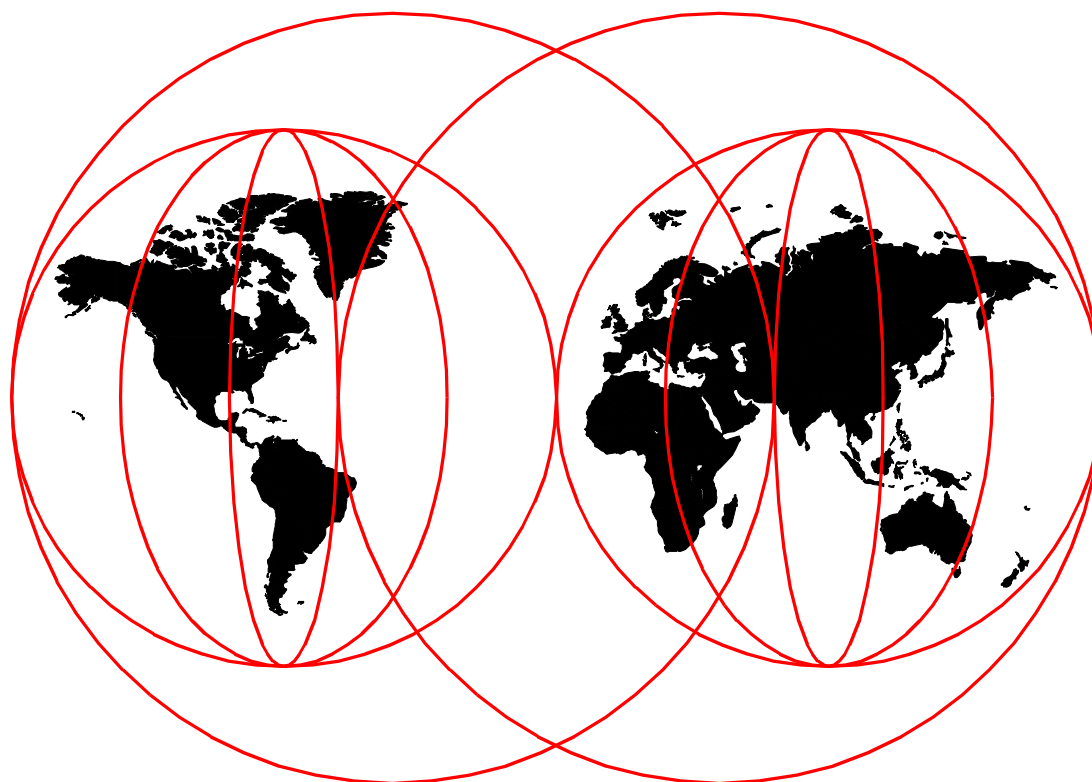


# Developing Cross-Platform DB2 Stored Procedures: SQL Procedures and the DB2 Stored Procedure Builder

*Patrick Dantressangle, Debra Eaton, Mark Leung, Ricardo D. Macedo, Ling Tay  
Maria Sueli Almeida, Jarek Mischczyk*



**International Technical Support Organization**

[www.redbooks.ibm.com](http://www.redbooks.ibm.com)





International Technical Support Organization

SG24-5485-00

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

November 1999

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix B, "Special notices" on page 223.

**First Edition (November 1999)**

This edition applies to DB2 for OS/390 Version 5 and Version 6 with applied service maintenance, for DB2 for AS/400 V4R2 and beyond, and for DB2 for UNIX, Windows, and OS/2 for the release after Version 6, and other current versions and releases of IBM products. Make sure you are using the correct edition for the level of the product. This edition is based on the latest beta version of SQL Procedures language support and IBM Stored Procedure Builder.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization  
Dept. QXXE Building 80-E2  
650 Harry Road  
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

**© Copyright International Business Machines Corporation 1999. All rights reserved.**

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

<b>Figures</b> .....	vii
<b>Tables</b> .....	ix
<b>Preface</b> .....	xi
The team that wrote this redbook .....	xi
Comments welcome .....	xiii
<b>Chapter 1. Introduction</b> .....	1
1.1 DB2 stored procedures — evolution .....	1
1.2 Cross-platform support of stored procedures written entirely in SQL .....	6
1.3 Building DB2 stored procedures from your workstation .....	6
1.4 Concepts and terminology .....	7
<b>Chapter 2. The SQL Procedures language</b> .....	9
2.1 What is it? .....	9
2.2 Planning to use the SQL Procedures language .....	10
2.2.1 Why use it? .....	10
2.2.2 When to use them? .....	13
2.3 Comparing SQL stored procedures and external stored procedures .....	16
2.3.1 Development .....	16
2.3.2 Runtime .....	17
2.4 Current implementation of SQL Procedures language .....	17
2.4.1 How does this work in general? .....	18
2.4.2 Declaring SQL local variables .....	19
2.4.3 Language Elements .....	20
2.4.4 Returning result sets .....	29
2.4.5 Handling errors in an SQL stored procedure .....	33
2.4.6 Current restrictions .....	36
2.5 SQL Procedures portability .....	36
2.6 New error messages .....	39
2.7 Migrating from OEM DBMS .....	40
2.7.1 Migrating the database structure .....	40
2.7.2 Migrating the database data .....	41
2.7.3 Migrating the business logic .....	41
2.7.4 Comparison with Sybase/Microsoft SQL Server Transact-SQL .....	43
2.7.5 Comparison with Oracle PL/SQL .....	48
2.7.6 Comparison with Informix SPL .....	55
<b>Chapter 3. The DB2 Stored Procedure Builder</b> .....	57
3.1 DB2 Stored Procedure Builder — overview .....	57
3.1.1 What is it? .....	57
3.1.2 Programming languages supported .....	58
3.2 Product Installation on Windows NT .....	60
3.2.1 Prerequisites for SPB .....	60
3.2.2 Installing the SPB .....	61
3.3 Advanced configuring of the SPB .....	65
3.3.1 Concepts and terminology .....	75
3.3.2 What are its components? .....	77
3.3.3 Working with SPB projects .....	79
3.4 Using the Stored Procedure Builder .....	83

3.4.1	Viewing existing stored procedures . . . . .	83
3.4.2	Creating new stored procedures . . . . .	85
3.4.3	Building stored procedures . . . . .	101
3.4.4	Modifying existing stored procedures . . . . .	102
3.4.5	Copying and pasting stored procedures across connections . . . . .	104
3.4.6	Debugging stored procedures . . . . .	105
<b>Chapter 4.</b>	<b>SQL Procedures for DB2 UDB for OS/390 . . . . .</b>	<b>109</b>
4.1	General considerations . . . . .	109
4.2	System requirements and planning . . . . .	109
4.2.1	Requirements for DB2 for OS/390 Version 5 . . . . .	109
4.2.2	Requirements for DB2 UDB for OS/390 Version 6 . . . . .	111
4.2.3	Remote Debugger and Debug tool . . . . .	112
4.2.4	Creating non-catalog DB2 tables . . . . .	112
4.2.5	WLM requirements for OS/390 Procedure Processor . . . . .	114
4.3	Coding considerations . . . . .	115
4.3.1	Length and size limits . . . . .	115
4.3.2	Parameters and variables . . . . .	115
4.3.3	Handling SQLCODE and SQLSTATE values . . . . .	117
4.3.4	SQL statements . . . . .	117
4.3.5	Client application . . . . .	118
4.4	Stored procedure preparation . . . . .	118
4.4.1	Process . . . . .	119
4.4.2	Authorization . . . . .	120
4.5	Setting up DSNTPSMP . . . . .	120
4.5.1	Using the SPB . . . . .	122
4.5.2	Using OS/390 Procedure Processor (DSNTPSMP) . . . . .	125
4.5.3	Using JCL . . . . .	136
4.6	Stored procedure debugging . . . . .	142
4.6.1	Process . . . . .	143
4.6.2	If the debugger does not start . . . . .	143
<b>Chapter 5.</b>	<b>SQL Procedures for DB2 UDB for UNIX, Windows, OS/2 . . . . .</b>	<b>145</b>
5.1	General considerations . . . . .	145
5.2	Supported platforms . . . . .	146
5.3	System requirements and planning . . . . .	146
5.3.1	Requirements for the Windows NT platform . . . . .	146
5.3.2	Requirements for the UNIX platform . . . . .	148
5.3.3	Changing compiler options . . . . .	149
5.3.4	Retaining intermediate files . . . . .	150
5.4	Coding considerations . . . . .	151
5.4.1	Recommendations for writing portable stored procedures . . . . .	151
5.4.2	Structure of SQL stored procedures . . . . .	151
5.4.3	Coding the SQL stored procedures body . . . . .	152
5.5	Stored procedures preparation . . . . .	159
5.5.1	Privileges required to prepare an SQL stored procedure . . . . .	161
5.5.2	Preparing an SQL stored procedure from the DB2 CLP . . . . .	162
5.5.3	Preparing an SQL stored procedure from the DB2 tools . . . . .	162
5.5.4	Preparing an SQL stored procedure from application programs . . . . .	164
5.5.5	Preparing an SQL stored procedure from the SPB . . . . .	164
5.5.6	Copying SQL stored procedures between DB2 UDB servers . . . . .	165
5.6	Stored procedure debugging . . . . .	166
5.6.1	Platforms supported for remote debugging . . . . .	166

5.6.2	The DB2DBG.ROUTINE_DEBUG debugger table . . . . .	166
5.6.3	DB2 environment variables for debugging . . . . .	167
5.6.4	Starting the debugger client . . . . .	168
5.6.5	Debugging stored procedures through SPB. . . . .	168
<b>Chapter 6. SQL Procedures for DB2 UDB for AS/400 . . . . .</b>		
6.1	General Considerations . . . . .	169
6.2	System requirements and planning . . . . .	169
6.3	System Catalog Tables . . . . .	169
6.4	Creating an SQL stored procedure . . . . .	170
6.4.1	Creating an SQL SP with traditional tools . . . . .	170
6.4.2	Creating an SQL SP with Operations Navigator GUI . . . . .	174
6.4.3	Creating an SQL SP with the Run SQL Scripts utility. . . . .	176
6.4.4	Verifying the stored procedure properties . . . . .	179
6.5	Deleting or replacing the SQL stored procedure . . . . .	180
6.6	Debugging SQL stored procedures . . . . .	181
6.6.1	The ILE Source Debugger. . . . .	181
6.6.2	Preparing the SQL stored procedure for debugging. . . . .	182
6.6.3	Testing the SQL stored procedure in traditional environment . . . . .	184
6.6.4	Testing the SQL stored procedure in client/server environment. . . . .	188
<b>Appendix A. Sample SQL stored procedure programs . . . . .</b>		
A.1	Naming convention . . . . .	195
A.2	OS/390 samples . . . . .	195
A.2.1	DSN8ES1 . . . . .	195
A.2.2	SDK0LMS . . . . .	197
A.2.3	SDK1LMS . . . . .	198
A.2.4	SDK2LMS . . . . .	199
A.2.5	SDK3LMS . . . . .	199
A.2.6	SDK4LMS . . . . .	200
A.2.7	SDK5LMS . . . . .	201
A.2.8	SDK6LMS . . . . .	201
A.2.9	SDK7LMS . . . . .	202
A.2.10	SDK8LMS . . . . .	203
A.2.11	SDK9LMS . . . . .	204
A.2.12	SMP0LMS . . . . .	204
A.2.13	SMP1LMS . . . . .	205
A.2.14	SMP2LMS . . . . .	205
A.2.15	SMP3LMS . . . . .	206
A.2.16	SMP4LMS . . . . .	206
A.2.17	SMP5LMS . . . . .	207
A.2.18	SMP5LMS2 . . . . .	207
A.2.19	SMP7LMS . . . . .	207
A.2.20	SMP8LMS . . . . .	208
A.2.21	SMP8LMS2 . . . . .	209
A.3	NT and AIX samples . . . . .	209
A.3.1	SDK0LNS . . . . .	209
A.3.2	SDK1LNS . . . . .	210
A.3.3	SDK2LNS . . . . .	211
A.3.4	SDK3LNS . . . . .	211
A.3.5	SDK4LNS . . . . .	212
A.3.6	SDK5LNS . . . . .	213
A.3.7	SDK6LNS . . . . .	213

A.3.8	SDK7LNS	214
A.3.9	SDK8LNS	215
A.3.10	SDK9LNS	216
A.3.11	SDKALNS	216
A.3.12	SMP1LNS	217
A.3.13	SMP2LNS	218
A.3.14	SMP3LNS	218
A.3.15	SMP4LNS	219
A.3.16	SMP5LNS	219
A.3.17	SMP7LNS	219
A.3.18	SMP8LNS	220
A.3.19	SMP9LNS	220
A.3.20	SMPALNS	221
<b>Appendix B. Special notices</b>		223
<b>Appendix C. Related publications</b>		227
C.1	International Technical Support Organization publications	227
C.2	Redbooks on CD-ROMs	227
C.3	Other publications	227
<b>How to get ITSO redbooks</b>		229
	IBM Redbook Fax Order Form	230
<b>List of abbreviations</b>		231
<b>Index</b>		233
<b>ITSO redbook evaluation</b>		237



---

## Figures

1. The usual SQL client, where the logic sends many SQL queries . . . . .	12
2. SQL client using the same logic, but within an SQL stored procedure. . . . .	12
3. An example of shielding tables from users . . . . .	13
4. Behavior summary of different condition handlers . . . . .	34
5. SPB environment . . . . .	57
6. Invoking SPB through VisualAge for Java . . . . .	62
7. Specify Database Connection Window . . . . .	63
8. SPB main frame invoked through VA Java Workbench. . . . .	64
9. Customizing Microsoft Visual Studio . . . . .	65
10. Stored Procedure Builder . . . . .	67
11. SPB: Previous Projects . . . . .	68
12. Environment Properties: Connection . . . . .	68
13. Environment Properties: Editor . . . . .	69
14. Environment Properties: Assistance . . . . .	69
15. Environment Properties: Output . . . . .	70
16. Environment Properties: Debug . . . . .	70
17. Environment Properties: OS/390 Options . . . . .	71
18. Environment Properties: SQL Types . . . . .	71
19. Environment Properties: Type Mapping. . . . .	72
20. SPB main panel . . . . .	77
21. The New Stored Procedures SmartGuide . . . . .	78
22. SQL Assistant window . . . . .	79
23. SPB projects (*.spp), connections and stored procedures. . . . .	80
24. Creating new project "ITSO SG245485" . . . . .	81
25. Tree view of existing stored procedures . . . . .	83
26. Detailed view of existing stored procedures. . . . .	84
27. Filtering the list of stored procedures . . . . .	84
28. The Filter dialog window . . . . .	85
29. Creating a new SQL stored procedure. . . . .	86
30. The Name panel of the New Stored Procedures SmartGuide . . . . .	87
31. The Pattern panel of the New Stored Procedures SmartGuide . . . . .	88
32. The SQL Query panel of the New Stored Procedures SmartGuide . . . . .	90
33. The Parameters panel of the New Stored Procedures SmartGuide . . . . .	91
34. The Define Parameter dialog . . . . .	91
35. The Options panel of the New Stored Procedures SmartGuide . . . . .	92
36. The Advanced options for DB2 for OS/390 SQL stored procedures . . . . .	93
37. The Advanced build options for DB2 for OS/390 SQL stored procedures. . . . .	94
38. The Tables panel of the SQL Assistant . . . . .	95
39. The Join panel of the SQL Assistant . . . . .	96
40. Changing the type of Join created by SQL Assistant. . . . .	96
41. The Conditions panel of the SQL Assistant . . . . .	97
42. Specifying a variable for a condition . . . . .	97
43. The Columns panel of the SQL Assistant . . . . .	98
44. The Sort panel of the SQL Assistant . . . . .	99
45. The SQL panel of the SQL Assistant . . . . .	100
46. Entering values for variables in the SQL statement. . . . .	101
47. Displaying the results of the SQL Statement . . . . .	101
48. Modifying an existing stored procedure . . . . .	103
49. Copying one SQL stored procedure. . . . .	104
50. Paste the stored procedure at the target server. . . . .	105

51. Debugging process . . . . .	106
52. IBM Distributed Debugger daemon . . . . .	106
53. IBM Distributed Debugger main window . . . . .	107
54. Monitoring variables . . . . .	108
55. Multiple WLM environments . . . . .	114
56. Three methods for preparing SQL stored procedures . . . . .	119
57. Build Name field on OS/390 Options . . . . .	122
58. OS/390 Options from SPB — 1/2 . . . . .	123
59. OS/390 Options from SPB — 2/2 . . . . .	123
60. SQL Costing Information panel . . . . .	125
61. Input / Output for SQL Procedures Processor . . . . .	126
62. The BUILD process . . . . .	130
63. The DESTROY process . . . . .	131
64. DSNHSQL process . . . . .	137
65. Using the same name for variables and parameters . . . . .	154
66. Using variables with the same name as a column . . . . .	154
67. Assigning special registers to variables . . . . .	155
68. Assigning results of built-in functions to variables . . . . .	156
69. Setting both SQLCODE and SQLSTATE variables to program variables . . . . .	157
70. Nested compound statement . . . . .	157
71. Setting a SAVEPOINT . . . . .	158
72. Ways to create SQL stored procedures in DB2 UDB . . . . .	160
73. Preparation steps for SQL stored procedures in DB2 UDB . . . . .	161
74. STP.DB2 file containing SQL stored procedures using \$ as a delimiter . . . . .	162
75. Changing the terminating character for the DB2 Command Center . . . . .	163
76. Changing the termination character for DB2 tools . . . . .	164
77. Using SPB to prepare SQL stored procedures . . . . .	165
78. Entering source code . . . . .	172
79. Creating the SQL stored procedure . . . . .	172
80. Working with spool files . . . . .	173
81. Displaying SQL precompiler error messages . . . . .	174
82. Parameters definition for SQL stored procedure . . . . .	175
83. Entering SQL statements . . . . .	176
84. Creating SQL SP with script utility . . . . .	177
85. Job log window . . . . .	178
86. Saving SQL Script File . . . . .	179
87. Displaying the stored procedure properties . . . . .	180
88. Deleting a stored procedure . . . . .	181
89. RUNSQLSTM command . . . . .	183
90. Specifying the DBGVIEW and OUTPUT parameters . . . . .	184
91. Starting a debug session . . . . .	185
92. Debug session . . . . .	186
93. INVDSK2LMS source code . . . . .	187
94. Running Java client . . . . .	188
95. Finding the database server job . . . . .	189
96. Job log for a database server job . . . . .	189
97. Calling the stored procedure from Java . . . . .	190
98. Naming convention for samples . . . . .	195

---

## Tables

1. SQL Procedures portability across DB2 platforms . . . . .	36
2. RDBMS Stored Procedure language comparison . . . . .	42
3. Comparison between SQL/PSM and T/SQL control statements . . . . .	44
4. Comparison between DB2 and Sybase SQL Server . . . . .	45
5. Comparison between SQL/PSM and PL/SQL control statements . . . . .	49
6. Comparison between DB2 and Oracle . . . . .	50
7. Comparison between SQL/PSM and SPL control statements . . . . .	55
8. DB2SPB.INI file sections and keywords . . . . .	72
9. SYSIBM.SYSPSM . . . . .	113
10. SYSIBM.SYSPSMOPTS . . . . .	113
11. SYSIBM.SYSPSMOUT . . . . .	114



---

## Preface

This redbook is intended for DB2 application developers who are familiar with Structured Query Language (SQL) and stored procedures, and who want to learn about developing stored procedures in the SQL Procedures language, as well as using and exploring the Stored Procedure Builder (SPB). Our discussion particularly applies to Version 6 of DB2 Universal Database Server for OS/390, for AS/400, and for distributed platforms, Version 5 of DB2 Server for OS/390, and other current versions and releases of IBM products.

First, we present the evolution of the IBM stored procedures support, describing in detail the new stored procedures language, SQL Procedures; and the new tool, Stored Procedure Builder.

In addition, we cover the implementation of these new features across platforms such as OS/390, Windows, and UNIX. The sample SQL stored procedures programs implemented during this project are documented in detail. Most of those samples are delivered with the SPB product. The sample SQL stored procedures programs illustrate the theory discussed in this redbook. These programs are useful for getting started with the SQL Procedures language in your own environment and gaining some hands-on experience on whatever platform you may have.

The support for the SQL Procedures language provides the customer with the facility for developing their stored procedures in a standard and portable language across the DB2 family and OEM DBMSs.

---

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

**Patrick Dantressangle** is an advisory software engineer working for IBM Santa Teresa Laboratory, San Jose, CA. He joined IBM in 1996 to work for the Net.Data development team, where he modified Net.Data V2 to perform flawlessly for all the 645.9 millions hits of the Nagano Winter Olympic Games Web server. Since 1998, he has been involved in DB2 UDB Stored Procedures enhancements. Since 1999, he has been working with Toronto teams on SQL Procedures implementation on DB2 UDB distributed platform version 6.1 and version 7. He has 15 years of experience in different fields such as client/server project management and application development, high performance Web development, and security software (smart cards).

**Debra Eaton** is a Field Technology Sales Specialist working at the IBM Software Migration Project Office in Chicago, IL, USA. She has 10 years of experience in the application development field. She has worked at IBM for 10 years. Her areas of expertise include database management and application development. She has written extensively on migrating from non-DB2 databases to DB2 databases.

**Mark Leung** is a systems specialist in Australia. He has 11 years of experience in the OS/390 field, and has worked at IBM for 11 years. His areas of expertise include application performance testing and DB2 connectivity. He is also a co-author of the redbook, *Getting Started with DB2 Stored Procedures*, SG24-4693.

**Ricardo Darriba Macedo** is a Senior DB2 Product Specialist, working at the IBM Software Business Unit, in Rio de Janeiro, Brazil. He joined IBM in 1987, and since then has been responsible for supporting customers in the database and application development areas. Ricardo has helped implement DB2 for many large customers in Brazil. He is also a co-author of the redbooks *Getting Started with DB2 Stored Procedures*, SG24-4693 and *DB2 DRDA Supports TCP/IP*, SG24-2212.

**Ling Tay** is DB2 technical support in Australia. She has 6 years of experience in DB2, mainly working on the OS/390 platform. Her areas of expertise are in application development, as well as DB2 system and application tuning.

**Maria Sueli Almeida** is a Certified I/T Specialist - Systems Enterprise Data, and is currently a DB2 for OS/390 and Distributed Relational Database System (DRDS) specialist at the International Technical Support Organization, San Jose Center. Before joining the ITSO in 1998, Maria Sueli worked at IBM Brazil assisting customers and IBM technical professionals on DB2, data sharing, database design, performance, and DRDA connectivity.

**Jarek Miszczyk** is an international Technical Support Organization specialist for the AS/400 system at the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide on all areas of AS/400 database. Before joining the ITSO, he worked in IBM Poland as a Systems Engineer and AS/400 Sales Specialist. He has over 10 years experience in the computer field and his areas of expertise include cross-platform database programming, SQL, and object-oriented (OO) programming.

Thanks to the following people for their invaluable contributions to this project:

Bob Carr  
Thomas Eng  
Marion Farber  
Gerry Fisher  
Greg Kim  
Susan Malaika  
Rick Mandel  
Bruce McAlister  
Claire McFeely  
Jessica Mignone  
Katherine A Morgan  
Connie Nelin  
Eugene Phu  
Marichu Scanlon  
Judy Tobias  
Ronald Trueblood  
Dirk Wollscheid  
IBM Santa Teresa Laboratory

Gustavo Arocena  
Serge Boivin  
Judy Chan  
Serge Rielau  
Dan Scott  
IBM Toronto Laboratory

Mark Anderson  
Kathy Passe  
IBM Rochester

Luca Montini  
IBM Italy

Paolo Bruni  
Joerg Reinschmidt  
IBM International Technical Support Organization, San Jose Center

---

## Comments welcome

### **Your comments are important to us!**

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO redbook evaluation” on page 237 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)





---

## Chapter 1. Introduction

In this chapter we discuss the evolution of DB2 cross-platform stored procedures support, leading up to the delivery of these two new features: SQL Procedures language and the Stored Procedure Builder — which are the main content of this redbook.

---

### 1.1 DB2 stored procedures — evolution

Stored procedures are user-written programs that are stored at the database server and can be invoked by client applications using SQL statements.

Stored procedures are predefined processes that execute the DB2 server side of applications. They can be called locally, on the same system where the application runs, or remotely from a different system. The SQL CALL statement is used to invoke a stored procedure. A stored procedure can send and/or receive parameters from the calling program.

The main goal of the stored procedure is the reduction of network message flow between the requester (the application) and the data server (DB2) in a distributed environment. Another usage of stored procedures is to have functions callable from any application on any platform.

Your programming productivity can be improved using stored procedures when you develop client/server applications. Stored procedures are the easiest way to perform a remote call and to distribute the logic of an application program.

Using stored procedures gives two main advantages to Information Technology departments:

- The application development department needs to maintain only one source of each function, even if applications calling the functions are executed in a different environment, such as transactional, distributed, batch, or distributed access. Source languages can be mixed between the calling application and the stored procedures, which permits the applications designer to choose the right language at the right place.
- Management rules are enforced by the centralization and uniqueness of function using stored procedures. In case of new application development, the developer can call stored procedures already developed easily without the burden of integrating these functions into the new source.

Since the stored procedure support by the DB2 family has been available, it has been enhanced to meet the pace of our customers and business. One of the most important enhancements is the support for a new stored procedure programming language: SQL Procedures. This support allows SQL-only stored procedures based on the ISO/ANSI standard SQL/PSM. This makes it possible to port the same stored procedure to any member of the DB2 family. It also simplifies the process of migrating stored procedures between other DBMSs and the DB2 family.

The implementation of SQL stored procedures is based on the SQL standard, and supports constructs that are common to most programming languages. This support has been available on DB2 UDB for AS/400 for more than one year, and now has been deployed for other members of the DB2 family, which will be covered in this book.

This is part of what has been known as SQL 3, and will shortly be called SQL 99 as it becomes a formal international standard. DB2 has adopted this SQL Procedures language from the SQL standard. We believe that customers will find value in using a standard compliant stored procedures language rather than proprietary stored procedures languages invented by other database vendors.

It is important to emphasize that the Database Language SQL - Part 4: Persistent Stored Modules of ISO/IEC 9075 specifies the syntax and semantics of a database language for declaring and maintaining persistent database language routines in SQL-server modules. The scope of the current implementation is limited to SQL Procedures and does not include support for SQL functions and Feature P01, "Stored Modules". See 2.4, "Current implementation of SQL Procedures language" on page 17.

The following sections describe the evolution of the stored procedure support by the DB2 family.

#### ***DB2 for OS/390***

Stored procedures support was introduced in DB2 for OS/390, Version 4. In this first deployment, stored procedures support was focused on application programming; for example:

- Improving the application security
  - Sensitive business logic run on DB2 server
  - End users do not need to have table privileges
- Improving application maintenance
  - Business logic is centralized
  - Online changes to application code
  - Client systems not sensitive to underlying tables
- Good integration with desktop tools
- Improving performance

In DB2 for OS/390 Version 5, stored procedures take advantage of the DRDA related enhancements introduced in this version of DB2, such as: native TCP/IP support, which improves the connectivity for workstation users; direct connection to DB2 for Windows through DB2 Connect for Windows; and many other performance improvements. From the stored procedures point of view, the main enhancements in this version were:

- Returning one or more query result sets
- Multiple DB2 stored procedure address spaces, managed by OS/390 Workload Manager (WLM)
- Ability to invoke utilities from a stored procedure, which means you can invoke utilities from an application that uses the SQL CALL statement

- Support to IMS Open Database Access (ODBA), which means that a DB2 stored procedure can directly connect to IMS DBCTL and access IMS data.

DB2 for OS/390 Version 6 deploys considerable enhancements for stored procedures, such as:

- SQL capability
  - CREATE and ALTER SQL statements and enhancements to the DROP statement for creating, modifying, and deleting stored procedure definitions
  - GRANT and REVOKE SQL statements to manage execution privileges for stored procedures
  - CURRENT PATH special register and PATH bind option to implicitly qualify stored procedures names in CALL statements
- Improved data transfer, allowing the DRDA server to send multiple DRDA query blocks
- Support for nested calls for stored procedures, so a stored procedure can call another stored procedure
- CALL SQL statement embedded in application programs or dynamically invoked from IBM's ODBC and CLI drivers
- Support for new data types on CALL SQL statement

### ***DB2 for UNIX, Windows, and OS/2***

The history of DB2 on the PC and UNIX platforms starts with DB2 Common Server Version 2, then DB2 Universal Database Version 5, and then DB2 Universal Database Version 6. The following paragraphs describe the development of stored procedures for DB2 Universal Database Version 6.

Below is a list of the stored procedures features supported in DB2 Common Server, Version 2:

- The external stored procedure programming technique can be used for database manager applications running in a client/server environment.
- Stored procedures invoked through DB2 CLI provide the capability to return one or more result sets to the client applications.
- The stored procedure must be written in one of the supported languages (C, C++, COBOL, Fortran, REXX) for that database server.
- A special table, DB2CLI.PROCEDURES (a pseudo-catalog table), which is defined by DB2, lists and describes the available stored procedures, along with the associated parameters of those stored procedures.
- Stored procedures stored at the location of the database are invoked from the client application via the SQL CALL statement.
- The SQL CALL statement can accept a series of host variables or an SQLDA structure.
- The SQL\_API\_FN macro is required when you write stored procedures.
- The client procedure should set the indicator for output-only SQLVARs to -1.
- The server procedure should set the indicator for input-only SQLVARs to -128.

Below is a list of the stored procedures features supported in DB2 Universal Database, Version 5 (see *DB2 UDB Version 5 Embedded SQL Programming Guide*, S10J-8158):

- The external stored procedure programming technique can be used for database manager applications running in a client/server environment.
- Stored procedures invoked through DB2 CLI provide the capability to return one or more result sets to the client applications.
- The stored procedure must be written in one of the supported languages (C, C++, Java, COBOL, Fortran, REXX) for that database server.
- When creating a stored procedure in the Java language, the CREATE PROCEDURE statement is used to register the procedure to the system catalog table SYSCAT.PROCEDURES.
- The SYSCAT.PROCEDURES table is defined by DB2, and it lists and describes available stored procedures, along with the associated parameters of those stored procedures.
- Stored procedures stored at the location of the database are invoked from the client application via the SQL CALL statement.
- The SQL CALL statement can accept a series of host variables or an SQLDA structure.
- The SQL\_API\_FN macro is required when you write stored procedures.
- The client procedure should set the indicator for output-only SQLVARs to -1.
- The server procedure should set the indicator for input-only SQLVARs to -128.

Below is a list of the stored procedures features supported in DB2 Universal Database, Version 6 (see *DB2 UDB Version 6 Application Development Guide Embedded SQL*, SC09-2845):

- The external stored procedure and SQL stored procedure programming techniques can be used for database manager applications running in a client/server environment.
- Stored procedures invoked through DB2 CLI, ODBC, JDBC and SQLJ clients provide the capability to return one or more result sets to the client applications.
- The stored procedure must be written in one of the supported languages (C, C++, Java, COBOL, Fortran, REXX) for that database server.
- Stored procedures stored at the location of the database are invoked from the client application via the SQL CALL statement.
- The parameter style with which you register the stored procedure in the database manager with the CREATE PROCEDURE statement determines how the stored procedure receives data from the client application. The parameter styles are GENERAL, GENERAL WITH NULLS, JAVA, DB2SQL, DB2DARI and DB2GENERAL.
- For LANGUAGE C stored procedures with a PARAMETER TYPE of GENERAL, GENERAL WITHNULLS, or DB2SQL, you have the option of writing your stored procedure to accept parameters like a main function in a C program (MAIN) or like a subroutine (SUB).

- You must declare every parameter passed from a client application to the stored procedure, and from the stored procedure back to the client application, as either an IN, OUT, or INOUT parameter.
- For Fortran or REXX stored procedures, you must write the stored procedure as a DB2DARI stored procedure.
- The SQL Procedures language is used to create the SQL stored procedure CREATE PROCEDURE statement with a procedure body.
- The two ways to create an SQL stored procedure are to use the IBM DB2 Stored Procedure Builder or to write a CREATE PROCEDURE statement for the SQL stored procedure manually.
- The SQL CALL statement can accept a series of host variables or an SQLDA structure.
- The SQL\_API\_FN macro is required when you write C/C++ stored procedures.
- The CREATE PROCEDURE statement is used to register the procedure to the system catalog table SYSCAT.PROCEDURES.
- A SYSCAT.PROCEDURES table, is defined by DB2, and lists and describes available stored procedures, along with the associated parameters of those stored procedures.
- The C/C++ client procedure should set the indicator for output-only SQLVARs to -1.
- The C/C++ parameter style DB2DARI server procedure should set the indicator for input-only to -128.

### ***DB2 UDB for AS/400***

From the AS/400 point of view, there are two ways to implement stored procedures:

- External stored procedures

When stored procedure support was delivered in V3R1, the SQL standards for procedural extensions were still unclear and not well defined. Thus, DB2 for AS/400 first delivered support for external stored procedures. An external stored procedure can be written in any high level language available on the AS/400 platform, including CLI and REXX. This approach gives you the flexibility to use a programming language you are most comfortable with. The external stored procedure may contain SQL statements (embedded SQL), or it may perform only native access to the database. The following steps illustrate the typical scenario for using external stored procedure in your client SQL application:

- Code your business logic in the high level language of your choice.
- Define the store procedure through the DECLARE PROCEDURE or CREATE PROCEDURE SQL statement.
- Invoke the stored procedure using the SQL CALL statement, passing parameters.
- Check the completion status of the stored procedure.

- SQL stored procedures:

In the years that have passed since V3R1, the SQL standards have matured, and now include an entire addendum defining the procedural language extensions for SQL. Starting with V4R2, the AS/400 programmers have the option of writing an entire program in SQL. Some SQL programmers new to the AS/400, were not comfortable learning how to use AS/400 compilers and how to embed SQL in a C program. The SQL procedural language gives them a way to produce DB2 for AS/400 stored procedures without having to learn these system-specific operations.

The SQL procedural language also makes it easier to port stored procedures from other databases to the AS/400. For example, Oracle and Microsoft have created their own proprietary languages (PL/SQL and T/SQL) for SQL stored procedures.

The SQL procedural language should not be considered a new language for the AS/400 like RPG or COBOL. Stored Procedures can be leveraged the most in network computing environments where processing is divided between the client and the server. They provide an easy way to package related database operations into a single object to reduce network traffic.

To create a SQL stored procedure on the AS/400 system you can use either native interface using the RUNSQLSTM command or the GUI interface provided by the Operations Navigator.

---

## 1.2 Cross-platform support of stored procedures written entirely in SQL

With SQL Procedures, you can now write stored procedures consisting entirely of SQL statements. SQL Procedures functionality has been supported by DB2 UDB for AS/400 for more than one year, and support has recently been rolled out across all members of the DB2 Universal Database Family, as well as to DB2 Server for OS/390 Version 5.

SQL Procedures functionality provides you the benefit of writing stored procedures in a standard, portable language. An SQL stored procedure consists of a CREATE PROCEDURE statement to define the procedure and a single or compound SQL statement. A compound SQL statement can include declarations (of variables, conditions, cursors, and handlers), flow control, assignment statements, and traditional SQL for defining and manipulating relational data. These extensions provide a procedural language for writing stored procedures, and they are consistent with the Persistent Stored Modules (PSM) portion of the SQL standard.

---

## 1.3 Building DB2 stored procedures from your workstation

The IBM DB2 Stored Procedure Builder (SPB) provides an easy-to-use development environment for creating, installing, and testing stored procedures. With the DB2 SPB, you can focus on creating your stored procedure logic rather than on the details of registering, building, and installing stored procedures on a DB2 server.

With DB2 Stored Procedure Builder, you can develop stored procedures on one operating system, such as Windows NT, Windows 98, or Windows 95, and deploy it on any DB2 platform on any operating system that supports DB2, such as DB2 for

AIX, DB2 for Sun Solaris, or DB2 for OS/390.

The Stored Procedure Builder allows you to create stored procedures in Java (dynamic SQL through JDBC support, or Static SQL through SQLJ support) and the SQL Procedures language. Creating stored procedures in Java and entirely in SQL produces stored procedures that are highly portable among operating systems.

Using the Stored Procedure Builder, you can perform a variety of tasks that are associated with stored procedures, such as:

- Viewing existing stored procedures
- Modifying existing stored procedures
- Creating new stored procedures
- Running existing stored procedures
- Copying and pasting stored procedures across connections
- One-step building of stored procedures on target databases
- Customizing the settings to enable remote debugging of installed stored procedures

---

## 1.4 Concepts and terminology

This section describes the terms and concepts used during the development of this book.

### ***Stored procedure, or external stored procedure***

This is a program developed in embedded SQL or in CLI, using host programming language such as C, C++, COBOL, Fortran, Java or REXX. The executable as well as the source code is stored as files in the file system.

### ***SQL stored procedure***

This is a program developed using the SQL Procedure language, according to the standard definition of SQL3. SQL stored procedures programs are made of a collections of SQL statements and control-of-flow language written by program developers. They are stored at the database and can be invoked by client applications.

### ***SQL function***

This is a program for user-defined-function, developed entirely using the SQL Procedure language. The SQL function is not supported by this first delivery of SQL Procedure language support on DB2 across platforms.

### ***SQL/PSM***

PSM means Persistent Stored Module. It is the SQL3 definition of a procedural language for relational databases.

The following chapters of this book cover in detail the SPB and SQL Procedures language support across platforms.





---

## Chapter 2. The SQL Procedures language

The SQL Procedures language is a common programming language across the DB2 family for writing stored procedures. The use of SQL Procedures provides to customers the benefit of writing their stored procedures in a standard and portable language. This chapter describes in detail the SQL Procedures programming language.

Any differences in current releases are due to differences in how we roll out features for a given platform. However, DB2 will be continuing to enhance the SQL Procedures language on all platforms and reduce any current platform differences.

---

### 2.1 What is it?

The SQL Procedures language is based on SQL extensions as defined by the SQL/PSM (Persistent Stored Modules) standard. SQL/PSM (an ISO/ANSI standard for SQL3) is a high level language — similar to other RDBMS languages such as Transact SQL (T/SQL) from Sybase, and Procedural Language (PL/SQL) from Oracle — that extends SQL to procedural support.

The ISO/ANSI SQL3 is an open solution for SQL among database management system vendors that support the SQL ISO/ANSI standard.

Modules are collections of SQL stored procedure and function declarations that are stored “persistently” inside the database (more likely, inside the SQL system for a possible dependency checking between the procedures and functions and the database objects). Inside SQL/PSM modules, one can specify the character set, the default schema name to be prepended to unqualified names in SQL statements in the module, the search path within schemas, and temporary table declarations.

Following is an example of a CREATE MODULE statement:

```
CREATE MODULE mymodule
CHARACTER SET "latin-1"
SCHEMA MYSCHEMA
PATH 'SCHEMA1,SCHEMA2'
create procedure mySQLprocedure1(out parm1 integer)
begin
    SET parm1 = select max(id) from mytable;
end;
create function mySQLfunction1 returns integer
begin
    return( select max(id) from mytable);
end;
END MODULE;
```

Where:

- "latin-1" is the character set.
- MYSCHEMA is a default schema name to be prepended to unqualified names in SQL statements in the module.
- 'SCHEMA1,SCHEMA2' is the search path within schemas SCHEMA1 and SCHEMA2.

In the first releases of DB2 UDB SQL Procedures support, SQL/PSM modules and SQL functions are not implemented. Only SQL stored procedures are implemented.

A DB2 UDB SQL stored procedure can be created without being part of a module. The SQL stored procedure source code, that is the CREATE PROCEDURE statement, is stored in the database after a successful compilation and registration of the SQL stored procedure in the appropriate DB2 tables (see chapter for specific platform).

As explained later, SQL stored procedures are different from external stored procedures that are written in a third generation language like C, COBOL, or Java.

Local client as well as remote client applications, connected to the DB2 server through network stacks (for example, TCP/IP), can invoke an SQL stored procedure by executing the SQL CALL statement. The SQL CALL statement is also part of ISO/ANSI SQL3.

The ability to write stored procedures greatly enhances the power, efficiency, and flexibility of SQL. The client program can pass parameters to the SQL stored procedure and receive parameters from it, as well as result sets. The SQL CALL statement can be executed as either static or dynamic SQL. Parameters in the CALL statement, including the stored procedure name, can be supplied at execution time. The SQL CALL statement can be used to invoke dynamically any SQL stored procedure supported by DB2.

The following DB2 servers currently support SQL stored procedures:

- DB2 UDB for UNIX, Windows, and OS/2
- DB2 UDB for AS/400
- DB2 UDB for OS/390

---

## 2.2 Planning to use the SQL Procedures language

This section contains information about planning for the use of DB2 stored procedures, and provides specific information to help you develop stored procedures using the SQL Procedures language. For more information about specific hardware and software requirements, see the chapter in this book that covers the specific platform you are planning to use.

### 2.2.1 Why use it?

This section describe the benefits of using SQL stored procedures.

#### ***Consistency with the data***

In theory, SQL stored procedures are fully SQL, which means that they are written using only SQL statements. Like other SQL objects, and because they are part of module statements, they are always stored in the database system in which they are used and for which they were developed. Storing SQL stored procedures within the database system allows dependencies to be checked between the SQL schema objects (tables, views, and so on) and the procedure, as soon as a manipulation is done (like dropping or altering an object). An SQL

stored procedure must not get *out of sync* with the SQL schema objects it works with.

On the other hand, this *out of sync* status can happen with external stored procedures, where the *executable* is stored at the system level, away from the database data management system. The backup and restore operations could be done out of sync with the backup and restore of the data, leading to inconsistencies in the processing.

The actual DB2 UDB support for SQL Procedure translates an SQL stored procedure into an external stored procedure, allowing potential inconsistencies. However, the source code of each stored procedure is stored inside the database itself, making possible to recreate the correct *executable* of the procedure.

