE of '02000' will result.
- Cursor may be closed at any time.
- Result set FETCHing is the same as FETCHing from a cursor that has been DECLAREd and OPENed by a conventional DB2 program.
CLOSE

CLOSE C1, C2;

- Good programming protocol would CLOSE all cursors.
- CURSOR may be closed at any time.
- Any unclosed cursors will be closed at process end.

Notes:
Although the end of the process will close all open cursors, it is a good practice to close cursors as soon as logically feasible. The database server is still holding information about that cursor until the close is executed.
Checkpoint

Exercise — Unit 5 Checkpoint

___ 1.  **T/F**  A stored procedure may not call another stored procedure.

___ 2.  A DECLARE for a result set locator in a calling application would indicate that what clause was coded in the called stored procedure's DECLARE CURSOR?

___ 3.  Is an OPEN cursor statement needed for processing a result set in the calling program?
Unit Summary

Since completing this unit, you should be able to:

- Identify correct CALL syntax
- Set up for Client CALL to stored procedure
- Process result sets

Notes:
Unit 6. Create Procedure

What This Unit Is About

This unit will describe the CREATE PROCEDURE DDL statement.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Code the CREATE PROCEDURE DDL
• Describe EXTERNAL NAME
• Create JAR files

How You Will Check Your Progress

Accountability:

• Checkpoint

References

SG24-5485 Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826 IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827 IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844 IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845 IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932 DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933 DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
SC26-9944 DB2 UDB for OS/390 and z/OS V7 SQL Reference
Unit Objectives

After completing this unit, you should be able to:

- Code the CREATE PROCEDURE DDL
- Describe EXTERNAL NAME
- Create JAR files

Notes:
6.1 Create Procedure
CREATE PROCEDURE

- A DDL statement used to register a stored procedure with the database manager.

- May be executed from a program.

- Appropriate authorizations must be in place.

Notes:

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- CREATE_EXTERNAL_ROUTINE authority and least one of:
  - IMPPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the procedure does not exist
- CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema.

To create a not-fenced stored procedure, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_NOT_FENCED authority on the database.
- SYSADM or DBADM authority.

To create a fenced stored procedure, no additional authorities or privileges are required.

If the authorization ID has insufficient authority to perform the operation, an error (SQLSTATE 42502) is raised.

Although we will be focusing on the DDL to create a procedure, do not forget that there is the ALTER statement as well. The ALTER statement can be used to change some of the characteristics that we will discuss, such as:

- EXTERNAL NAME
- FENCED characteristics
- THREADSAFE
- FEDERATED characteristics
CREATE Syntax

```
CREATE PROCEDURE myschema.mypname
  (IN myParm1 data-type,
   OUT myParm2 data-type,
   INOUT myParm3 data-type)
  EXTERNAL NAME 'schema.mypname'
  DYNAMIC RESULT SETS n
```

- Procedure name may be qualified by schema.
- Procedure name and number of parameters must be unique within a schema.
- External name identifies the executable module.

**Notes:**

**procedure-name**

Names the procedure being defined. It is a qualified or unqualified name that designates a procedure.

The unqualified form of procedure-name is an SQL identifier (with a maximum length of 128). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a schema-name followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters, must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of the parameters, while of course unique within its schema, need not be unique across schemas.

The two-part name is specified, the schema-name cannot begin with SYS. Otherwise, an error (SQLSTATE 42939) is raised.
IN | OUT | INOUT

- IN - parameter is input only
- OUT - parameter is output only
- INOUT - parameter is both input and output

EXTERNAL

This parameter provides the external name of the program to be executed if this is not an SQL procedure. For DB2 for OS/390 and z/OS, it will be the load module name in a library available to the server. For DB2 for Linux, UNIX, and Windows, it would be the path name. A Java stored procedure may include the JAR filename.
CREATE Syntax - SPECIFIC

CREATE PROCEDURE ....
    ....
    SPECIFIC specific-name
    NO SQL or
    CONTAINS SQL or
    READS SQL DATA or
    MODIFIES SQL DATA

- SPECIFIC is optional and used for control purposes, not for client program CALLs.

- SQL may be controlled by specification in DDL.