### ***Modular programming***

An SQL stored procedure can be created once and then stored in the database. Any userid with the necessary authorization can access the stored procedure through the CALL SQL statement. For instance, an SQL stored procedure could be created or modified by a specialist in SQL stored procedure language programming, and be called by client programs developed by other programmers. This allows the DBA to control the development of the business logic at the server, which is a very sensitive area.

### ***Faster execution***

If an operation requires a large number of SQL statements or is performed repetitively, an SQL stored procedure can be faster than an SQL query sent directly from a client program, because it may be already loaded in memory (or cached) after the first execution.

The benefit of SQL stored procedures is even greater if the client expects to send dynamic SQL to the server. The compilation of the dynamic SQL statements in an SQL stored procedure would be done at the first execution and would stay in the DB2 cache as long as the memory space is not needed.

In addition, the fact that SQL stored procedures are translated to C is another performance advantage.

### ***Network traffic reduction***

An operation requiring many SQL statements can be performed through a single CALL statement that executes the same SQL code inside a procedure, rather than sending every individual SQL query over the network. For instance, Figure 1 shows what a client application written in C or CLI or any other embedded SQL language has to send and receive through the network to execute two SQL statements.

Figure 2 shows the same logic embedded in an SQL stored procedure. The client has to send one SQL CALL statement to the server. The results sent back are the same. As you can see, the two SQL statements from Figure 1 are reduced to one SQL CALL statement.

The savings can be important. The SQL CALL statement is a very short network message. On the contrary, a single SQL statement with many synchronized queries could be a few kilobytes long, up to 32k or 64k, depending of the version of DB2 UDB installed, and this could result in many network messages sent to the server. The savings are greater if the logic you want to execute requires many

SQL statements or many network messages, like the FETCH operation with cursors. The network transfer time between the client and the server could become longer than the execution of the SQL statement themselves, depending on network availability.

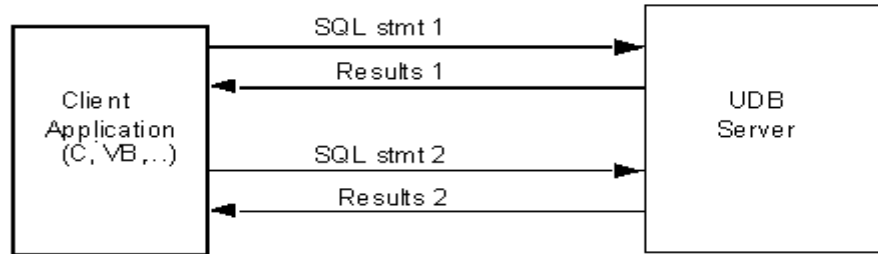


Figure 1. The usual SQL client, where the logic sends many SQL queries

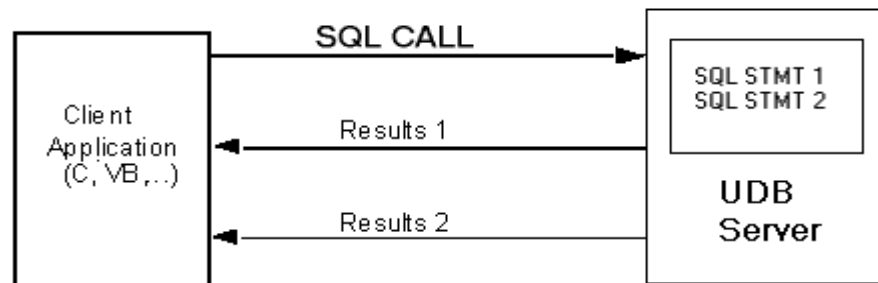


Figure 2. SQL client using the same logic, but within an SQL stored procedure.

Another point to be considered is that, while the client based query must return every row across the network, a stored procedure can return only a subset of the rows, or potentially none at all, and can return values derived from subsequent logic.

**Can be used as a security/shield mechanism**

It is often useful to shield the data and the tables from developers and database users. This can be done by restricting access to tables to only a few *trusted users*, and by granting the right to execute specific stored procedures, developed for updating, inserting or deleting rows to *less trusted users*. This can ensure that applications cannot execute direct SQL statements against important data, but must make modifications only through the filter of stored procedures.

These stored procedures can even have more complex logic than just basic SQL statements. One stored procedure that deletes one customer could also update other tables related to that customer to keep information for audit or further references. Referential integrity should be able to do this, but sometimes, it is easier and faster to use a stored procedure, because depending on the context of the modification, different actions will have to be executed. Triggers or referential integrity constraints on a table are not context sensitive, because only the data in the rows are available. There is no possibility to have a specific DELETE, UPDATE or INSERT operation according to the current context of the application.

For example, for the following SQL statement:

```
DELETE CUSTOMER WHERE CUSTID=11111
```

You may want to take one of the following actions:

- Delete this customer, but insert a log record of it in the custlog table for audit later
- Delete this customer, but keep the statistics in the custstats table until end of year
- Delete this customer, but also delete corresponding rows in custlog and custstats tables

**Note:** The custlog and custstats tables mentioned here are tables you created for your application design.

The delete trigger on the customer table will not be able to choose one of these different actions because it has to be generic. When you delete the customer, you cannot specify another action *along with* the delete SQL statement. A stored procedure would be able to decide what to do, according to an input parameter value.

Another example is shielding tables from users. Figure 3 shows that the developer shielded the tables A, B and C from the client applications C11, C12, C13 and C14 by inserting a layer of SQL stored procedures SP1, SP2 and SP3. These SQL stored procedures can execute complex SQL queries like insert, update and delete on one or more tables. When called, SP1 will modify only table A, SP2 will modify table A and table C, and SP3 will modify table B and table C.

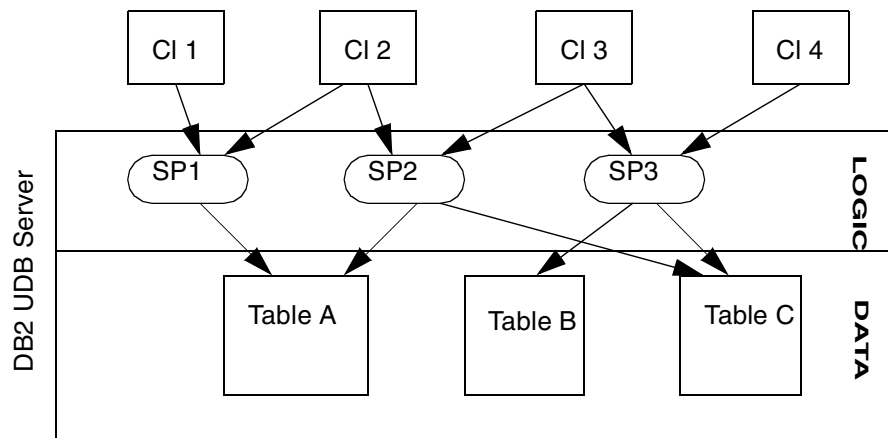


Figure 3. An example of shielding tables from users

## 2.2.2 When to use them?

In general, SQL stored procedures can be used as soon as a client application needs to send dynamic SQL statements, or multiple SQL operations, like FETCH, to the server. Grouping the same functional SQL statements in an SQL stored procedure allows performance and network bandwidth improvement. It also simplifies programming, improves maintainability, and facilitates deployment of applications.

The following sections describe in more detail why SQL stored procedures are really useful:

- **Easier and faster to program than external stored procedure**

SQL stored procedures are very similar to SQL. The control-flow statements are really simple, and the programmer does not need to understand C or COBOL.

SQL stored procedures are the perfect solution for business logic, since they can be totally developed with SQL statements and control-flow logic. That is, you do not need to access any other external resource besides the RDBMS. They are fast to develop and maintain.

- **No use of specific system services or resources**

If the business logic needs to access system services or external resources, or call other external programs, SQL stored procedures may not be applicable. In fact, there is no possibility yet, in the SQL/PSM standard, to call a shared library, execute a system command, or send e-mail.

For the time being, the only solution is to use external stored procedures written in C or COBOL. For instance, a stored procedure that has to send an e-mail to the credit manager every time a credit limit is exceeded, or execute a system command when a threshold is reached, has to be an external stored procedure developed in C or COBOL.

**Note:** This is true except for the Windows platforms. DB2 UDB for Windows allows OLE stored procedures which can invoke operating system and other platform capabilities like email through the use of OLE services. Refer to the DB2 UDB for Windows manuals for more details on use of OLE automation. So you can use more than C or COBOL stored procedures when you need system services and are running on Windows.

- **No vital performance needed**

Due to the actual implementation of SQL stored procedures, the execution performance may not be as fast as an optimized external stored procedure written in C or COBOL. Although the difference is very small for most of the business logic functionality, some applications may need the extra time difference that can be gained only by using an optimized procedure programmed in C or COBOL language.

How do we choose, then? Usually, only a few parts of a company business logic application needs real performance. The designer of the application should find the right compromise between development time and maintainability versus execution performance.

**Important:**

SQL stored procedures are not interpreted. They are translated to C, so the build process has more steps, but the resulting procedure is compiled. The preprocessors and translators also add optimization.

- **Lots of business logic**

The more business logic you have, the more you will save in development, maintenance, and execution time. The modularity allowed by stored

procedures is certainly a gain in maintenance, especially when the client programs are developed in different languages.

Changing some business logic does not mean changing all the different COBOL and C or Visual Basic (VB) clients that were executing these SQL statements. Only the stored procedure involved in this business logic needs to be changed, assuming that the signature — which means the list of parameters and types of the stored procedure — is still the same.

- **Highly distributed application (deployment)**

Deployment of clients having business logic embedded is hard to maintain. Every time the business logic change or an error is fixed, all the clients have to be updated again. This can lead to *out of sync* clients accessing good data in the wrong way.

However, fixing one stored procedure fixes all the clients at once. The maintenance is faster, and problems arise less often.

- **What partition of business logic should reside on the server?**

Lots of documents have been written on this subject, and not everybody agrees on the conclusions. But let us consider the two major solutions that are seen most often.

Usually, the two rules of 80% on client 20% and on server, or 20% on client and 80% on server, could apply for most applications, depending on the level of risk the developers and company expect.

- 80%-20% client to server ratio:

This corresponds to applications that do not want to rely too heavily on specific stored procedure languages. The 20% consists mainly of SQL statements. This is typically the case for most independent software vendors (ISVs) that want to deal with a minimum of different stored procedures languages (T/SQL or PL/SQL ,or even DB2 SQL Procedures language) because their programs will have to be ported on different RDBMSs. They are usually constrained to a subset of SQL, portable across all the RDBMSs. The counterpart of this solution is that this kind of application cannot use the full possibilities of an RDBMS, and would usually not perform as well as a 20%-80% solution (because of using more network bandwidth, for instance).

- 20%-80% client to server ratio:

This kind of application is tuned to access business logic on the server, mainly as stored procedures. This application will rely heavily on RDBMS features, and should hopefully be more efficient than an 80%-20% solution by optimizing bandwidth, deployment, and development time. This is typically the kind of application developed by a company for itself. The drawback of this type of application is that the company relies on one RDBMS, which could be troublesome for future migrations.

Of course, all kind of partitions can be seen, and the company's final choice will depend only on the evaluation between level of risk/RDBMS dependency versus application performance/savings during the development and maintenance phase.

---

## 2.3 Comparing SQL stored procedures and external stored procedures

External stored procedures are stored procedures developed using host programming language such as C, C++, COBOL, Fortran, Java, and REXX. This was the original way to develop stored procedures with DB2, which means that stored procedures could not be written totally in SQL, on all platforms (except for DB2 UDB on AS/400). Such stored procedures are stored in files on the machine where the database server is located and not in the database itself. That is why they are named external stored procedures.

### 2.3.1 Development

SQL stored procedures are stored procedures written in the SQL Procedures programming language. An SQL stored procedure is developed totally in SQL, and its source code is stored in the database, so it can be executed directly within the DBMS environment.

#### 2.3.1.1 Lower the development cost

External stored procedures are developed in a procedural language such as C, Java or COBOL. Using programming languages for database programming requires much experience and a deep understanding of how the SQL data types and database engine features are mapped to the host language.

For example, in the C language, an SQL VARCHAR variable called *myvar* would have to be represented as a structure like the following:

```
struct {  
    short len;  
    char data[31];  
} myvar;
```

You would represent the same VARCHAR variable like this, using SQL Procedures language:

```
DECLARE myvar VARCHAR;
```

As you can see, it is simpler in SQL Procedures language to declare variables of an SQL type.

The conceptual differences between SQL and host languages adds complexity to the programming of external stored procedures, making them longer in the number of lines, with more possibilities for mistakes, and making them more difficult to debug. Programmers of external stored procedures must be well trained database SQL programmers as well as having extensive experience working with third generation languages.

On the other hand, SQL Procedures rely on SQL. Variable handling is fully SQL (no structures to deal with VARCHAR) and the procedural extensions use an easy common syntax very close to BASIC or PASCAL. The benefit of programming with SQL Procedures language is immediate. Once you know SQL, you can learn the SQL Procedures language in a matter of hours. You do not have to deal with obscure representation and manipulation of your variables, but can just use them directly.



For example, assuming that two VARCHAR variables v1 and v2 are declared, the code in SQL Procedures language is as follows to assign v2 to v1:

```
/* in SQL Procedures language*/  
SET v1 = v2;
```

The same code in C is as follows:

```
/* in C */  
memcpy(v1.data, v2.data, v2.len);  
v1.len=v2.len;
```

As you can see, you need to know the internal representation of a VARCHAR variable in C, to be able to copy it into a C program.

Another advantage of using SQL Procedures is that you can develop them anywhere, as soon as you have a DB2 UDB connection ready. The source code is stored in the database, allowing the programmer to fetch it back for modification. Unlike external stored procedures, there is no need to access the file system. This simplifies the development by focusing only on the DB2 UDB functionality, that is, programmers do not have to learn system tools such as file transfer protocols, system specific editors or compiler options. They just learn SQL and its procedural extensions.

The Stored Procedure Builder, provided with DB2 UDB, relies on this feature, allowing the same easy, graphical interface to build SQL or Java stored procedures on any DB2 UDB platforms with nothing more needed than a DB2 UDB connection. (See Chapter 3, “The DB2 Stored Procedure Builder” on page 57 for details.)

#### **2.3.1.2 Leverage programming skills**

By using SQL Procedures, you leverage the programming skills needed in your company, lowering the cost of development of application. For example, the same person that knows SQL can also be the programmer.

It also makes things easier for porting, maintaining the source code, and deployment (same code everywhere). This can be an important saving for companies that have many programmers.

### **2.3.2 Runtime**

Calling an SQL stored procedure is no different than calling an external stored procedure. The same SQL CALL statement works for both of them.

Because the SQL Procedures language is higher level, it may have to do more tasks for the programmer (like checking for exception condition after every SQL stored procedure statement). Also, it may be slower to execute than the usual C or COBOL stored procedure written by a specialist or expert (which may bypass some error checking). But the cost performance is justified because of the savings that can be obtained during development and maintenance.

---

## **2.4 Current implementation of SQL Procedures language**

The actual choice made by IBM is to translate an SQL Procedures language program into an external C stored procedure. This is done under-the-covers by the engine, and the programmer does not need to understand how it is done. The

only thing needed is a C or C++ compiler to be installed along with the DB2 UDB server on the server machine.

## 2.4.1 How does this work in general?

Once you send the SQL stored procedure source code (the CREATE PROCEDURE statement) to the DB2 UDB engine from the Stored Procedure Builder or from the DB2 UDB command line, the DB2 UDB engine processes it, creates a C file with embedded SQL, compiles it and installs it in the right place for its first execution. The implementation is slightly different on OS/390, as compared to AS/400 and distributed platforms due to platform differences. For details on each platform, see Chapter 4, "SQL Procedures for DB2 UDB for OS/390" on page 109; Chapter 5, "SQL Procedures for DB2 UDB for UNIX, Windows, OS/2" on page 145; and Chapter 6, "SQL Procedures for DB2 UDB for AS/400" on page 169.

**Note:** When you build your SQL stored procedures with the DB2 Stored Procedure Builder, of course, all the application developer needs to do is press the "Build" button. All the work is done for you, and any differences in processing between platforms is handled for you. We recommend that approach for building stored procedures in the SQL Procedures language.

### 2.4.1.1 Statements supported

The Database Language SQL - Part 4: Persistent Stored Modules of ISO/IEC 9075 specifies the syntax and semantics of a database language for declaring and maintaining persistent database language routines in SQL-server modules. The scope of the current implementation is limited to SQL stored procedures and does not include support for SQL functions and Feature P01, "Stored Modules". The following stored database language capabilities are supported:

- Statements to direct flow control:
  - CASE statement
  - IF statement
  - ITERATE statement
  - FOR statement
  - LEAVE statement
  - LOOP statement
  - REPEAT statement
  - WHILE statement
  - Compound statement:
    - BEGIN
    - END
- Assignment statement:
  - SET
- Specification of condition handlers:
  - DECLARE.....HANDLER.....FOR.....
  - DECLARE.....CONDITION.....



```

declare cdate DATE default '03/01/1999';
declare ctime TIME default '11:11:22';
declare cstamp TIMESTAMP default '1990-01-01-12.01.00.000000';
declare c4 REAL default 12345.456;
declare c5 DECIMAL(9,2) default 12345.456 ;
declare c6 BIGINT default 1234567890123456789;
declare LOC1 result_set_locator varying;

SET c1 = (select count(*) from employee);
SET c2 = (select max(empno) from employee);
SET c3 = (select min(empno) from employee);
insert into result(proc,res)
values ( 'exec P1', 'C1=' || CHAR(C1) || ' C2=' || C2 || ' C3=' || C3);
insert into result(proc,res)
values ( 'exec P1', 'date=' || CHAR(cdate));
insert into result(proc,res)
values ( 'exec P1', 'time=' || CHAR(ctime));
insert into result(proc,res)
values ( 'exec P1', 'timestamp=' || CHAR(cstamp));
insert into result(proc,res)
values ( 'exec P1', 'REAL=' || CHAR(c4));
insert into result(proc,res)
values ( 'exec P1', 'decimal(9,2)=' || CHAR(c5));
insert into result(proc,res)
values ( 'exec P1', 'LONG VARCHAR=' || c34);
insert into result(proc,res)
values ( 'exec P1', 'BIG INT=' || CHAR(c6));
END

```

**Important:**

OS/390 does not have a BIGINT data type, or RESULT SET LOCATOR.

The example shows SET varname = (SELECT ...) - which is not yet supported on OS/390.

## 2.4.3 Language Elements

The following sections describe and show examples of each type of statement and language element.

The examples shown assume that a table *result* is created in the database that keeps an execution trace of the SQL stored procedure. The following is the DDL used to create the table *result*:

```
create table result ( proc char(22), res varchar(128))
```

Note that, because the SQL Procedures language is SQL, names (variables, labels, etc.) are not case sensitive.

### 2.4.3.1 Assignment statement

The assignment statement assigns a value to an output parameter or to an SQL variable, which is a variable that is defined and used only within a procedure body.

```
CREATE PROCEDURE B(OUT var1 INTEGER,OUT var2 INTEGER)
LANGUAGE SQL
```

```

BEGIN
    SET var1 = 10;
    SET var2 = (SELECT count(*) FROM customer);
END

```

**Important:**

The assignment statement: SET var2 = (SELECT count(\*) FROM customer); shown in the example above is not yet supported on the OS/390 platform.

### 2.4.3.2 CASE statement

The CASE statement selects an execution path based on the evaluation of one or more conditions. This statement is similar to the CASE expression, which is described in the SQL Reference of DB2 UDB, Chapter 3, Language Elements, topic CASE expression.

```

CREATE PROCEDURE I1(IN var1 CHAR(3) )
LANGUAGE SQL
BEGIN
CASE var1
    WHEN 'AAA' THEN
        insert into result(proc,res) values ('exec of I1','AAA found. ');
    WHEN 'BBB' THEN
        insert into result(proc,res) values ('exec of I1','BBB found. ');
    WHEN 'CCC' THEN
        insert into result(proc,res) values ('exec of I1','CCC found. ');
    ELSE
        insert into result(proc,res) values ('exec of I1','default case
value=' || CHAR(tt) );
    END CASE;
END

```

Here is another example of a CASE statement:

```

CREATE PROCEDURE I6(IN var1 CHAR(3),IN var2 CHAR(3))
LANGUAGE SQL
BEGIN
CASE
    WHEN var1='AAA' THEN
        INSERT INTO result(proc,res) VALUES ('exec of I6','var1=AAA');
    WHEN var2='AAA' THEN
        INSERT INTO result(proc,res) VALUES ('exec of I6','var2=AAA');
    END CASE;
END

```

### 2.4.3.3 IF statement

The IF statement selects an execution path based on the evaluation of a condition.

```

CREATE PROCEDURE B1(IN v CHAR(1))
LANGUAGE SQL
BEGIN
IF (v >'F') THEN
    INSERT into result(proc,res) VALUES ('exec of B1',' v > F v=' || v);
ELSEIF (v >'D') THEN
    INSERT INTO result(proc,res) VALUES ('exec of B1',' v > D v=' || v);
ELSEIF (v >'B') THEN
    INSERT INTO result(proc,res) VALUES ('exec of B1',' v > B v=' || v);

```

```

ELSEIF (v >'A') THEN
    INSERT INTO result(proc,res) VALUES ('exec of B1',' v > A v=' || v);
ELSE
    INSERT INTO result(proc,res) values ('exec of B1','Else branch
done. ');
END IF;
END

```

Here is another example of an IF statement:

```

CREATE PROCEDURE MM (in PA INTEGER)
LANGUAGE SQL
BEGIN
    IF ( PA in (12,13,10)) THEN
        INSERT INTO result(proc,res) VALUES ('exec of MM',' PA=' || CHAR(PA)
|| ' found in (12,13,10)');
    ELSE
        INSERT INTO result(proc,res) VALUES ('exec of MM',' PA=' || CHAR(PA)
|| ' not found in (12,13,10)');
    END IF;
END

```

#### 2.4.3.4 LEAVE statement

The LEAVE statement transfers program control out of a loop or a block of code (see statement LEAVE myloop in the sample below). When a LEAVE statement transfers control out of a compound statement, all open cursors in the compound statement, except cursors that are used to return result sets, are closed.

In addition, the LEAVE statement can be used to exit the stored procedure (see statement LEAVE PP in the sample below).

```

CREATE PROCEDURE F (IN ASSEMBLY_NUM CHAR(10) )
LANGUAGE SQL
PP:BEGIN
    DECLARE a INTEGER DEFAULT 0;
myloop:LOOP
    INSERT INTO result(proc,res) VALUES ( 'proc F', 'LOOP ' || CHAR(a) );
    IF (a> integer(assembly_num) ) THEN
        LEAVE myloop; /* exit the loop */
    ELSE
        SET a = a + assembly_num;
        IF (a> 1000) ) THEN
            LEAVE PP; /* exit the procedure*/
        END IF;
    END IF;
END LOOP myloop;
END

```

#### 2.4.3.5 ITERATE statement

The ITERATE statement causes the flow of control to return to the beginning of a labelled loop.

```

CREATE PROCEDURE E()
LANGUAGE SQL
BEGIN
    DECLARE zz INTEGER DEFAULT 11;
    xx1: while ( zz < 5) do

```

```

        INSERT INTO result(proc,res) VALUES ('proc E','ascending=
' || CHAR(zz) );
        SET zz = zz +1;
        ITERATE xx1;
        INSERT INTO result(proc,res) VALUES ( 'proc E', 'iterate not done=
' || CHAR(zz) );
        end while;
END

```

#### 2.4.3.6 FOR statement

The FOR statement executes a group of statements repeatedly. It must be associated with a query expression and terminates after the group of statements is executed for every row in the result of query expression.

The FOR statement is a labelled statement. The following example shows FOR usage in SQL stored procedures:

```

DECLARE X INTEGER DEFAULT 0;
FOR L1 AS SELECT balance FROM accounts DO
    SET X = X + balance;
END FOR;

```

The body of a FOR statement is not allowed to contain a LEAVE statement that refers to L1. But cursor columns can be prefixed with the name of the FOR statement L1. This is also valid.

```
SET X=X+L1.balance.
```

It is very useful to refer to variables in nested FOR loops that access the same table, as in this example:

```

DECLARE X INTEGER DEFAULT 0;
FOR L1 AS SELECT balance FROM accounts WHERE DEP > 'C02' DO
    FOR L2 AS SELECT balance FROM accounts WHERE DEP<'D01' DO
        SET X = L2.balance + L1.balance;
    END FOR;
/* do something here with X*/
END FOR;

```

A cursor is implicitly opened at the beginning of execution; closed automatically at the end of execution. In the preceding example, a cursor is created for the SQL statement `SELECT balance FROM accounts` and for each line of the result set instruction `SET X= L2.balance + L1.balance` is executed.

It is also possible to specify a name for the implicit cursor as in the following example:

```

FOR L1 AS curs1 CURSOR FOR
    SELECT * FROM accounts WHERE balance = 0 DO
    DELETE FROM accounts WHERE CURRENT OF curs1;
END FOR;

```

The body of a FOR statement is not allowed to contain an OPEN, FETCH, or CLOSE statement that refers to curs1.

**Important:**

The **FOR** statement is not yet supported on the OS/390 platform.

### 2.4.3.7 LOOP statement

The LOOP statement executes a group of statements repeatedly until a LEAVE statement transfers program control out of the loop. The ITERATE statement causes the flow of control to return to the beginning of the loop.

```
CREATE PROCEDURE F (IN ASSEMBLY_NUM CHAR(10) )
LANGUAGE SQL
BEGIN
    DECLARE a INTEGER DEFAULT 0;
myloop:LOOP
    INSERT INTO result(proc,res) VALUES ( 'proc F', 'LOOP ' || CHAR(a) );
    IF (a > integer(assembly_num) ) THEN
        LEAVE myloop;
    ELSE
        SET a = a +1;
    END IF;
    END LOOP myloop;
END
```

### 2.4.3.8 REPEAT statement

The REPEAT statement executes a group of statements until a search condition is true. Within the group of statement of a REPEAT statement, a LEAVE statement transfers program control out of the REPEAT and an ITERATE statement causes the flow of control to return to the beginning of the REPEAT.

```
CREATE PROCEDURE F (IN ASSEMBLY_NUM CHAR(10) )
LANGUAGE SQL
BEGIN
    DECLARE b INTEGER DEFAULT 0;
    REPEAT
        INSERT INTO result(proc,res) VALUES ('proc F','REPEAT ' || CHAR(b) );
        SET b = b +1;
    UNTIL (b>5)
    END REPEAT;
END
```

### 2.4.3.9 WHILE statement

The WHILE statement repeats the execution of a statement or group of statements while a specified condition is true. The LEAVE statement transfers program control out of the WHILE. The ITERATE statement causes the flow of control to return to the beginning of the WHILE.

```
CREATE PROCEDURE E()
LANGUAGE SQL
BEGIN
    DECLARE aaa char(30);
    DECLARE zz INTEGER DEFAULT 11;
-----
-- THIS WHILE LOOP WILL TEST THE WHILE STATEMENT ITSELF
-----
    while ( zz > 0) do
        INSERT INTO result(proc,res) VALUES('proc E','descending =
' || CHAR(zz));
        SET zz = zz -1;
    end while;
-----
-- THIS WHILE LOOP WILL TEST THE LEAVE STATEMENT
```



```

-----
xx:while ( zz < 5) do
    INSERT INTO result(proc,res) VALUES ('proc E','ascending=
'|CHAR(zz));
    SET zz = zz +1;
    LEAVE xx;
    INSERT INTO result(proc,res) VALUES ('proc E','leave not done=
'|CHAR(zz));
    end while;
-----
-- THIS WHILE LOOP WILL TEST THE ITERATE STATEMENT
-----
SET zz = 0;
xx1: while ( zz < 5) do
    INSERT INTO result(proc,res) VALUES ('proc E','ascending=
'|CHAR(zz) );
    SET zz = zz +1;
    ITERATE xx1;
    INSERT INTO result(proc,res) VALUES ( 'proc E', 'iterate not done=
'|CHAR(zz) );
    end while;
INSERT INTO result(proc,res) VALUES ( 'proc E', 'end ' );
END

```

#### 2.4.3.10 Dynamic SQL statements

The dynamic SQL statements, that is, PREPARE, EXECUTE and EXECUTE IMMEDIATE, are allowed in an SQL stored procedure code. The following is an example of the use of those statements:

```

CREATE PROCEDURE H1 ()
LANGUAGE SQL
BEGIN
    declare stmt varchar(254);
    declare v1  varchar(20) default 'foofoo';
    insert into result(proc,res) values('exec H','Start');

    SET stmt = 'insert into result (proc,res)  values (''exec H'', ''this is
the parameter marker ''||? )';
    prepare s2 from stmt;
    execute s2 using v1;

    SET stmt = 'create table mytab (c integer) ';
    execute immediate stmt;
    set stmt='insert into mytab values ( 1 )';
    execute immediate stmt;
    insert into result(proc,res) select 'exec H',CHAR(c) from mytab;
    insert into result(proc,res) values('exec H','End');
    SET stmt = 'drop table mytab ';
    execute immediate stmt;
END

```

#### 2.4.3.11 SIGNAL statement

The SIGNAL statement signals an exception condition:

- If at least one handler in all the nested compound statements is defined to handle this exception, it will be called immediately by the SIGNAL statement, as in the example below:

```

CREATE PROCEDURE G10()
LANGUAGE SQL
BEGIN
    DECLARE C1 CONDITION FOR SQLSTATE '04000';
    DECLARE EXIT HANDLER FOR C1
        INSERT INTO result(proc,res) VALUES ('exec of G','EXIT handler fired');

    BEGIN /* nested compound statement*/
        INSERT INTO result(proc,res) VALUES ('exec of G','This line should stay here');
        SIGNAL SQLSTATE '04000';/*the handler will be fired here*/
        INSERT INTO result(proc,res) VALUES ('exec of G','This line shouldn't be
here');
    END;
    INSERT INTO result(proc,res) VALUES ('exec of G','This lines is executed after the
handler is fired');
    INSERT INTO result(proc,res) VALUES ('exec of G','aEND of Proc');
END

```

- If no handler is defined to catch the SQLSTATE in the SIGNAL statement, the exception will be propagated to the caller, as in the example below:

```

CREATE PROCEDURE G11()
LANGUAGE SQL
BEGIN
    DECLARE C1 CONDITION FOR SQLSTATE '04000';
    INSERT INTO result(proc,res) VALUES ('exec of G', 'START: This line should stay
here');
    SIGNAL C1;/*exit the procedure with SQLSTATE=C1=04000*/
    INSERT INTO result(proc,res) VALUES ('exec of G', 'END of Proc');
END

```

#### 2.4.3.12 RESIGNAL statement

The RESIGNAL statement resignals an exception condition, and it can only be coded as part of a condition handler.

The use of a RESIGNAL statement without an operand causes the identical condition to be passed outwards, while a RESIGNAL statement with an operand causes the original condition to be replaced with the new condition you have specified.

Following is an example using the RESIGNAL statement:

```

CREATE PROCEDURE G8 ()
LANGUAGE SQL
BEGIN
    DECLARE not_found condition for SQLSTATE '02000';
    DECLARE found condition for SQLSTATE '01000';

    DECLARE CONTINUE HANDLER FOR SQLSTATE '12345' BEGIN
        RESIGNAL SQLSTATE '22345';
        RESIGNAL ;
        RESIGNAL not_found;
    END;

    insert into result(proc,res) values ( 'exec of G8','Start');
    SIGNAL SQLSTATE '12345';
    insert into result(proc,res) values ( 'exec of G8','After signal 123245');
    SIGNAL found;
    insert into result(proc,res) values ( 'exec of G8','After signal Found');
    insert into result(proc,res) values ( 'exec of G8','End');
END

```

**Important:**

The **SIGNAL** and **RESIGNAL** statements are not yet supported on the OS/390 platform.

### 2.4.3.13 Compound statement

Roughly, we can say that a compound statement is a set of one or more SQL statements between the `BEGIN` and `END` keywords.

A compound statement may contain SQL variable declarations, condition handlers declaration, or cursor declarations. The order of statements in a compound statement is:

1. SQL variable and condition declarations
2. Cursor declarations
3. Handler declarations
4. Assignment statements, control-flow statements such as `CASE`, `IF`, `LOOP`, `REPEAT`, and `WHILE`, and SQL statements such as `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CALL`, `CREATE TABLE`, etc.

A compound statement may be declared as:

- **ATOMIC**, which means that all actions performed by the compound statement must succeed, or the entire set of database modifications made by those actions are rolled back. The following is an example of an **ATOMIC** compound statement:

```
CREATE PROCEDURE D(PARM1 char(10) )
LANGUAGE SQL
BEGIN
SC:
BEGIN ATOMIC
    DECLARE zz INTEGER DEFAULT 11;
    INSERT INTO result(proc, res)
        VALUES ('exec of D', 'In atomic block before error test:' || PARM1);

    IF (Parm1 = '1') THEN
        SET aaa = (select job from employee );
    ELSE
        SET aaa = (SELECT job FROM employee WHERE empno = '000020');
    END IF;
    INSERT INTO result(proc, res)
        VALUES ('exec of D', 'In atomic block after error test:' || aaa);
END;
END
```

- **NOT ATOMIC**, which means that an error occurring within the compound statement does not cause all actions performed by the compound statement to be rolled back. The **NOT ATOMIC** string is optional. It is the default for a compound statement. The following is an example of a **NOT ATOMIC** compound statement:

```
CREATE PROCEDURE D(PARM1 char(10) )
LANGUAGE SQL
BEGIN
```

```

SC:
  BEGIN NOT ATOMIC
    declare zz integer default 11;
    insert into result(proc, res)
      values ('exec of D', ' In non atomic block before error test
:' || PARM1);

    if (Parm1 = '1') then
      set aaa = (select job from employee );
    else
      set aaa = (select job from employee where empno = '000020');
    end if;
    insert into result(proc, res)
      values ('exec of D', ' In non atomic block after error test :'
|| aaa);
  END;
END

```

**Important:**

ATOMIC compound statements are not yet supported on the OS/390 platform.

#### 2.4.3.14 SQL statements

Following is a list of SQL statements that may be used in an SQL stored procedure body. Refer to the SQL Reference Guide for your platform for a more detailed explanation of these SQL statements. Keywords are not case sensitive.

- ALLOCATE
- ASSOCIATE
- CALL
- CLOSE
- COMMENT ON
- CREATE
- DECLARE CURSOR
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- GRANT
- INSERT
- LABEL ON
- LOCK TABLE
- OPEN
- PREPARE FROM

- RELEASE
- RENAME
- RELEASE SAVEPOINT
- REVOKE
- ROLLBACK TO SAVEPOINT
- ROLLBACK
- SAVEPOINT
- SELECT INTO
- UPDATE
- VALUES INTO

**Important:**

ASSOCIATE, ALLOCATE, ROLLBACK (except ROLLBACK TO SAVEPOINT), VALUES INTO are not yet supported on the OS/390 platform.

REVOKE is supported on the OS/390 platform.

LABEL ON is supported on the OS/390 platform.

## 2.4.4 Returning result sets

### 2.4.4.1 Creating result sets in an SQL stored procedure

Passing back a result set to a calling application (an embedded SQL application or an SQL stored procedure) is done by declaring with a SELECT statement a cursor on the rows that are to be passed back. Many result sets can be returned, but each requires an independent DECLARE CURSOR statement.

If the caller is an application, the cursor declaration has to use a "WITH RETURN TO CLIENT" clause, as in the following example:

```
CREATE PROCEDURE sp_called_from_app
LANGUAGE SQL
BEGIN
    DECLARE result_set_1 cur1 CURSOR WITH RETURN TO CLIENT FOR
        Select empno,firstnme from employee ;
    OPEN result_set_1;
END
```

If the result set has to be passed back to another stored procedure, the cursor declaration has to specify a "WITH RETURN TO CALLER" clause, as in the example below:

```
CREATE PROCEDURE sp_called_from_sp
LANGUAGE SQL
BEGIN
    DECLARE result_set_1 cur1 CURSOR WITH RETURN TO CALLER FOR
        Select empno,firstnme from employee ;
    OPEN result_set_1;
END
```

**Important:**

WITH RETURN TO CLIENT or TO CALLER syntax in DECLARE CURSOR are not yet supported on the OS/390 platform.

#### 2.4.4.2 Retrieving result sets in the caller

To retrieve the result sets properly, the caller (embedded SQL application or another SQL stored procedure) must perform certain operations. We will focus on the SQL stored procedure syntax only, but the steps are similar in embedded SQL or CLI applications using host variables.

Here is a high level view for retrieving a result set in the caller:

- Declare a result set locator for each result set expected (DECLARE RESULT SET LOCATOR...).
- CALL the stored procedure.
- ASSOCIATE each locator to the procedure.
- ALLOCATE each cursor for each result set locator.
- FETCH the data for each cursor.
- CLOSE each cursors opened with the ALLOCATE statement.

Every step is detailed below.

Following is an example of an SQL stored procedure Y4 that retrieves one result set from a SQL stored procedure Y41.

```
CREATE PROCEDURE Y41 (IN parm1 INTEGER )
LANGUAGE SQL
BEGIN
    DECLARE cur1 CURSOR WITH RETURN TO CALLER
        FOR Select empno,firstnme from employee ;
    OPEN cur1 ;
END
```

In the following procedure Y4, note the use of the CONTINUE HANDLER declaration to detect the end of the result set using the NOT FOUND condition. When this handler is fired, it sets the RESULT\_SET\_END variable to 1. After the execution of this handler, execution resumes after the FETCH statement, and the loop will finally terminates.

```
CREATE PROCEDURE Y4(IN parm1 INTEGER)
LANGUAGE SQL
BEGIN
    DECLARE LOC_RES1 RESULT_SET_LOCATOR VARYING;
    DECLARE rc1,rc2 CHAR(20);
    DECLARE RESULT_SET_END integer default 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        BEGIN
            SET RESULT_SET_END = 1;
        END;
```

```

CALL Y41(parm1);
ASSOCIATE RESULT SET LOCATOR( LOC_RES1) WITH PROCEDURE Y41;
ALLOCATE RES1 CURSOR FOR RESULT SET LOC_RES1;
SET RESULT_SET_END = 0;
WHILE(RESULT_SET_END = 0) DO
    FETCH FROM RES1 INTO rc1,rc2;
    IF(RESULT_SET_END=0) THEN
        insert into result(proc,res) values ('exec Y4', 'rc1=' || rc1 || 'rc2=' || rc2);
    END IF;
END WHILE;
CLOSE RES1;
END

```

#### 2.4.4.3 CALL statement

This statement is the regular CALL SQL statement, which calls a stored procedure. It can be an external stored procedure (C, Java, COBOL) or a SQL stored procedure.

```

CREATE PROCEDURE F (IN ASSEMBLY_NUM CHAR(10) )
LANGUAGE SQL
BEGIN
    CALL MYSP2( ASSEMBLY_NUM);
END

```

The CALL statement accepts only parameters that are variables or constants. Expressions are not allowed.

#### 2.4.4.4 ASSOCIATE statement

The ASSOCIATE LOCATORS statement gets the result set locator value for each result set returned by a stored procedure. One result set locator variable is required for each result set that the stored procedure will return.

When the ASSOCIATE LOCATORS statement is executed, the procedure name or specification must identify a stored procedure that the requester has already executed using the CALL statement. The procedure name in the ASSOCIATE LOCATORS statement must be specified in the same way that it was specified on the CALL statement. For example, if a two-part name was specified on the CALL statement, you must use a two-part name in the ASSOCIATE LOCATORS statement. If the CALL statement was made with a three-part name and the current server is the same as the location in the three-part name, you can omit the location name and specify a two-part name.

In the following example, two result sets locators (LOC1,LOC2) are associated with the stored procedure P1.

```

DECLARE LOC1 RESULT_SET_LOCATOR VARYING;
DECLARE LOC2 RESULT_SET_LOCATOR VARYING;
CALL P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2) WITH PROCEDURE P1;

```

#### 2.4.4.5 ALLOCATE statement

The ALLOCATE CURSOR statement defines a cursor and associates it with a result set locator variable. The cursor name must not identify a cursor that has already been declared in the source program.

The result set locator variable must contain a valid result set locator value, as returned by a ASSOCIATE LOCATOR statement.

The following rules apply when you use an allocated cursor:

- You cannot open an allocated cursor with the OPEN statement.
- You can close an allocated cursor with the CLOSE statement. Closing an allocated cursor closes the associated cursor in the stored procedure.
- You can allocate only one cursor to each result set.

The life of an allocated cursor is: a rollback operation, an implicit close, or an explicit close that destroys allocated cursors.

For example, define and associate cursor C1 with the result set locator variable LOC1 for the first result set returned by the stored procedure, cursor C2 with result set locator LOC2 for the second result set returned by the stored procedure:

```
DECLARE LOC1 RESULT_SET_LOCATOR VARYING;
DECLARE LOC2 RESULT_SET_LOCATOR VARYING;
CALL P1;
ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2) WITH PROCEDURE P1;
ALLOCATE C1 CURSOR FOR RESULT SET LOC1;
ALLOCATE C2 CURSOR FOR RESULT SET LOC2;
```

#### 2.4.4.6 Processing the result set

Once the ASSOCIATE and ALLOCATE statements are done, processing a results set in the stored procedure is achieved by FETCHing the result set rows into an SQL variable until the end of the result set is reached. After a result set is processed, a CLOSE cursor statement must be executed.