Notes:

SPECIFIC specific-name

Provides a unique name for the instance of the procedure that is being defined. This specific-name can be used when dropping the procedure or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of specific-name is an SQL identifier (with a maximum length of 18). The qualified form is a schema-name followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The specific-name may be the same as an existing procedure-name.

If no qualifier is specified, the qualifier that was used for procedure-name is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier of procedure-name or an error (SQLSTATE 42882) is raised.

If specific-name is not specified, a unique name is generated by the database manager. The unique name is SQL followed by a character timestamp, SQLyymmddhhmssshhn.
SPECIFIC is not valid in DB2 for OS/390 and z/OS.

NO SQL

Indicates that the stored procedure cannot execute any SQL statements (SQLSTATE 38001).

NO SQL is not valid in DB2 for OS/390 and z/OS.

CONTAINS SQL

Indicates that SQL statements that neither read nor modify SQL data can be executed by the stored procedure (SQLSTATE 38004 or 42985). Statements that are not supported in any stored procedure return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA

Indicates that some SQL statements do not modify SQL data can be included in the stored procedure (SQLSTATE 38002 or 42985). Statements that are not supported in any stored procedure return a different error (SQLSTATE 38003 or 42985).

MODIFIES SQL DATA

Indicates that the stored procedure can execute any SQL statement except statements that are not supported in stored procedures (SQLSTATE 38003 or 42985).

DBINFO/NO DBINFO

Specifies whether specific information known by DB2 is passed to the stored procedure when it is invoked as an additional invocation-time argument (DBINFO) or not (NO DBINFO). NO DBINFO is the default. DBINFO is not supported for LANGUAGE OLE (SQLSTATE 42613). It is also not supported for PARAMETER STYLE JAVA or DB2GENERAL.

If DBINFO is specified, a structure containing the following information is passed to the stored procedure:

- Data base name - The name of the currently connected database.
- Application ID - Unique application ID which is established for each connection to the database.
- Application Authorization ID - The application run-time authorization ID.
- Code page - Identifies the database code page.
- Database version/release - Identifies the version, release and modification level of the database server invoking the stored procedure.
- Platform - Contains the server's platform type.

The DBINFO structure is common for all external routines and contains additional fields that are not relevant to procedures.

This option is only valid for external stored procedures.
Figure 6-5. CREATE Syntax - PARAMETER STYLE

Notes:

PARAMETER STYLE
This clause is used to specify the conventions used for passing parameters to and returning the value from stored procedures.

DB2GENERAL
This means that the stored procedure will use a parameter passing convention that is defined for use with Java methods. This can only be specified when LANGUAGE JAVA is used.

The value DB2GENRL may be used as a synonym for DB2GENERAL.

DB2GENERAL is not valid in DB2 for OS/390 and z/OS.

GENERAL
This means that the stored procedure will use a parameter passing mechanism where the stored procedure receives the parameters specified on the CALL. The parameters
are passed directly as expected by the language, the SQLDA structure is not used. This can only be specified when LANGUAGE C or COBOL is used.

Null indicators are NOT directly passed to the program.

The value SIMPLE CALL may be used as a synonym for GENERAL.

GENERAL WITH NULLS

In addition to the parameters on the CALL statement as specified in GENERAL, another argument is passed to the stored procedure. This additional argument contains a vector of null indicators for each of the parameters on the CALL statement. In C, this would be an array of short ints. This can only be specified when LANGUAGE C or COBOL is used.

The value SIMPLE CALL WITH NULLS may be used as a synonym for GENERAL WITH NULLS.

DB2SQL

In addition to the parameters on the CALL statement, the following arguments are passed to the stored procedure:

- A NULL indicator for each parameter on the CALL statement
- The SQLSTATE to be returned to DB2
- The qualified name of the stored procedure
- The specific name of the stored procedure
- The SQL diagnostic string to be returned to DB2

This can only be specified when LANGUAGE C, COBOL or OLE is used.

JAVA

This means that the stored procedure will use a parameter passing convention that conforms to the Java language and SQLJ Routines specification. IN/OUT and OUT parameters will be passed as single entry arrays to facilitate returning values. This can only be specified when LANGUAGE JAVA is used.

PARAMETER STYLE JAVA procedures do not support the DBINFO or PROGRAM TYPE clauses.

In DB2 UDB/UNO, an additional style is allowed:

DB2DARI

This means that the stored procedure will use a parameter passing convention that conforms to C language calling and linkage conventions. This can only be specified when LANGUAGE C is used.
CREATE Syntax - PROGRAM TYPE

```
CREATE PROCEDURE ....
    ....
    PROGRAM TYPE SUB or MAIN
    LANGUAGE C or JAVA or COBOL or SQL
```

- PROGRAM TYPE SUB is the default.
- If LANGUAGE SQL is specified, then the SQL procedure follows.

**Notes:**

**PROGRAM TYPE (valid for z/OS and Linux, UNIX, and Windows**

- Specifies whether the stored procedure expects parameters in the style of a main routine or a subroutine.
- SUB - The stored procedure expects the parameters to be passed as separate arguments. This is the default value for DB2 for OS/390 and z/OS when the CURRENT RULES value is STD or LANGUAGE is REXX.
- MAIN - The stored procedure expects the parameters to be passed as an argument counter, and a vector of arguments (argc, argv). The name of the stored procedure to be invoked must also be main. Stored procedures of this type must still be built in the same fashion as a shared library as opposed to a stand-alone executable.
- LANGUAGE - This mandatory clause is used to specify the language interface convention to which the stored procedure body is written.
- C - The stored procedure must conform to the C language calling and linkage convention as defined by the standard ANSI C prototype.
JAVA - This means the database manager will call the stored procedure as a method in a Java class.

COBOL

SQL - The specified SQL-procedure-body includes the statements which define the processing of the stored procedure.

DB2 for Linux, UNIX, and Windows Only

OLE - This means the database manager will call the stored procedure as if it were a method exposed by an OLE automation object. The stored-procedure must conform with the OLE automation data types and invocation mechanism. Also, the OLE automation object needs to be implemented as an in-process server (DLL). These restrictions are outlined in the OLE Automation Programmer's Reference.

LANGUAGE OLE is only supported for stored procedures stored in DB2 for Windows 32-bit operating systems.

DB2 for iSeries Additional Language Support

C++
CL
COBOL
FORTRAN
PLI
REXX
RPG
RPGLE

DB2 for OS/390 and z/OS Additional Language Support

ASSEMBLE - Stored procedure is written in Assembler.
COBOL
COMPJAVA - Compiled Java
PLI
REXX

DB2 for OS/390 and z/OS Additional CREATE Parameters

PARAMETER CCSID - Sets code conversion to ASCII | EBCDIC | UNICODE.
COLLID - Allows override of the collection id to be used by the stored procedure.
WLM ENVIRONMENT - Identifies the WLM address space used to execute this stored procedure.
ASUTIME - Limits the amount of service to be consumed by this stored procedure.
STAY RESIDENT - Set to YES | NO. Directs DB2 to retain load module in address space even if use count drops to zero.

SECURITY - Determines how non-DB2 resource authorization is handled.

SECURITY - Determines how non-DB2 resource authorization is handled.

COMMIT ON RETURN - Set to YES | NO.

INHERIT | DEFAULT SPECIAL REGISTERS - Determines whether the stored procedure will inherit the special register values of the calling process. Now supported in Create SQL Procedure statement.

CALLED ON NULL INPUT - If specified, makes the stored procedure responsible for handling of NULL values on input parameters.
CREATE Syntax - FENCED TYPE

CREATE PROCEDURE ....
    ....
    NOT FENCED or
    FENCED
        THREADSAFE or
        NOT THREADSAFE

- FENCED is the default.
- THREADSAFE is the default for FENCED.

Notes:

NOT FENCED

A NOT FENCED routine runs in the same process as the database manager. In general, running your routine as NOT FENCED results in better performance as compared with running it in FENCED mode, because FENCED routines run in a special DB2 process outside of the engine’s memory.

While you can expect improved routine performance when running routines in NOT FENCED mode, user code can accidentally or maliciously corrupt the database or damage the database control structures.

For Java routines running on UNIX platforms, scalability may be an issue if NOT FENCED is specified. This is due to the nature of the DB2 UNIX process model, which is one process per agent. As a result, each invocation of a NOT FENCED Java routine will require its own JVM. This can result in poor scalability, because JVMs have a large memory footprint. Many invocations of NOT FENCED routines on a UNIX-based DB2 server will use a significant portion of system memory. This is not the case for Java
routines running on Windows NT and Windows 2000, where each DB2 agent is represented by a thread in a process shared with other DB2 agent threads.