In the example below, the SQL stored procedure processes two result sets by inserting each row into the result table.

```
CREATE PROCEDURE P1()
  LANGUAGE SQL
BEGIN
  DECLARE LOC1 RESULT_SET_LOCATOR VARYING;
  DECLARE LOC2 RESULT_SET_LOCATOR VARYING;
  DECLARE AT_END INTEGER DEFAULT 0;
  DECLARE column1, columns2 VARCHAR(30);
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET AT_END = 1;

  CALL P1; /*retrieve 2 results sets, both of them with 2 varchar(30)
columns*/
  ASSOCIATE RESULT SET LOCATORS (LOC1, LOC2) WITH PROCEDURE P1;
  ALLOCATE C1 CURSOR FOR RESULT SET LOC1;
  ALLOCATE C2 CURSOR FOR RESULT SET LOC2;
  SET AT_END=0;

  WHILE (AT_END = 0 ) DO /* processing result set #1 */
    FETCH C1 INTO column1, columns2;
    INSERT INTO RESULT(proc,res) VALUES ('result
1', column1 || columns2);
  END WHILE;

  CLOSE C1;

  SET AT_END=0;
```



```

        WHILE (AT_END = 0 ) DO /* processing result set #1 */
            FETCH C2 INTO column1, columns2;
            INSERT INTO RESULT(proc,res) VALUES ('result
1',column1|columns2);
        END WHILE;
        CLOSE C2;
END

```

**Important:**

From sections 2.4.4.2, “Retrieving result sets in the caller” on page 30 to 2.4.4.6, “Processing the result set” on page 32:

The OS/390 platform does not support result set locators yet (DECLARE *xx* RESULT SET LOCATOR, ASSOCIATE, ALLOCATE).

The example shown in section 2.4.4.2, “Retrieving result sets in the caller” on page 30 with CONTINUE HANDLER shows a compound statement in the handler, which is not supported on OS/390.

## 2.4.5 Handling errors in an SQL stored procedure

You cannot use the `WHENEVER` statement in an SQL stored procedure body. Instead, you can declare handlers to tell the SQL stored procedure what to do when an SQL error or an SQL warning occurs, or when no more rows are returned from a query. In addition, you can declare condition handlers for specific `SQLSTATE` values like '22000', '2300' or whatever you like.

You can refer to an `SQLSTATE` by its number in a condition handler, or you can declare a name for the `SQLSTATE`, then use that name in the condition handler.

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 after handling the condition.

The action specified in a condition handler can be any SQL statement, including a compound statement.

A condition handler gets executed automatically when a condition it is prepared to handle is detected anytime during the execution of the containing compound statement.

The general form of a handler declaration is:

```
DECLARE handler-type HANDLER FOR condition SQL-procedure-statement;
```

Conditions specified in a condition handler can be:

- `SQLSTATE` value
- Condition name ( user defined)
- `SQLEXCEPTION` (all `SQLSTATE` values with class other than 00, 01, or 02)
- `SQLWARNING` (all `SQLSTATE` values with class 01)
- `NOT FOUND` (all `SQLSTATE` values with class 02)

In general, the way that a handler works is that when an error occurs that matches the *condition*, the *SQL-procedure-statement* executes. When *SQL-procedure-statement* completes, DB2 performs the action that is indicated by the *handler-type*.

There are three types of handlers:

- CONTINUE  
Specifies that after *SQL-procedure-statement* completes, execution continues with the statement after the statement that caused the error.
- EXIT  
Specifies that after *SQL-procedure-statement* completes, execution continues at the end of the compound statement that contains the handler.
- UNDO  
Specifies that all the SQL statements done from the beginning of the containing compound statement, until the error point, are rolled back, and then, after the *SQL-procedure-statement* completes, execution continues at the end of the compound statement that contains the handler.

Figure 4 shows the behavior of each condition handler, when the *statement-2* reaches the condition:

- CONTINUE: after the execution of the *handler-action* is successful, the execution of the stored procedure will resume at the next statement (the CONTINUE point) which in this case is *statement-3*.
- EXIT: after the execution of the *handler-action* is successful, the execution of the stored procedure will resume at the end of the continuing compound statement (EXIT point).
- UNDO: after the execution of the *handler-action* is successful (including the rollback of statement 1), the execution of the stored procedure will resume at the UNDO point, the end of the continuing compound statement.

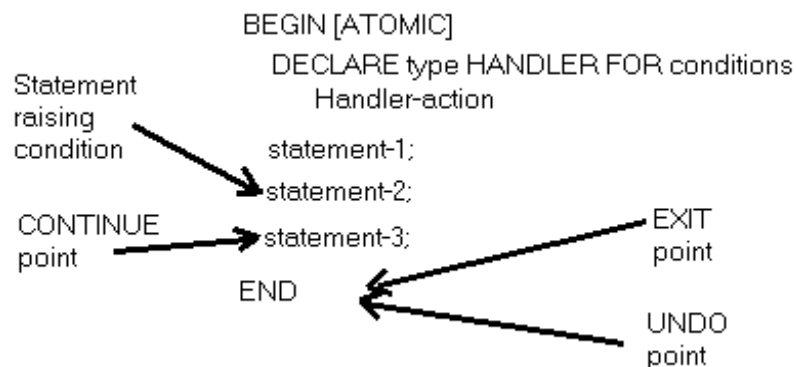


Figure 4. Behavior summary of different condition handlers

The following is a short example that will trigger an exception handler because the variable *mydate* is not correct (missing the *'* character).

```

CREATE PROCEDURE G3 ()
LANGUAGE SQL
BEGIN
    declare mydate DATE;
    DECLARE C1 condition for SQLSTATE '22007';
    DECLARE C2 condition for SQLSTATE '22017';
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        insert into result(proc,res)
values ('exec of G3','EXIT HANDLER fired:SQLSTATE='||SQLSTATE||'
SQLCODE='||CHAR(SQLCODE));

    DECLARE EXIT HANDLER FOR C1, SQLSTATE '22008', SQLSTATE '22006', C2
        insert into result(proc,res) values ('exec of G3','EXIT HANDLER
fired:SQLSTATE='||SQLSTATE||' SQLCODE='||CHAR(SQLCODE));

    insert into result(proc,res)
        values ('exec of G3','start of procedure');

    SET mydate = '12' || '13' || '99';

END

```

The following is another example that uses exception handlers.

```

CREATE PROCEDURE PSM031 (IN NUM_PARTS CHAR(10) )
LANGUAGE SQL
BEGIN
    DECLARE lc1,i, nb, lc2,lc1c2 integer default 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND BEGIN
        insert into result(proc,res)
            values ( 'exec from PSM031', 'NOT FOUND Handler fired');
    END;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN
        insert into result(proc,res)
            values ( 'exec from PSM031', 'SQLEXCEPTION handler fired
SQLCODE='||CHAR(SQLCODE));
    END;
    DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN
        insert into result(proc,res)
            values ( 'exec from PSM031', 'SQLWARNING handler fired');
    END;

    DECLARE cur1 CURSOR FOR Select c1,c2 from t1;
    delete from t1;
    insert into t1(c1,c2) values (1,1);
    insert into t1(c1,c2) values (1,2);
    set nb = (select count(*) from t1);
    OPEN cur1;
    set i = 0;
    while (i < nb) do
        FETCH FROM cur1 INTO lc1,lc2;
        insert into result(proc,res) values ( 'exec from PSM031',
            'row:'||CHAR(i)||' c1='|| CHAR(lc1) || ' c2='|| CHAR(lc2));
        set i=i+1;
    end while;
    CLOSE cur1;

END

```

**Important:** UNDO handlers are not yet supported on the OS/390 platform.

### 2.4.6 Current restrictions

The first release of SQL Procedures support on all DB2 platforms does not implement the following SQL/PSM features:

- Persistent modules
- Nested condition handlers
- Nested ATOMIC compound statements
- SQL functions

---

## 2.5 SQL Procedures portability

It is our intention to provide SQL Procedures support across the DB2 Universal Database family of products using a common SQL Procedures language and a common application development tool, the DB2 Stored Procedure Builder on every platform. We also expect that a significant number of customers will want to develop their stored procedures on an individual workstation, for example, a PC running Windows NT, using a local database, for example, DB2 for Windows NT; and as they move that application and its stored procedures into production, they will deploy it on DB2 for Windows NT on a different server or a DB2 for OS/390 platform.

Some customers will initially use their stored procedures against a departmental server running DB2 for AIX or DB2 for Sun Solaris, and then as usage scales up and more capacity is needed, they will consider moving their stored procedures to DB2 for OS/390. We will support these development and usage scenarios by providing common SQL Procedures language and Stored Procedure Builder support across all platforms.

Because of differences in release cycles and platform priorities, it is not always possible (or desirable) to deliver the same functionality on all platforms at exactly the same time. To help application developers identify and use the very large set of SQL Procedures language features that are common across all of the DB2 platforms, we have produced the following portability matrix (see Table 1). It is important to emphasize, however, that DB2 will be continuing to enhance the SQL Procedures language on all platforms and reduce any current platform differences.

Table 1 shows the potential differences between each of the IBM platforms.

Table 1. SQL Procedures portability across DB2 platforms

Functional Item	Windows, UNIX, and OS/2	OS/390	AS/400
Savepoint	Named savepoints. single, non-nested	Not supported in V5 Supported in V6 (Multiple)	Not yet supported
Source code size limit	64K size limit in 12/99	32K	32K

Functional Item	Windows, UNIX, and OS/2	OS/390	AS/400
Authorization behavior with dynamic SQL statements; will use Executor's authority for both DML and DDL	Supported	Supported	Supported only through RUNSQLSTM command
Nested NOT ATOMIC compound statement	Supported	Not yet supported	Not yet supported
Nested ATOMIC compound statements	Not yet supported	Not yet supported	Not yet supported
Nested stored procedure calls	Supported	Not supported in V5 Supported in V6	Supported
Returning result sets	Supported	Supported	Supported
ALLOCATE CURSOR statement for processing result sets from nested procedures	Supported	Not yet supported	Supported
Dynamic SQL statement within an SQL stored procedure	Supported (CALL statement supported in V8)	Supported	Supported
Multi-rowed result sets	Supported	Supported	Supported from a client via Client Access ODBC and Java Toolbox JDBC.  Supported from the server through CLI and JDBC.
COMMIT	Not yet supported	Not yet supported	Supported (only allowed for NOT ATOMIC procedures that are NOT invoked through DRDA)
ROLLBACK	Supported	Not yet supported	Supported (only allowed for NOT ATOMIC procedures that are NOT invoked through DRDA)
CONNECT	Not yet supported	Not Supported in V5 Supported in V6	Supported
FOR	Supported	Not yet supported	Supported
GRANT statement in procedure	Supported	Supported	Supported
REVOKE statement in procedure	Not yet supported	Supported	Supported
SIGNAL	Supported	Not yet supported	Not yet supported
RESIGNAL statement	Supported	Not yet supported	Not yet supported
Stand-alone SQLCODE/SQLSTATE	Supported	Supported	Supported

Functional Item	Windows, UNIX, and OS/2	OS/390	AS/400
CREATE PROCEDURE statements	Not yet supported	Supported	Supported
Static DDL	Supported	Supported	Supported
Single statement procedure	Supported	Supported	Supported
ITERATE statement (not in PSM-96)	Supported	Not yet supported	Not yet supported
GOTO (extension to the standard)	Not yet supported	Not yet supported	Not yet supported
C comment	Supported	Not yet supported	Supported
Overriding of PREP and compile options	Supported	Supported	Supported via options in Client Access. Supported on the RUNSQLSTM interface
New line stored in the catalog - consistency	Supported	Newline markers are stored in SYSIBM.SYSPSM for SPB to recover the initial "look" of the SP. The debugger will display PSM source as 80 byte wide lines (a DB2 OS/390 precompiler restriction)	Currently strip out all extra blanks and control characters. This is partially because the catalog column is VARCHAR(18432), but a procedure body can be larger than that.
Ambiguous names resolved in the following sequence: 1) Check if it is a column name (table exists), 2) Check if it is a SQL variable/parameter name; 3) Assume to be a column name (table does not exist and VALIDATE RUN option is used).	Supported	Not yet supported	Supported
Parameter names	128	8 characters in V5 18 characters in V6	128
Max length of character variable	254 bytes for VARCHAR 32 Kbytes for LONG VARCHAR	255 bytes	32 Kbytes
DATE arithmetic (for example: mydate+5 days)	Supported	Supported	Supported
SELECT statement on right hand side of SET statement	Supported	Not yet supported	Supported
DECIMAL data types	Not yet supported	Supported	Supported

---

## 2.6 New error messages

Below we provide a short description of the new error messages that will be provided by the precompiler when errors occur during the SQL stored procedure program preparation.

The error messages will be prefixed with the specific platform precompiler identification. For example, for error code -775, DB2 for OS/390 will send DSNH29775I.

-060 Data-type was specified in the definition of object-name. object-type in an SQL stored procedure parameter or variable. data-type is not supported for SQL stored procedure parameters or variables.

-061 One unexpected reason code was returned from Language Environment.

-775 In an SQL stored procedure, compound SQL statement contains an SQL stored procedure statement that is not allowed.

-776 In an SQL stored procedure, a FOR statement contains an OPEN, FETCH, or CLOSE statement for cursor *cursor-name*. Cursor operations are not allowed in FOR statement.

-777 An SQL stored procedure contains nested compound statements, which are not allowed.

-778 An SQL stored procedure statement contains an ending label and a beginning label that do not match.

-779 In an SQL stored procedure, the label on a LEAVE statement does not match the label for a block of code or loop that contains the LEAVE statement.

-780 An SQL stored procedure specifies an UNDO statement for a handler, and the ATOMIC statement was not specified.

-781 In an SQL stored procedure, a handler is declared for condition *condition-name*, but the SQL stored procedure does not contain a condition declaration statement that defines *condition-name*.

-782 In an SQL stored procedure, a condition handler is not valid for one of the following reasons:

- The handler specifies an SQLSTATE value that is not valid.
- The handler specifies duplicate conditions.
- The handler specifies SQLWARNING, SQLEXCEPTION, or NOT FOUND with other condition.

-783 An SQL stored procedure contains a FOR in which the select list in the cursor declaration has a column that is not valid. That column is a duplicate of another column in the select list, or the column is not named.

-785 In an SQL stored procedure, the name SQLCODE or SQLSTATE is used in one of the following invalid ways:

- An SQLCODE is declared as an SQL variable with a data type other than an INTEGER.

- An SQLSTATE is declared as an SQL variable with a data type other than CHAR(5).
- An SQLCODE or SQLSTATE is declared as an SQL variable with DEFAULT NULL.
- An SQLCODE or SQLSTATE is assigned the value NULL in an assignment statement.
- An SQLCODE or SQLSTATE is the same as an SQL stored procedure parameter.

**Important:**

- . Error code -061 is specific for OS/390 platform.
- . -775 is not defined for distributed platforms because they support nested compound statements.
- . -780 is not defined for OS/390 because UNDO is not currently supported on OS/390 platform.

---

## 2.7 Migrating from OEM DBMS

The SQL Procedures language is at the same conceptual level as the T/SQL of Sybase/Microsoft SQL Server, PL/SQL of Oracle, or SPL from Informix. Migrating the business logic, programmed in stored procedures language from another database vendor, to DB2 UDB SQL Procedures, should be easy. Most of the time, every stored procedure statement in the Oracle, Sybase, and Informix stored procedure language has an equivalent in the DB2 stored procedure language. Some differences in concepts, like error handling or result sets processing, may be the most difficult things to migrate.

But migration from another RDBMS implies many phases, not only business logic. The following sections briefly discuss all of these considerations before focusing on the business logic and stored procedures migration.

**Note:** Refer to the URL <http://www.ibm.com/solutions/softwaremigration>, for assistance with migrations.

### 2.7.1 Migrating the database structure

This is usually the easiest part. The data definition language between RDBMSs is quite similar. A few differences exist, though, most often on referential integrity constraint declarations, domain declarations, and storage management directives. There are tools that actually that convert structures rather well, with a minimum need for the DBA to intervene.

Some of these tools are:

- Platinum ERwin ERX v3.5. This one appears to be the favorite in this area.
- InfoModelers InfoModeler v3.1.
- DataJunction v6.5, which is primarily a data movement tool, but also provides support for DDL.



## 2.7.2 Migrating the database data

This can be pretty complex if transformation or conversion of the data is needed, because certain data types do not exist in the target RDBMS. This could result in complex and expensive operations, for instance on Binary Large Objects (BLOB). Another problem is the format of the export or import files. Every vendor uses different ones, and an extra conversion of the data files may be needed. Most of the time, import/export files have binary proprietary formats that allow fast load/unload of data for the same kind of RDBMS. They have to win load/unload benchmarks, and any trick is valuable. These binary files are usually incompatible between different RDBMS.

Fortunately, most of the actual RDBMSs can usually create SQL scripts, which means files with SQL statements, or tabulated text files, to transfer data to disk. This method is slower and requires more space on disk, but usually will work without too many manipulations. Some tools exist on the marketplace that handle data migration for you. Some of them even allow you to transfer and transform the data before saving it to disk. This can be really helpful if you have to change data types or format between the two RDBMS.

Some of these tools are:

- DataJunction v6.5.
- IBM DataJoiner allows you to copy data from any RDBMS (Sybase, Oracle, Informix) seamlessly into your DB2 UDB, exactly as if all the other databases were all on the same machine. Using it can be a good solution for long projects, where customers have to deal with different RDBMS for a long period of time.
- SQL Conversion Workbench (SQL-CW), from Mantech Systems Solutions Corporation, allows you migrate database objects and contents.

## 2.7.3 Migrating the business logic

This is the most complex part, depending on how much business logic has been developed in a language different from the target one.

### 2.7.3.1 The client applications

If the client application relies on a 4GL tool such as PowerBuilder or SQL Windows, which accepts different RDBMSs, it is usually enough to change the data source in the project, recompile the application, and fix the errors that show up. But sometimes, differences in the SQL syntax, incompatibilities between data types for parameters and variables, transaction control, and locking model may be difficult issues to solve, and may require longer investigations to find solutions that work on the new RDBMS.

There are some migration processing tools available that accept one language such as T/SQL and generate stored procedures in another language such as SQL Procedures. These tools generally perform some significant percentage of the work automatically but typically require some manual effort for conversion as well.

Proprietary 4GL tools do not usually connect to other databases, such as Oracle FORMS, and migrating that kind of application may require more manual effort for conversion.

If the client application uses a 3GL tool like C with ODBC (standard API) it will probably be relatively easy to migrate to DB2 UDB. That is because DB2 UDB supports ODBC as well as supporting IBM CLI (an IBM ODBC layer that has the same APIs as ODBC, but is optimized for DB2 UDB). In fact, ODBC was designed to be insensitive to different RDBMSs. Unfortunately, because of the different implementations by the ODBC resellers, most ODBC flavors have minor portability issues.

If the client application was developed using Oracle proprietary APIs, or proprietary Sybase APIs (CT-LIB) or some other proprietary API on other RDBMSs, then more work will have to be done, that is, the part of the application that deals with those API will have to be rewritten, because the API calls will probably not have any corresponding calls in DB2 UDB APIs.

Because this is a lot of work, tools should be available to ease the process of converting client applications. Unfortunately, no tools really do this kind of conversion automatically. It is really too difficult, and human intervention is required.

### 2.7.3.2 The stored procedures

Other RDBMS vendors did not choose the SQL Procedures language, which is a standard. Since each database vendor has a different implementation of stored procedure programming languages, migration of stored procedures can become an important issue, depending on the extent of them in the application. The future of stored procedures in the area of relational databases is currently at a turning point, as Java is being considered as a possible solution to the portability and flexibility. Java is a full programming language, and people who are familiar with writing in simple languages such as PL/SQL or T/SQL prefer to move to another language like SQL procedures language that is much easier to learn. For many customers, Java is an interesting programming language for experienced programmers, but is not necessarily for developers who only know T/SQL, for example.

It is still much easier and faster to migrate to SQL stored procedures from other RDBMS than migrating to C or Java external stored procedures. Not only will the training cost of the teams involved in development be reduced (a few hours are enough to learn the SQL Procedures language if you know T/SQL or PL/SQL), but also, the development cost itself will go faster by reducing the amount of code written.

To ease the migration, some help can be found with tools like the SQL Conversion Workbench (SQL-CW) from Mantech Systems Solutions Corporation, that can translate most simple procedures from other RDBMSs. For more complex stored procedures, which often rely heavily on engine features, it may sometimes be necessary to redesign the application itself, instead of spending time to mimic a behavior that cannot be created on the new DB2 UDB engine. (See Table 2.)

Table 2. RDBMS Stored Procedure language comparison

RDBMS Vendor	Language	Benefits	Drawbacks
Oracle V7.x / V8.x	PL/SQL	ease of use used in Ad tools (Forms)	proprietary, flexibility

RDBMS Vendor	Language	Benefits	Drawbacks
Sybase SQL Server V 10.x & V11.x	Transact SQL	ease of use	proprietary, flexibility
Microsoft SQL Server V6.5 / V7.0	Transact SQL	ease of use	proprietary, flexibility
Informix v7.x	Informix SPL	ease of use	proprietary, flexibility
DB2 UDB V6.x / DB2 Server for OS/390 V5 SQL stored procedure	SQL Procedures language	ease of use, standard compliant	flexibility
DB2 UDB V5.x / V6.x external SP	C, COBOL, Java	flexibility, power	skill require, complex

Migrating the control statements from other RDBMS stored procedures languages manually is usually not that difficult. The most difficult part of this is dealing with engine-specific features not available on the target platform.

For example, until recently, temporary tables like those found in Sybase and Informix (check DECLARE GLOBAL TEMPORARY TABLE statement) were not supported in any version of DB2 UDB on any platform, and simulating them with base tables can lead to poor performance in the migrated applications because of the catalog contention. This can also lead to expensive development costs during the migration phase, sometimes requiring a thorough redesign of the business logic itself, and resulting in a slower implementation of the new application.

To lower the cost of migration from other RDBMSs, DB2 UDB version 7.x will implement a full set of new features in the engine, such as temporary tables, savepoints, identity columns, and nested stored procedure calls, which will be incorporated in the new releases.

Although all these new features will ease migration, some stored procedures logic will have to be redesigned to fit in the new engine. That is, there may be changes needed in naming conventions or reengineering what modules work best for a given RDBMS.

#### 2.7.4 Comparison with Sybase/Microsoft SQL Server Transact-SQL

The SQL Procedures language is very similar to T/SQL, and in fact, most of the T/SQL control flow statements represent a subset of the SQL Procedures language control flow statements. However, there are some differences; for example, T/SQL does not have support for:

- Error handler statements
- LOOPS
- FOR loops
- REPEAT UNTIL
- ELSEIF
- ATOMIC blocks

Table 3 shows the SQL/PSM (standard) control statements and Sybase/Microsoft T/SQL control statements.

Table 3. Comparison between SQL/PSM and T/SQL control statements

SQL/PSM control statements	Sybase/Microsoft SQL Server T/SQL control statements
CALL	EXECUTE
Result set processing after a CALL statement	No equivalent
LEAVE <procedure-body-name>	RETURN
BEGIN (Compound statements) END	BEGIN END
Handler declaration	No equivalent
Condition declaration	No equivalent
SQL variable declaration declare <var-name> <datatype> default <defvalue>	declare <var-name> <datatype>
Assignment declaration SET <var-name> = <expression>	SELECT <var-name> = <expression>
IF <expression> THEN...ELSEIF...ELSE...END IF;	IF <expression> ...ELSE... (no THEN, no ELSE IF, no END IF)
CASE statement	No equivalent
LEAVE statement	BREAK
LOOP	No equivalent
WHILE statement	WHILE statement
REPEAT	No equivalent
FOR statement	No equivalent equivalent done by tricks with WHILE, temporary table and set rowcount=1 and 0.
SIGNAL <sqlstate> statement	RAISERROR <error-number>
RESIGNAL statement	No equivalent
ITERATE	Continue
GET DIAGNOSTICS <varname>=ROWCOUNT	SELECT <varname>=rowcount
No equivalent	goto <label>
No equivalent	waitfor DELAY 'time' waitfor TIME 'time'
No equivalent	SET options

Table 4 shows a quick comparison between DB2 and Sybase/Microsoft SQL Server.

Table 4. Comparison between DB2 and Sybase SQL Server

DB2	Sybase/Microsoft SQL Server
ATAN	ATAN
-- comment	-- comment
/* comment */	/* comment */
ABS	ABS
ABSVVAL	ABS
ACOS	ACOS
AND	AND
ASCII	ASCII
ASIN	ASIN
ATAN2	ATAN2
AVG	AVG
BLOB	CONVERT(IMAGE, ... )
CALL statement	execute
CEILING	CEILING
CHAR	CONVERT(CHAR,...)
CLOB	CONVERT(TEXT, ...)
CLOSE	CLOSE
COALESCE	ISNULL(...,...)
COMMIT (no nested commits)	COMMIT savepoint_name
Compound ATOMIC statement	Begin tran xx.... commit tran xx
Compound SQL statement	Begin ... End block
COS	COS(:1P)
COT	COT(:1P)
COUNT	COUNT
COUNT DISTINCT	COUNT DISTINCT
Create temporary table ***	Select <expression> into table
cursor with hold	HOLDLOCK
DATE	CONVERT(DATETIME, ...)
DAY	DATEPART(DAY,CONVERT(DATETIME,...))
DAYNAME	DATENAME(DAY, CONVERT(DATETIME,...))
DAYOFWEEK	DATEPART(DAY,CONVERT(DATETIME,...))

DB2	Sybase/Microsoft SQL Server
DAYOFYEAR	DATEPART(YEAR, CONVERT(DATETIME,...))
DB2 allows setting transaction isolation level to RR, RS, CS, UR at package bind	SET TRANSACTION isolation level <n>
DB2 ordinarily runs under transaction control	BEGIN Transaction <transaction name> or <savepoint name>
DDL for create table, create index, grant, revoke, .....*	DDL for create table, create index, grant, revoke, .....
DECIMAL	CONVERT(DECIMAL(...),...)
DECIMAL	CONVERT(DECIMAL(...),...)
DECLARE Cursor	DECLARE Cursor
DEGREES	DEGREES(...)
DELETE	DELETE
DELETE WHERE CURRENT OF cursor	delete where current of
DIFFERENCE	DIFFERENCE(...,...)
DOUBLE_PRECISION	CONVERT(FLOAT(2),...)
EXP	EXP(...)
FETCH	FETCH
FLOOR	FLOOR(...)
GROUPBY (ROLLUP(...))	COMPUTE
HOUR	DATEPART(HOUR, CONVERT(DATETIME,...))
IDENTITY COLUMN	IDENTITY COLUMNS
IF cd1 THEN St1 ELSE St2 END IF	IF cd1 THEN st1 ELSE st2
INSERT	INSERT
INT	CONVERT(INT,...)
IS NOT NULL	IS NOT NULL
IS NULL	IS NULL
ITERATE	continue
LCASE	LOWER(...)
LEAVE statement	break
LENGTH	DATALENGTH(...)
LIKE	LIKE
LN	LOG(...)
LOCATE	PATINDEX(...,...)
LOG	LOG(...)
LOG10	LOG10(...)

DB2	Sybase/Microsoft SQL Server
LTRIM	LTRIM(...)
MAX	MAX
MICROSECOND	DATEPART(MILLISECOND, ...)
MIN	MIN
MINUTE	DATEPART(MINUTE, CONVERT(DATETIME, ...))
MOD	%
MONTH	DATEPART(MONTH, CONVERT(DATETIME,...))
MONTHNAME	DATENAME(MONTH,CONVERT(DATETIME,...))
NOT	NOT
OPEN	OPEN
OR	OR
POSSTR	CHARINDEX(...,...)
QUARTER	DATEPART(QUARTER,CONVERT(DATETIME,...))
RADIANS	RADIANS(...)
REPEAT	REPLICATE(...,...)
RETURN statement	RETURN integer
RIGHT	RIGHT
ROLLBACK TO SAVEPOINT <savepoint-name>	ROLLBACK <savepoint-name>
ROLLBACK WORK	ROLLBACK WORK
RTRIM	RTRIM
SAVEPOINT <savepoint-name>**	SAVE transaction savepoint-name
SECOND	DATEPART(SECOND, CONVERT(DATETIME,...))
SELECT	SELECT
SET statement(into many vars)	SELECT var1=column1,var2=columns2,....
SET var=statement	SELECT var = statement
SIGN	SIGN
SIGNAL sqlstate	RAISERROR error-number
SIN	SIN
SMALLINT	CONVERT(SMALLINT, ...)
SOUNDEX	SOUNDEX
SPACE	SPACE
SQL variable declaration	declare @variable-name data-type
SQRT	SQRT
STDDEV	No equivalent

DB2	Sybase/Microsoft SQL Server
SUM	SUM
TAN	TAN
TIME	CONVERT(DATETIME, ...,108)
UCASE	UPPER
UNION	UNION
UPDATE	UPDATE
UPDATE WHERE CURRENT OF cursor	update where current of
VARCHAR	CONVERT(VARCHAR, ...)
VARIANCE	No equivalent
WEEK	DATEPART(WEEK, CONVERT(DATETIME,...))
WHILE statement	while expression statement
YEAR	DATEPART(YEAR, CONVERT(DATETIME,...))
— (string concat)	+
No equivalent	@@rowcount
No equivalent	bitwise operations ( ^ &   ~ )
No equivalent	deallocate
No equivalent	goto label
No equivalent	LIKE 'regular_expression'
No equivalent	PRINT ...
No equivalent	SET ROWCOUNT

### 2.7.5 Comparison with Oracle PL/SQL

The SQL Procedures language is also very similar to PL/SQL. But, just as with T/SQL, there are some differences between SQL Procedures and PL/SQL; for example:

- PL/SQL does not have the same error handling, although it does have exceptions
- Does not support compound atomic
- Does not have result sets, although can still use the dbms\_output package to send information back to the client application
- PL/SQL has features such as:
  - %TYPE,
  - %ROWTYPE,
  - %TABLE,
  - SQL%ROWCOUNT,
  - SQL%FOUND,
  - SQL%NOTFOUND,



- ROWID
- Packages (that should look similar to SQL PSM modules)

Table 5 shows the SQL/PSM (standard) control statements and Oracle PL/SQL control statements.

Table 5. Comparison between SQL/PSM and PL/SQL control statements

SQL/PSM control statements	Oracle PL/SQL control statements
CALL	No equivalent
result set processing after a CALL statement	No equivalent
LEAVE <procedure-body-name>	RETURN
BEGIN (Compound statements) END	BEGIN END
Handler declaration	EXCEPTION (differences)
Condition declaration	exception-name EXCEPTION PRAGMA EXCEPTION_INIT(exception-name,error-number) (no SQLSTATE values)
SQL variable declaration declare <var-name> <datatype> default <defvalue>	declare <var-name> <datatype>
Assignment declaration SET <var-name> = <expression>	<var-name> = <expression>
IF <expression> THEN ... ELSEIF...THEN ... ELSE... END IF;	IF <expression> THEN.. ELSIF...THEN... ELSE... END IF;
CASE statement	No equivalent
LEAVE statement	EXit <label>
LOOP	LOOP
WHILE statement	WHILE statement or FOR <indx-name> IN lowerbound..upperbound LOOP.....END LOOP
REPEAT	No equivalent
FOR statement	FOR <record-name> IN <cursor-name> LOOP .... END LOOP
SIGNAL <sqlstate> statement	RAISE <error-number>
RESIGNAL statement	RAISE
ITERATE	continue
GET DIAGNOSTICS <varname>=ROWCOUNT	<varname>=SQL%ROWCOUNT
No equivalent	goto <label>

Table 6 shows a comparison between DB2 and Oracle:

Table 6. Comparison between DB2 and Oracle

DB2	Oracle
-- comment	-- comment
/* comment */	/* comment */
ABS	ABS
ABSVAL	ABS
ACOS	ACOS
AND	AND
ASCII	ASCII
ASIN	ASIN
ATAN	ATAN
ATAN2	ATAN2
AVG	AVG
AVG	AVG
BLOB	LONG()
BLOB	RAW()
CALL statement	CALL statement
CASE (expr) WHEN..THEN..ELSE..END	DECODE( expr,.....,....)
CEILING	CEIL
CHAR()	CHAR()
CHAR()	TO_CHAR()
CLOB	LONG()
CLOSE	CLOSE
COALESCE	ISNULL(..., ...)
COMMIT (no nested commits)	COMMIT savepoint_name
Compound ATOMIC statement	Begin tran xx.... commit tran xx
Compound SQL statement	Begin ... End block
CONTINUE/EXIT/UNDO handlers	EXCEPTION
COS	COS
COT	COT
COUNT	COUNT
COUNT DISTINCT	COUNT DISTINCT
cursor with hold	HOLDLOCK
DATE	TO_DATE()

DB2	Oracle
DAY	TO_CHAR(....,DD)
DAYNAME	TO_CHAR(....,DAY)
DAYOFWEEK	TO_CHAR(....,D)
DAYOFYEAR	TO_CHAR(....,DY)
DB2 allows setting transaction	SET TRANSACTION
DB2 ordinarily runs under transaction control	BEGIN Transaction <transaction name> or <savepoint name>
DDL for create table, create index, grant, revoke, .....*	DDL for create table, create index, grant, revoke, .....
DECIMAL ()	DECIMAL()
DECIMAL ()	NUMBER()
DECIMAL()	DEC()
DECLARE Cursor	DECLARE Cursor
DEGREES	DEGREES
DELETE	DELETE
DELETE WHERE CURRENT OF cursor	delete ... where current of
DOUBLE()	DOUBLE_PRECISION()
EXP	EXP
FETCH	FETCH
FLOAT()	FLOAT()
FLOOR	FLOOR
FOR rec IN csr LOOP ...END LOOP	FOR rec IN csr LOOP ...END LOOP
GENERATE_UNIQUE (slightly different)	ROWID
GROUPBY (ROLLUP(...))	No equivalent
HOUR	TO_CHAR(....,HH)
IDENTITY COLUMN	CREATE SEQUENCE xx START WITH yy
IF cd1 THEN St1 ELSE St2 END IF	IF cd1 THEN st1 ELSE st2 END IF
INSERT	INSERT
INT	BINARY_INTEGER()
IS NOT NULL	IS NOT NULL
IS NULL	IS NULL
isolation level to RR,RS,CS,UR at	No equivalent
LCASE	LOWER
LEAVE	EXIT
LEAVE statement	EXIT WHEN ...

DB2	Oracle
LENGTH	LENGTH()
LIKE	LIKE
LN	LN
LOCATE	INSTR()
LOG	LOG
LOG10	LOG10
LOOP.... END LOOP	LOOP.... END LOOP
LTRIM	LTRIM
MAX	MAX
MIN	MIN
MINUTE	TO_CHAR(....,MI)
MOD	MOD
MONTH	TO_CHAR(....,MM)
MONTHNAME	DATENAME(MONTH,CONVERT(DATETIME,...))
NOT	NOT
OPEN	OPEN
OR	OR
package bind	No equivalent
POWER	**
POWER	POWER
QUARTER	DATEPART(QUARTER,CONVERT(DATETIME,...))
RADIANS	RADIANS
REPLACE	REPLACE
RESIGNAL	RAISE
RETURN statement	return integer
RIGHT	RIGHT
ROLLBACK TO SAVEPOINT <savepoint-name>	ROLLBACK <savepoint-name>
ROLLBACK WORK	ROLLBACK WORK
ROUND(m,n)	ROUND(m,n)
RTRIM	RTRIM
SAVEPOINT <savepoint-name>**	SAVE transaction savepoint-name
SECOND	TO_CHAR(....,SS)
SELECT	SELECT
SET var=(statement)	var := (statement)

DB2	Oracle
SIGN	SIGN
SIGNAL	RAISE exception_name
SIN	SIN
SINH	SINH
SMALLINT	SMALLINT
SOUNDEX	SOUNDEX
SPACE	SPACE
SQL variable declaration	declare variable-name data-type
SQRT	SQRT
STDDEV(exp)	STDDEV(exp)
SUBSTR	SUBSTR
SUM	SUM
TAN	TAN
TANH	TANH
TIME	TO_DATE()
TRANSLATE()	TRANSLATE()
UCASE	UPPER
UNION	UNION
UPDATE	UPDATE
UPDATE WHERE CURRENT OF cursor	update ... where current of
USER	UID
VALUE(exp1,exp2)	NVL(exp1,exp2)
VARCHAR()	VARCHAR2()
VARIANCE	VARIANCE
WEEK	DATEPART(WEEK, CONVERT(DATETIME,...))
WHILE statement	WHILE expression statement
YEAR	TO_CHAR(...,IYY), TO_CHAR(...,YYYY)
— (string concat)	( string concat)
No equivalent	%ISOPEN
No equivalent	%TYPE
No equivalent	BOOLEAN()
No equivalent	COSH()
No equivalent	COUNT
No equivalent	CREATE PACKAGE .....

<b>DB2</b>	<b>Oracle</b>
No equivalent	CURRVAL
No equivalent	dbms_output_package.print()
No equivalent	deallocate
No equivalent	define RECORD Type
No equivalent	DELETE
No equivalent	EXISTS
No equivalent	FIRST
No equivalent	GREATEST or GREATEST_LB
No equivalent	HEXTORAW()
No equivalent	INITCAP()
No equivalent	LAST
No equivalent	LAST_DAY
No equivalent	LEAST or LEAST_UB
No equivalent	LEVEL
No equivalent	LPAD
No equivalent	MONTHS_BETWEEN
No equivalent	nested RECORD
No equivalent	NEW_TIME
No equivalent	NEXT
No equivalent	NEXT_DAY
No equivalent	NEXTVAL
No equivalent	NLSSORT
No equivalent	PRINT ...
No equivalent	PRIOR
No equivalent	ROUND(date)
No equivalent	RPAD()
No equivalent	SINH()
No equivalent	SQL%FOUND
No equivalent	SQL%NOTFOUND
Get DIAGNOSTIC <varname> = ROWCOUNT	SQL%ROWCOUNT
No equivalent	TANH()
No equivalent	CONVERT()

## 2.7.6 Comparison with Informix SPL

Table 7 shows the SQL/PSM (standard) control statements and Informix SPL control statements.

Table 7. Comparison between SQL/PSM and SPL control statements

SQL/PSM control statements	Informix SPL control statements
CALL	CALL EXECUTE
result set processing after a CALL statement	No equivalent
LEAVE <i>&lt;procedure-body-name&gt;</i>	RETURN
BEGIN (Compound statements) END	BEGIN END
Handler declaration	ON EXCEPTION (but differences)
Condition declaration	No equivalent
SQL variable declaration declare <i>&lt;var-name&gt;</i> <i>&lt;datatype&gt;</i> default <i>&lt;defvalue&gt;</i>	define <i>&lt;var-name&gt;</i> <i>&lt;datatype&gt;</i>
Assignment declaration SET <i>&lt;var-name&gt;</i> = <i>&lt;expression&gt;</i>	LET <i>&lt;var-name&gt;</i> = <i>&lt;expression&gt;</i>
IF <i>&lt;expression&gt;</i> THEN ... ELSEIF...THEN ... ELSE... END IF;	IF <i>&lt;expression&gt;</i> THEN.. ELIF...THEN... ELSE... END IF;
CASE statement	No equivalent
LEAVE statement	EXIt <i>&lt;loop-type&gt;</i>
LOOP	No equivalent
WHILE statement	WHILE statement or FOR <i>&lt;indx-name&gt;</i> IN <i>&lt;criteria&gt;</i> .... END FOR
REPEAT	No equivalent
FOR statement	FOREACH <i>&lt;cursor-name&gt;</i> FOR <i>&lt;select-stmt&gt;</i> ....END FOREACH
SIGNAL <i>&lt;sqlstate&gt;</i> statement	RAISE EXCEPTION <i>&lt;error-number&gt;</i>
RESIGNAL statement	RAISE
ITERATE	CONTINUE
No equivalent	SYSTEM
No equivalent	goto <i>&lt;label&gt;</i>





---

## Chapter 3. The DB2 Stored Procedure Builder

This chapter describes the new DB2 Stored Procedure Builder (SPB) tool, and explains how it can be used to help with the development and maintenance of stored procedures for both DB2 for OS/390 and DB2 Universal Database (UDB) environments.

---

### 3.1 DB2 Stored Procedure Builder — overview

This section describes the DB2 SPB, its main components, prerequisites, installation, and customization.

#### 3.1.1 What is it?

The DB2 SPB is a graphical tool designed to help with the development of DB2 stored procedures. It provides all the functions required to create, build, test, and deploy new stored procedures. It also provides functions to work with existing stored procedures.

The SPB provides a single development environment that supports the entire DB2 family ranging from the workstation to System/390. Figure 5 shows the environment for development and deploy stored procedures with SPB. With the SPB you can focus on the logic of your stored procedure rather than on the process details of creating stored procedures on a DB2 server.

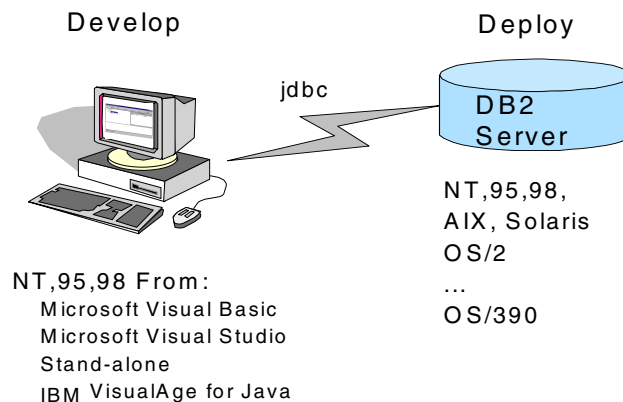


Figure 5. SPB environment

The SPB supports two languages for the development of new stored procedures: Java, and SQL Procedures language.

It is important to notice that the SPB is not a prerequisite to write stored procedures for DB2 servers. The support for stored procedures, even for the SQL Procedures language, is built-in to the DB2 base code. For more details on the new SQL Procedures language, refer to Chapter 2, "The SQL Procedures language" on page 9. For details on the implementation of SQL stored procedures in different DB2 servers, refer to Chapter 4, "SQL Procedures for DB2 UDB for OS/390" on page 109, Chapter 5, "SQL Procedures for DB2 UDB for UNIX, Windows, OS/2" on page 145, and Chapter 6, "SQL Procedures for DB2 UDB for AS/400" on page 169.

In summary, using the SPB you can perform a variety of tasks associated with stored procedures, such as:

- Creating new stored procedures
- Listing existing stored procedures
- Modifying existing stored procedures (Java and SQL stored procedures)
- Running existing stored procedures
- Copying and pasting stored procedures across connections
- One-step building of stored procedures on target databases
- Customizing the settings to enable remote debugging of installed stored procedures

### 3.1.2 Programming languages supported

The SPB supports both Java and SQL Procedures languages to create, build, and debug stored procedures on DB2 UDB servers. For Java stored procedures, SPB allows the use of both supported Java interfaces: JDBC and SQLJ. The main difference between JDBC and SQLJ is that JDBC stored procedures execute as a dynamic SQL program, while SQLJ stored procedures execute as a static SQL program.

Following are examples of the same stored procedure written in SQL Procedures language, Java with JDBC, and Java with SQLJ.

#### Procedure STP using SQL Procedures language:

```
CREATE PROCEDURE DRDARES1.STP ( IN v_id int )
    SPECIFIC DRDARES1.S5521448
    RESULT SETS 1
    LANGUAGE SQL

-----
-- SQL stored procedure DRDARES1.STP
-----

P1: BEGIN
    -- Declare cursor
    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT
            DRDARES1.STAFF.ID,
            DRDARES1.STAFF.NAME,
            DRDARES1.STAFF.DEPT,
            DRDARES1.STAFF.JOB,
            DRDARES1.STAFF.YEARS,
            DRDARES1.STAFF.SALARY,
            DRDARES1.STAFF.COMM
        FROM
            DRDARES1.STAFF
        WHERE
            (
                (
                    DRDARES1.STAFF.ID > v_id
                )
            )
        );

    -- Cursor left open for client application
    OPEN cursor1;
```

END P1

### Procedure STP using JDBC:

```
/**
 * JDBC Stored Procedure DRDARES1.STP
 */
import java.sql.*;          // JDBC classes

public class STP
{
    public static void sTP ( short v_id,
                            ResultSet[] rs ) throws SQLException, Exception
    {
        // Get connection to the database
        Connection con =
        DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement stmt = null;
        String sql;

        sql = "SELECT"
            + "      DRDARES1.STAFF.ID,"
            + "      DRDARES1.STAFF.NAME,"
            + "      DRDARES1.STAFF.DEPT,"
            + "      DRDARES1.STAFF.JOB,"
            + "      DRDARES1.STAFF.YEARS,"
            + "      DRDARES1.STAFF.SALARY,"
            + "      DRDARES1.STAFF.COMM"
            + " FROM"
            + "      DRDARES1.STAFF"
            + " WHERE"
            + "      ("
            + "          ( "
            + "              DRDARES1.STAFF.ID > ? "
            + "          )"
            + "      )";

        stmt = con.prepareStatement( sql );
        stmt.setShort( 1, v_id );
        rs[0] = stmt.executeQuery();
        if (con != null) con.close();
    }
}
```

### Procedure STP using SQLJ:

```
/**
 * SQLJ Stored Procedure DRDARES1.STP
 */
import java.sql.*;          // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator Stp_Cursor1 ( short, String, short, String, short,
java.math.BigDecimal, java.math.BigDecimal );

public class Stp
{
    public static void stp ( short v_id,
```

```

                                                                    ResultSet[] rs ) throws SQLException,
Exception
{
    Stp_Cursor1 cursor1 = null;
    #sql cursor1 =
    {
        SELECT
            DRDARES1.STAFF.ID,
            DRDARES1.STAFF.NAME,
            DRDARES1.STAFF.DEPT,
            DRDARES1.STAFF.JOB,
            DRDARES1.STAFF.YEARS,
            DRDARES1.STAFF.SALARY,
            DRDARES1.STAFF.COMM
        FROM
            DRDARES1.STAFF
        WHERE
            (
                (
                    DRDARES1.STAFF.ID > :v_id
                )
            )
    };
    rs[0] = cursor1.getResultSet();
}
}

```

**Note:** For DB2 for OS/390 servers, SPB supports only the SQL Procedures language. SPB does not yet support Java stored procedures for DB2 for OS/390, and it does not yet support SQL Procedures for DB2 UDB for AS/400.

Existing stored procedures written in any supported language, and registered in the DB2 server, are also listed. However, stored procedures written in other languages can only be executed from SPB. You will not be able to modify, build, get source for, or debug these procedures.

---

## 3.2 Product Installation on Windows NT

The following sections describe the prerequisites and the steps required to install SPN in the Windows NT environment.

### 3.2.1 Prerequisites for SPB

There are a few requirements to run the SPB in your workstation. If you plan to develop stored procedures for remote DB2 servers, you have to define the connection to the server. To access DB2 UDB servers, you only need to catalog the remote database in your workstation.

To access a DRDA server (only DB2 for OS/390 is supported at this time), you need to install the DB2 Connect product in your workstation or in a gateway, and catalog the remote database. To work with SPB and DB2 Server for OS/390 Version 5, you must also ensure that APARs PQ29866, PQ24199, and PQ29706 are installed in your system.

SPB is implemented in Java, and all database connections are managed by using Java Database Connectivity (JDBC). To write stored procedures with SPB, you only need to be able to connect to a local or remote DB2 database alias.

### 3.2.2 Installing the SPB

The SPB is included with the DB2 Software Developer's Kit (SDK) and is available for the Windows 95, 98, and NT environments. The DB2 SDK is included with DB2 UDB server editions (Workgroup Edition, Enterprise Edition, and Enterprise-Extended Edition), DB2 Developers editions (Personal Edition, and Universal Edition), and DB2 Connect. You can install SDK together with DB2 UDB server or in a separate client workstation.

DB2 for OS/390 users can also get the SPB from the DB2 Management Tools Package.

Although the SPB executes only in Windows systems, it can be used to develop stored procedures for any DB2 UDB platform and for DB2 for OS/390. Support for the development of DB2 UDB for AS/400 stored procedures should be available soon.

If you are using SPB to develop stored procedures for a DB2 UDB server, before you run SPB, you must configure the DB2 UDB server in the following ways:

- Set the path for the Java Development Kit (JDK) by entering the following command from the DB2 Command Window:

```
DB2 UPDATE DATABASE MANAGER CONFIGURATION USING JDK11_PATH  
x:\sql1lib\java\jdk
```

where x: is the drive on which you installed DB2 UDB

- We recommend that you set the Java heap size to 4096 bytes. From the DB2 Command Window, enter the following command:

```
DB2 UPDATE DATABASE MANAGER CONFIGURATION USING JAVA_HEAP_SZ 4096
```

- We recommend that you set the application heap size to 1024 bytes. From the DB2 Command Window, enter the following command:

```
DB2 UPDATE DATABASE MANAGER CONFIGURATION FOR database_name USING  
APPLEHEAPSZ 1024
```

- Set the DB2 parameter KEEPDAIRI to NO if you are frequently rebuilding and testing stored procedures. In the DB2 Command Window, enter the following command:

```
DB2 UPDATE DATABASE MANAGER CONFIGURATION USING KEEPDAIRI NO
```

For these configuration settings described above, you must stop and restart the database server before the new settings will take effect.

If you are planning to use SPB to develop stored procedures for a DB2 for OS/390 server, see Chapter 4, "SQL Procedures for DB2 UDB for OS/390" on page 109 for detailed information.

When you install the SDK in your machine, a path to SPB is automatically included in the DB2 UDB program group. You can launch the SPB from the DB2 UDB program group, or you can launch SPB as an add-in tool from any of the following applications:

- IBM VisualAge for Java
- Microsoft Visual Studio Version 5 and Version 6
- Microsoft Visual Basic Version 5 and Version 6

### 3.2.2.1 Starting SPB from IBM VisualAge for Java

The integration of SPB and VisualAge for Java that is currently available is: VisualAge for Java 3.0. The SPB that is integrated with VA Java 3.0 is the DB2 UDB version 6 code base.

Follow the steps below to invoke the Stored Procedure Builder driver:

1. Right-click the VisualAge for Java project you want to use the Stored Procedure Builder with.
2. Select Tools from the menu.
3. Select IBM DB2 Stored Procedure Builder.

SPB projects save connection information and stored procedure objects that have not yet been built to a database.

Figure 6 shows how to invoke the SPB through VA Java Workbench.

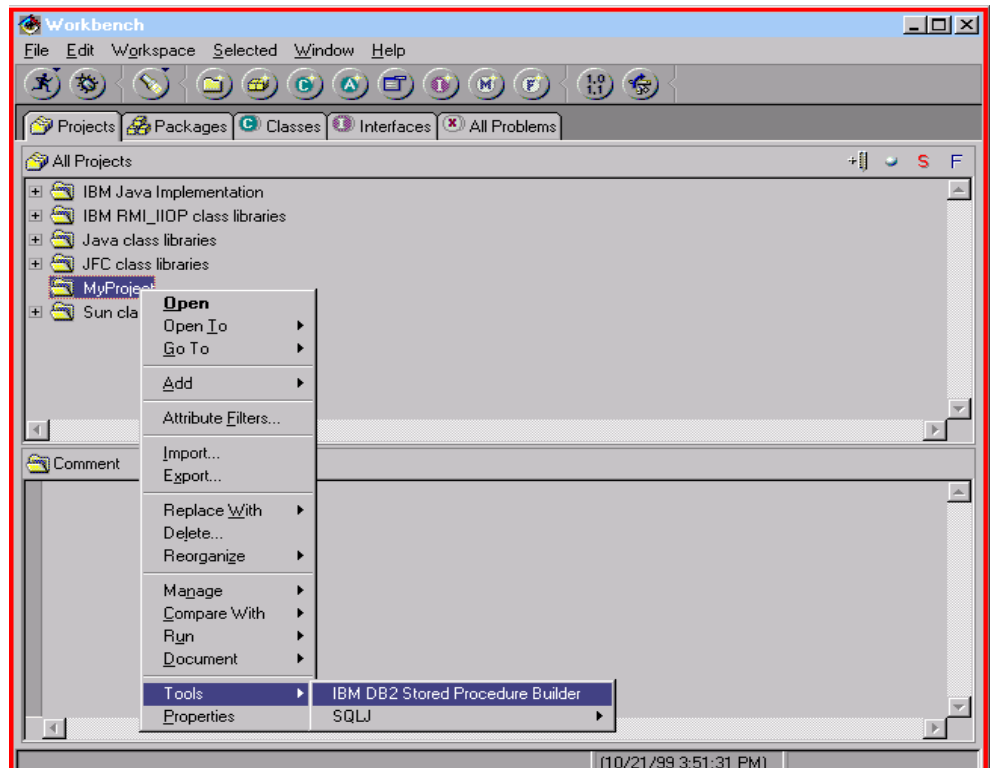


Figure 6. Invoking SPB through VisualAge for Java

If the SPB project file is NOT found in the VisualAge for Java that you use to invoke SPB, then the "Specify Database Connection" dialog appears (Figure 7).

If the SPB project file is found in the VisualAge for Java that you use to invoke SPB, then the "Specify Database Connection" does not appear (since we use the

connection information stored in the SPB project file found in the VA Java project).

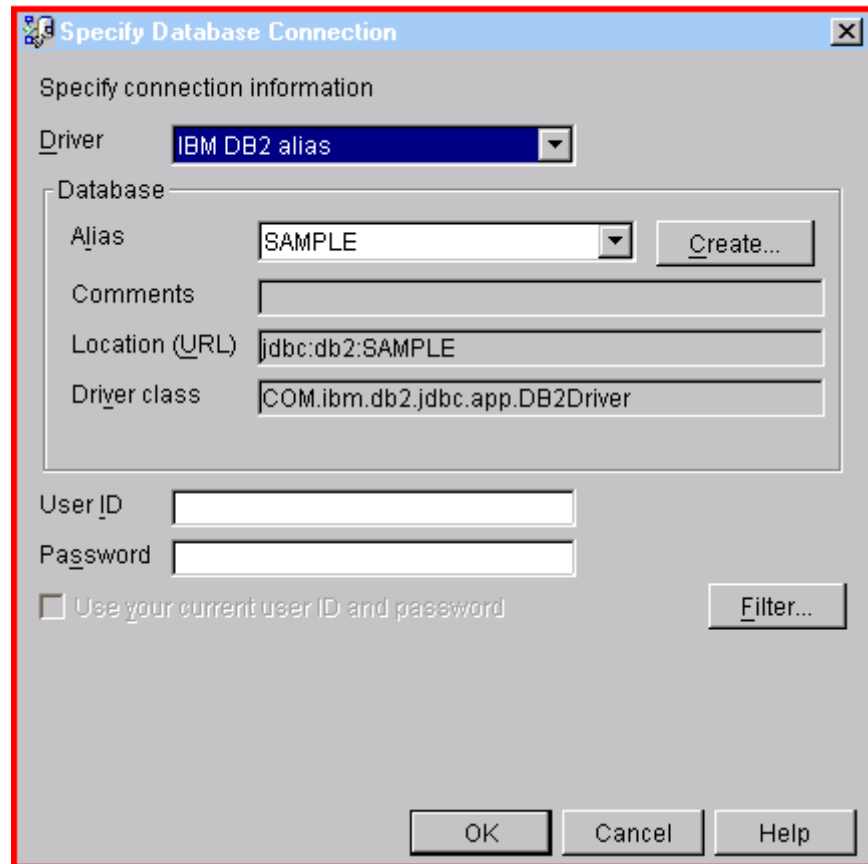


Figure 7. Specify Database Connection Window

The SPB main frame is displayed once you have specified the connection information (see Figure 8).

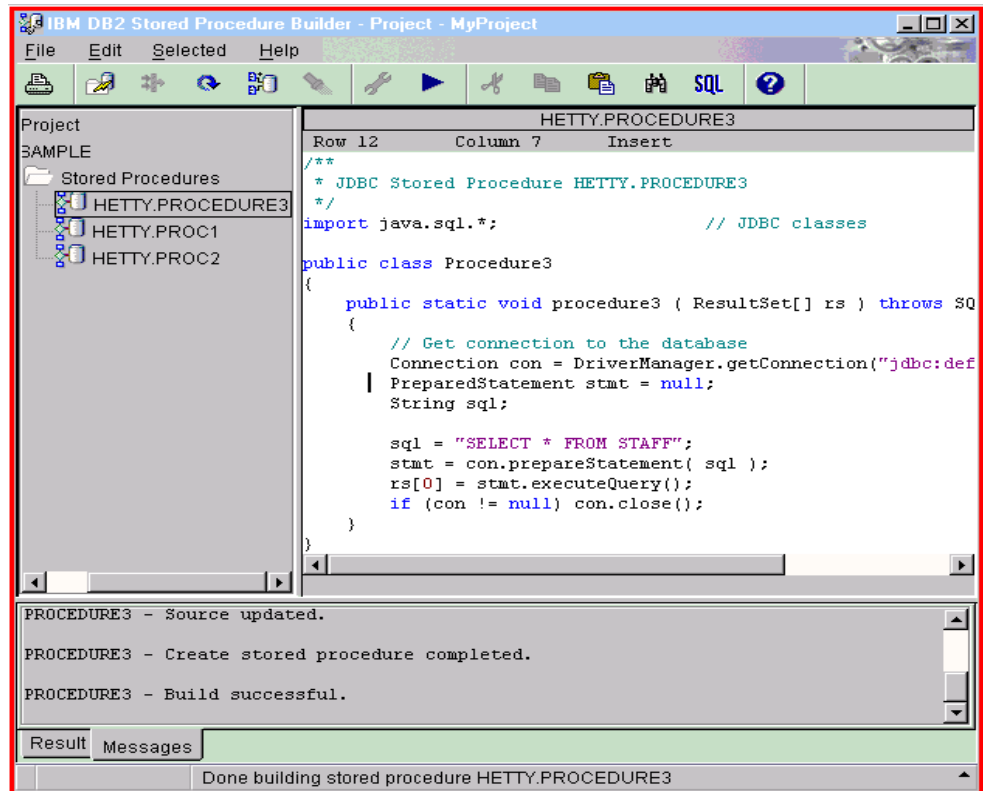


Figure 8. SPB main frame invoked through VA Java Workbench

### 3.2.2.2 Starting SPB from Microsoft Visual Studio

If Microsoft Visual Studio was not installed when you installed DB2 SDK, you must perform one of the following steps to register the add-in with Visual Studio. If Microsoft Visual Studio was installed when you installed DB2 SDK, you should skip these steps.

- If you have Visual Studio 5, copy the file DB2SSPB.DLL from the directory x:\sqllib\bin to y:\Program Files\DevStudio\SharedIDE\AddIn, where x: is the drive on which you have installed DB2 SDK, and y: is the drive on which you have installed Visual Studio 5.
- If you have Visual Studio 6, copy the file DB2SPBVS.DLL from the directory x:\sqllib\bin to y:\Program Files\Microsoft Visual Studio\Common\MSDev98\AddIns, where x: is the drive on which you have installed DB2 SDK, and y: is the drive on which you have installed Visual Studio 6.

To launch SPB from Microsoft Visual Studio, you have to enable the SPB add-in. This can be done using the following steps:

1. From the main Visual Studio window select **Tools --> Customize...** The popup window showed in Figure 9 is displayed.



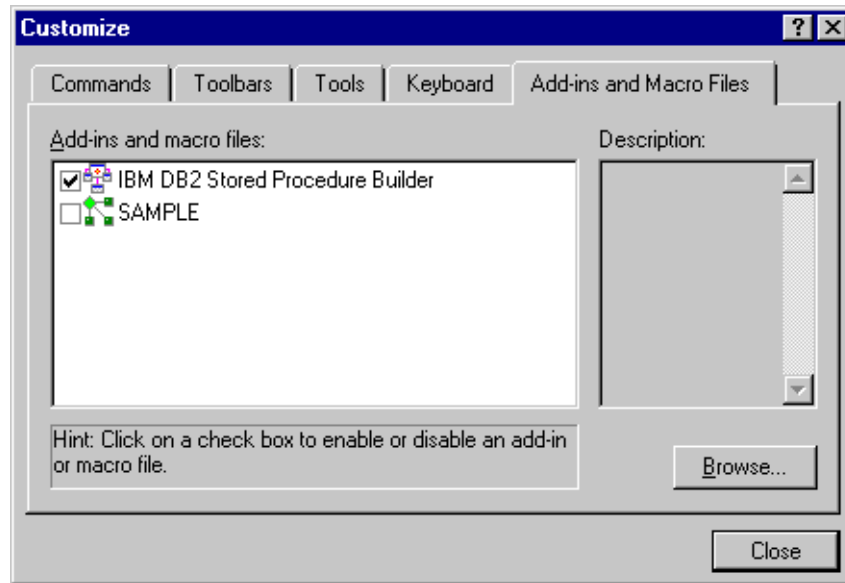


Figure 9. Customizing Microsoft Visual Studio

2. Select the **Add-ins and Macro Files** tab, and click on the check box near the IBM DB2 Stored Procedure Builder add-in. Close the window.

You should now have a fast path for SPB in your Microsoft Visual Studio desktop.

### 3.2.2.3 Starting SPB from Microsoft Visual Basic

If Microsoft Visual Basic was not installed when you installed DB2 SDK, you must perform the following steps to register the add-in with Visual Basic:

1. Open a DOS command prompt and change to the directory x:\sqllib\bin, where x: is the drive on which you have installed DB2 SDK.
2. Enter the following command:

```
db2spbv -addtoini
```

To launch SPB from Microsoft Visual Basic, you have to enable the SPB add-in. This can be done using the following steps:

1. Select **Add-Ins --> Add-In Manager**. The Add-In Manager window opens.
2. Select IBM DB2 Stored Procedure Builder and click OK.

The SPB is added to the Add-Ins menu.

---

## 3.3 Advanced configuring of the SPB

You can customize your SPB environment by using a Windows initialization (INI) file. SPB uses the DB2SPB.INI file to save information about your preferences and environment. The DB2SPB.INI file is located in the SPB subdirectory x:\sqllib\spb, where x: is the drive on which you have installed DB2 SDK.

You need to edit the DB2SPB.INI file to change stored procedure option defaults. You can also use the Environment Properties notebook in SPB to change all other defaults.

### 3.3.0.1 The DB2SPB.INI file

The DB2SPB.INI file contains several sections marked by a section name surrounded by brackets ([ ]). The entries in each section contain keynames and their associated values. Following is an example of the DB2SPB.INI file we used in our project:)

```
[IBM DB2 Stored Procedure Builder 2.1.0b] (1) (see Figure 10)
```

```
ENABLE_STDERR_CONSOLE = FALSE
BUILD_KEEP_TMPDIR_AFTER_FAILURE = FALSE
```

```
[Previous projects]----- (see Figure 11)
```

```
MRU_FILE1 = D:\SQLLIB\spb\projects\sg245485.spp
MRU_FILE0 = D:\SQLLIB\spb\projects\sueli.spp
```

```
[Debug information]----- (see Figure 16)
```

```
DEBUG_PORT = 8000
DEBUG_IPADDR = 9.1.151.109
```

```
[Logon information]----- (see Figure 12)
```

```
DEFAULT_JDBC_CLASS = COM.ibm.db2.jdbc.app.DB2Driver
DEFAULT_JDBC_DRIVER = 378
DEFAULT_JDBC_DATABASE = SAMPLE
DEFAULT_JDBC_URL = jdbc:db2:SAMPLE
```

```
[Data type preferences]----- (see Figure 18 and Figure 19)
```

```
TYPE_MAP_BYTES = 7
TYPE_MAP_STRING_MAGNITUDE =
TYPE_MAP_BYTES_LENGTH = 254
TYPE_MAP_DEFAULT_LENGTH = 254
TYPE_MAP_DEFAULT = 7
TYPE_SYNONYM_7 = varchar
TYPE_SYNONYM_6 = char
TYPE_SYNONYM_4 = double
TYPE_SYNONYM_2 = dec
TYPE_MAP_BYTES_MAGNITUDE = B
TYPE_SYNONYM_1 = int
TYPE_CASE = LOWER
TYPE_MAP_DEFAULT_BITDATA = FALSE
TYPE_MAP_STRING_LENGTH = 254
TYPE_MAP_STRING = 7
```

```
[Output preferences]----- (see Figure 15)
```

```
OUTPUT_COMMIT_RUN = FALSE
OUTPUT_MAX_ROWS =
OUTPUT_MAX_COLWIDTH =
OUTPUT_ALL_ROWS = TRUE
OUTPUT_ALL_COLWIDTH = TRUE
OUTPUT_STATEMENT_SEPARATOR = @
```

```
[User-assistance preferences]----- (see Figure 14)
```

```
ASSISTANCE_BEEPS = TRUE
ASSISTANCE_WINDOW_SIZE_WIDTH = 660
ASSISTANCE_BORDERS = TRUE
ASSISTANCE_SPLIT_HORZ_LOCATION = 339
ASSISTANCE_SPLIT_VERT_LOCATION = 163
ASSISTANCE_WINDOW_SIZE_HEIGHT = 350
ASSISTANCE_TIPS = TRUE
ASSISTANCE_INFOPOPS = TRUE
```

```

[Stored-procedure options] (2) (see Figure 17)
SPOPTION_COMPILE_TEST_OPTION = NOTEST(block,noline,nopath)
SPOPTION_WLM_ENVIRONMENT = wlmenv1
SPOPTION_SQLPROC_BUILDER = DSNTPSMP
SPOPTION_EXTERNAL_SECURITY = DB2
SPOPTION_STAY_RESIDENT = FALSE
SPOPTION_LINK_OPTIONS =
SPOPTION_LE_TEST_OPTION = NOTEST(ALL,*,,VADTCPIP&9.1.151.109:*)
SPOPTION_COMPILE_OPTIONS = list,longname
SPOPTION_LE_OPTIONS =
SPOPTION_TEST = FALSE
SPOPTION_COLLID = TEST
SPOPTION_PRELINK_OPTIONS = nomap
SPOPTION_PSM_PRECOMPILE = source
SPOPTION_BIND_OPTIONS =

```

```

[Editor preferences]----- (see Figure 13)
EDITOR_LINE_NUMBERS = TRUE
EDITOR_FONT_SIZE = 12
EDITOR_TAB_SIZE = 4
EDITOR_LANGUAGE_PARSING = TRUE

```

The first section **(1)**, IBM DB2 Stored procedure Builder 2.1.0b, is internal only and must not be modified.

The Stored Procedure Option **(2)** is for OS/390 only. Part of the information will be externalized in a future release of SPB to facilitate updating via the Environment Properties dialog.

Figure 10 through Figure 19 show the various menus of the SBP.

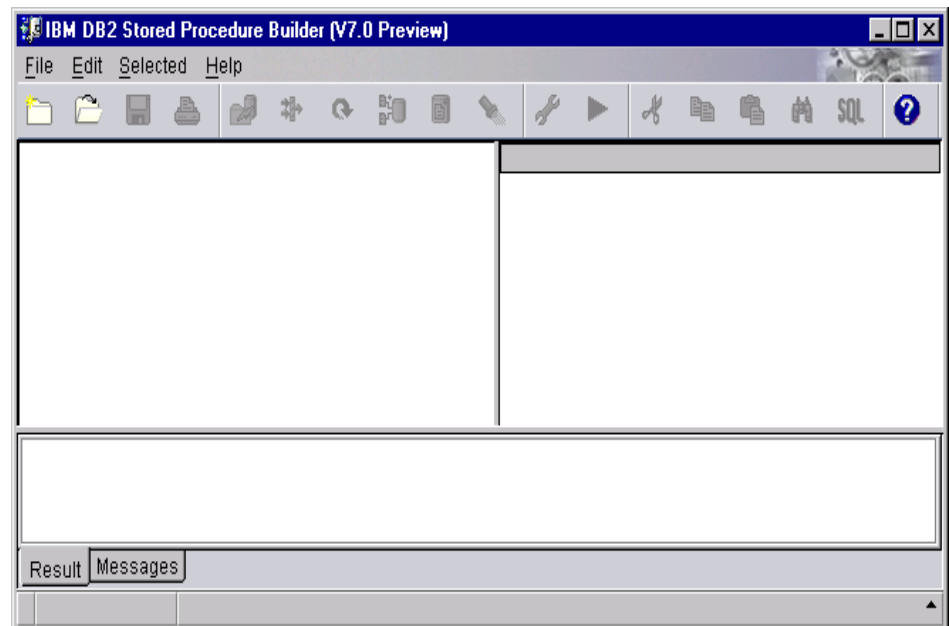


Figure 10. Stored Procedure Builder

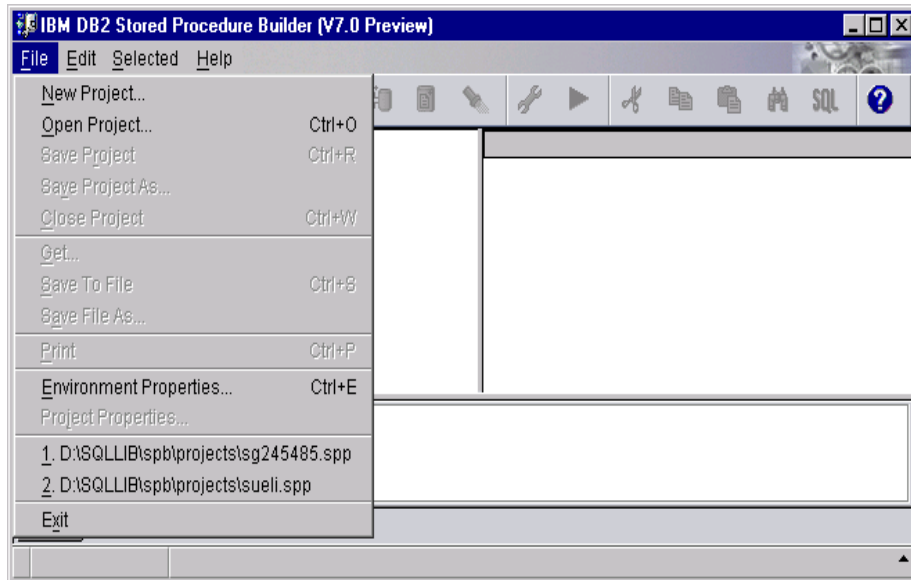


Figure 11. SPB: Previous Projects

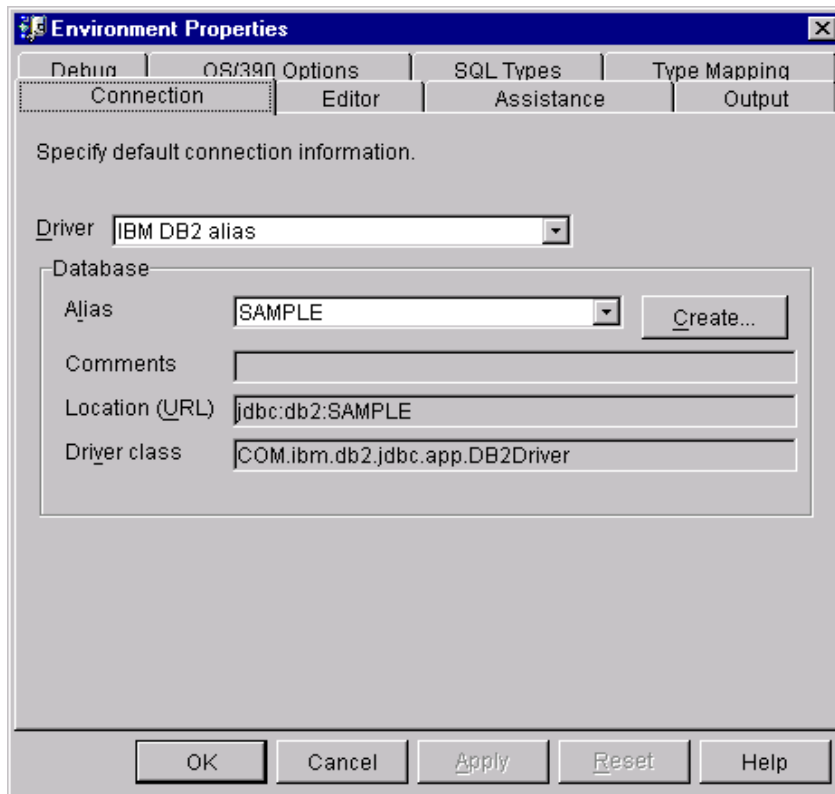


Figure 12. Environment Properties: Connection

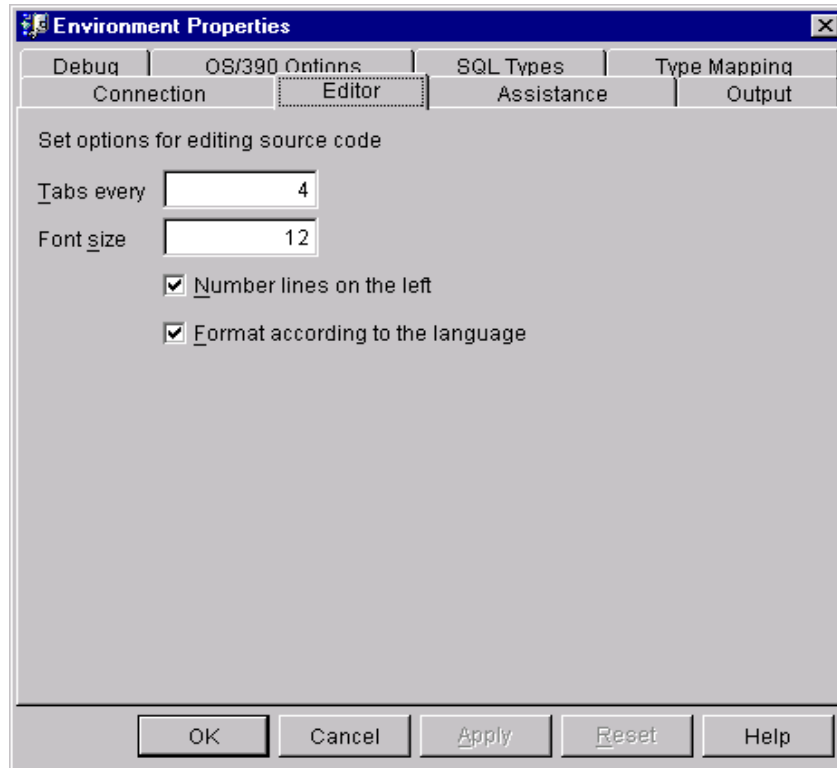


Figure 13. Environment Properties: Editor

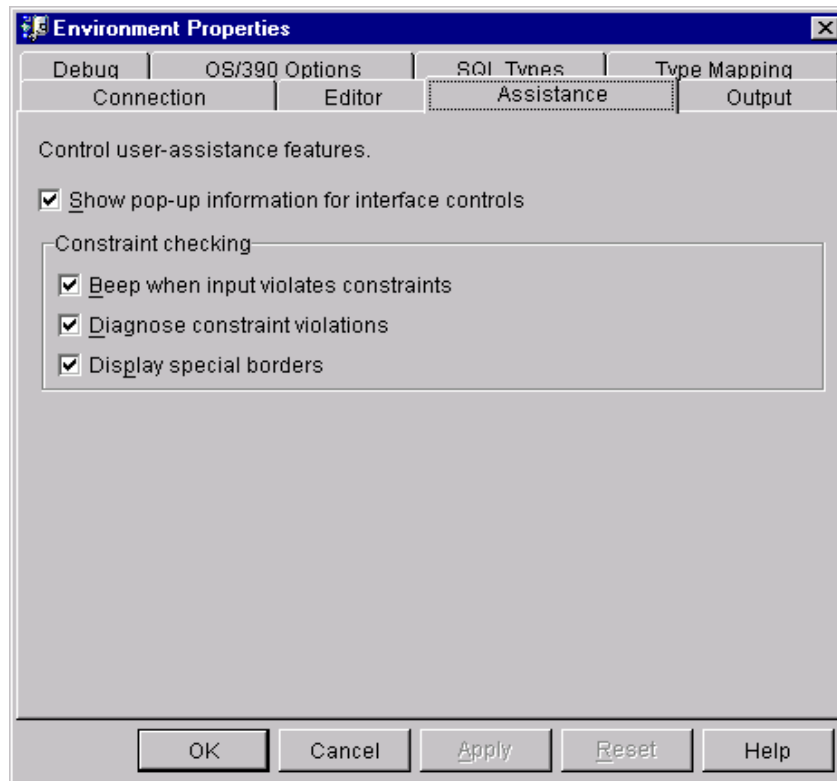


Figure 14. Environment Properties: Assistance

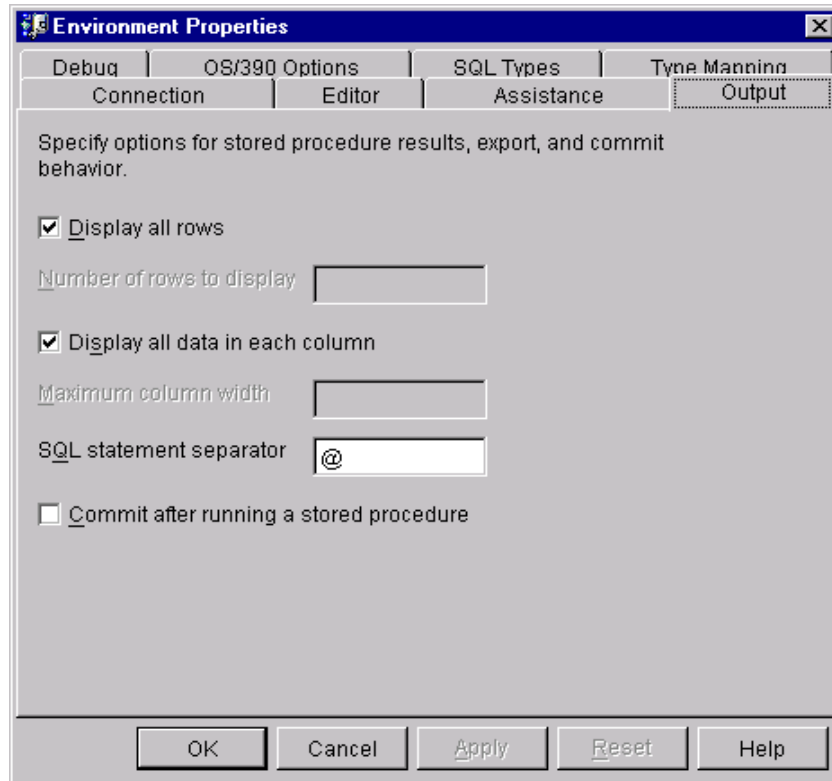


Figure 15. Environment Properties: Output

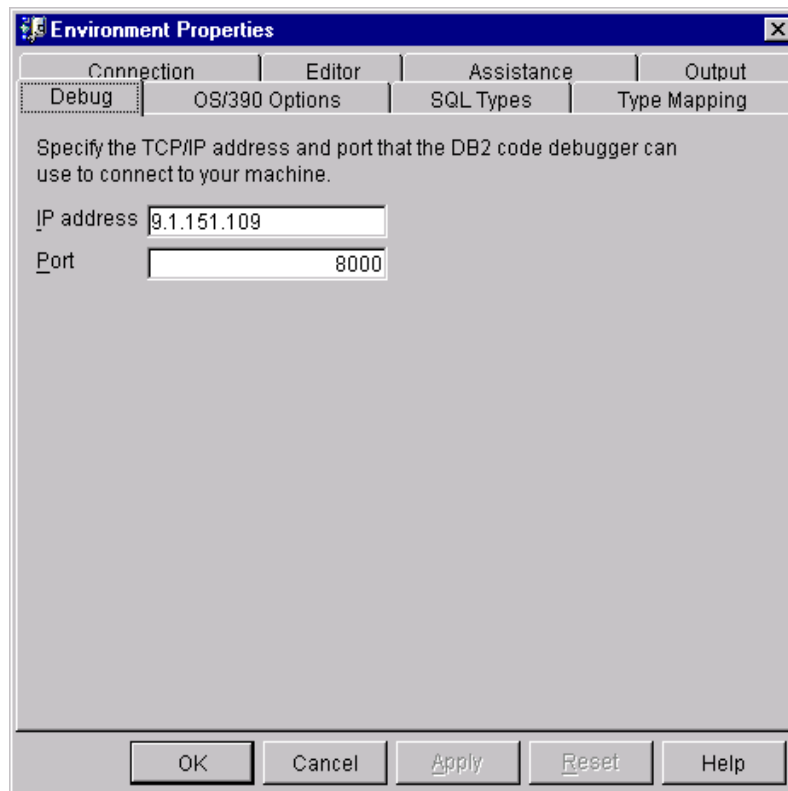


Figure 16. Environment Properties: Debug

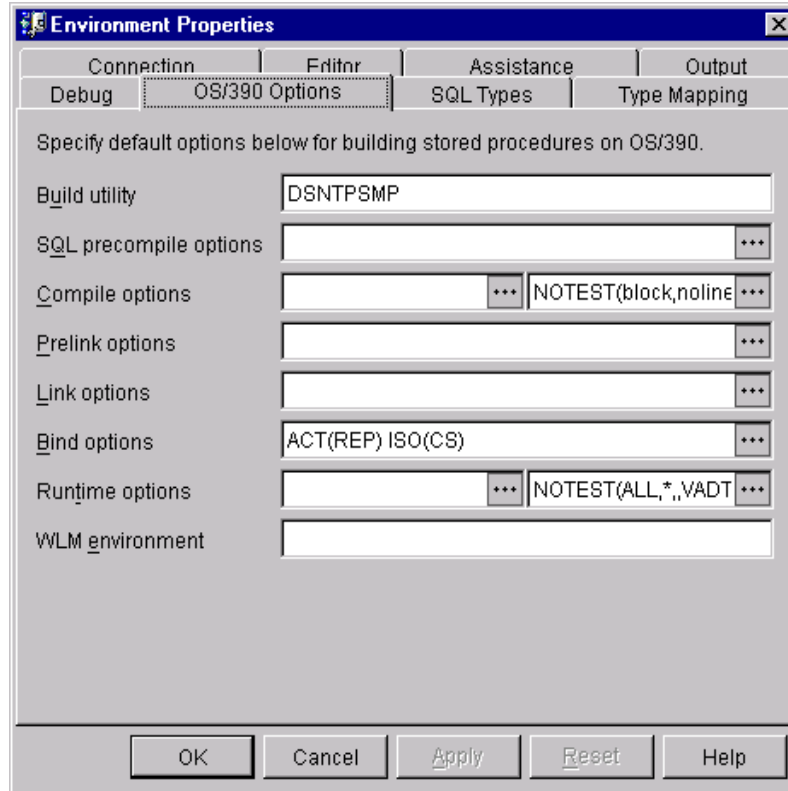


Figure 17. Environment Properties: OS/390 Options

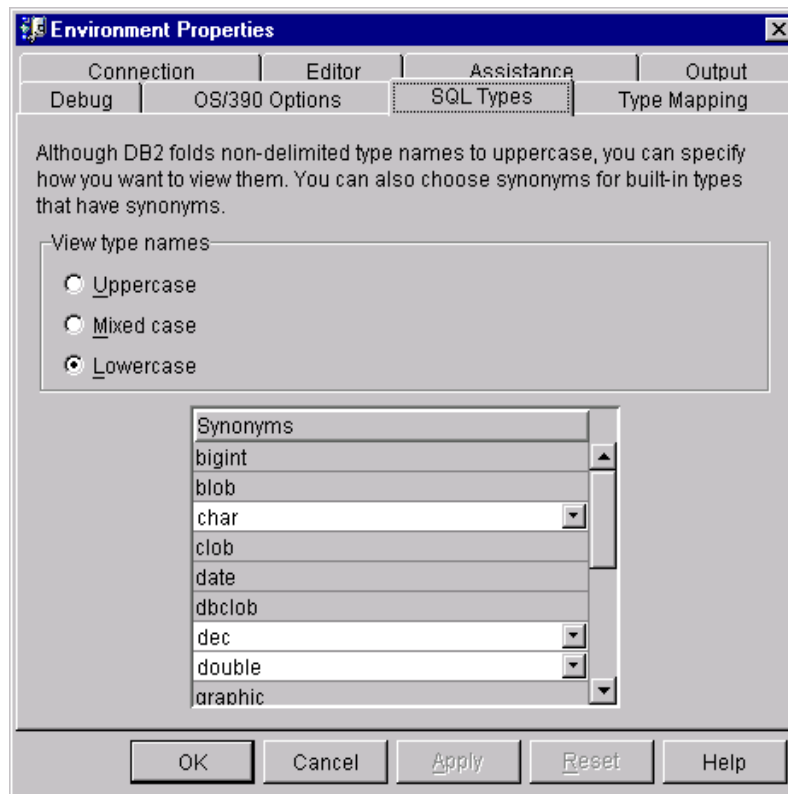


Figure 18. Environment Properties: SQL Types

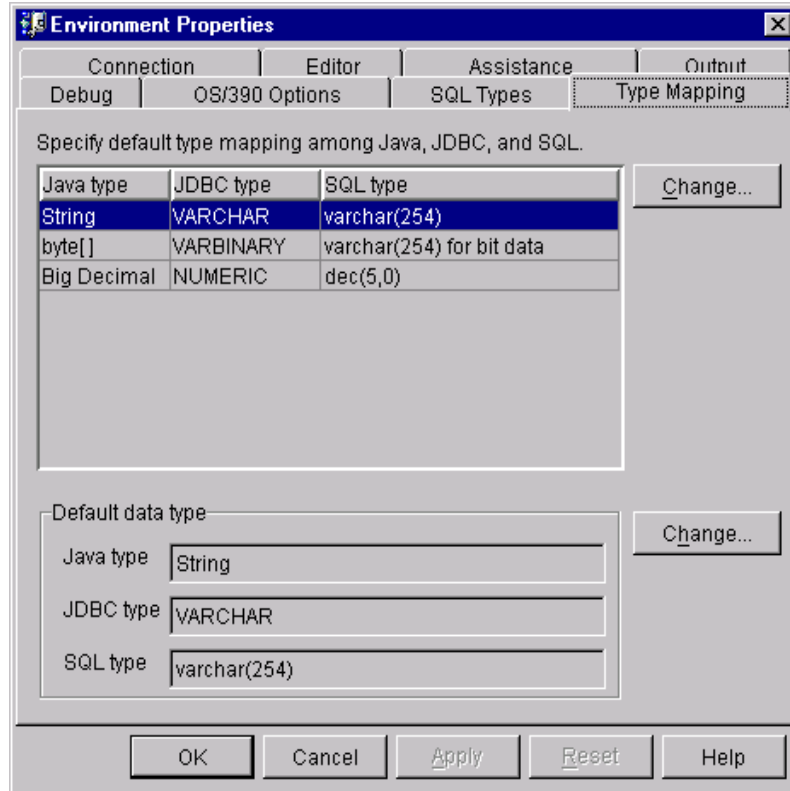


Figure 19. Environment Properties: Type Mapping

### 3.3.0.2 Entries in the DB2SPB.INI file

Each section of the DB2SPB.INI file contains keywords that affect the behavior of SPB. Table 8 shows the keywords associated with each section, their possible values, and a description of each keyword.