**FENCED THREADSAFE**

FENCED THREADSAFE routines run in the same process as other routines. More specifically, non-Java routines share one process, while Java routines share another process, separate from routines written in other languages. This separation protects Java routines from the potentially more error prone routines written in other languages. Also, the process for Java routines contains a JVM, which incurs a high memory cost and is not used by other routine types. Multiple invocations of FENCED THREADSAFE routines share resources, and therefore incur less system overhead than FENCED NOT THREADSAFE routines, which each run in their own dedicated process.

If a FENCED THREADSAFE routine abends, only the thread running this routine is terminated. Other routines in the process continue running.

**Note:** THREADSAFE is the default if the LANGUAGE is JAVA. However it is not a valid option if LANGUAGE OLE is specified.

**FENCED NOT THREADSAFE**

FENCED NOT THREADSAFE routines each run in their own dedicated process. If you are running numerous routines, this can have a detrimental effect on database system performance. If the routine is not safe enough to run in the same process as other routines, use the NOT THREADSAFE clause when registering the routine.

On UNIX, NOT THREADSAFE processes appear as db2fmp (pid) (where pid is the process id of the agent using the fenced mode process) or as db2fmp (idle) for a pooled NOT THREADSAFE db2fmp.

If you intend to run a Java routine with large memory requirements, it is recommended that you register it as FENCED NOT THREADSAFE.

**Note:** NOT TREADSAFE is the default for all languages other than JAVA.
CREATE Syntax - Example 1

CREATE PROCEDURE PARTS_ON_HAND
    (IN PARTNUM INTEGER,
     OUT COST DECIMAL(7,2),
     OUT QUANTITY INTEGER)
    EXTERNAL NAME 'parts.onhand'
    LANGUAGE JAVA PARAMETER STYLE JAVA

- **Example 1**: Create the procedure definition for a stored procedure, written in Java, that is passed a part number and returns the cost of the part and the quantity that are currently available.

*Notes:*
CREATE Procedure ASSEMBLY_PARTS
(IN ASSEMBLY_NUM INTEGER,
  OUT NUM_PARTS INTEGER,
  OUT COST DOUBLE)
EXTERNAL NAME 'parts.assembly'
LANGUAGE C
DYNAMIC RESULT SETS 1

• Example 2: Create the procedure definition for a stored procedure, written in C, that is passed an assembly number and returns the number of parts that make up the assembly, total part cost, and a result set that lists the part numbers, quantity, and unit cost of each part.

Notes:
CREATE Syntax - Example 3

CREATE PROCEDURE MEDIAN_RESULT_SET
  (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
BEGIN
  ...
END

• Example 3: Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

Notes:

CREATE PROCEDURE MEDIAN_RESULT_SET
  (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
BEGIN
  DECLARE v_numRecords INT DEFAULT 1;
  DECLARE v_counter INT DEFAULT 0;

  DECLARE c1 CURSOR FOR
    SELECT CAST(salary AS DOUBLE)
    FROM staff
    ORDER BY salary;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, CAST(salary AS INTEGER)
    FROM staff
    WHERE salary > medianSalary
ORDER BY salary;
DECLARE EXIT HANDLER FOR NOT FOUND
  SET medianSalary = 6666;
SET medianSalary = 0;
SELECT COUNT(*) INTO v_numRecords
  FROM STAFF;
OPEN c1;
WHILE v_counter < (v_numRecords / 2 + 1)
  DO FETCH c1 INTO medianSalary;
  SET v_counter = v_counter + 1;
END WHILE;
CLOSE c1;
OPEN c2;
END
JAR Files

1. Create the Java function (stored procedure) as a Java method. Compile the Java source code into a Java class file.

   `sqlj..., db2profc..., javac MyProc.java`

2. Collect the class file(s) containing the Java function in a JAR file.

   `jar cvf SPJar.jar MyProc*.class`

3. Install the JAR file in the DB2 instance.

   `db2 call sqlj.install_jar
      ('file:\javasp\SPJar.jar', 'MyJar')`

Notes:

- Step one consists of preparing the Java program. This will vary depending on use of SQLJ or JDBC.
- Step two creates the archive file (JAR) containing all the compiled Java code.
- Step three identifies the JAR to the database manager.
- Step four is the DDL to register the stored procedure to the database manager.
CREATE Java PROCEDURE

```sql
CREATE PROCEDURE MYPROC (INOUT HOST1 CHAR(15),
   IN HOST2 DOUBLE, OUT HOST3 INTEGER)
   EXTERNAL NAME 'MyClass.MyProc2'
   LANGUAGE JAVA
   PARAMETER STYLE JAVA
   READS SQL DATA
```

Notes:

- This visual illustrates the minimum DDL for creating a Java stored procedure.
- The CALL statement is in SQLJ format, but the procedure could be called from any supported program.
- The parameters being passed on the call must be previously defined by the calling application and should match the data types in the CREATE parameter list.
**EXTERNAL NAME**

CREATE PROCEDURE `MYPROC` (...)

EXTERNAL NAME `'MyClass.MyProc2'`

EXTERNAL NAME `'MyJar.MyClass.MyProc'`

**Notes:**

The external name must provide either the JAR file or the compiled Java code in the correct directories listed above.
Checkpoint

Exercise — Unit 6 Checkpoint

__ 1. What data types are allowed in the definition of stored procedure parameters?
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

__ 2. The EXTERNAL NAME parameter of the CREATE PROCEDURE DDL statement identifies what object in the database manager's operating environment?
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

__ 3. If a value is specified for the EXTERNAL NAME parameter, what can be concluded about the value for the LANGUAGE parameter in a CREATE PROCEDURE DDL statement.
__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________
Unit Summary

Since completing this unit, you should be able to:

- Code the CREATE PROCEDURE DDL
- Describe EXTERNAL NAME
- Create JAR files

Notes:
Unit 7. DB2 Development Center Debug

What This Unit Is About

This unit will describe the DB2 Development Center support of stored procedure debugging using the IBM Distributed Debugger.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Describe the Development Center debug capabilities
• Describe platforms supported for debugging
• Set up Development Center procedures for debugging

How You Will Check Your Progress

Accountability:

• Checkpoint

References

SG24-5485  Cross-Platform DB2 Stored Procedures: Building and Debugging
SC09-4826  IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827  IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844  IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845  IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9932  DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java
SC26-9933  DB2 UDB for OS/390 and z/OS V7 Application Programming and SQL Guide
SC26-9944  DB2 UDB for OS/390 and z/OS V7 SQL Reference
Unit Objectives

After completing this unit, you should be able to:

- Describe the Development Center's stored procedure debug capabilities
- Describe platforms supported for debugging
- Set up procedures for debugging

Notes:
7.1 Development Center Debug
Platforms Supported

- **Debug support**
  - DB2 UDB (UNIX and Windows)
  - DB2 UDB for OS/390 and z/OS

- **IBM Distributed Debugger**
  - Executes in Windows 2000
  - Connected to a DB2 UDB server
  - Included with Java

- **IBM Debug Tool for OS/390**
  - SQL stored procedures on OS/390 and z/OS

---

**Notes:**

By using the remote debugging capabilities available for DB2, you can debug SQL and Java stored procedures as they execute on the DB2 server. From your client desktop, you can watch the source code and state of variables of the stored procedure as it executes. For debugging SQL stored procedures running on DB2 for Intel and UNIX, the debugging support is directly integrated into the DB2 Development Center. For debugging Java stored procedures running on DB2 for the Intel and UNIX platforms, you need the IBM Distributed Debugger (available separately as part of IBM VisualAge for Java). For debugging SQL stored procedures running on DB2 for OS/390 and z/OS, you need the IBM Distributed Debugger and the IBM Debug Tool for OS/390.
Build for Debug

Figure 7-3. Build for Debug

Notes:

Right-clicking the stored procedure name will present a drop-down menu that contains **Build for Debug**. Selecting **Build for Debug** will drop the stored procedure, if it has been previously built, and rebuild it in debug mode.