Table 8. DB2SPB.INI file sections and keywords

Section	Keyname	Possible Values	Description
[Most recently used projects]	MRU_FILE<number>	<project-file-name-with-absolute-path>	Specifies absolute path of an SPB project file. The number appended to the keyname represents how recently the file was used; for example, MRU_FILE0 is the most recently used file.
[Stored procedure options]	SPOPTION_STAY_RESIDENT	TRUE   FALSE	OS/390 only: Specifies whether the stored procedure load module remains in memory when the stored procedure ends
	SPOPTION_EXTERNAL_SECURITY	DB2   USER   DEFINER	OS/390 only: Specifies the type of external security environment for the stored procedure
	SPOPTION_COLLID	<collection-ID>   <user-ID>	OS/390 only: Specifies the package collection used when the stored procedure is executed
	SPOPTION_LE_OPTIONS	<Language-Environment-options>	OS/390 only: Specifies Language Environment run-time options



Section	Keyname	Possible Values	Description
	SPOPTION_WLM_ENVIRONMENT	<Work-Load-Manager-environment>	OS/390 only: Specifies the MVS workload manager (WLM) environment where the stored procedure runs
	SPOPTION_PSM_PRECOMPILE	<PSM-precompile-options>   CONNECT(1), DATE(LOC), DEC(31), FLAG(I), HOST(SQL), LINECOUNT(0), MARGINS(0), NOSOURCE, SQLFLAG(STD), TIME(LOC)	OS/390 only: Specifies precompilation options for building SQL stored procedures on a DB2 for OS/390 server
	SPOPTION_TEST	TRUE   FALSE	Specifies whether or not to build the Java or SQL stored procedure for debugging
[User-Assistance preferences]	ASSISTANCE_INFOPOPS	TRUE   FALSE	Specifies whether pop-up help for interface controls is on or off
	ASSISTANCE_BEEPS	TRUE   FALSE	Specifies whether the system beep for input constraint violations is on or off
	ASSISTANCE_TIPS	TRUE   FALSE	Specifies whether pop-up information for fields with constraint checking are on or off
	ASSISTANCE_BORDERS	TRUE   FALSE	Specifies whether the blue or red border around constraint checking fields is on or off
[Output preferences]	OUTPUT_ALL_COLWIDTH	TRUE   FALSE	Specifies whether all columns in a stored procedure result set display
	OUTPUT_ALL_ROWS	TRUE   FALSE	Specifies whether all rows in a stored procedure result set display
	OUTPUT_MAX_COLWIDTH	<maximum-column-width>   20	Specifies maximum number of characters displayed per column in a stored procedure result
	OUTPUT_MAX_ROWS	<maximum-row_count>   10	Specifies maximum number of rows displayed in a stored procedure result
[Debug information]	DEBUG_IPADDR	<your-IP-address>	Specifies IP address of the workstation on which SPB is installed
	DEBUG_PORT	<your-port>   8000	Specifies port number which the debugger can use to connect to the client workstation
[Logon information]	DEFAULT_JDBC_DRIVER	app   net   odbc   other	Specifies the default driver used to establish a database connection
	DEFAULT_JDBC_DATABASE	<database-or-alias>   <first-alias>	Specifies the name of the database with which to establish a connection
	DEFAULT_JDBC_HOST	<host>   <empty>	Specifies the host name of the DB2 server

Section	Keyname	Possible Values	Description
	DEFAULT_JDBC_PORT	<port>   <empty>	Specifies the port defined by the DB2 server
	DEFAULT_JDBC_URL	<connection-URL>   jdbc:db2:<first-alias>	Specifies the location path of the database
	DEFAULT_JDBC_CLASS	<driver-class>   COM.ibm.db2.jdbc.app.DB2 Driver	Specifies the driver class associated with the chosen driver
[Data type preferences]	TYPE_SYNONYM_7	varchar   char varying   character varying	Specifies default synonym for SQL data type varchar
	TYPE_SYNONYM_6	char   character	Specifies default synonym for SQL data type char
	TYPE_SYNONYM_4	double   double precision	Specifies default synonym for SQL data type double
	TYPE_SYNONYM_2	decimal   dec   numeric   num	Specifies default synonym for SQL data type dec
	TYPE_SYNONYM_1	int   integer	Specifies default synonym for SQL data type int
	TYPE_CASE	INITIAL   UPPER   LOWER	Specifies default type case settings for SQL data type names
	TYPE_MAP_STRING	6   7   8   9   10   11   12   13 <sup>2</sup>	Specifies default SQL type mapped to the Java type string
	TYPE_MAP_STRING_LENGTH	<length>   256	Specifies default SQL type length for the Java type string
	TYPE_MAP_STRING_MAGNITUDE	B   K   M   G3	Specifies default SQL type unit size for the Java type string
	TYPE_MAP_BYTES	6   7   8   172	Specifies the default SQL type mapped to the Java type string
	TYPE_MAP_BYTES_LENGTH	<length>   256	Specifies default SQL type length for the Java type byte
	TYPE_MAP_BYTES_MAGNITUDE	B   K   M   G3	Specifies default SQL type unit size for the Java type byte
	TYPE_MAP_DECIMAL_PRECISION	<precision>   5	Specifies the default precision for the SQL type mapped to Java type decimal
	TYPE_MAP_DECIMAL_SCALE	<scale>   0	Specifies the default scale for the SQL type mapped to Java type decimal
	TYPE_MAP_DEFAULT	6   7   8   9   10   11   12   132	Specifies default SQL type mapped to unknown Java and JDBC types (for SQL written outside of SQL Assistant)
	TYPE_MAP_DEFAULT_LENGTH	<length>   256	Specifies default SQL type length
	TYPE_MAP_DEFAULT_MAGNITUDE	B   K   M   G3	Specifies default SQL type unit size

Section	Keyname	Possible Values	Description
	TYPE_MAP_DEFAULT_PRECISION	<precision>   5	Specifies the default precision for the SQL type mapped to unknown Java and JDBC types
	TYPE_MAP_DEFAULT_MAGNITUDE	B   K   M   G3	Specifies default SQL type unit size
	TYPE_MAP_DEFAULT_PRECISION	<precision>   5	Specifies the default precision for the SQL type mapped to unknown Java and JDBC types
	TYPE_MAP_DEFAULT_SCALE	<scale>   0	Specifies the default scale for the SQL type mapped to unknown Java and JDBC types
	TYPE_MAP_DEFAULT_BIT DATA	TRUE   FALSE	Specifies whether SQL types use the bit data subtype for character strings
[Editor]	EDITOR_TAB_SIZE	<width>   4	Specifies the default tab width in character spaces
	EDITOR_FONT_SIZE	<size>   12	Specifies the default font size in points
	EDITOR_LINE_NUMBERS	TRUE   FALSE	Specifies whether line number display is on or off
	EDITOR_LANGUAGE_PARSING	TRUE   FALSE	Specifies whether color-coded text is on or off

### 3.3.1 Concepts and terminology

SPB uses some terms to define objects and operations related to the creation and maintenance of stored procedures. Following are some of the most important terms associated with SPB objects and operations:

#### 3.3.1.1 Project

When using SPB, you must create a project. A project stores the connection information to the databases you are accessing and stored procedures sources that have not been build to the DB2 server. The connections information specified in the project file is used for all the functions performed by SPB.

#### 3.3.1.2 Create

The create function is associated with the creation of new stored procedures using the New Stored Procedures SmartGuide. At the end of the creation of the stored procedure you can choose to generate the stored procedure, or to generate and build the stored procedure.

#### 3.3.1.3 Generate

The process of generating a new stored procedure is started by the New Stored Procedures SmartGuide. The generate option creates a base skeleton for your stored procedure based on the information provided through the SmartGuide. Note that the generate function runs locally and does not include any information about the stored procedure on the DB2 server, that is, the source code is generated in memory.

If the stored procedure is in Java, the Build action compiles the source in local temporary directories, creates a jar file to contain the executables, and then calls the `sqlj_install.jar()` utility to add the jar file to the server. When developing SQL stored procedures, the source is generated in local memory, and the CREATE PROCEDURE statement is compiled and processed on the server.

#### **3.3.1.4 Build**

The build function is responsible for performing all the tasks necessary to create and register the stored procedure on the DB2 server. Only when you build the stored procedure does it become available for use at the DB2 server.

#### **3.3.1.5 Register**

Inserting the row for a stored procedure into the DB2 table for stored procedures, along with the associated parameters of those stored procedures.

#### **3.3.1.6 Run**

After building the stored procedure, the SPB allows you to execute the stored procedure without writing a client program. The run function invokes the stored procedure on the DB2 server, prompts for parameters, and display the results of the stored procedure. You can use the run function to execute any stored procedure registered at the DB2 server, regardless of the language of the stored procedure.

#### **3.3.1.7 Get Source**

Stored procedures written in Java or SQL Procedures language have their source codes stored at the DB2 server in control tables. When working with SPB, you can request the stored procedure source code from the DB2 server by using the get source function.

With DB2 UDB, procedures written in SQL Procedures language always have their source stored in the DB2 tables, even if they were created outside SPB, however, Java procedures created outside SPB may not have their sources stored in the DB2 tables. In this case, if the procedures were registered with LANGUAGE JAVA and PARAMETER STYLE JAVA, when SPB cannot find the source code in the DB2 tables, a popup window is displayed and you can associate a file with the Java procedure source code, and this source code will be stored by SPB in the DB2 tables.

With DB2 for OS/390, your SQL stored procedures may not have their source codes in DB2 tables if they were not created using SPB. In this case, when SPB cannot find the source code in the DB2 tables, a popup window is displayed and you can associate a file with the SQL stored procedures source code, and this source code will be stored by SPB in the DB2 tables. The file with the source code must be downloaded from the mainframe, and has to reside on a disk that can be accessed by SPB.

#### **3.3.1.8 Modify**

Once you have the stored procedures source code available in SPB, you can change the stored procedure using the stored procedures editor, or the SPB assistants, such as the SQL Assistant. You can only modify stored procedures written in Java or SQL Procedures language.

### 3.3.1.9 Dirty Procedures

A dirty procedure is indicated in SPB by having its name in bold. A stored procedure is considered dirty if you have made modifications to the stored procedures source code that have not been built back to the DB2 server.

### 3.3.1.10 Database Connection

SPB can be used to develop stored procedures to access many servers. Within a project, you can have different database connections to the DB2 servers being accessed.

## 3.3.2 What are its components?

The SPB has many components to help you create, build, and debug your stored procedures. The SPB main panel, shown in Figure 20, provides icons and fast paths to other components.

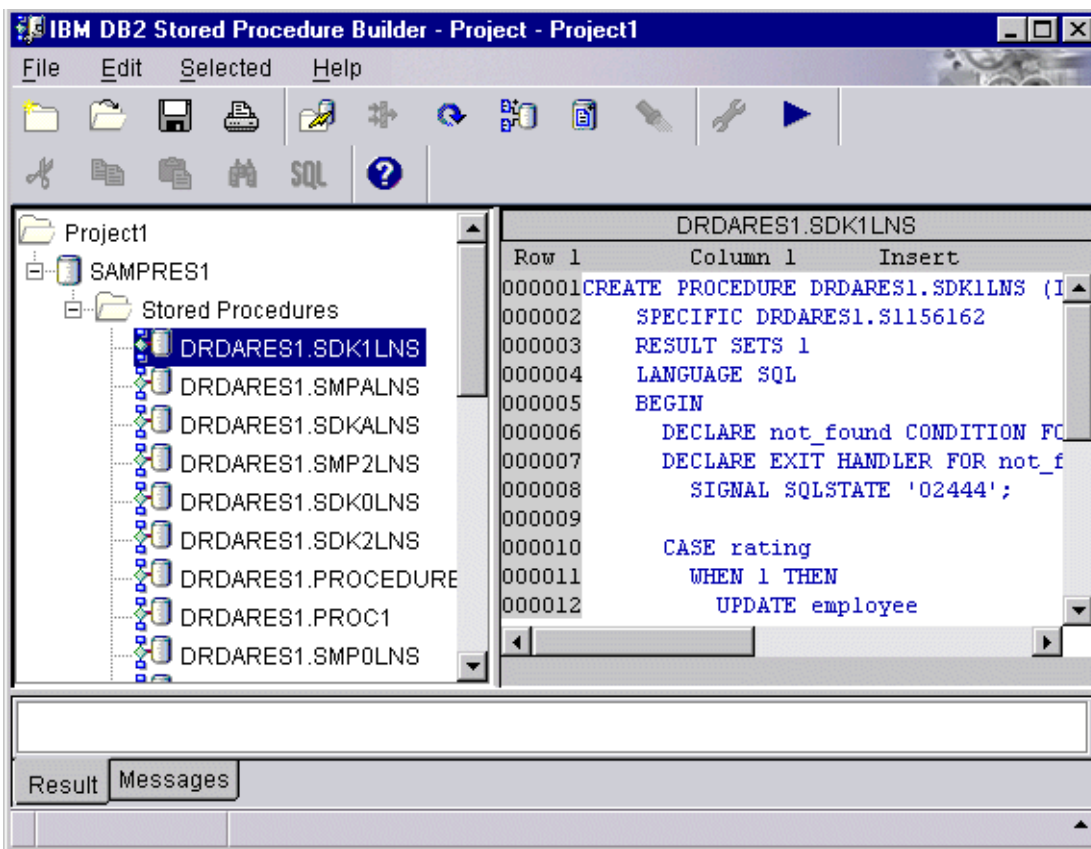


Figure 20. SPB main panel

Following is a description of the main components of SPB.

### 3.3.2.1 New Stored Procedure SmartGuide

The New Stored Procedures SmartGuide is a graphical user interface that helps you to create a new stored procedure written in Java or SQL Procedures language. The SmartGuide helps you to specify the name, the general pattern for the source code, the query to run, the SQL data types for the parameters, and the build options for a new stored procedure.

You can start the New Stored Procedures SmartGuide by clicking on the **Insert Java Procedure** icon or the **Insert SQL Procedure** icon. You can also start the SmartGuide by right-clicking the entry **Stored Procedures**, in the tree-view, and choosing **Insert Java Stored Procedure** or **Insert SQL Stored Procedure**. Figure 21 shows the initial window of the New Stored Procedures SmartGuide.

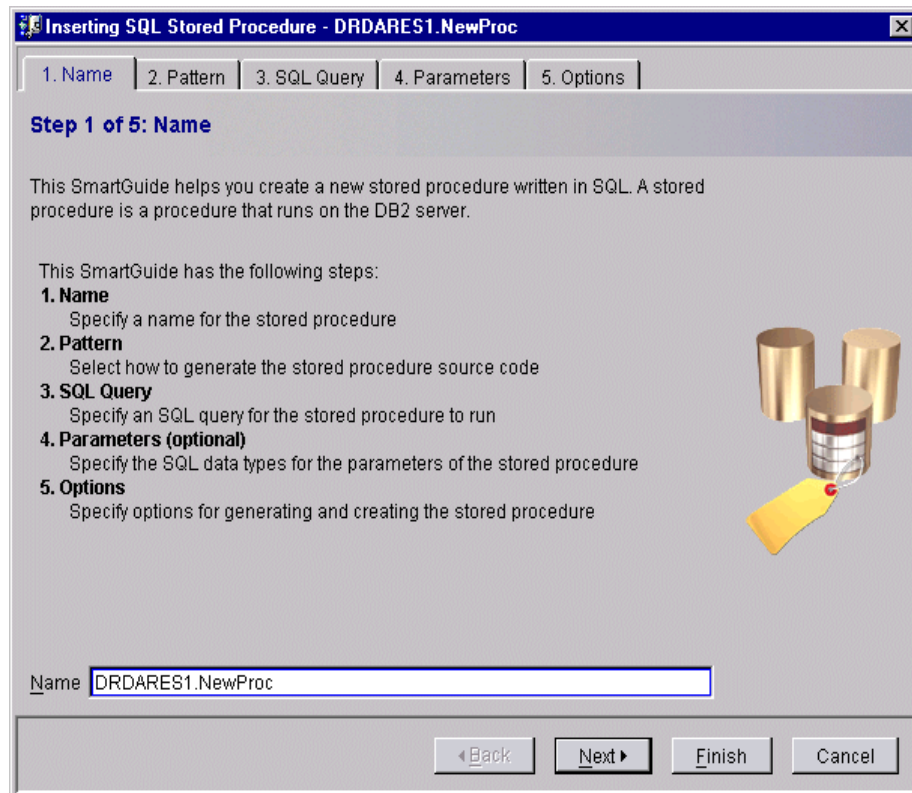


Figure 21. The New Stored Procedures SmartGuide

The SmartGuide has many features to help you build your stored procedure:

- **Field Sensitive Help:** You can click on any object in the SmartGuide windows and press F1. Help information and tips are presented in a yellow box near the object.
- **Smart Fields:** Whenever you need to type information in the SmartGuide, smart input fields will help you. If the border is blue, the field is selected and contains valid information. If the border turns red, the SmartGuide has found a problem with your input. When a problem is detected, a grey popup message appears showing the error. If you press F1, a tip with a possible solution is displayed.

### 3.3.2.2 The SQL Assistant

The SQL Assistant is a SmartGuide that steps you through the processing of creating SQL statements. You can select tables on which to run queries, join tables, enter conditions and columns, determine how to sort the result, and display the SQL statement so that you can copy or test the SQL query. In SQL Assistant, the tables that you can view and build queries from are listed from the catalog tables of the current database alias.

The SQL Assistant can be invoked from the editor, or from the New Stored Procedure SmartGuide dialogs. Figure 22 shows the first window of the SQL Assistant.

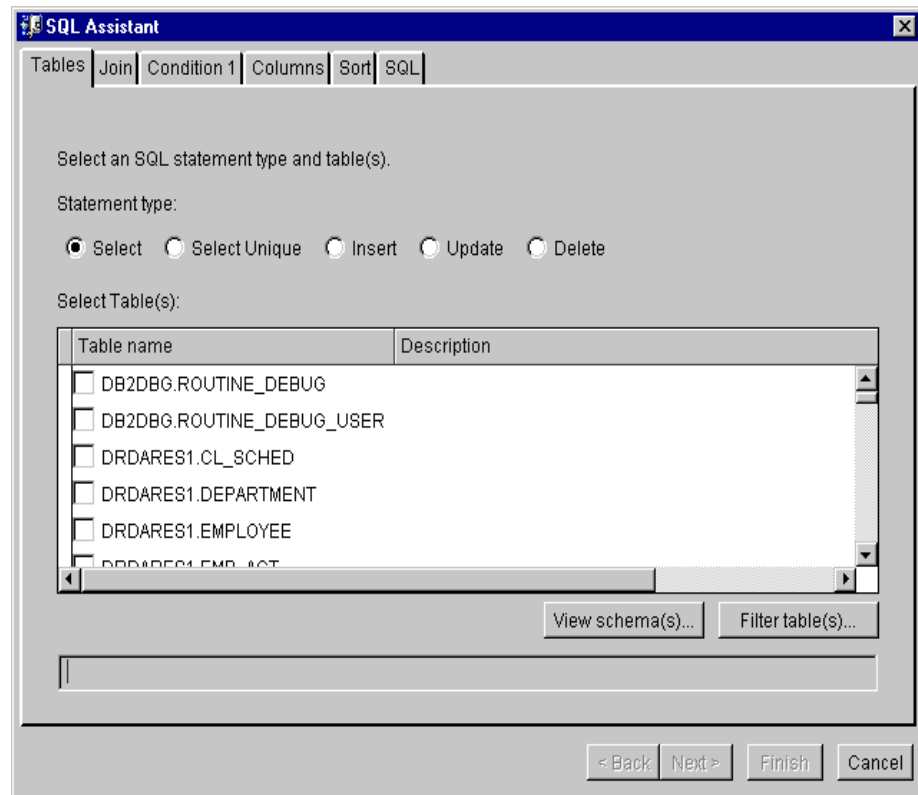


Figure 22. SQL Assistant window

### 3.3.2.3 Client Configuration Assistant

The DB2 Client Configuration Assistant SmartGuide can be invoked from SPB to catalog new databases, if necessary. During the creation of a new project or when inserting a new database connection, if you click on the **Create...** button near the database alias pulldown, the Client Configuration Assistant is started.

### 3.3.2.4 IBM Distributed Debugger

The IBM Distributed Debugger, also referred as VisualAge Remote Debugger, is the client debugger application running on your workstation that allows you to remotely debug a stored procedure executing on the server. The client debugger must be connected to the debugger backend on the DB2 server. You can debug stored procedures written in Java or SQL Procedures language executing on DB2 servers on Windows NT, AIX, or OS/390. Other platforms will be added in the future for remote debugging.

## 3.3.3 Working with SPB projects

SPB manages the work by using projects. A project stores the information about the databases you are working and also stored procedures that have not yet been built into the DB2 server. A project can contain many connections to different DB2 databases or servers; each of these databases can contain many stored procedures. Figure 23 shows the relationships among projects, connections, and stored procedures.

The SPB does not control concurrent access on a project file and/or stored procedures. You must be careful when many developers are working with the same databases, to avoid unintentional destruction of stored procedures or changes.

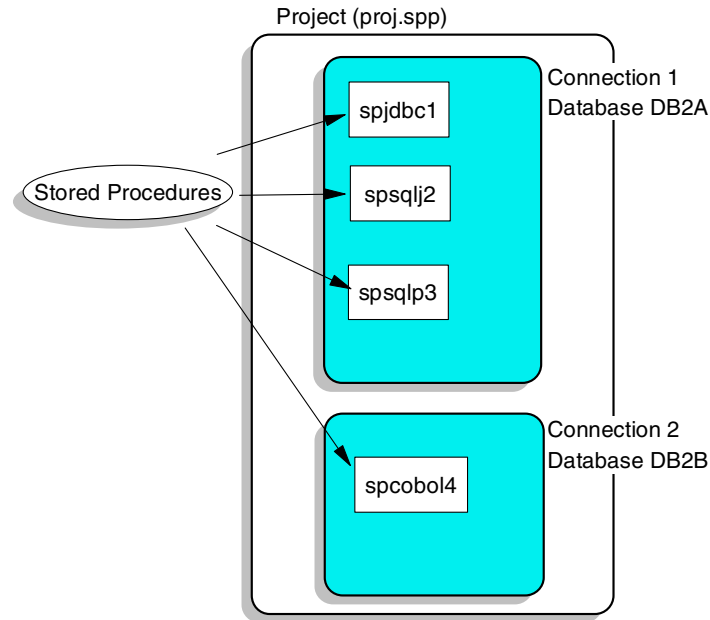


Figure 23. SPB projects (\*.spp), connections and stored procedures

When SPB starts, it prompts you to either create a new project, or work with an existing project. The following sections describe how to create and manage SPB projects.

### 3.3.3.1 Creating Stored Procedure Builder projects

The first thing you have to do when working with SPB is to specify your project. You can open an existing project or create a new one. When creating a new project, you have to provide some initial information about the project, such as the project name, and the database being accessed. Figure 24 shows the **New Project** window used to create a new project.



### 1 - Creating new project "ITSO SG245485"

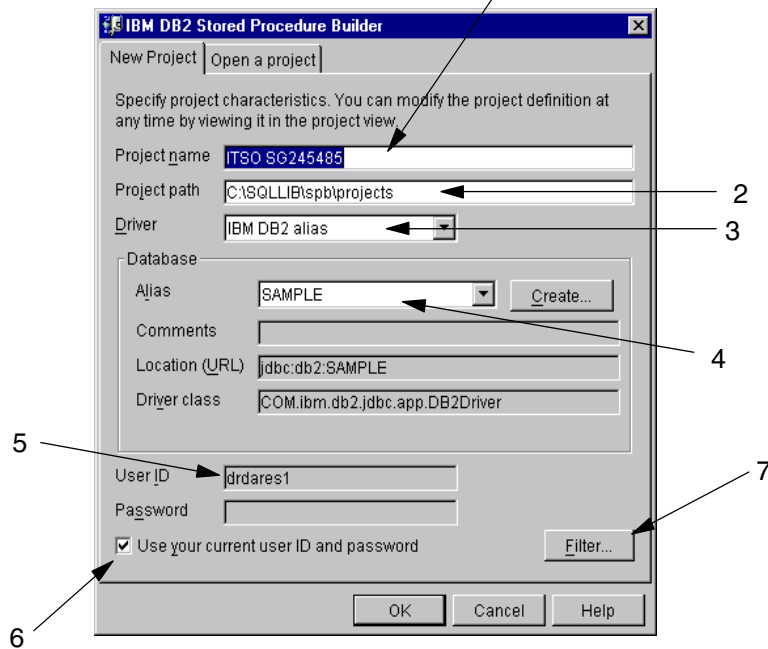


Figure 24. Creating new project "ITSO SG245485"

Note the following items in Figure 24:

1. **Project name.** This is the name of your new project. The project name can be up to 256 characters, and can contain any alphanumeric character and the underscore ' \_ ' character. If you violate any of these rules, when you click **OK**, a popup window with an error message is displayed. SPB will save your project in a file with extension `.spp` and the same name as the project. This `.spp` file is created in the project path.
2. **Project path.** The project path defines the location of your project in your workstation or in a shared disk. By default, the project path points to `x:\sqllib\spb\projects`, where `x`: is the drive on which you have installed DB2 SDK.
3. **Driver.** Select the appropriate driver for your connection to the database on which you want to build stored procedures. When you install the DB2 SDK, the following IBM DB2 drivers are installed on your workstation. When you select one of these drivers for a new SPB project, the JDBC URL and driver class are automatically entered for you. In this release of the product, only the following driver is supported:
  - **IBM DB2 alias** — When the DB2 database to which you want to connect is defined as an alias in your DB2 database client, select the IBM DB2 alias driver. When you use the IBM DB2 alias driver, you do not need to enter the host name and port for the DB2 database in SPB. By default, SPB selects the IBM DB2 alias driver for a connection. You may also click on the **Create** pushbutton; this will invoke the DB2 Client Configuration Assistant, and allow you to create a new alias to a database.

4. **Alias.** Select the database alias you plan to use. You may also click on the **Create** pushbutton; this will invoke the DB2 Client Configuration Assistant, and allow you to create a new alias to a database. After you create your project, you will be able to insert connections to other databases, in the same project, by using the **Insert Connection** dialog.
5. **Userid and Password.** Type the userid and password for accessing the DB2 server database. For security reasons, even if you specify your password here, whenever you open your project, you will be prompted for the password to access remote DB2 servers.
6. **Use your current user ID and password** check box. Select this if you want to use the current window userid and password to connect to the DB2 server.
7. **Filter.** If the database you are connecting to contains a large number of existing stored procedures and you want to limit the stored procedures displayed in the tree view, you may click on the **Filter...** pushbutton. You will be able to filter the procedures displayed using the name, the schema, or the collection id (OS/390 only) of the stored procedure.

### 3.3.3.2 Managing SPB projects

After creating your SPB projects you are ready to start working with SPB. You may change your project properties such as name and description using the **Project Properties** dialog clicking on **File --> Project Properties**. You cannot change the path of you project.

When you first create your project, only one database connection is defined. You may however, define other connections using the **Insert Connection** dialog. If you right-click on the connection at any time, you will also be able to delete, refresh, filter, or change properties of your connections.

A good practice is to refresh your connections periodically to check changes that other users of the database might have made.

You can save your project at any time you want. If you have made changes to stored procedures, but have not built them to the database, the changes are saved with your project, so you can continue working with them later.

**Note:** Other developers working in different projects with connections to the same database will not be able to see your changes until you build them to the database.

### 3.3.3.3 Sharing SPB projects

The way that SPB currently works is not suitable for a team development environment. The current version of SPB does not control concurrence in projects and does not provide check-in/check-out mechanisms, versioning, or any other feature to control accesses in your project.

Saving a project in a shared disk is possible, and may be helpful when more than one developer wants to copy a project with changes from another developer. However, you should not have one project file being used by many developers, since this may lead to unintentional destruction of changes not built into the database.

When sharing SPB projects, it is important to have the same database aliases pointing to the same database servers in all the developers' workstations.

## 3.4 Using the Stored Procedure Builder

The SPB provides you a single development environment for your stored procedures. You will be able to perform all tasks related to the creation and maintenance of stored procedures in both DB2 for OS/390 and DB2 UDB servers. In the future, SPB will also work with DB2 UDB for AS/400, but not in the current version.

### 3.4.1 Viewing existing stored procedures

The SPB allows you to view all the stored procedures that are registered in the DB2 server catalog tables for stored procedures. Regardless of the language in which the stored procedure was written, you can view the existing procedures and parameters being passed; however, you can only get the source, modify, or rebuild existing stored procedures written in SQL Procedures language or Java.

Since only SQL stored procedures and Java stored procedures are mandatory to be registered in the DB2 UDB catalog, it is possible that you have stored procedures in other languages that will not be shown, because they were not registered in the DB2 UDB catalog using the CREATE PROCEDURE statement.

When you create or open an SPB project, all the registered stored procedures in the defined database connections are displayed in the SPB main window tree view. Figure 25 shows the tree view of existing stored procedures in SPB.

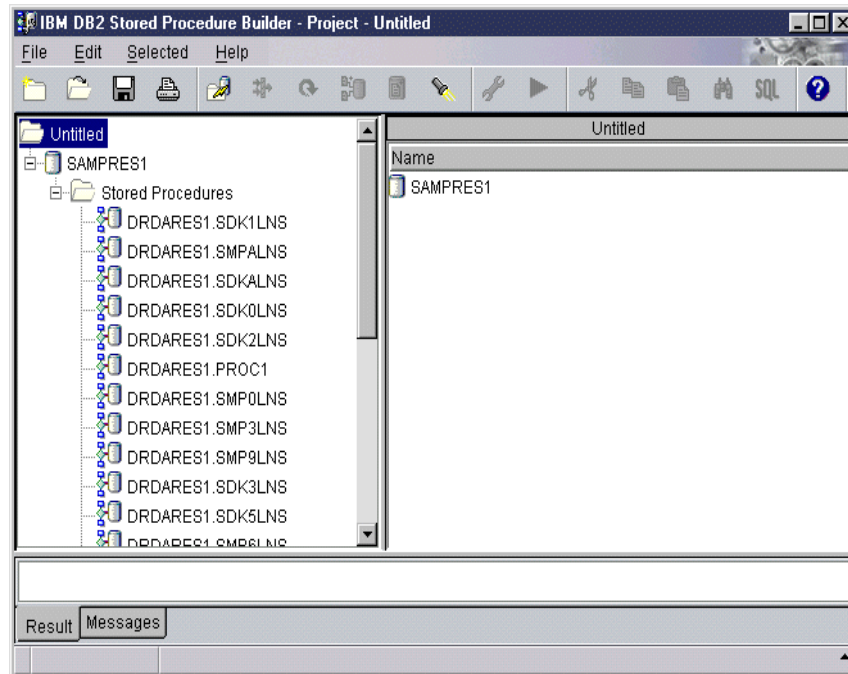


Figure 25. Tree view of existing stored procedures

Note that in the tree view, the stored procedures are not presented in alphabetical order, and also, the parameters are not shown. You can easily access detailed information on the existing stored procedures, such as parameter lists, specific name, and language, by double-clicking the folder **Stored Procedures** in the tree view. A detailed list of the stored procedures, in alphabetical order, is displayed in

the list view part of the SPB main window. Figure 26 shows the detailed view of the existing registered stored procedures.

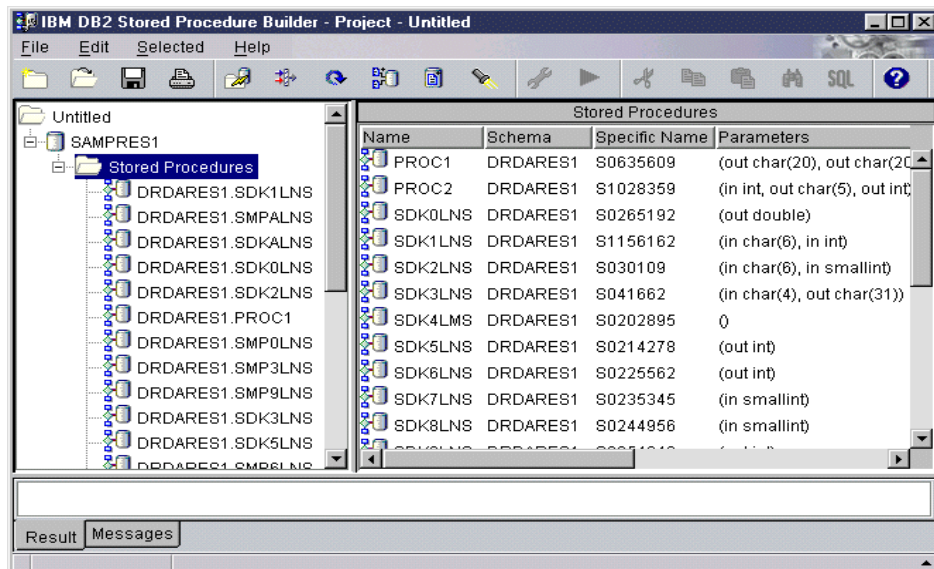


Figure 26. Detailed view of existing stored procedures

If you have a large number of registered stored procedures, you can filter the list of the stored procedures to be displayed. To open the filter dialog, right-click on the **Stored Procedures** folder in the tree view, and choose **Filter**, as shown in Figure 27.

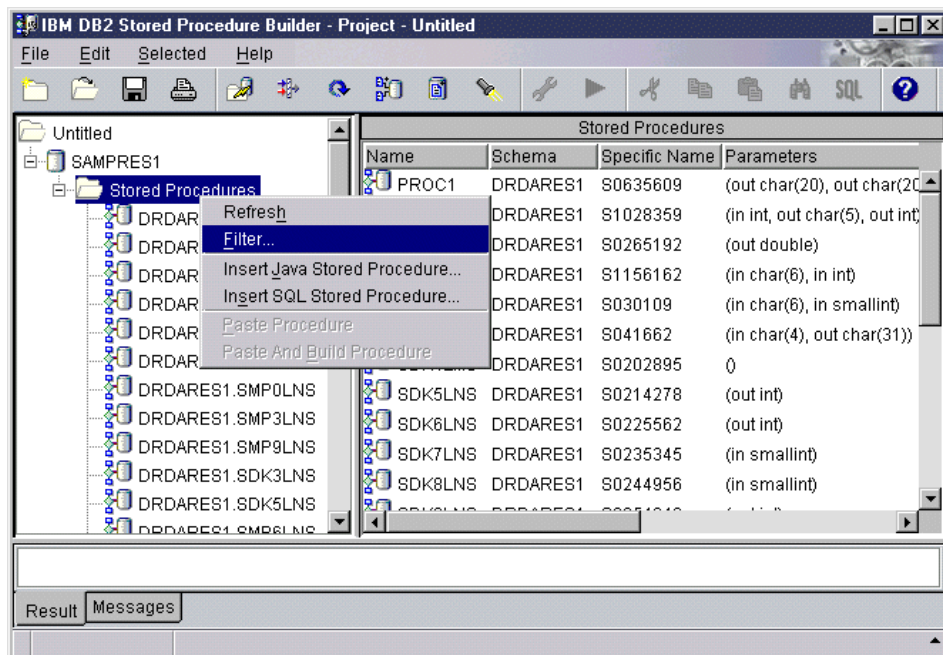


Figure 27. Filtering the list of stored procedures

The **Filter** dialog window is displayed, and you can filter the list of stored procedures based on the name or the schema of the registered stored procedures. You can filter using the complete name of the stored procedure or schema or just a substring of the name or schema. If you are using a substring, you can choose to list procedures that start or end with the substring, or that contains the substring. Figure 28 shows the **Filter** dialog window, and some of the filter options.

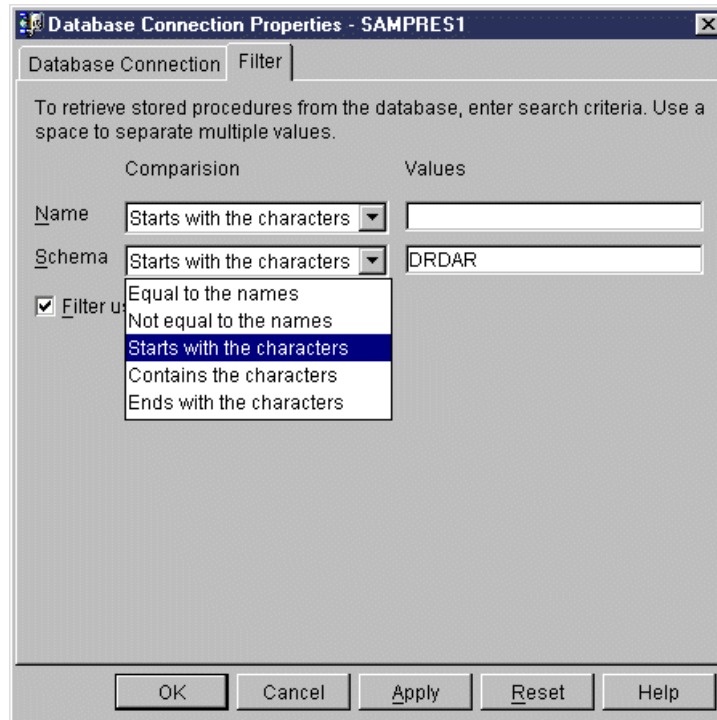


Figure 28. The Filter dialog window

### 3.4.2 Creating new stored procedures

Using SPB, you can create new stored procedures in the SQL Procedures or Java languages. The **New Stored Procedures** SmartGuide helps you in creating the base skeleton of the stored procedure, so you can later include your business logic in it. For DB2 for OS/390 servers, SPB can only create new SQL stored procedures. Support for Java stored procedures for DB2 for OS/390 through SPB is not yet available.

To invoke the **New Stored Procedures** SmartGuide to create a new SQL stored procedure, right click on the **Stored Procedures** folder, and choose **Insert SQL Stored Procedure**, as shown in Figure 29. You can also click on the **Insert SQL Procedure** icon in the SPB toolbar.

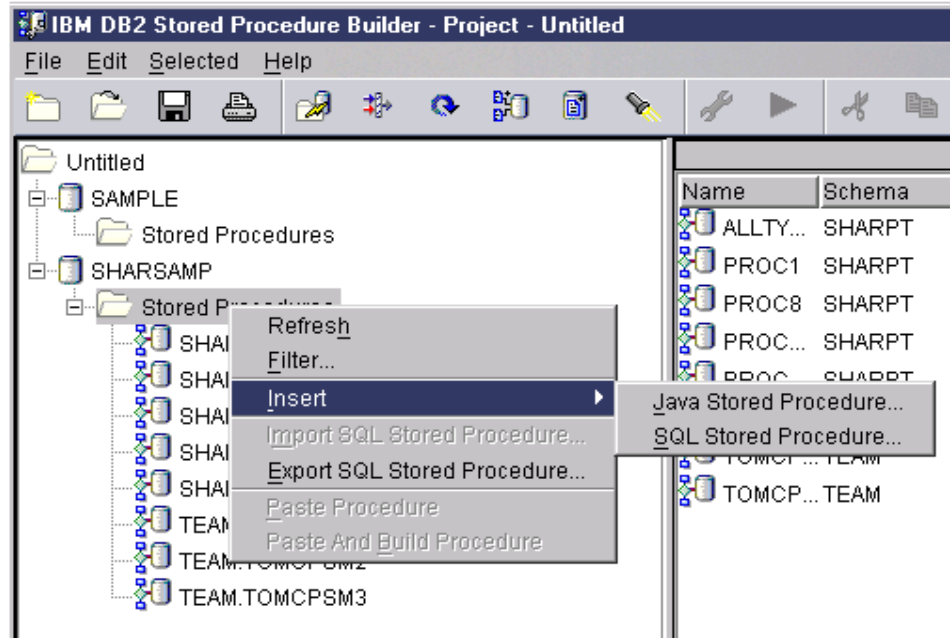


Figure 29. Creating a new SQL stored procedure

The **New Stored Procedures** SmartGuide guides you through 5 steps to create your stored procedure, as follows:

1. **Name:** In this step, you provide the name of the new stored procedure.
2. **Pattern:** In this step, you provide the characteristics that describe the pattern of your stored procedure, such as, if your stored procedure returns a result set, or if your stored procedure runs a single SQL.
3. **SQL Query:** In this step, you can type one SQL query to be run by your stored procedure. If in the **Pattern** panel, you have chosen **Run one query from a set of queries**, you can specify multiple queries in the **SQL Query** panel. From the **SQL Query** panel, you can also invoke the **SQL Assistant** SmartGuide to help you build your query, by clicking on the **Define SQL** button.
4. **Parameters:** In this step, you provide the parameters and associated datatypes being passed to or from your stored procedure. This step is optional, and if your stored procedures do not expect parameters, you may skip this panel.
5. **Options:** In this panel, you specify options for generating the basic skeleton of you SQL stored procedure, and for building your stored procedure at the DB2 server.

#### 3.4.2.1 The New Stored Procedures SmartGuide

When you start the **New Stored Procedures** SmartGuide, the **Name** panel appears, as shown in Figure 30. In any panel of the SmartGuide, if you position your mouse cursor on any field, a popup window appears with more information related to that field. This can be very helpful when you are in doubt of the meaning of the fields in a specific panel.

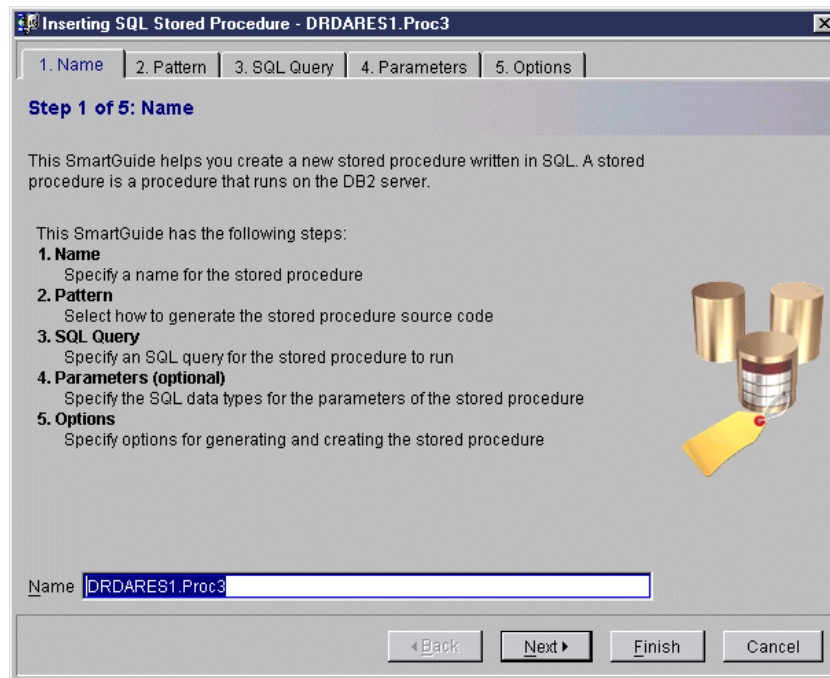


Figure 30. The Name panel of the New Stored Procedures SmartGuide

When you are creating a new SQL stored procedure for a DB2 UDB server, the name of the stored procedure is presented already prefixed with your userid as the schema name. If you are creating a new SQL stored procedure for a DB2 for OS/390 server, the name of the stored procedure is not prefixed, and the default schema SYSPROC is used. If you want, you can type a new entry or change the schema in the stored procedure name field.

After providing a name to your stored procedure, you can click on the **Next** pushbutton. The Pattern panel is displayed, as shown in Figure 31.

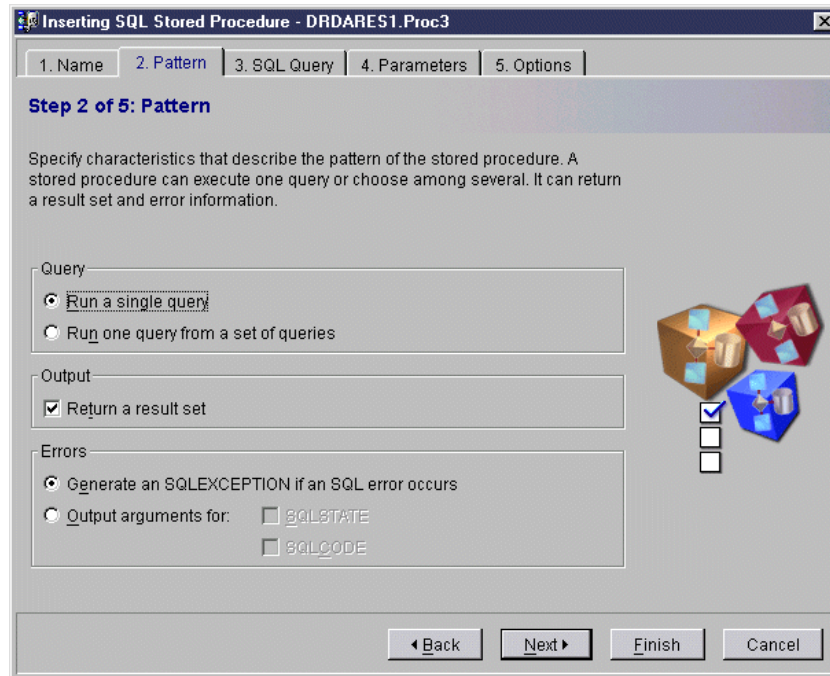


Figure 31. The Pattern panel of the New Stored Procedures SmartGuide

The **Pattern** panel is divided into three sections: **Query**, **Output**, and **Errors**.

The **Query** section allows you to specify if your stored procedure executes only one query or a set of queries. If you specify that your stored procedure executes a set of queries, the **SQL Query** panel will allow multiple SQL statements to be defined, and your skeleton source code will include a CASE statement and an input parameter to select which SQL statement to run. If you specify that your stored procedure executes a single query, only one SQL statement will be allowed in the **SQL Query** panel. However, even if you specify a single query, after the SQL stored procedure skeleton code is generated, you can modify it to include other SQL statements required for your stored procedure function.

In fact, usually stored procedures execute more than one SQL statement throughout the procedure logic. For these cases, you can select the **Run a single query** radio button, and later modify your procedure to include the additional statements.

The **Output** section allows you to specify if your stored procedure returns a result set to the calling application. If you select the **Return a result set** checkbox, a RESULT SETS 1 parameter is included in the CREATE PROCEDURE statement for your stored procedure. If your stored procedure returns more than one result set, you can change the numbers of result sets in the CREATE PROCEDURE statement manually after the source code generation.

The **Errors** section allows you to specify how you want your stored procedure to handle errors. For SQL stored procedures, the first option **Generate SQLEXCEPTION** does not generate any code in the source, and if an SQL error occurs, the stored procedure will be terminated.



The second option, **Output arguments for SQLSTATE and SQLCODE**, includes output parameters and a handler in your SQL stored procedure source code to provide the client application with the values of SQLSTATE and SQLCODE variables. SQL stored procedures can return SQLSTATE and SQLCODE information, but not SQLMESSAGE.

Following is an example of the code generated by this option:

```
CREATE PROCEDURE DRDARES1.PROC3 ( OUT SQLSTATE_OUT char(5),
                                OUT SQLCODE_OUT int )
    SPECIFIC DRDARES1.S1022844
    RESULT SETS 1
    LANGUAGE SQL
P1: BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE SQLCODE INT DEFAULT 0;

-----
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        IF (1 = 1) THEN
            SET SQLSTATE_OUT = SQLSTATE;
            SET SQLCODE_OUT = SQLCODE;
        END IF;
-----
END P1
```

At the time we were writing this book, the above code would only work properly with DB2 for OS/390 servers. In DB2 UDB, the values of SQLCODE and SQLSTATE variables are set after every statement, and in the above generated example, the IF and the SET statements would reset the original SQLCODE and SQLSTATE values. For more information on handling errors in SQL stored procedures running on DB2 UDB, refer to Chapter 5, "SQL Procedures for DB2 UDB for UNIX, Windows, OS/2" on page 145.

After choosing your options in the Pattern panel, when you click the **Next** pushbutton, the **SQL Query** panel is displayed, as shown in Figure 32.

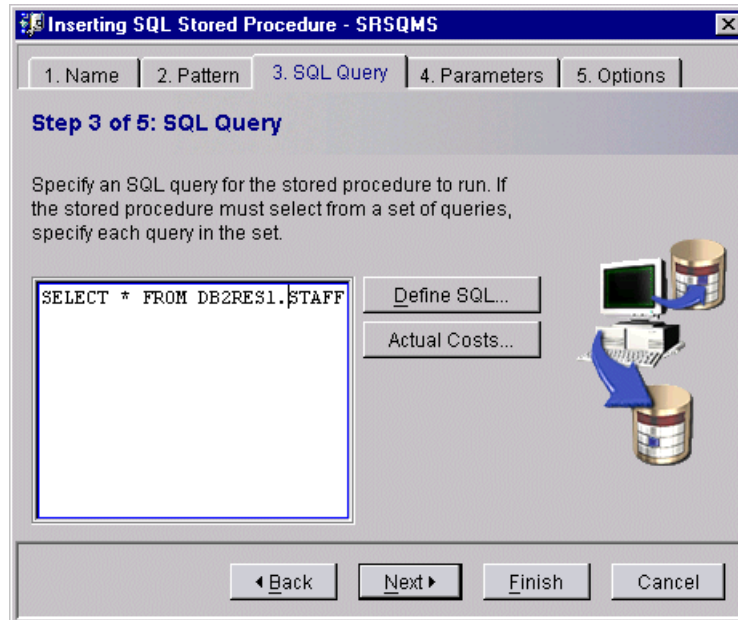


Figure 32. The SQL Query panel of the New Stored Procedures SmartGuide

The **SQL Query** panel allows you to create one or multiple SQL statements to be included in your SQL stored procedures code. You can type your SQL statement in the left area of the panel, or you can use the **SQL Assistant** SmartGuide. The **Define SQL** pushbutton invokes the **SQL Assistant** SmartGuide, that helps you to create SELECT, INSERT, UPDATE, and DELETE statements through dialogs that access the DB2 catalog. For more information on the **SQL Assistant** SmartGuide, refer to “SQL Assistant” on page 94.

The **Actual Costs** pushbutton is only available when you are creating an SQL stored procedure for a DB2 for OS/390 server. After you define your SQL statement, if you click the **Actual Costs** pushbutton, the stored procedure DSNWSPM is invoked at the DB2 for OS/390 server to evaluate the cost of your SQL statement and the results are presented by SPB.

Click on the **Next** pushbutton, and the **Parameters** panel is displayed, as shown in Figure 33.

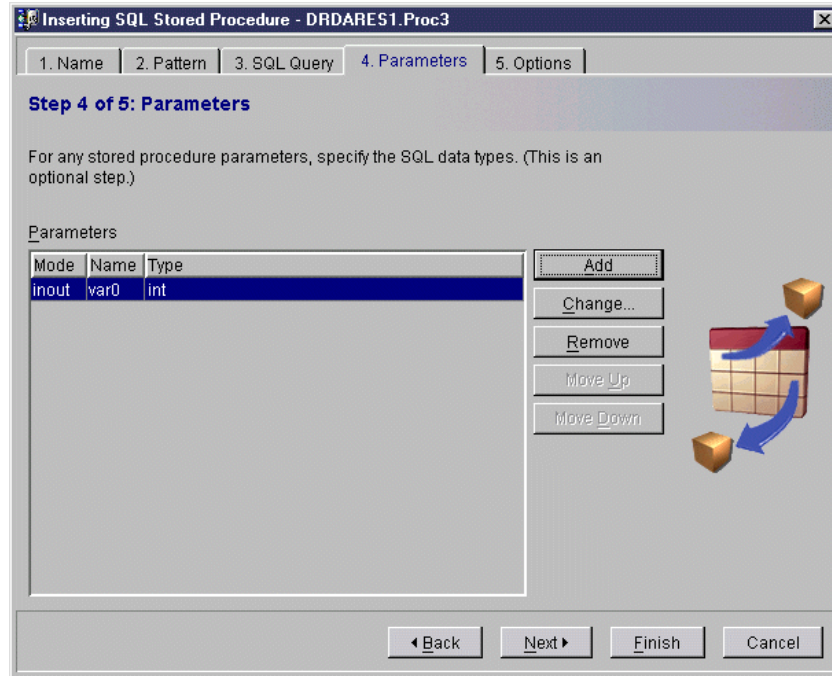


Figure 33. The Parameters panel of the New Stored Procedures SmartGuide

In the **Parameters** panel, you define the parameters that are sent to, or received from, the SQL stored procedure. If you click on the **Add** pushbutton, the **Define Parameter** dialog is displayed, as shown in Figure 34.

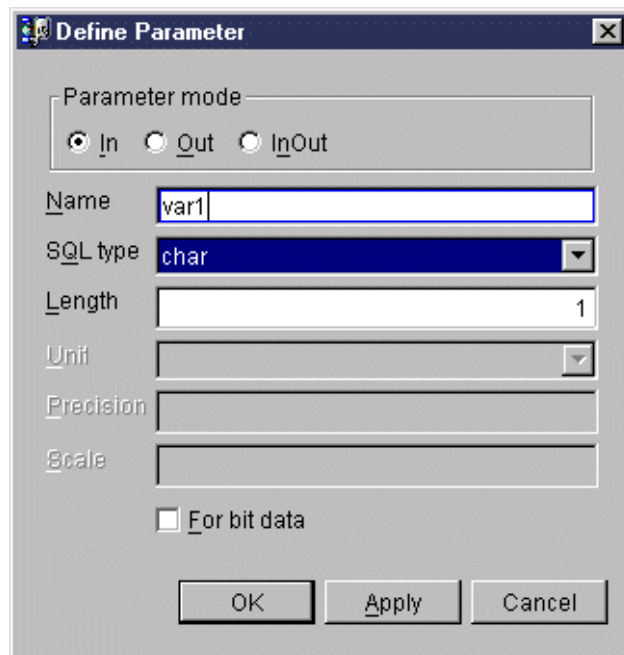


Figure 34. The Define Parameter dialog

In the **Define Parameter** dialog, you specify the characteristics of your SQL stored procedure parameters. The parameter mode defines if a parameter is used as an input, output, or input/output for the stored procedure. In this dialog, you also define the name and SQL type of the parameter. The length, unit, precision, and scale fields are requested according to the SQL type of the parameter. You cannot use user defined datatypes as SQL types for SQL stored procedures parameters.

The parameters you define in the **Parameters** panel are included in the generated CREATE PROCEDURE statement for the SQL stored procedure. Before generating the code for your SQL stored procedure, you can also change, delete, or change the order of the parameters in the **Parameters** panel. After the code is generated, you can change the definition of your parameters by editing the CREATE PROCEDURE statement.

After you define your stored procedure parameters, you can click on the Next pushbutton to go to the last panel of the **New Stored Procedures SmartGuide**, the **Options** panel, as shown in Figure 35.

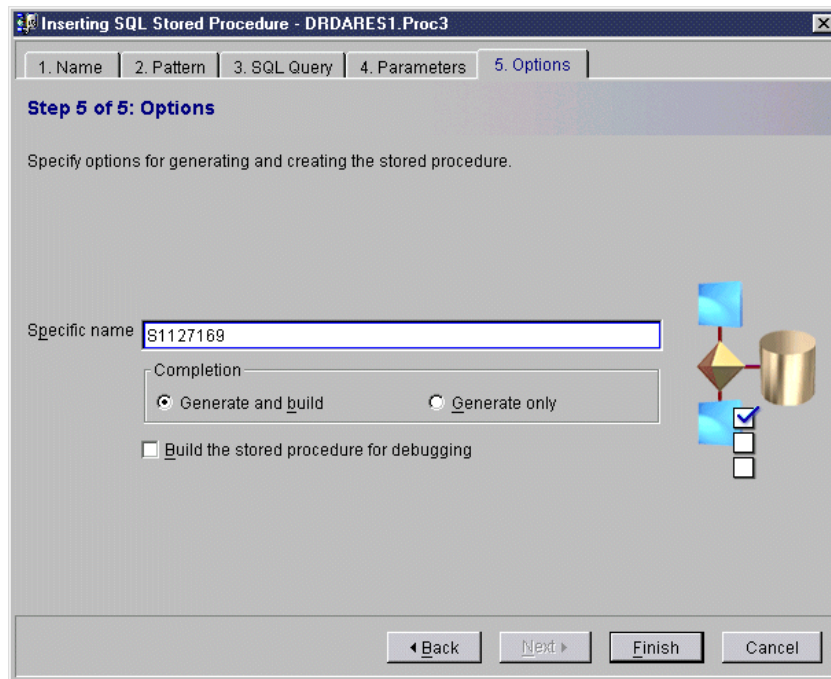


Figure 35. The Options panel of the New Stored Procedures SmartGuide

The **Options** panel allows you to specify options for generating and building your stored procedure. The options available are different for DB2 for OS/390 and DB2 UDB servers.

The **Options** panel displayed in Figure 35 is for DB2 UDB servers. For DB2 UDB servers, you may define the specific name of your SQL stored procedure. The specific name you type in the input field is included in the generated CREATE PROCEDURE statement.

In the **Completion** area of the panel, you can specify if you want to generate the source code and automatically build the procedure at the DB2 server, or if you want only to generate the source code. In most cases, you should choose the **Generate only** option, to add specific logic to your SQL stored procedure. After the generation, you will be able to change the SQL stored procedure source to include your changes, and then build the procedure.

The **Build the stored procedure for debugging** checkbox should be selected if you want to use the IBM Distributed Debugger to debug your procedure. When selected, this option includes an entry in the DB2 UDB debugger table.

The Options panel for DB2 for OS/390 SQL stored procedures allows you to specify the collection id and load module name for the SQL stored procedure in entry fields. For DB2 for OS/390, the Options panel also includes an **Advanced** options pushbutton. If you click on the **Advanced** pushbutton, the OS/390 Options window is displayed, as shown in Figure 36 and Figure 37. In this window, you can specify parameters used during the build process, that are passed to the DSNTPSMP stored procedure in the mainframe.

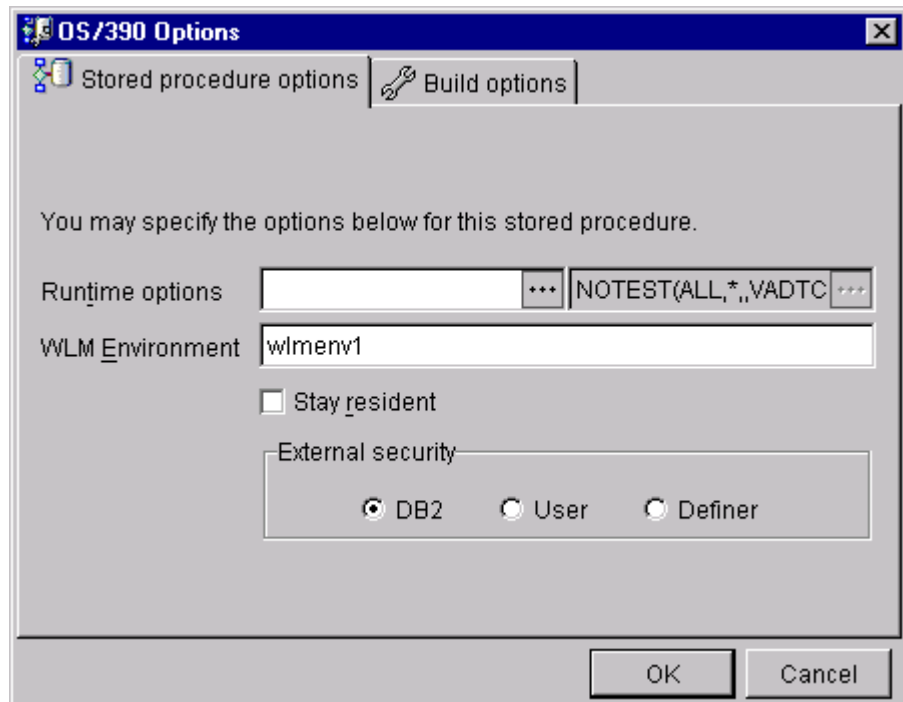


Figure 36. The Advanced options for DB2 for OS/390 SQL stored procedures

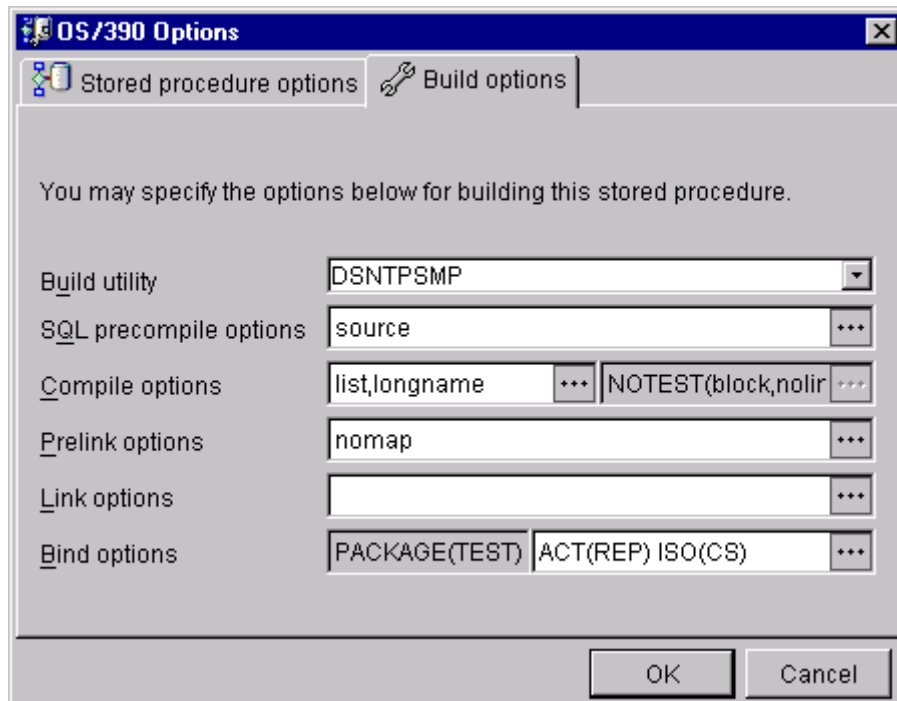


Figure 37. The Advanced build options for DB2 for OS/390 SQL stored procedures

Note that in the OS/390 Options, you can also specify the name of the stored procedure to be invoked for the build process in the Build name entry field. By default, SPB invokes the DSNTPSMP REXX stored procedure, but you can change this procedure for your installation, and specify the name of your customized build procedure in the OS/390 options.

### 3.4.2.2 SQL Assistant

The **SQL Assistant** is a SmartGuide that steps you through the processing of creating SQL statements. You can select tables on which to run queries, join tables, enter conditions and columns, determine how to sort the result, and display the SQL statement so that you can copy or test the SQL query.

You can invoke the **SQL Assistant** during the creation of a new stored procedure, by clicking on the **Define SQL** pushbutton in the **SQL Query** panel of the **New Stored Procedures SmartGuide**. You can also invoke the **SQL Assistant** when modifying your stored procedure by clicking on the **Insert SQL** icon from the SPB toolbar.

The SQL Assistant guides you through a series of steps to create your SQL statement. In the first step, you choose the type of SQL statement being generated and the tables that are referenced by your statement, as shown in Figure 38.

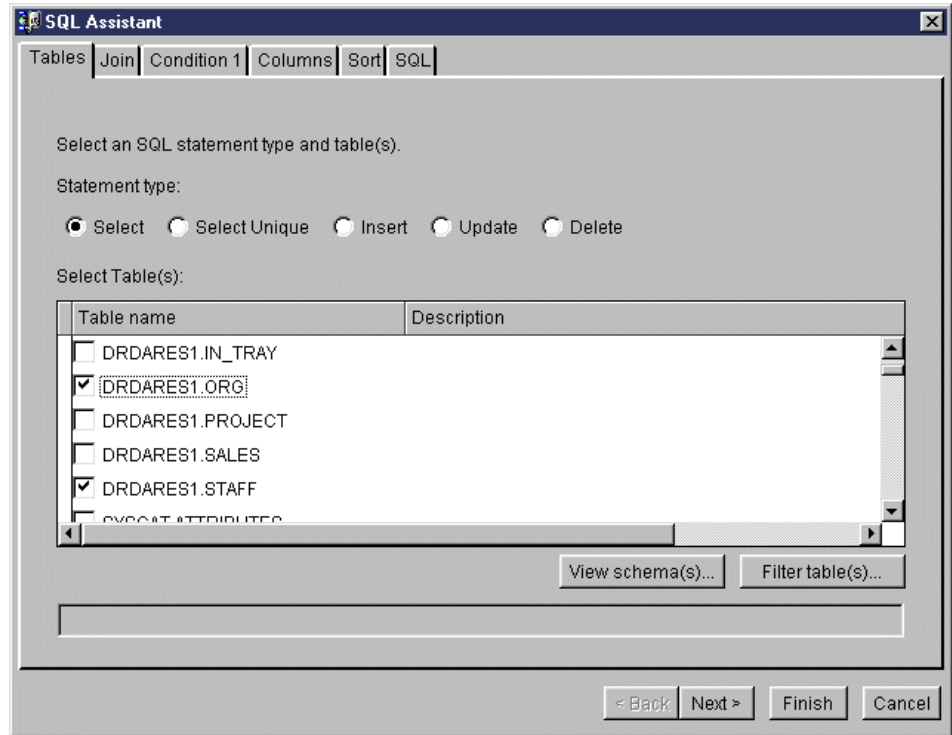


Figure 38. The Tables panel of the SQL Assistant

In the **Tables** panel of the SQL Assistant, you can choose if you want to generate a SELECT, INSERT, UPDATE, or DELETE statement. However, if you select more than one table from the list, the SQL Assistant only allows SELECT statements to be generated. You can refine the list of tables being presented by using the **View schemas** and the **Filter tables** pushbuttons.

After selecting the tables for your SQL statement, the **Join** panel of the SQL Assistant is displayed, as shown in Figure 39.

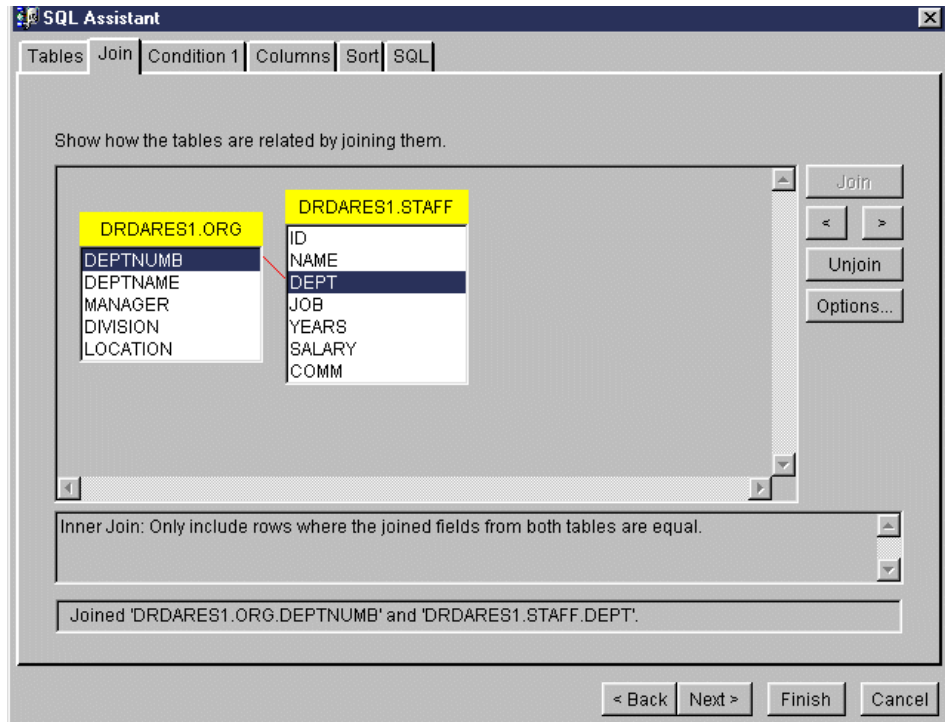


Figure 39. The Join panel of the SQL Assistant

In the **Join** panel, you can click on the columns on each table that are used to join the tables to highlight them. After that, you can click on the **Join** pushbutton to create the join. By default, an inner join is created, but if you click on the **Options** pushbutton, you can change the type of the join being created, as shown in Figure 40. In the **Join properties** panel, you can choose if you want to create an inner join, a left outer join, or a right outer join between the selected tables.

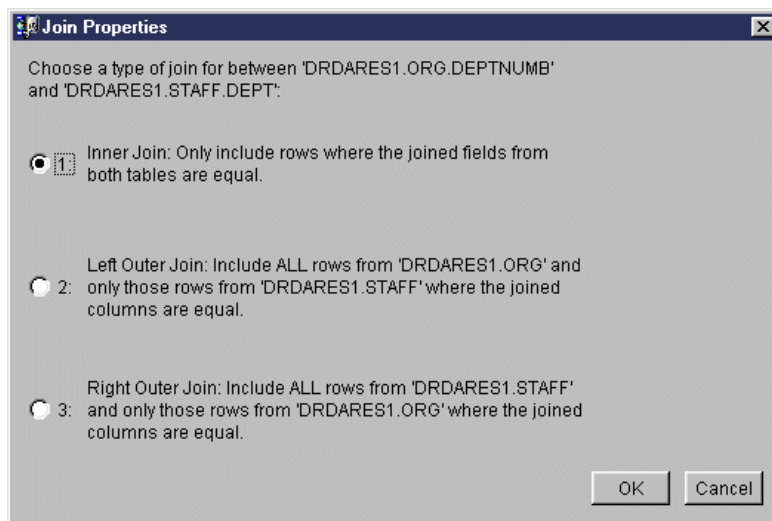


Figure 40. Changing the type of Join created by SQL Assistant

The next panel of the SQL Assistant is the Conditions panel. In this panel you can specify search conditions that are included in the WHERE clause of your SQL statement. The Conditions panel is shown in Figure 41.



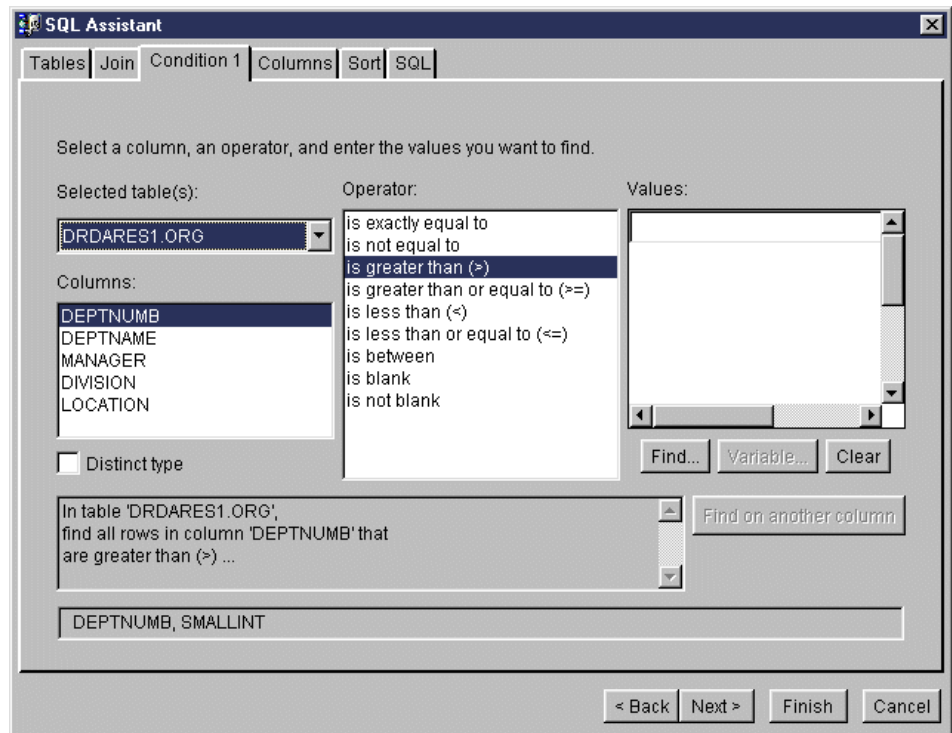


Figure 41. The Conditions panel of the SQL Assistant

In the **Conditions** panel, you can include search conditions based in any column of any of the tables included in your SQL statement. Just select the table you want to add the condition, click on the column name and operator you want to highlight them. Then, in the **Values** section of the panel, type the search value for your condition. To check the existing values for the column you are defining your condition, just click on the **Find** pushbutton below the **Values** section. If you want, you can click on the **Variables** pushbutton to define a variable for your search condition as shown in Figure 42.

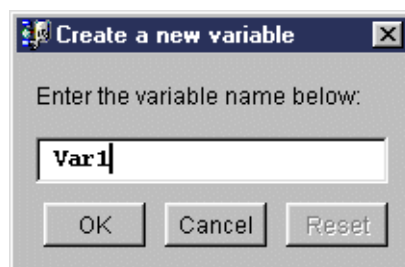


Figure 42. Specifying a variable for a condition

You can define as many conditions as you want. To define new conditions for your SQL statement, just click on the **Find on another column** pushbutton.

The next panel of the SQL Assistant is the **Columns** panel, as shown in Figure 43.

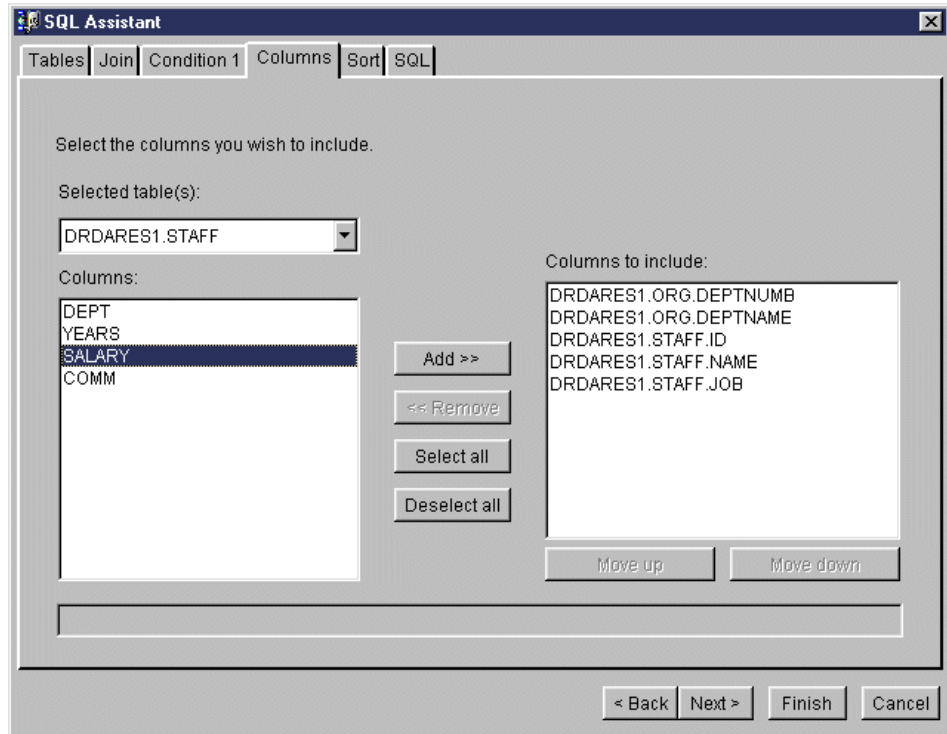


Figure 43. The Columns panel of the SQL Assistant

In the **Columns** panel, just select the columns of each table that you want to include in your SQL statement, by highlighting the column name and then clicking on the **Add** pushbutton.

After selecting the columns for your SQL statement, you can specify one or more columns to be used to sort the output of your statement, using the **Sort** panel, as shown in Figure 44.

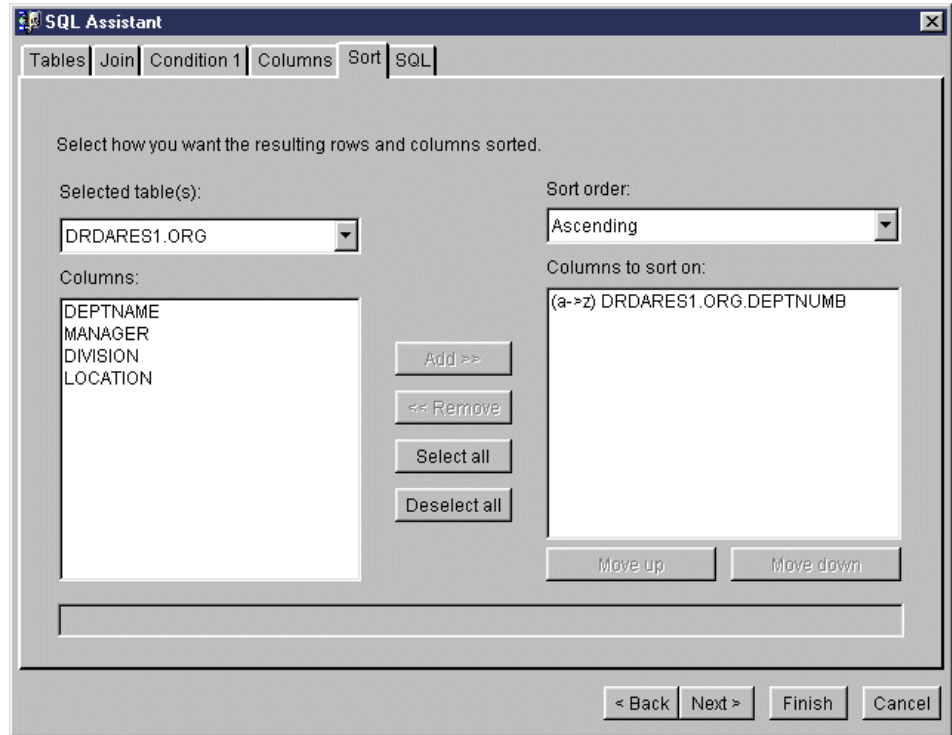


Figure 44. The Sort panel of the SQL Assistant

In the **Sort** panel, you can select the columns of any of the referenced tables to be used for sorting the output. The columns you select are included in the **ORDER BY** clause of your SQL statement.

The last panel of the SQL Assistant is the **SQL** panel, as shown in Figure 45.

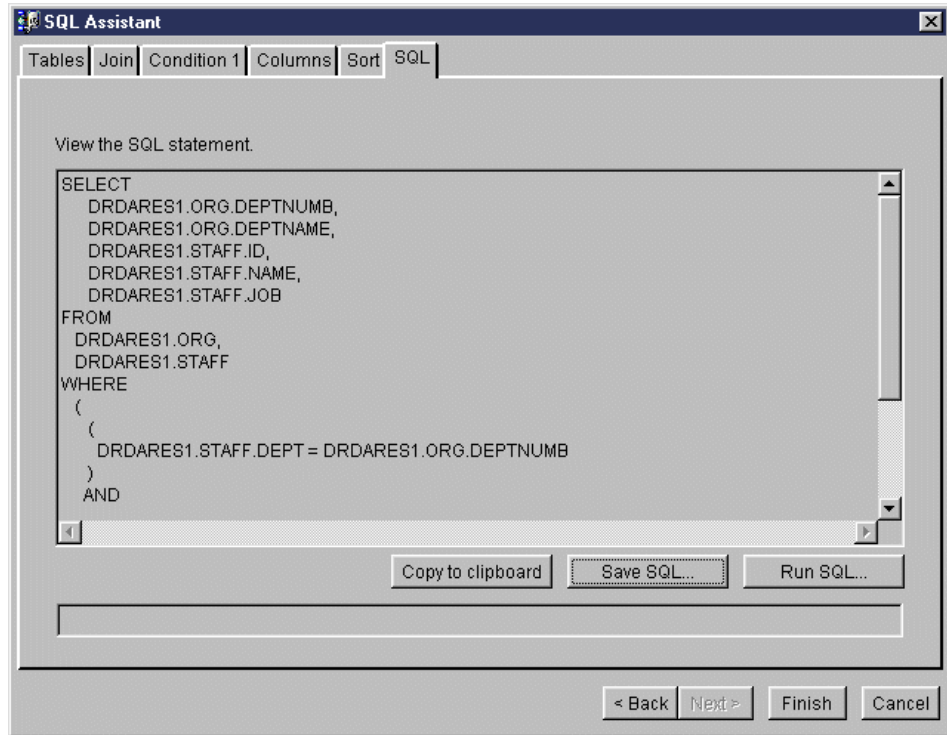


Figure 45. The SQL panel of the SQL Assistant

The **SQL** panel displays the generated SQL statement that will be included in your stored procedure. If you click on the **Finish** pushbutton, the SQL Assistant ends and the SQL statement is inserted in the stored procedure source code. Before closing the SQL Assistant, in the SQL panel you have three pushbuttons that you can use.