There is also the option, when creating the stored procedure using the wizard, to build the stored procedure and enable it for debugging. Although, since modifications are usually required to the generated stored procedure, this particular feature will not be used very often.
Debug Connect

Notes:
Clicking the **Project** menu item will allow access to Environment Settings window. Click **Debug** to specify various properties. The default Port setting is **8000**.
Debug Session

![Debug Session Image]

Notes:

This stored procedure, ADMIN.DebugSP, is an SQL procedure that was built for debug and executed. The current line to be executed is the highlighted line. Lines with breakpoints are displayed in the source and listed under Breakpoints.

- Breakpoints - The breakpoint, line 8, was set by clicking in the column to the left of the statement where the breakpoint was desired. Enabled breakpoints display as a solid dot. Disabled breakpoints display as a red ring.

- Call Stack - Since there is only one procedure involved in this process, only the one entry is found in the call stack.

- Variables - This panel shows the value of all variables as of the breakpoints encountered. At the time of this capture, some variables had not yet been initialized.
Debug Support

- Procedure source code is displayed
  - Step through program statements
- Monitoring of procedure variables
  - Display of variables
  - Setting of variables
  - Break if variable changes
- Breakpoint setting
  - Break at procedure line number
  - Breaks set by double-click
- Call Stack
  - View all call levels
  - Switch between procedures

Notes:

Beware that only certain statements in a stored procedure will support breakpoints. For example, if a statement spans multiple lines, then the breakpoint should be on the last line. The debugger will allow you to put a breakpoint anywhere. Only when the store procedure is being debugged will the validity of the breakpoint be checked.
Troubleshooting Checklist

☐ Check CALL statement.
   – Ensure procedure name used and variables passed agree with procedure declaration.

☐ Check Stored Procedure entry point.
   – Ensure class name, method name, and variable definition agree with procedure declaration.

☐ Check Stored Procedure creation statement.

☐ Ensure all class files and serialized profiles can be found by DB2:
   – In \sqllib\function, or
   – JAR file

☐ First found is executed, so if you’ve copied files to \sqllib\function and created JAR file, the copied files will be executed.

Figure 7-7. Troubleshooting Checklist

Notes:

The mismatch of parameters being passed to the stored procedure is one of the most common errors encountered when implementing stored procedures. Documentation should be clear as to the use of and the nature of the variables being used in the CALL to the stored procedures that you develop.
Debugging Stored Procedures

- Prepare to Debug.
  1. Compile stored procedure to debug code.
  2. Prepare the server.
  3. Set the client environment variables.
  4. Create the debug table.
  5. Start the debugger daemon on the client.

- Populate the debug table.

- Invoke the debugger.

OR

- Use the Development Center.

Notes:

Using the DB2 Development Center to debug stored procedures would allow debugging to be done prior to the design, coding, and preparation of the client programs. As stated earlier, the IBM Distributed Debugger is independent of whether you choose to use the Development Center, but the Development Center is recommended.
Checkpoint

Exercise — Unit 7 Checkpoint

__ 1. Which of the following is not true regarding the IBM Distributed Debugger:
   • Executes in Windows 2000
   • Connected to a DB2 UDB server
   • Not included with DB2 UDB

__ 2. What is the name of the table used by IBM Distributed Debugger?

__ 3. What major debugging functions are provided for stored procedures by the DB2 Distributed Debugger?
Unit Summary

Since completing this unit, you should be able to:

- Describe the Development Center's stored procedure debug capabilities
- Describe platforms supported for debugging
- Set up procedures for debugging

Notes:
Appendix A. Checkpoint Solutions

Unit 1 - DB2 Stored Procedures and Platform Considerations

__ 1. A stored procedure is an executable program under control of the Database Server.

__ 2. List three cases when a stored procedure may be justified.
   - Performance not being met.
   - Large number of distributed clients.
   - Heavy database access with little program execution.
   - Program changes often.
   - More control of program code is required.
   - Many SQL statements are executed.

__ 3. The same procedure name appears to be registered more than once with the DB2 database manager. What does this imply about the parameters?
   - The number of parameters must be different for each.

Unit 2 - DB2 Development Center

__ 1. List three DB2 Development Center functions that would aid application developers.
   - Create stored procedures
   - Build on local and remote DB2 server
   - Modify and rebuild existing stored procedures
   - Run stored procedures for testing and debugging

__ 2. List three ways of launching the DB2 Development Center.
   - IBM DB2 UDB Program Group
   - Any Microsoft Application Development Environment
   - IBM DB2 Control Center or any other DB2 Center

__ 3. IBM DB2 Development Center may be used to create stored procedures in which two languages?
   - SQL PL and Java

Unit 3 - DB2 SQL Procedure Language

__ 1. T/F The SQL Procedure Language is part of the DDF CREATE statement.
   - True
2. What would you find in a DECLARE statement within a stored procedure?