The **Copy to clipboard** pushbutton copies the generated SQL statement to the Windows clipboard, so you can paste it in any application that has access to the Windows clipboard. The **Save SQL** pushbutton saves the generated SQL statement to a file in your hard disk. The **Run SQL** pushbutton allows you to test the generated SQL statement, before inserting the statement in your SQL stored procedure code.

If your SQL statement is expecting variables, when you click on the **Run SQL** pushbutton, you are prompted to enter the values for the variables, as shown in Figure 46.

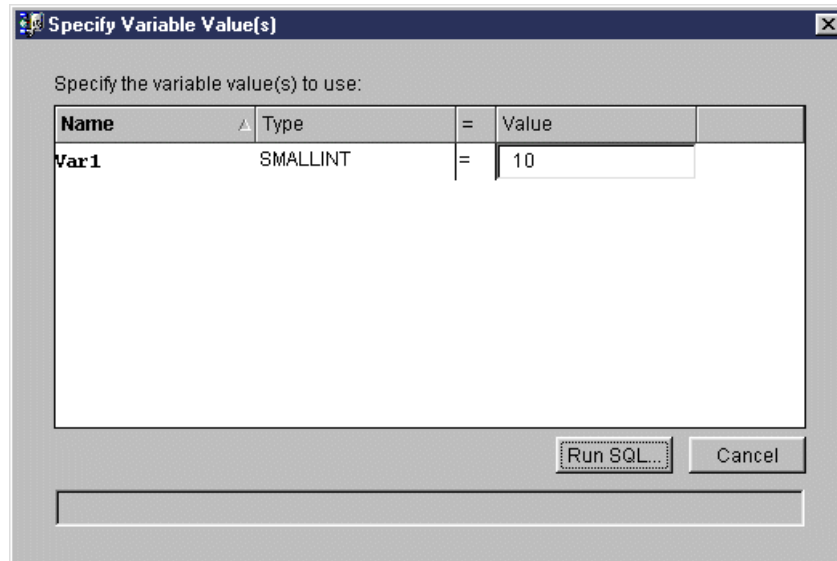


Figure 46. Entering values for variables in the SQL statement

Enter the values for the variables and click on **Run SQL** pushbutton again. The results of your SQL statement are displayed as shown in Figure 47. You can copy these results to the clipboard, or save them to a file if you want. Click on the **OK** pushbutton to return to the **SQL** panel of the SQL Assistant.

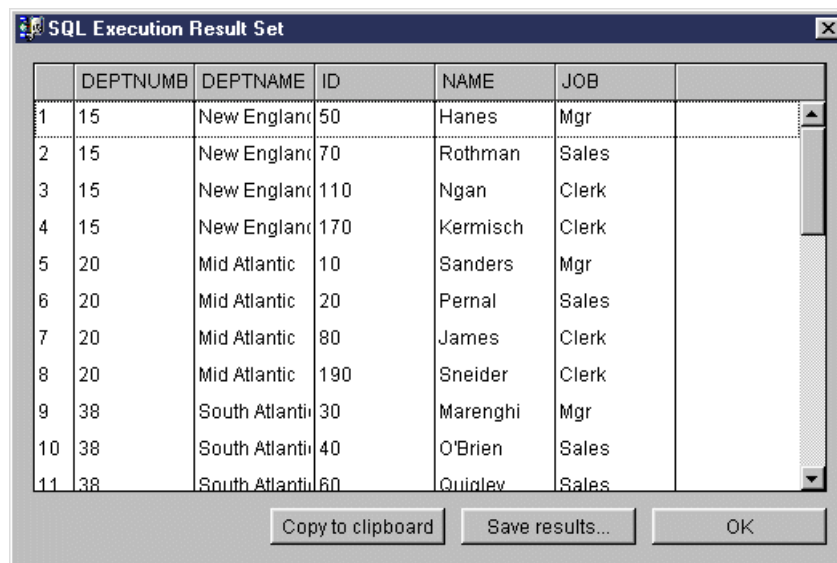


Figure 47. Displaying the results of the SQL Statement

### 3.4.3 Building stored procedures

The process of building a stored procedure is different between DB2 for OS/390 and DB2 UDB servers. SPB handles this difference and builds the stored procedure according to the DB2 server being accessed.

For DB2 for OS/390 servers, the current version of SPB only supports creation and building of stored procedures written in the SQL Procedures language. The build process of SPB, by default, invokes the OS/390 Procedure Processor,

which is a REXX stored procedure (DSNTPSMP), in the mainframe to build the new or changed stored procedure. You can customize the DSNTPSMP procedure at the mainframe to better fit the needs of your environment, and change SPB parameters, so it will invoke your customized procedure instead of the default DSNTPSMP.

For DB2 UDB (Windows, UNIX), the build process invokes the command processor of DB2 to build the new or changed procedure.

For DB2 UDB for AS/400, you cannot use SPB, however, if you have a local database with tables defined with the same structure of your AS/400 database, you can create the procedure with SPB working with the local database. You can then, save the stored procedure in a sequential file and upload it to AS/400. The stored procedure can be built using AS/400 commands with minimum changes. That was what we did in this project. Refer to Chapter 6, “SQL Procedures for DB2 UDB for AS/400” on page 169 for detailed information about how to build an SQL stored procedure for DB2 UDB for AS/400.

In both cases, prior to building the stored procedure, SPB drops the existing version of the stored procedure on the DB2 server.

The process of building an SQL stored procedure involves the creation of the executable file and the registration of the stored procedure at the DB2 server.

To create the executable file, both DB2 UDB and DB2 for OS/390 engines generate an intermediate C source code that is precompiled, compiled, and linked at the DB2 server. For more information refer to Chapter 4, “SQL Procedures for DB2 UDB for OS/390” on page 109, Chapter 5, “SQL Procedures for DB2 UDB for UNIX, Windows, OS/2” on page 145 and Chapter 6, “SQL Procedures for DB2 UDB for AS/400” on page 169.

#### **3.4.4 Modifying existing stored procedures**

Using SPB you can easily modify SQL stored procedures already built in the DB2 for OS/390 or DB2 UDB servers.

To modify an existing stored procedure, double-click on the name of the stored procedure. SPB gets the SQL stored procedure source from the DB2 server and opens the edit window, as shown in Figure 48.

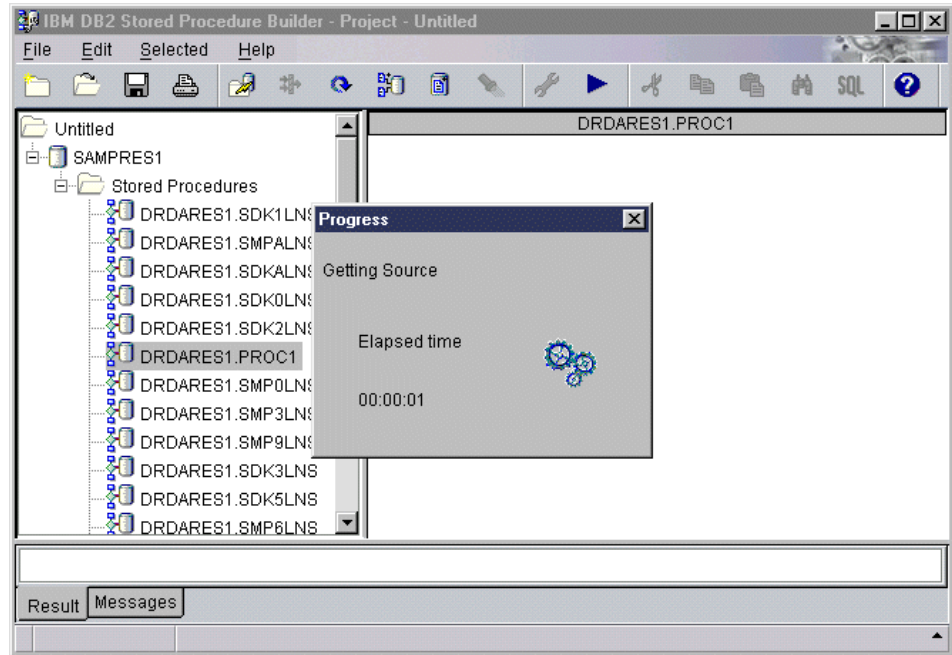


Figure 48. Modifying an existing stored procedure

For DB2 UDB servers, the source of the SQL stored procedure is saved in the SYSIBM.SYSPROCEDURES table, regardless of the method used to create the procedure. In this case, SPB always find the source code on the DB2 server and displays it for your modifications.

For DB2 for OS/390 servers, when you create your SQL stored procedure using SPB, the source code is stored in the SYSIBM.SYSPSM table. However, if you did not use the SPB to build your procedure, it is possible that the source code is not stored in the server. In this case, when you try to get the source of your SQL stored procedure, SPB displays a window prompting you to specify a file, residing on your workstation, containing the source code.

After the source code is displayed in the edit window, you can type any modifications you want to your stored procedure. The current version of the SPB editor does not check the syntax of the statements you are including or changing. Syntax errors are only detected during the build process of the stored procedure.

Any changes you do to your stored procedure are not included in the DB2 server until you build it again. While you are changing the SQL stored procedure, the name of the procedure is shown in bold characters in the tree view part of the SPB main window. In this case, the procedure is referred as a **dirty procedure**, meaning that the code you have in SPB is not the same as in the DB2 server.

If you make changes to your SQL stored procedures, and do not build the procedure back to the DB2 server, when you close SPB you are prompted to save your changes locally, so you do not lose any of your modifications. Remember that other developers will not be able to see your changes until you build them to the DB2 server, and that SPB does not control concurrent access to the same SQL stored procedure.





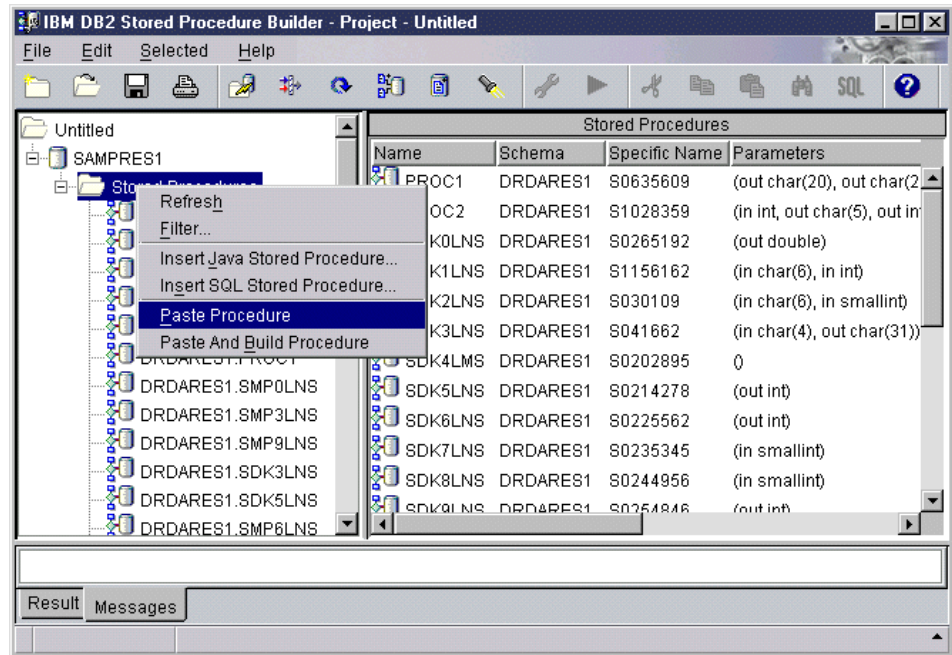


Figure 50. Paste the stored procedure at the target server

When pasting the stored procedure at the target server, you have two options. If you choose **Paste Procedure**, the procedure is created only in SPB but not on the target DB2 server. In this case, you can change the stored procedure code before building it to the DB2 server. If you do not plan to change the stored procedure, you can select **Paste and Build Procedure**, that copies the procedure and builds it to the DB2 server without any changes to the original source code.

### 3.4.6 Debugging stored procedures

The DB2 SPB can help you to debug your SQL stored procedures. The IBM Distributed Debugger is shipped with DB2 SDK, and can be used to debug SQL stored procedures.

There are a few tasks to perform at both the DB2 server and client workstations. If you plan to debug your stored procedure, it has to be prepared with parameters that will trigger the debugging process during the execution of the stored procedure. The steps to prepare the SQL stored procedure at the DB2 server for debugging are different for DB2 UDB and DB2 for OS/390 servers. For more information on how to prepare your DB2 for OS/390 SQL stored procedures for debugging, refer to 4.6, “Stored procedure debugging” on page 142. For more information on how to prepare your DB2 UDB for UNIX, Windows, and OS/2 SQL stored procedures for debugging, refer to 5.6, “Stored procedure debugging” on page 166.

The client workstation for debugging remote stored procedures must be executing a Windows NT environment. During our project, we worked with a beta version of the IBM Distributed Debugger.

The DB2 SPB is not a prerequisite for debugging your SQL stored procedures. You can debug your SQL stored procedures even if you did not create them with

SPB. The SPB can help you with panels to customize the DB2 server for debugging, but the process of debugging is independent of SPB. Figure 51 shows how the debugger process is triggered for stored procedures.

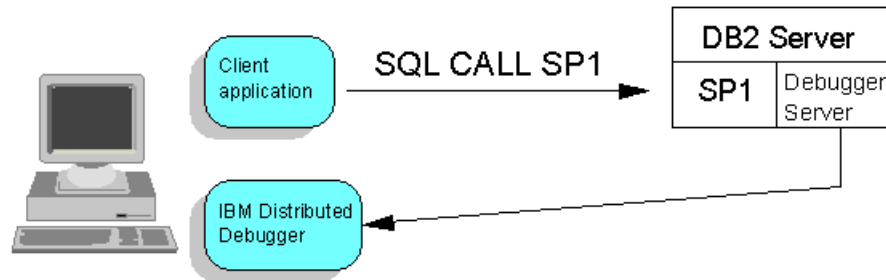


Figure 51. Debugging process

SPB also avoids the need to create a client application to debug your stored procedure. You can invoke your procedure from SPB and the debugging process is triggered.

To be able to debug remote stored procedures, you must have the IBM Distributed Debugger client daemon executing on your workstation. To start the debugger client daemon, you can issue the following command:

```
idebug -qdaemon -quiport=8000
```

The above command starts the debugger client listener in TCP/IP port 8000. In your DB2 server running the procedure, you need to inform this port and the IP address of your client machine, so when the stored procedure executes on the server, the debugger is started in your workstation. Figure 52 shows the IBM Distributed Debugger daemon window, when waiting for a remote connection.

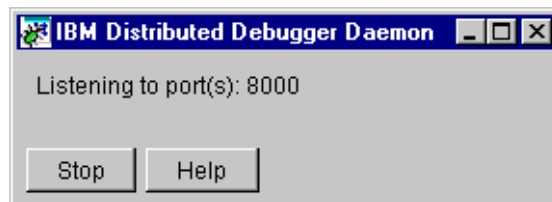


Figure 52. IBM Distributed Debugger daemon

When your procedure starts in the DB2 server, the debugger code in the server, sends a message to the debugger client, and the IBM Distributed Debugger main window is started. Figure 53 shows the IBM Distributed Debugger main window.

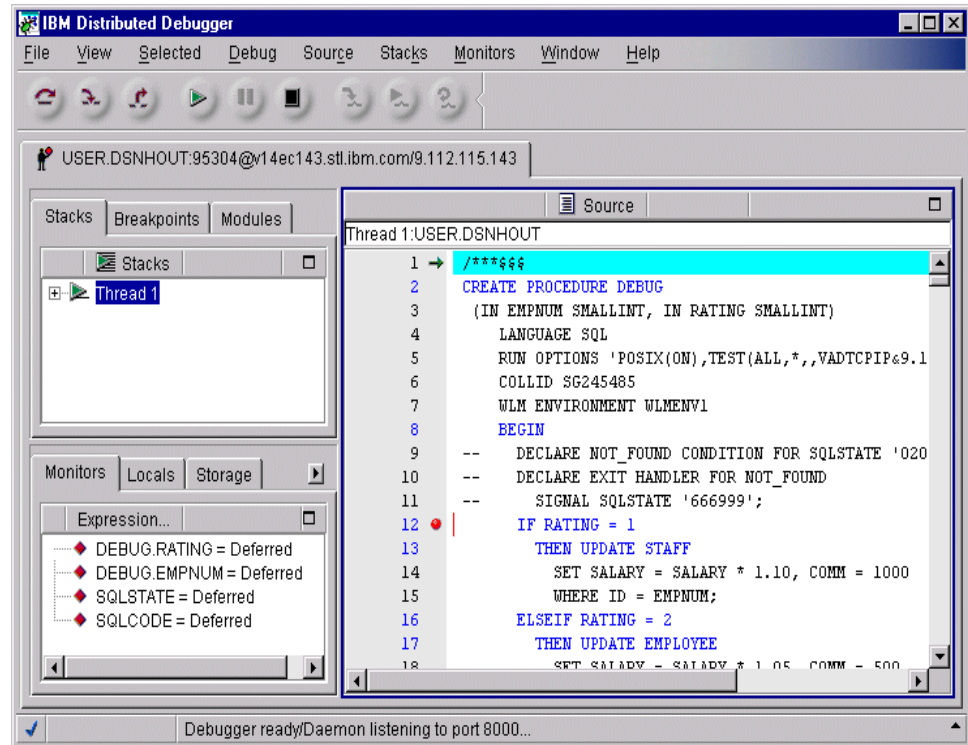


Figure 53. IBM Distributed Debugger main window

The IBM Distributed Debugger main window is divided into three main parts: one containing the source code of your stored procedure, one with monitors, and one with stacks. On the top of the window, you have controls that allow you to manage the execution of your procedure, step-by-step if you want.

In the source code part of the main window, an arrow shows the current statement being executed. Due to the fact that SQL Procedures generates a C code, during our project with the beta version of the debugger, we had to step many times to go from one SQL stored procedure statement to the following, because it was actually stepping on the C code generated. This may change when the final version is released.

You can also set breakpoints in your source code, indicated by a red dot next to the line number. To set breakpoints, all you have to do is double-click next to the line number and the breakpoint is set. You can only set breakpoints in lines that actually execute some code, so you will not be able to set breakpoints in lines with comments, for example.

In the monitor part of the main window, you can monitor and change values of variables and parameters of your stored procedure. To start monitoring the values of a variable, just click on **Monitor -> Add variable to program monitor**. The Monitor Expression window appears and you can type the name of the variable you want to monitor. Figure 54 shows the Monitor Expression window.

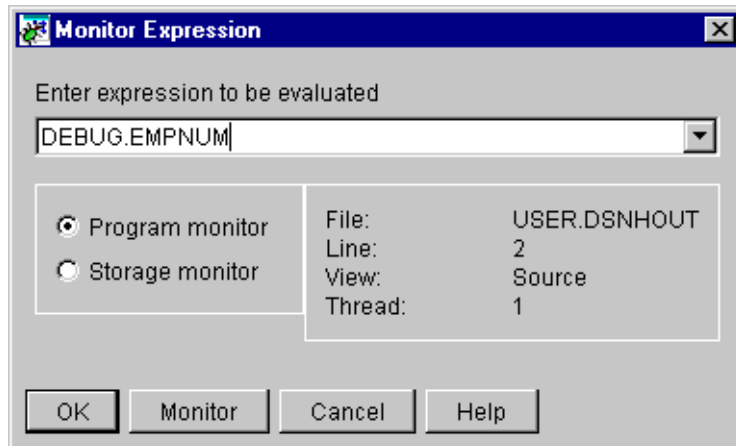


Figure 54. Monitoring variables

Remember that during the generation of the C code your parameters are prefixed with the procedure name, and your variables declared within a compound statement are prefixed with the label of the compound statement. You must remember to type the prefixed name of the variable, or you will not be able to add the variable to the monitor. In our example, this is the name of the stored procedure, `DEBUG`, as typed before the name of the parameter we wanted to monitor, `EMPNUM`.

After you have added your variable or parameter to the monitor, you can easily change the contents of it, by simply double clicking on the name.

With the IBM Distributed Debugger graphical interface, it is very easy to understand and debug the logic of your stored procedures running in any DB2 server in your network.

---

## Chapter 4. SQL Procedures for DB2 UDB for OS/390

In this chapter, we explain how to code SQL Procedures for DB2 UDB for OS/390 version 5 and version 6 servers. We focus on system requirements and planning, as well as the rules for writing, preparing, and debugging an SQL Procedures program. Also, we show how the new tool, Stored Procedure Builder (SPB), can be used in conjunction with the SQL Procedures support on OS/390.

---

### 4.1 General considerations

The SQL Procedures language support on the OS/390 platform has been implemented for DB2 for OS/390 version 5 and DB2 UDB for OS/390 version 6.

The SQL Procedures support improves the development and usability of stored procedure, ensures the portability of SQL stored procedures across DB2 platforms, and simplifies migration to DB2 from other environments through the use of a translation tool (expected to be made available by first quarter of 2000).

There are three main methods, describe in detail in this chapter, for developing of SQL stored procedures on OS/390:

- Method 1 — Using the Stored Procedure Builder (SPB) tool, which runs on all current Windows platforms, and later will be made available on UNIX. The SPB invokes the OS/390 SQL Procedure Processor for building the stored procedure.
- Method 2 — Using the OS/390 SQL Procedure Processor (DSNTPSMP).
- Method 3 — Using Job Control Language (JCL) or Command List (CLIST).

For most situations, we recommend that you use Method 1, because the SPB tool helps you in coding, testing and debugging of your stored procedure, thereby improving the development process (see Chapter 3, “The DB2 Stored Procedure Builder” on page 57 for more information about the SPB). However, you should also read the section for each corresponding method and choose the one that fits best in your environment.

---

### 4.2 System requirements and planning

If you already have version 6 already installed, or if you are still using version 5 and are planning to use this new stored procedure programming language, you need to follow the steps below, which describe the prerequisites for this support. Otherwise, this support will be provided as part of installing or migrating to version 6.

#### 4.2.1 Requirements for DB2 for OS/390 Version 5

The trial beta version of the SQL Procedures language support is shipped as a zip file, `sqlproc1.zip`, which you can download from:

<http://www.software.ibm.com/data/db2/os390/sqlproc>

This Web download includes the following software, which you will need to apply to your OS/390 environment:

- Load modules for SQL Procedures language support
- JCL and SQL samples, including DSNHSQL, used to create your SQL stored procedures
- A `readme` file, which contains detailed installation instructions

Once you have done this, you will be able to develop and prepare stored procedures written in the SQL Procedures language. At this point, the preparation process for your SQL stored procedures can be done manually, that is, through JCL, as described in 4.5.3, “Using JCL” on page 136.

If you plan to use the OS/390 Procedure Processor, either directly or through the SPB, you must download an additional zip file, `sqlprocp.zip`. This additional zip file includes the following software:

- PTF for DB2 APAR PQ24199 — Dynamic invocation of the bind
- PTF for DB2 APARs PQ29706 and PQ32467— REXX stored procedure support
- REXX exec DSNTPSMP — The OS/390 Procedure Processor
- JCL job DSNTIJSQ, which must be manually customized by the user. Directions are provided in the JCL prologue. This job performs the following:
  - Creates and defines the DSNTPSMP stored procedure to DB2 and grants EXECUTE to PUBLIC.
  - Executes the DDL to create the SQL Procedures database, tablespaces, tables and indexes. See 4.2.4, “Creating non-catalog DB2 tables” on page 112 for details about this database.
  - Grants SELECT access to the SQL Procedures tables.
- JCL and samples, including DSNWLMP, which is a sample JCL procedure to start the WLM-managed stored procedures address space required by DSNTPSMP. Customize this procedure for your site by following the directions in the prologue, and then copy it to your system PROCLIB.
- A `readme` file, which contains detailed instructions.

The REXX language support feature must also be installed if you plan to use the OS/390 Procedure Processor. The following feature numbers are orderable through your IBM Representative:

- 5861 for install using 6250 tape
- 5862 for install using 3480 cartridge
- 5275 for install using 4mm DAT

If you plan to use the OS/390 Procedure Processor through the SPB, then you need to install the SPB product on your PC (see Chapter 3, “The DB2 Stored Procedure Builder” on page 57). If you want to use the SPB SQL Costing Information, you need to install the following PTFs on your OS/390 system:

- PTF for DB2 APAR PQ23162
- PTF for DB2 APAR PQ24230

**Note:** For detailed information about how to install the SQL Procedures code, refer to the document *DB2 for OS/390 Version 5 Preview of SQL Procedures*, available at the following URL (download site):

<http://www.software.ibm.com/data/db2/os390/sqlproc>

Details will also be available in the `readme` file that is downloaded with the `zip` files.

#### 4.2.2 Requirements for DB2 UDB for OS/390 Version 6

The following PTFs must be applied in your environment:

- PTF for DB2 APARs PQ29782 and PQ30467 — SQL Procedures support for DB2 pre-compiler
- PTF for DB2 APAR PQ24199 — Dynamic invocation of the bind
- PTF for DB2 APAR PQ30219 — REXX stored procedure support
- PTF for DB2 APAR PQ30492 — This includes the following components:
  - Modifications to DB2 Install parts in support of SQL Procedures
  - JCL and SQL samples
  - Jobs to create the SQL Procedures database
  - The OS/390 Procedure Processor (DSNTPSMP)
  - JCL job DSNTIJSQ, a post-install job that creates objects required for DB2 SQL Procedures. This job must be manually customized by the user. Directions are provided in the JCL prologue. This job performs the following:
    - Creates and defines the DSNTPSMP stored procedure to DB2 and grants EXECUTE to PUBLIC.
    - Executes the DDL to create the SQL Procedures database, tablespaces, tables, and indexes. See 4.2.4, “Creating non-catalog DB2 tables” on page 112 for details about this database.
    - Grants SELECT access to the SQL Procedures tables.
    - Copies JCL procedure DSNHSQL to the system PROCLIB.
    - Copies DSNTPSMP to prefix.NEW.SDSNCLST, where the WLM Startup procedure for DSNTPSMP will expect to find it.
- A sample JCL procedure (DSN8WLMP), which starts the WLM-managed stored procedures address space required by DSNTPSMP. Customize this procedure for your site by following the directions in the prologue, and then copy it to your system PROCLIB.
- The REXX language support feature is a prerequisite for REXX stored procedure and must also be installed if you plan to use the OS/390 Procedure Processor.
- PTF for DB2 APAR PQ30439 — External Savepoint support
- PTF for DB2 APAR PQ32670 — Declared Temporary Table support
- PTF for DB2 APARs PQ30684 and PQ30652 — Identity Columns support
- PTF for DB2 APAR PQ24891 - SPB SQL Costing Information

Further components will be forthcoming through APARs as they become available.

**Note:** For detailed information about how to install the version 6 SQL Procedures code, refer to the document *DB2 UDB for OS/390 Version 6 Preview of SQL Procedures*, available at the URL:

<http://www.software.ibm.com/data/db2/os390/spb>

**Important:**

This is the site to use to get the trial code for SQL Procedures and the Stored Procedure Builder:

<ftp://ftp.software.ibm.com/software/os390/db2server/fixes/db2apars/>

### 4.2.3 Remote Debugger and Debug tool

If you plan to use the Remote Debugger and the Debug tool to debug your stored procedures (see 4.6, “Stored procedure debugging” on page 142), you need to:

- Apply DB2 PTF APAR PQ30773 (for DB2 version 5 and version 6).
- Install the Remote Debugger on your PC (see 3.4.6, “Debugging stored procedures” on page 105).
- Install the Debug tool on your OS/390 system.

The beta for the Debug tool code and information can be downloaded from the following Web site:

<http://www.software.ibm.com/ad/c390/cmvsbeta.htm>

Following are the requirements to install the Debug tool on your OS/390 system environment:

- TCP/IP version 3.2
- LE/390 base C compile with debug (OS/390 optional feature codes 5962, 5963, 5712).
- 5655-B85 IBM C/C++ Productivity Tools for OS/390 Release 1.

Once you have installed the Debug tool, you have to:

- Apply the PTF for Debug tool APARs PQ27247 and PQ25905, on your OS/390 system. These PTFs contain additions to the Debug tool for stored procedure debugging.

### 4.2.4 Creating non-catalog DB2 tables

The SQL Procedures database DSNPSPM contains the tablespace DSNPSPM, the three non-catalog DB2 tables SYSIBM.SYSPSM, SYSIBM.SYSPSMOPTS, and SYSIBM.SYSPSMOUT, and the indexes DSNPSMX1, DSNPSMX2, and DSNPXM0X1. These are required for the OS/390 Procedures Processor. You do not need to create these if you are planning to build your SQL stored procedures using JCL only (see 4.5.3, “Using JCL” on page 136).

This SQL Procedures database is created by:

- For customers using version 5: Running the job DSNTIJSQ when installing from the `sqlprocp.zip` download.



- For customers installing a new DB2 version 6 system: Running the normal installation scenario, using the DB2 install CLIST and customized install and sample jobs.
- For customers migrating to version 6: Running installation job DSNTIJSJG.
- For customers who have already installed or migrated to version 6: Running post-installation job DSNTIJSQ, which is delivered with the PTF for APAR PQ30492.

#### 4.2.4.1 Creating SYSIBM.SYSPSM

Table 9 shows the definition of SYSIBM.SYSPSM, which stores the unmodified source code of the SQL stored procedure.

Table 9. SYSIBM.SYSPSM

Column name	Format(Length)	Description	Nullable	Default
SCHEMA	CHAR(8)	Owner ID	Y	Y
PROCEDURENAME	CHAR(18)	SQL stored procedure name	N	N
SEQNO	SMALLINT	Used to sequence lines for SQL stored procedure source code greater than 3800 characters	N	N
PSMDATE	DATE	Creation date	N	N
PSMTIME	TIME	Creation time	N	N
PROCCREATESTMT	VARCHAR(3800)	SQL stored procedure source	N	N

#### 4.2.4.2 Creating SYSIBM.SYSPSMOPTS

Table 10 shows the definition of SYSIBM.SYSPSMOPTS, which stores the precompile, compile, prelink, link and bind options.

Table 10. SYSIBM.SYSPSMOPTS

Column name	Format(Length)	Description	Nullable	Default
SCHEMA	CHAR(8)	Owner ID	Y	Y
PROCEDURENAME	CHAR(18)	SQL stored procedure name	N	N
BUILDSHEMA	CHAR(8)	Owner ID of build module. Usually SYSPROC	Y	Y
BUILDNAME	CHAR(18)	Build module name. Usually DSNTPSMP.	Y	Y
BUILDOWNER	CHAR(8)	Builder ID of SQL stored procedure (current SQLID)	Y	Y
PRECOMPILE_OPTS	VARCHAR(255)	Precompile options	Y	Y
COMPILE_OPTS	VARCHAR(255)	Compile options	Y	Y
PRELINK_OPTS	VARCHAR(255)	Prelink options	Y	Y
LINK_OPTS	VARCHAR(255)	Link options	Y	Y

Column name	Format(Length)	Description	Nullable	Default
BIND_OPTS	VARCHAR(1024)	Bind options	Y	Y
SOURCEDSN	VARCHAR(255)	Data set name of SQL stored procedure source code	Y	Y

#### 4.2.4.3 Creating SYSIBM.SYSPSMOUT

Table 11 shows the definition of SYSIBM.SYSPSMOUT, which is a global temporary table for DSNTPSMP.

Table 11. SYSIBM.SYSPSMOUT

Column name	Format(Length)	Nullable	Default
STEP	VARCHAR(16)	Y	Y
FILE	VARCHAR(8)	Y	Y
SEQN	INTEGER	Y	Y
LINE	VARCHAR(255)	Y	Y

Refer to 4.4, “Stored procedure preparation” on page 118 for details about how these tables are updated.

#### 4.2.5 WLM requirements for OS/390 Procedure Processor

The OS/390 Procedure processor (DSNTPSMP) should be set up in a WLM environment of its own. It is recommended that no other stored procedure should be defined to this environment. The OS/390 Procedure Processor requires that the NUMTCB in the WLM managed region be 1, which means that only one instance of DSNTPSMP will be able to execute in its own WLM environment at any time.

The main consequence is that only one SQL stored procedure can be built in each WLM environment at one time, but it is possible to have more than one WLM environment, each running its own version of DSNTPSMP. See Figure 55.

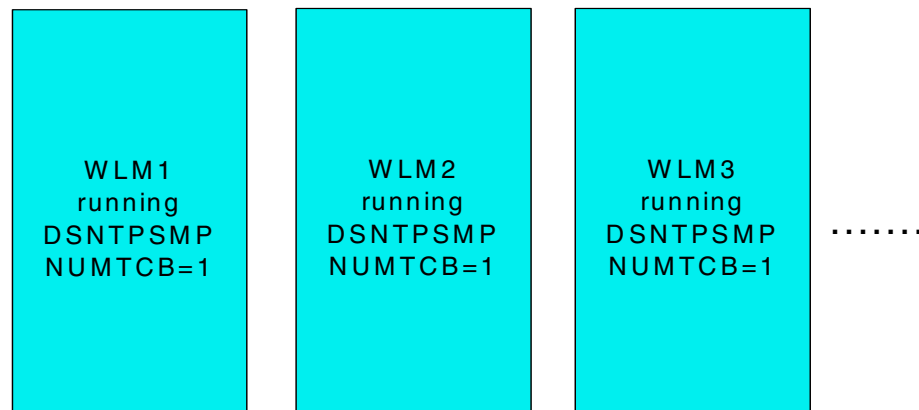


Figure 55. Multiple WLM environments

See 4.5, “Setting up DSNTPSMP” on page 120 for more details on how to set up the OS/390 Procedure Processor.

---

## 4.3 Coding considerations

In this section we document some of the issues that we found while creating SQL stored procedures on the OS/390 platform.

### 4.3.1 Length and size limits

The size of your SQL stored procedure code is limited to 32K. Note that if you are using DSNTPSMP, the tables SYSIBM.SYSPSM, SYSIBM.SYSPSMPOPTS, and SYSIBM.SYSPSMOUT will be updated. Although the field PROCCREATESTMT in SYSIBM.SYSPSM, which contains the source code, is limited to 3800 characters, DSNTPSMP will break up your source into 3800 character "chunks" and use the SEQNO field to sequence these lines to keep the order of your source code.

When creating your SQL stored procedure using the SPB, the width of your statements must be limited to 72 bytes. Although the edit screen in SPB is wider than this, the DB2 pre-compiler will only read a width of 72 bytes.

The name of the SQL stored procedure can be a maximum of 18 characters, but due to OS/390 limitations, the member name created at the DBRMLIB, LOADLIB, and other libraries is limited to 8 bytes. If you are creating your SQL stored procedure through the SPB or calling the OS/390 Procedure Processor directly, and your procedure name is longer than 8 characters, DSNTPMSP will generate an 8-character name for you, consisting of the characters "SP" followed by 6 random alphameric characters. We recommend that the first 8 characters be unique within one project, and that you limit your procedure name to 8 characters when building it, using the JCL batch job.

### 4.3.2 Parameters and variables

To store data used within an SQL stored procedure, you can declare SQL variables. SQL variables can have the same data types and lengths as SQL stored procedures parameters.

SQL stored procedures parameters and variables have the following restrictions:

- Since the precompiler folds all SQL variables to uppercase, two variables cannot be declared the same except for their case. For example, variables VAR1 and var1 declared in the same SQL stored procedure will receive a compile error.
- Variable and parameter names cannot be the same name. The following error message is issued:

```
DSNH590I Name <name> is not unique
```

- An SQL reserved word cannot be used as a parameter or variable.
- Do not precede the variable name with a colon.
- In version 5, parameters cannot contain underscores (in version 6 this restriction is removed).

Currently, this restriction is not picked up by the compiler; you will only see the error message when the START PROCEDURE command is issued. For example:

```
STC00044 DSNX904E . DSNX9CAT THE NAME DEPT_NUM IN THE PARMLIST
```

```

FOR PROCEDURE
GETMEDIANSALARY CONTAINS AN INVALID CHARACTER. REASON CODE IS 000.
STC00044 DSNX948I . DSNX9ST2 START PROCEDURE FAILED FOR *, DUE
TO PREVIOUSLY
REPORTED ERROR CONDITION

```

- In version 5, parameter names must be 8 bytes or less (in version 6 this restriction is removed).

Currently, this restriction is not picked up by the compiler; you will only see the error message when the START PROCEDURE command is issued. For example:

```

STC00044 DSNX903E . DSNX9CAT THE NAME MEDIANSALARY IN THE
PARMLIST COLUMN OF
SYSIBM.SYSPROCEDURES FOR PROCEDURE GETMEDIANSALARY IS TOO LONG.
STC00044 DSNX948I . DSNX9ST2 START PROCEDURE FAILED FOR *, DUE
TO PREVIOUSLY
REPORTED ERROR CONDITION

```

The following items should be qualified to avoid ambiguity and also prevent compilation or bind errors:

- When using a parameter in the procedure body, qualify the parameter name with the procedure name.
- Qualify variable names with the label of the compound statement in which the variables appear.
- Qualify column names with the table name. For example:

```

SELECT STAFF.ID INTO V_ID
FROM DB2RES1.STAFF WHERE STAFF.ID = V_ID;

```

- A parameter name can be the same as a column name, but the column name must be qualified with the table name.
- If you qualify a parameter with a misspelled procedure name, the following error message is issued:

```

DSNH312I Undefined or unusable host variable

```

This is also true of variables qualified with misspelled label names.

In version 5, although the maximum length of the parameter list (PARMLIST) column is defined as 3000 in SYSIBM.SYSPROCEDURES, you are limited to a maximum of 254 bytes, because when your SQL stored procedure is defined to DB2, it issues an INSERT statement containing a literal to update the SYSIBM.SYSPROCEDURES table. You will receive an SQLCODE -102 if the literal string to be inserted to PARMLIST is longer than 254. This limitation does not apply when using a host variable.

In version 6, the previous limitation does not apply, because a DDL CREATE PROCEDURE statement is executed to define your SQL stored procedure to DB2.

If you are planning to use the SPB to test your SQL stored procedures, we recommend that you prefix any table referenced in your SQL stored procedure code with the owner ID. Otherwise, the owner of the OS/390 Procedure Processor, which is a REXX stored procedure, will be assumed as the owner of the tables.

### 4.3.3 Handling SQLCODE and SQLSTATE values

The sample A.2.19, "SMP7LMS" on page 207 of Appendix A, "Sample SQL stored procedure programs" on page 195 shows how the SQLCODE and SQLSTATE values can be captured and handled in an SQL stored procedure. But it is important to be aware of the following warning when trying to capture BOTH the SQLCODE and SQLSTATE values:

**Important:**

Since a SET statement is an SQL statement, the SQLCODE and SQLSTATE are set to the values returned by the SELECT generated under-the-covers for the SET statement. So, in the example A.2.19, "SMP7LMS" on page 207, when you say SET PSQLST = SQLSTATE, it will set PSQLST to the SQLSTATE returned by the previously-executed statement, which is what you would expect. However, when you follow that with SET PSQLCO = SQLCODE, it will set PSQLCO to the SQLCODE returned by the SET PSQLST = SQLSTATE, which will be most certainly zero. You **will not** get the SQLCODE you expect, that is, from the statement executed just before the handler was invoked.

### 4.3.4 SQL statements

To define your SQL stored procedure to DB2 in version 5, an INSERT DML statement is made to SYSIBM.SYSPROCEDURES. This INSERT statement is created for you from the SQL Procedures pre-compiler in *ddname* SYSUT1. In version 6 this is kept as a CREATE PROCEDURE DDL statement and will be generated for you from the SQL Procedures precompiler in *ddname* SYSUT2.

If you are building your SQL stored procedure through JCL, you need to execute this generated statement through DSNTIAD to define your SQL stored procedure to DB2. If you are using the SPB or calling DSNTPSMP directly, this will be automatically executed for you.

Currently, in SQL Procedures for OS/390, you can have only one compound statement (see 2.4, "Current implementation of SQL Procedures language" on page 17). But you can overcome this limitation by coding all the compound statements within a single simple statement, for example in a LOOP or IF statement. The example below shows that the compound statements *test* and *test2* are contained within the *loop1* loop statement:

```
CREATE PROCEDURE spmd0211
    (INOUT ps1 SMALLINT, inout ps2 smallint,inout pc1 char(20))
    LANGUAGE SQL
    WLM ENVIRONMENT WLMENV1
    COLLID CLMD02
loop1: loop
    test: BEGIN
        DECLARE i1 INT;
        DECLARE p1 DOUBLE PRECISION;
        .....
        delete from tsttab1
            where num = test.n2;
        end if;
        set spmd0211.ps1 = spmd0211.ps1 + 10;      -- ps1 is 80
    end test;
test2: begin
```

```

        DECLARE s1 smallINT default 0;
        DECLARE c1, c2 CHAR (5);
        .....
        close cursor1;
        set spmd0211.ps2 = test2.s1;
        END test2;
    leave loop1;
end loop

```

This method can also be used within handlers which cannot currently support nested compound statements. For example, you can have a block of statements by using the IF statement as below:

```

DECLARE CONTINUE HANDLER FOR NOT FOUND
IF (1=1) THEN
    SET ENDTABLE = 1;
    SET OUTCODE = SQLCODE;
END IF;

```

### 4.3.5 Client application

No major change is needed in the way the client application is coded, or the way it calls a stored procedure written with the SQL Procedures programming language.

**Note:** Following the SQL/PSM standard, DB2 supports only the parameter style GENERAL WITH NULLS linkage convention for SQL stored procedures. This means that you should include a null indicator variable for each of the parameters that will be returned from the SQL stored procedure. Refer for *DB2 for OS/390 V5 Application Programming and SQL Guide, SC26-8958*, or *DB2 UDB for OS/390 V6 Application Programming and SQL Guide, SC26-9004*, for detailed information about linkage conventions.