Any of the following may be DECLAREd in a stored procedure:
- Cursor declarations
- Handler declaration
- SQL variable declarations
- Condition declarations

3. How is an assignment made to a parameter or a local variable?

The SQL SET statement is used to make such assignments. For example, SET v-date = CURRENT DATE

Unit 4 - DB2 Development Center and Java

1. The DB2 Development Center only supports two languages within the DB2 Development Center: SQL Procedure Language and Java.

2. The DB2 Development Center will allow you to build a stored procedure in Java that will access the DB2 database manager using either SQLJ or JDBC.

3. What is the defining difference between SQLJ and JDBC?

SQLJ statements will be statically bound into a DB2 package. JDBC statements are dynamically bound at execution time. In most applications, this would result in a run time performance advantage for SQLJ.

Unit 5 - The CALL SQL Statement

1. T/F A stored procedure may not call another stored procedure.

False. A stored procedure may call another stored procedure in the current releases of DB2.

2. A DECLARE for a result set locator in a calling application would indicate that what clause was coded in the called stored procedure’s DECLARE CURSOR?

The WITH RETURN clause of the DECLARE CURSOR in the stored procedure may produce a result set if that cursor is opened and left open. The locator variable in the calling program is used to link that result set.

3. Is an OPEN cursor statement needed for processing a result set in the calling program?

No, the ALLOCATE statement performs the same function as the open cursor. Once, the result set is ALLOCATEd to the calling program, the calling program may FETCH from that cursor.
Unit 6 - The CREATE PROCEDURE SQL Statement

__ 1. What data types are allowed in the definition of stored procedure parameters?

   The data types supported are the same as those for a CREATE TABLE in DB2.

__ 2. The EXTERNAL NAME parameter of the CREATE PROCEDURE DDL statement identifies what object in the database manager's operating environment?

   The EXTERNAL NAME parameter identifies the executable module that is the stored procedure being identified to the database manager.

__ 3. If a value is specified for the EXTERNAL NAME parameter, what can be concluded about the value for the LANGUAGE parameter in a CREATE PROCEDURE DDL statement.

   The absence of the EXTERNAL NAME parameter means that the LANGUAGE must be SQL; that is, the procedure is written in SQL Procedure Language.

Unit 7 - DB2 Development Center and Debugging

__ 1. Which of the following is not true regarding the IBM Distributed Debugger:

   • Executes in Windows 2000
   • Connected to a DB2 UDB server
   • Not included with DB2 UDB

   Not included with DB2 UDB is false. The IBM Distributed Debugger is included with DB2 UDB.

__ 2. What is the name of the table used by IBM Distributed Debugger?

   The name of the table used by IBM Distributed Debugger to control debugging functions is DB2DBG.ROUTINE_DEBUG.

__ 3. What major debugging functions are provided for stored procedures by the DB2 Distributed Debugger?

   Procedure source code is displayed.
   Monitoring of procedure variables.
   Breakpoint setting.
Bibliography

Manuals

SC09-4826 IBM DB2 UDB Application Development Guide: Programming Client Applications V8
SC09-4827 IBM DB2 UDB Application Development Guide: Programming Server Applications V8
SC09-4844 IBM DB2 UDB SQL Reference Volume 1 V8
SC09-4845 IBM DB2 UDB SQL Reference Volume 2 V8
SC26-9004 DB2 UDB for OS/390 Version 6 Application Programming and SQL Guide
SC26-9014 DB2 UDB for OS/390 Version 6 SQL Reference
SC26-9018 DB2 UDB for OS/390 Version 6 Application Programming Guide and Reference for Java

Technical Bulletins

SG24-5485 Developing Cross-Platform DB2 Stored Procedures: SQL Procedures and the DB2 Stored Procedure Builder

WEB URLs

http://www.ibm.net IBM's Internet Connection Web site
http://www.isi.edu/rfc-editor/ Internet “Requests for Comments” Editor