---

## 4.4 Stored procedure preparation

There are three methods available for preparing a stored procedure written in SQL Procedures language as shown in Figure 56:

1. The Stored Procedure Builder (SPB) Tool, which invokes the OS/390 Procedure Processor, can be used to build your SQL stored procedures.
2. The OS/390 Procedure Processor, called DSNTPSMP, can be used to build your SQL stored procedures.
3. All steps required to build an SQL stored procedure can be run using JCL. You can use the DB2-supplied JCL procedure (DSNHSQL). The option to compile using the DB2 Interactive (DB2I) panels will be available from version 6.

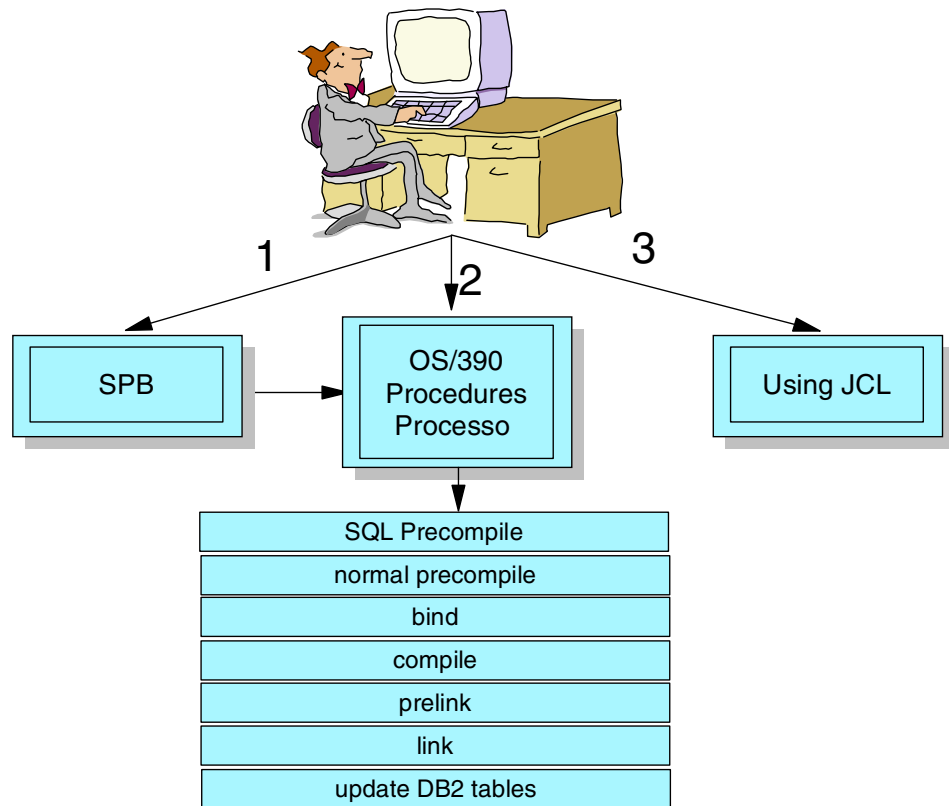


Figure 56. Three methods for preparing SQL stored procedures

#### 4.4.1 Process

The process involved in the creation of the SQL stored procedure is as follows:

1. The user creates the SQL stored procedures source (either manually or through prompted help via the SPB).
2. The source is then precompiled resulting in a C language program complete with SQL and logic.
3. The generated C program is then precompiled by the normal DB2 precompiler (DSNHPC) as per any other program using parameter of `HOST(C)`.
4. The modified C source is then compiled and linked.
5. The DBRM is bound.
6. The following tables might be updated when an SQL stored procedure is created successfully:
  - SYSIBM.SYSPROCEDURES (in version 5) or SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS (in version 6)
  - SYSIBM.SYSPSM
  - SYSIBM.SYSPSMOPTS
  - SYSIBM.SYSPSMOUT

**Note:** If you are using Method 1 or 2, a DML INSERT statement to populate the SYSIBM.SYSPROCEDURES (if against DB2 version 5), or a DDL CREATE PROCEDURE statement (against DB2 version 6), will be

automatically executed for you. But if you are using Method 3, you need to execute these statements to define the procedure to DB2.

#### 4.4.2 Authorization

The authorization needed to build SQL stored procedures in DB2 version 5 differs from that in version 6. In version 5, there is no explicit security for stored procedures, because they are not recognized as DB2 resources or objects, as they are in version 6.

The authorization used for DB2 when generating and binding the SQL stored procedure is the current SQLID or submitter of the job.

The extra privileges needed by the person requesting the build of the SQL stored procedure in version 5 and version 6 are:

- DB2 version 5 — SELECT, DELETE and INSERT from tables SYSIBM.SYSPROCEDURES, SYSIBM.SYSPSM, SYSIBM.SYSPSMOPTS, and SYSIBM.SYSPSMOUT.
- DB2 version 6 — CREATEIN, DROPIN or ALTERIN to the SCHEMA name under which you will be creating your SQL stored procedure.
- DB2 version 6 — (if using SPB or DSNTPSMP), you need EXECUTE ON PROCEDURE DSNTPSMP.
- Alternatively, you need SYSADM or SYSCTRL authority.

The normal privileges are needed to bind to the collection being used for your SQL stored procedure and any privileges to tables referenced in your SQL stored procedure.

Please also note that update access needs to be given on the datasets referenced in the WLM region, to the userid running the WLM.

---

### 4.5 Setting up DSNTPSMP

Since the OS/390 Procedure Processor itself is a stored procedure, it is associated with a particular WLM environment. The JCL to run this WLM environment is, therefore, set up with *ddnames* referencing a single set of application libraries.

The DSNTPSMP is written using REXX, and it was designed to attend customers in general. When you install the SQL Procedures support, you get the source code of the DSNTPSMP, which can be customized to a specific environment.

In addition, you might want the processor to be able to use more than one WLM environment, so that a different set of libraries can be referenced. You might do this if, for example, you are building SQL stored procedures for different projects, or if you want to set up different environments for test and production, or even if you just want to be able to test in a WLM environment without affecting or being affected by other people who may be testing.

To be able to set up your environment so that DSNTPSMP can be used in more than one WLM environment, you need to do the following:

- Define a new stored procedure based on the existing DSNTPSMP, but:



- Use a different procedure name.
- Use a different WLM environment.

Here is an example of defining a new OS/390 Procedure Processor called NEWTPSMP to use a WLM region called WLMENV2:

**(version 5)**

```
INSERT INTO SYSIBM.SYSPROCEDURES
VALUES ('NEWTPSMP',
       ' ',
       ' ',
       'DSNTPSMP',
       ' ',
       'DSNREXCS',
       'REXX',
       0,
       ' ',
       'N',
       'TRAP(OFF),MSGFILE(SYSPRINT)',
       'VARCHAR(20) IN, VARCHAR(18) IN, VARCHAR(32672) IN, VARCHAR(10
24) IN, VARCHAR(255) IN, VARCHAR(255) IN, VARCHAR(255) IN, VARCHAR(255)
IN, VARCHAR(254) IN, VARCHAR(80) IN, VARCHAR(8) IN, VARCHAR(18) IN, VARC
HAR(255) OUT',
       1,
       'WLMENV2', 'M', 'N', 'N');
```

**(version 6)**

```
CREATE PROCEDURE NEWTPSMP
(IN P1 VARCHAR(20),
 IN P2 VARCHAR(8),
 IN P3 VARCHAR(32672),
 IN P4 VARCHAR(255),
 IN P5 VARCHAR(255),
 IN P6 VARCHAR(255),
 IN P7 VARCHAR(255),
 IN P8 VARCHAR(255),
 IN P9 VARCHAR(254),
 IN P10 VARCHAR(80),
 OUT P11 VARCHAR(255))
EXTERNAL NAME DSNREXCS
PARAMETER STYLE GENERAL
WLM ENVIRONMENT WLMENV2
PROGRAM TYPE MAIN
LANGUAGE REXX
RESULT SETS 1
RUN OPTIONS 'TRAP(OFF),MSGFILE(SYSPRINT)';
```

- Note that the other definitions should be the same as what was originally defined for DSNTPSMP.
- Register the new WLM environment.
- Set up the JCL for the WLM started task JCL to reference the other set of libraries.

To make the SPB recognize this new procedure name for the DSNTPSMP, you have to change the **Build utility** field. This field is located from the "Advanced" button from the "Options" tab when building SQL Procedures for the OS/390 environment. It is also available from the Properties option when you click on the

procedure with the right mouse button. See Figure 57 for an example where we changed the Build utility to NEWTPSMP.

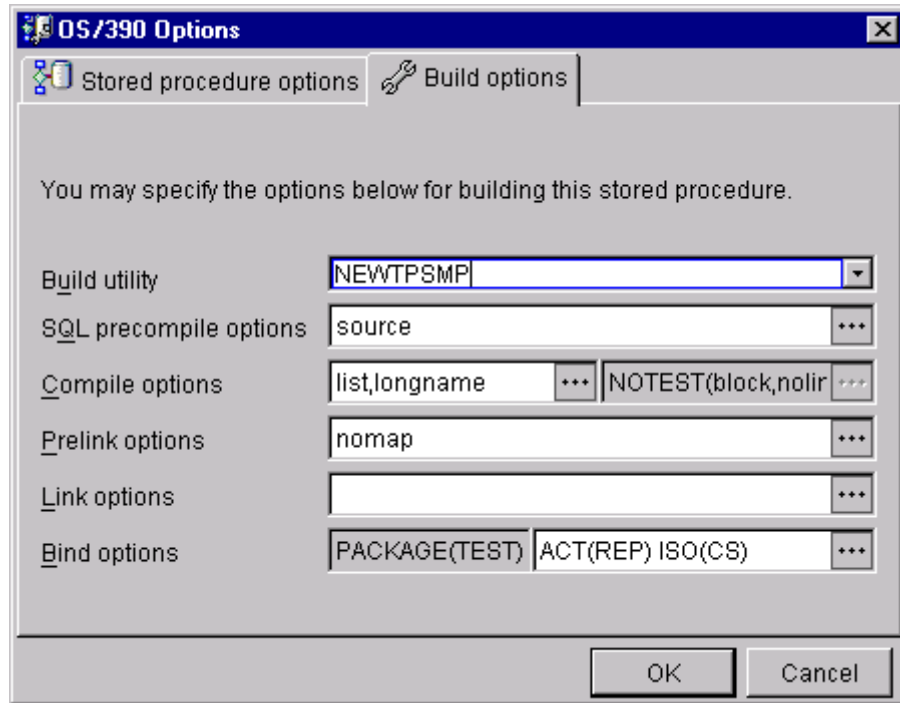


Figure 57. Build Name field on OS/390 Options

When you generate and build this new SQL stored procedure using NEWTPSMP, the dbrm, load, and source modules will be placed in the data sets referenced by the *ddnames* in the WLM environment associated with NEWTPSMP.

If you are building your SQL stored procedure by calling the procedure processor directly, it is just a simple matter of changing the called procedure name from DSNTPSMP to NEWTPSMP.

#### 4.5.1 Using the SPB

The easiest method of creating, generating, and testing your SQL stored procedure is to use the Stored Procedure Builder Tool. Only the OS/390 related options will be discussed here. See Chapter 3, “The DB2 Stored Procedure Builder” on page 57 for a general description of using the SPB to build your SQL stored procedure.

The OS/390 Options screen is available from the Advanced button on the Options tab when using the SmartGuide to create your SQL stored procedure. See Figure 58 and Figure 59. It is also available from the Properties option when you right-click on your procedure name.

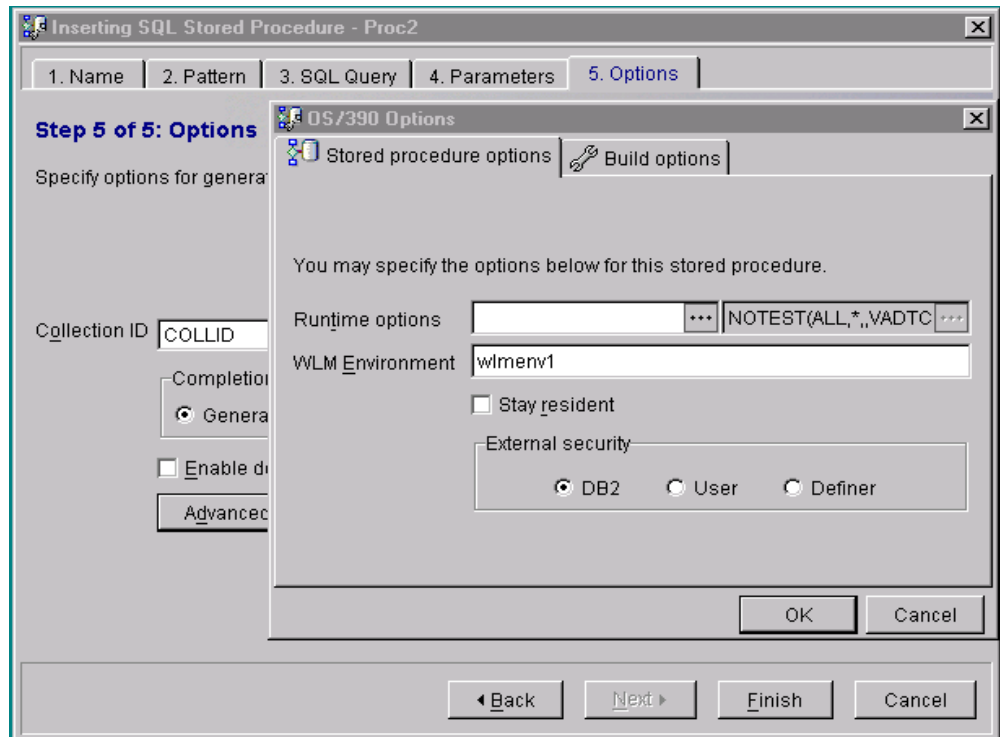


Figure 58. OS/390 Options from SPB — 1/2

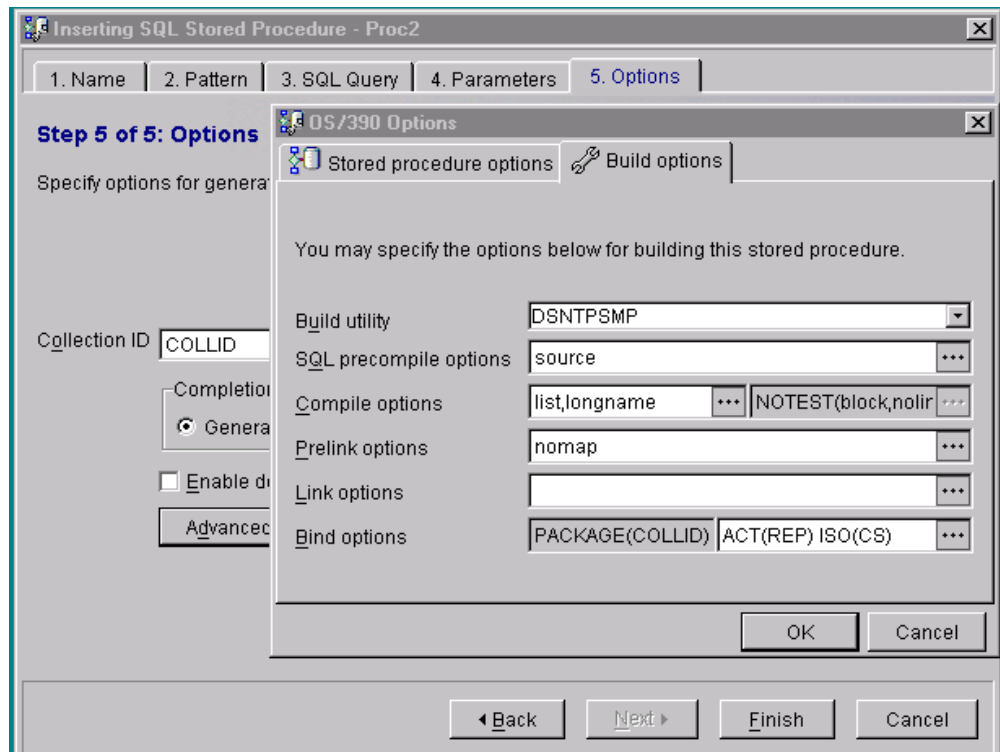


Figure 59. OS/390 Options from SPB — 2/2

You specify on this panel the options that will be used when SPB builds your procedure.

The Build utility name needs to be changed if you are running the OS/390 Procedure processor in multiple WLM environments. See 4.5, “Setting up DSNTPSMP” on page 120 for more details about this.

Once you have successfully generated and built your SQL stored procedure, you can then choose to test it by clicking the Run button from the main panel.

#### **4.5.1.1 SQL costing information**

The SPB tool allows you to measure the costing of the SQL statements in your SQL stored procedure. This functionality is only available for the OS/390 environment.

From the SPB tool, click on the Actual Costs button to get the costs of running the SQL stored procedure against the OS/390 environment. This button is located on the SQL Query tab when generating your SQL stored procedure using the SmartGuide. For this release of the SPB, the actual costs will only be calculated for the whole SQL stored procedure, but in future releases you will be able to get the cost of each SQL statement by highlighting it.

The SPB calls the stored procedure monitor program DSNWSPM which starts the monitor trace and formats the output. From the output you can then view and sort by different columns. Note that the first time this is called, the number of GETPAGEs is higher because the module needs to be loaded into the buffer.

#### ***Stored procedure monitor program***

The stored procedure monitor program DSNWSPM (which itself is an assembler stored procedure) returns CPU time and other DB2 costing information for the thread on which it is running. A client program can connect to DB2 on OS/390, execute SQL and then call DSNWSPM to find out how much CPU time it took.

DSNWSPM works on either a local or remote thread.

The instrumentation values that DSNWSPM provides are:

- CPU time in external format
- Latch/lock contention wait time, external format
- CPU time as an integer in hundredths of a second
- Latch/lock contention wait time
- Number of GETPAGEs in integer format
- Number of read I/Os in integer format
- Number of write I/Os in integer format

Figure 60 shows an output of the SQL Procedures monitor program. You can sort the information shown by clicking on the columns. Additionally, if you place your mouse pointer over any SQL statement, it will be expanded for you.

SQL statement	CPU time	Latch/lock wait time	Getpages	Read I/Os	Write I/Os
SQL SELECT * FROM Sysibm.sys...	00:00:00.040106	00:00:00.0	146	0	0
SQL SELECT * FROM Sysibm.sys...	00:00:00.04035	00:00:00.0	87	0	0

Delete Close

Figure 60. SQL Costing Information panel

### Setup and security

An accounting trace needs to be started on the host for this option to be successfully executed. The command to start the accounting trace is:

```
start trace(acctg) class(1,2,3)
```

The monitor trace also has to be on, but DSNWSPM starts that internally.

The authority you need to execute the stored procedure is:

- For version 5: MONITOR1 and TRACE
- For version 6: MONITOR1, TRACE, and EXECUTE ON PROCEDURE

For example, to allow USRT011 to execute DSNWSPM on version 6:

```
GRANT TRACE TO USRT011;
GRANT MONITOR1 TO USRT011;
GRANT EXECUTE ON PROCEDURE SYSPROC.DSNWSPM TO USRT011;
```

### Invoking the stored procedure monitor program (DSNWSPM)

You can call DSNWSPM in two ways:

1. Before and after executing the SQL
2. Only after executing the SQL

The preferred way to use DSNWSPM is the first method, because the time values are more accurate, as they are the delta values between beginning and ending time. If you call DSNWSPM using the second method, after executing the SQL only, the beginning CPU time returned to you will include the thread setup time, and may include the times for SQL that you have previously executed on the same thread.

## 4.5.2 Using OS/390 Procedure Processor (DSNTPSMP)

After coding the SQL stored procedure you can choose to prepare it using the OS/390 Procedure Processor (method 2). The processor automates and performs all the steps required to generate and build your SQL stored procedure.

The way to do this is to code a client program (coded in any language), which invokes the OS/390 Procedure Processor. The processor (DSNTPSMP) itself is a stored procedure which has input and output parameters. You pass to it the function that you wish performed on your SQL stored procedure source. The functions are: Build, Destroy, Rebuild, Rebind and Alter LE Run Options (see 4.5.2.4, "DSNTPSMP functions" on page 129).

Using this method, your client program will invoke DSNTPSMP and pass to it the parameters which it needs to completely build your stored procedure.

One of the reasons for coding a client program rather than using the SPB Tool is that the project administrator or DBA might wish to standardize the bind, link, or compile options within a project. One way of doing this is to only allow the programmer to pass a few parameters, for example: `function`, `SQL stored procedure name`, `SQL stored procedure source`. The other options can then be hardcoded in your client program and therefore "hidden" from the SQL stored procedure developer. See 4.5.2.5, "Example of a client program" on page 131 for a sample client program written in PL/1, which does this.

Figure 61 shows the input and output to DSNTPSMP. The WLM environment defined to be used by DSNTPSMP must be available before you call DSNTPSMP.

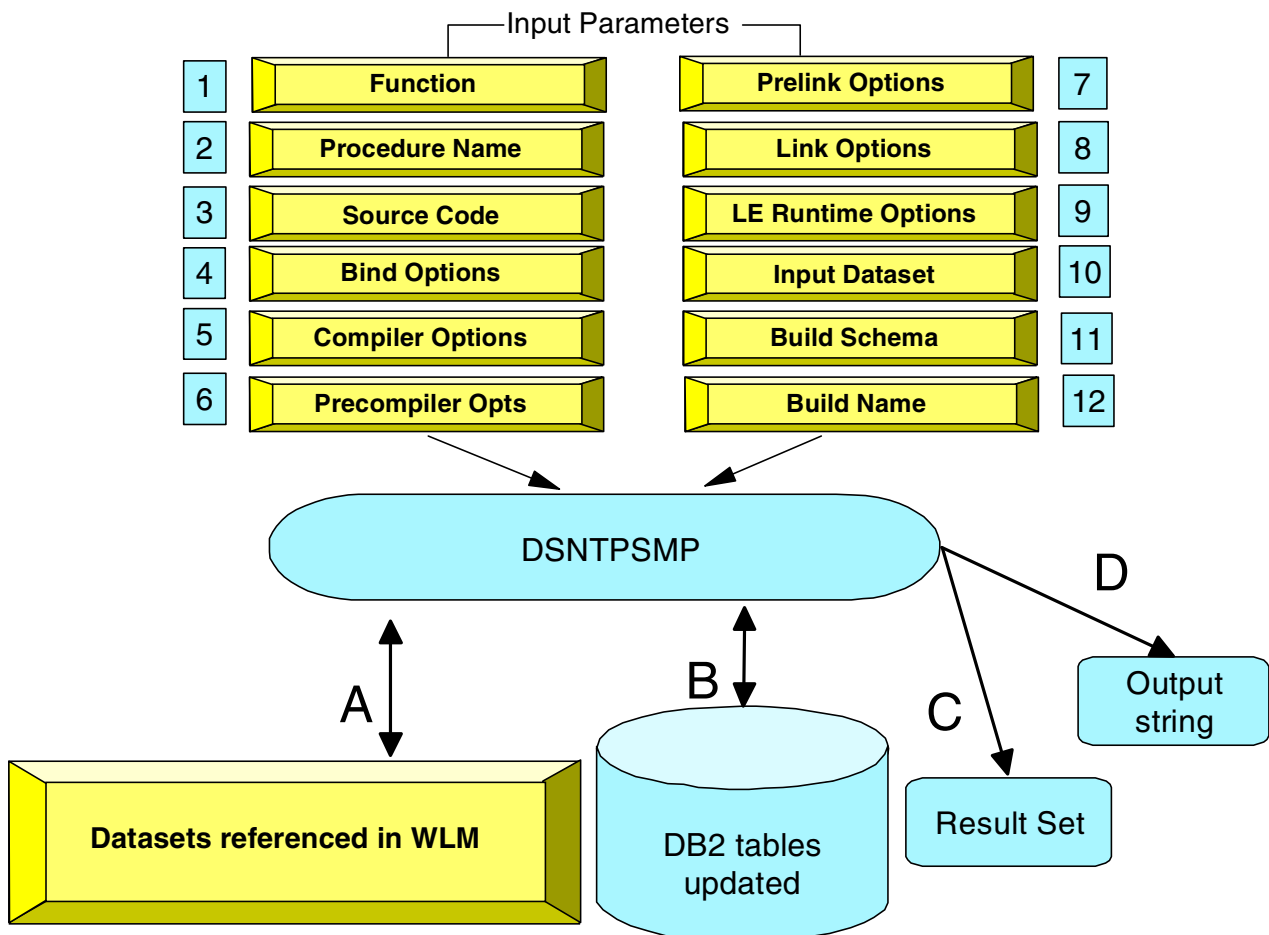


Figure 61. Input / Output for SQL Procedures Processor

#### 4.5.2.1 Input parameters for DSNTPSMP

The parameters passed to DSNTPSMP must be in this order:

1. **Function:** The action that you want DSNTPSMP to perform on your SQL stored procedure. See 4.5.2.4, "DSNTPSMP functions" on page 129 for a description of each function.

2. **Procedure Name:** The name of your SQL stored procedure (maximum 18 characters). This needs to be the same as the name you specify in your CREATE PROCEDURE statement. In version 6, the procedure name can be up to 27 characters (8-byte schema, 1-byte dot, 18-byte routine name).
3. **Source Code:** The SQL stored procedure source code. This is not needed if you pass the source via an Input File data set. But if both parameters are passed, source will be taken from this parameter in preference to source from the data set. In version 6, source code passed in as a parameter can be 32K. If using a dataset for the input, there is no limit.
4. **Bind Options:** The options with which you wish to bind your SQL stored procedure package. The maximum length is 1024 characters.
5. **Compiler Options:** The options which you want to pass for use in compiling your SQL stored procedure module, for example, TEST/NOTEST. The maximum length is 255 characters.
6. **Precompiler Options:** The options to be used during pre-compilation of your SQL stored procedure. Maximum length is 255 characters.
7. **Prelink Options:** The options to be used to pre-link your module. Maximum length is 255 characters.
8. **Link Options:** The options passed to the linkage editor. Maximum length is 255 characters.
9. **LE Runtime Options:** The options to be passed to the Language Environment for execution of your module. Input passed here will be ignored for other functions. The maximum length is 254 characters.
10. **Input Dataset:** The data set containing the SQL stored procedure source code. If you pass this parameter, then you do not need to pass the source in the Source Code parameter. But if both parameters are passed, the source in the Source Code parameter is used in preference.
11. **Build Schema:** This contains the schema name for the procedure name you pass in the Build Name (parameter 12). Usually this is SYSPROC for version 5, and could be any name in version 6.
12. **Build Name:** A name for the OS/390 Procedures Processor, usually DSNTPSMP. The maximum length is 18 characters.

#### 4.5.2.2 DDNAMES used by DSNTPSMP on WLM

The *DDNAMES* for the permanent datasets referenced on the WLM region that are used by DSNTPSMP are described here. They are used for both input and output:

- SQLDBRM — Your SQL stored procedure DBRM module
- SQLCSRC — Your SQL stored procedure precompiled C source code generated by DSNTPSMP
- SQLLMOD — SQL stored procedure generated load module
- SQLLIBC — Language C header files
- SQLLIBL — Link libraries, run libraries, DB2 load libraries
- SYSMSGSGS — SCEEMSG "C" messages library for prelinker

Other *ddnames* are also used, but permanent datasets do not need to be allocated to them.

Following is part of the JCL used in our WLM environment, which was set up for executing DSNTPSMP, showing the relevant *ddnames* :

```
//*****
//**** Data sets required by the SQL Procedures Processor ****
//*****
//SQLDBRM DD DISP=SHR,                <== DBRM Library
//          DSN=DSN.DBRMLIB.DATA
//SQLCSRC DD DISP=SHR,                <== Generated C Source
//          DSN=DSN.SRCLIB.DATA
//SQLMOD  DD DISP=SHR,                <== Application Loadlib
//          DSN=DSN.RUNLIB.LOAD
//SQLLIBC DD DISP=SHR,                <== C header files
//          DSN=CEE.SCEEH.H
//          DD DISP=SHR,
//          DSN=CEE.SCEEH.SYS.H
//SQLLIBL DD DISP=SHR,                <== Linkedit includes
//          DSN=CEE.SCEELKED
//          DD DISP=SHR,
//          DSN=DSN.SDSNLOAD
//SYMSGS  DD DISP=SHR,                <== Prelinker msg file
//          DSN=CEE.SCEEMSGP(EDCPMSGE)
//*****
//**** Workfiles required by the SQL Procedures Processor ****
//*****
//SQLSRC  DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=FB,LRECL=80)
//SQLPRINT DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=VB,LRECL=137)
//SQLTERM DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=VB,LRECL=137)
//SQLOUT  DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=VB,LRECL=137)
//SQLCPRT DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=VB,LRECL=137)
//SQLUT1  DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=FB,LRECL=80)
//SQLUT2  DD UNIT=SYSDA,SPACE=(16000,(20,20)),
//          DCB=(RECFM=FB,LRECL=80)
//SQLCIN  DD UNIT=SYSDA,SPACE=(32000,(20,20))
//SQLLIN  DD UNIT=SYSDA,SPACE=(8000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80)
//SQLWORK1 DD UNIT=SYSDA,SPACE=(16000,(20,20)),    <= Work C source
//          DCB=(RECFM=FB,LRECL=80)
//SQLWORK2 DD UNIT=SYSDA,SPACE=(16000,(20,20)),    <= Work LOADMOD
//          DCB=(RECFM=U)
//SYSMOD  DD UNIT=SYSDA,SPACE=(16000,(20,20)),    <= PRELINKER
//          DCB=(RECFM=FB,LRECL=80)
```

#### 4.5.2.3 Output parameters of DSNTPSMP

Depending on whether your SQL stored procedure was successfully generated, the output from your call to DSNTPSMP will be as follows:

- If your SQL stored procedure was successfully generated, the DBRM and load module members are placed into the data sets referenced by the *ddnames* on the WLM started task JCL. See "A" in Figure 61 on page 126.



- If your SQL stored procedure was successfully generated, the caller of DSNTPSMP (which might be SPB or your own client) must commit the changes made to the DB2 tables. See "B" in Figure 61 on page 126. If there are any problems found by DSNTPSMP, and rows are returned in the result set, the caller must issue a ROLLBACK.
- When an unrecoverable error was received, DSNTPSMP will store all diagnostic information into a temporary table and return these as a result set to the calling application (SPB Tool or your own) indicating the failing step and a return code. See "C" in Figure 61 on page 126. You need to code the normal result set processing to process the messages received. For example:
  - DESCRIBE PROCEDURE
  - ASSOCIATE LOCATORS
  - ALLOCATE CURSOR
  - DESCRIBE CURSOR
 See 4.5.2.5, "Example of a client program" on page 131 for a sample written in PL/1.

- Output received via the Output String (outstring) parameter. This contains the results of your call to DSNTPSMP. You should pass an output parameter (empty variable) of length 255 when you invoke DSNTPSMP. The outstring parameter returns with zero condition code or error messages (see "D" in Figure 61 on page 126). For the time being, the outstring returned is only an integer such as 0, 4, 8, and 999. No text is returned. This may be enhanced in the future. Below, the text after the = sign is an explanation or programmer action:

```

outstring:  0 = successful operation
            4 = successful operation with warnings, look at result set
            8 or higher = failure, look at result set
            999 = severe internal error look at result set
  
```

**Note:** If you are using DSNTPSMP directly, the input source string should be broken up into lines of code <=80 , with the separating character being an EBCDIC newline ('15'x) or linefeed ('25'x).

#### 4.5.2.4 DSNTPSMP functions

Following are the functions performed by the OS/390 Procedure Processor:

- **BUILD**

This function goes through all the necessary steps for preparing an SQL stored procedure. The build process will terminate if the build is requested for an SQL stored procedure which already exists:

```
MEDSALRY already exists in SQLDBRM.
```

Figure 62 shows the steps executed in a BUILD process. Given the SQL stored procedure source as input:

1. DSNTPSMP defines the SQL stored procedure to DB2 by updating the DB2 tables SYSIBM.SYSPROCEDURES (v5) or SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS (v6). The unmodified SQL stored procedure source is stored in the SYSIBM.SYSPSM table, and the process options are stored in SYSIBM.SYSPSMOPTS.

2. For DB2 version 5, the SQL Procedures pre-compiler is invoked to translate the source to a C host language stored procedure with embedded SQL.
3. A "normal" precompile is then done to produce a DBRM and a modified source.
4. A bind is done to create a stored procedure package.
5. The modified source is compiled with the C compiler, pre-linked and linked to produce an executable load module.

If BUILD abends halfway through this process, the caller (SPB or own client) must issue a ROLLBACK to back out of the updates made to the DB2 tables.

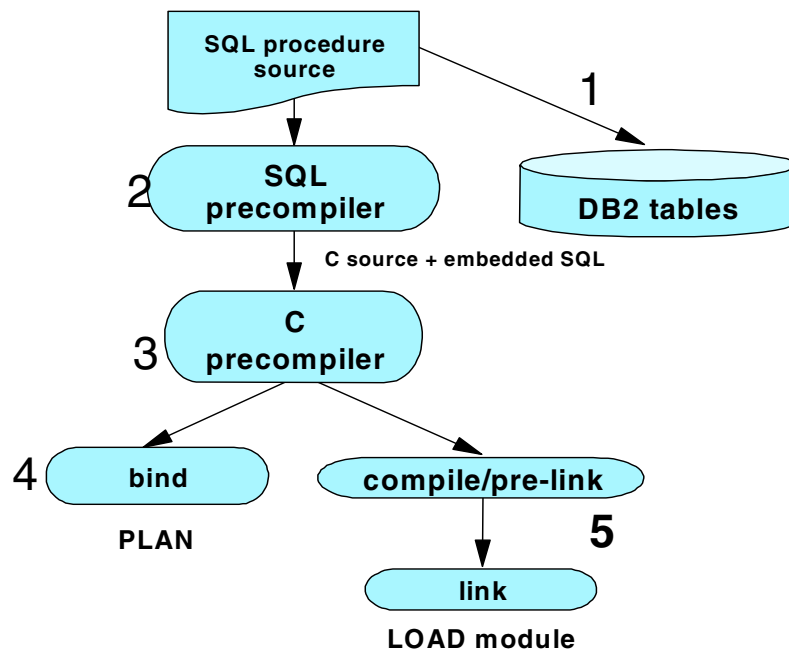


Figure 62. The BUILD process

Below is an example of a call to DSNTPSMP to perform a BUILD:

```

EXEC SQL
CALL DSNTPSMP( :func, :proc-name, :PSM-source, :bnd-opt,
:comp-opt, :pcomp-opt, :plnk-opt, :link-opt, :lert-opt,
:in-dsname, :build-schema, :build-name, :out-var )
END-EXEC.

```

#### • DESTROY

This function cleans up the definitions in the catalog and the members in the libraries which are associated with the SQL stored procedure.

Figure 63 on page 131 shows the actions taken for the destroy process.

1. First the definition of the stored procedure is deleted from SYSIBM.SYSPROCEDURES (v5) or dropped from SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS (v6).
2. The SQL stored procedure source is then deleted from SYSIBM.SYSPSM and SYSIBM.SYSPSMOPTS.
3. A drop package is done for the stored procedure package.

- The load module, DBRM, and C source are deleted from the relevant libraries referenced in the *ddnames* on the WLM JCL.

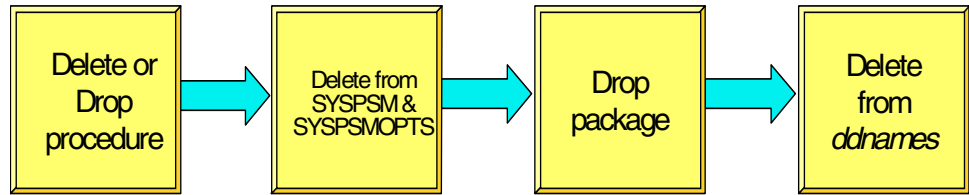


Figure 63. The DESTROY process

The parameters that need to be passed to DSNTPSMP are the function and the procedure name. The other parameters can be empty (nulls).

```
CALL DSNTPSMP( :func, :proc-name, , , , , , , )
```

- **REBUILD**

Performs the same functions as BUILD but will overwrite any DB2 rows and library members if necessary. It is generally performed against an existing SQL stored procedure but is also allowed for new non-existent routines. The parameters passed are the same as for Build function.

- **REBIND**

This function will be used against an existing SQL stored procedure to change bind options and rebind the stored procedure package.

#### 4.5.2.5 Example of a client program

Below is a sample of a PL/1 program which calls DSNTPSMP to perform a build of an SQL stored procedure.

It can be coded generically enough so that only three parameters need to be passed: function, procedure name, and the SQL stored procedure source data set.

You would only need to compile it once. Using this program, the DBA or project administrator can control the compile, link, and bind options being used to create SQL stored procedures for their project.

#### PL/1 sample client program

```

//*****
//PSMBUILD JOB 'USER=JONATHAN', 'JONATHAN', CLASS=K,
//          MSGCLASS=H,MSGLEVEL=(1,1)
//*****
/*  COMPILE/LINK THE CALLING APPLICATION
//*****
//STEP PROC EXEC PROC=DSNHPLIA,DB2LEV=DB2A,MEM=STUBPGM
//PC.SYSIN DD *
STUBPGM: PROCEDURE(PARMS) OPTIONS(MAIN);

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

//*****
/* INPUT PARAMETERS
//*****
  
```

```

DECLARE PARMS                CHAR(100) VARYING ;

DECLARE WHATTODO             CHAR(8) VARYING;
DECLARE SPNAME               CHAR(8) VARYING;
DECLARE SRCDSN               CHAR(80) VARYING;

DECLARE space1               BIN FIXED(15);
DECLARE space2               BIN FIXED(15);
DECLARE parm_len             BIN FIXED(15);

DECLARE action                CHAR(20)  VARYING;
DECLARE member_name          CHAR(18)   VARYING;
DECLARE psm_program          CHAR(32672) VARYING;
DECLARE bind_opts            CHAR(1024) VARYING;
DECLARE comp_opts            CHAR(255)  VARYING;
DECLARE precomp_opts        CHAR(255)  VARYING;
DECLARE prelink_opts        CHAR(255)  VARYING;
DECLARE link_opts            CHAR(255)  VARYING;
DECLARE le_opts              CHAR(254)  VARYING;
DECLARE input_dsn            CHAR(80)   VARYING;
DECLARE buildschema          CHAR(8)    VARYING;
DECLARE buildname            CHAR(18)   VARYING;
DECLARE out_string           CHAR(255)  VARYING;

DECLARE SPIRS    SQL TYPE IS RESULT_SET_LOCATOR VARYING;
DECLARE STEP     CHAR(16);
DECLARE FILE     CHAR(8);
DECLARE SEQN     BIN FIXED(31);
DECLARE LINE     CHAR(255);

action          = '';
member_name     = '';
psm_program     = '';
bind_opts       = '';
comp_opts       = '';
precomp_opts   = '';
prelink_opts   = '';
link_opts       = '';
le_opts         = '';
input_dsn       = '';
buildschema     = '';
buildname       = '';
out_string      = '';

buildschema='SYSPROC'; /* Schema name of builder */
buildname='DSNTPSMP'; /* Builder name */

/*****/
/* parse the input parameter to get - */
/*      function */
/*      SQL Procedure module name */
/*      SQL Procedure source dataset */
/*****/
parm_len = length(parms);
space1 = index(parms, ' ');
space2 = index(substr(parms, space1+1, parm_len-space1), ' ');

```

```

WHATTODO = SUBSTR (PARMS,1,space1-1);
SPNAME   = SUBSTR (PARMS,space1+1,space2-1);
SRCDSN   = SUBSTR (PARMS,space1+space2+1,param_len-space1-space2);

PUT SKIP LIST('#####');
PUT SKIP LIST('Function=',WHATTODO) ;
PUT SKIP LIST('Programe=',SPNAME) ;
PUT SKIP LIST('Dataset =',SRCDSN);
PUT SKIP LIST('#####');
CLOSE FILE(SYSPRINT) ;

SELECT (WHATTODO);
    WHEN('BUILDT') CALL program1; /* BUILD for debug */
    WHEN('BUILD') CALL program2; /* BUILD normally */
    WHEN('DESTROY') CALL program3; /* DESTROY */
    WHEN('REBUILD') CALL program4; /* REBUILD */
    WHEN('REBIND') CALL program6; /* REBIND */
    OTHERWISE
    DO;
        PUT SKIP LIST('** NO ACTION SPECIFIED, EXITING **');
        EXIT;
    END;
END;
/*****/

EXEC SQL
CALL :BUILDNAME ( :action,
                 :member_name,
                 :psm_program,
                 :bind_opts,
                 :comp_opts,
                 :precomp_opts,
                 :prelink_opts,
                 :link_opts,
                 :le_opts,
                 :input_dsn,
                 :buildschema,
                 :buildname,
                 :out_string)
;

IF SQLCODE<=0 THEN
    call Print_SQLCA;
PUT SKIP LIST ('*** SQLCODE from call == ',sqlcode);

PUT SKIP LIST('** STUBPGM Call return from REXX stored proc **');
PUT SKIP LIST ('REXX output parm follows....');
PUT SKIP EDIT('Returned outstring =', out_string) (A,A(100));

/* Will get result set only if errors received */
PUT SKIP LIST ('Doing associate locators....');
EXEC SQL
ASSOCIATE LOCATORS (:SP1RS) WITH PROCEDURE DSNTPSMP;
IF (SQLCODE <=0) & (SQLCODE <= -482)THEN
    call Print_SQLCA;

```

```

    PUT SKIP LIST ('Doing allocate cursors...');
    EXEC SQL
    ALLOCATE C1 CURSOR FOR RESULT SET :SP1RS;
    IF (SQLCODE=-0) & (SQLCODE -=- 423) THEN
        call Print_SQLCA;
    PUT SKIP LIST ('** Result set follows **');
    do while (sqlcode=0);
    EXEC SQL FETCH C1 INTO :STEP, :FILE, :SEQN, :LINE ;
        PUT SKIP DATA (LINE);
    end;
    PUT SKIP LIST ('** End of result set **');

program1: procedure;

    comp_opts='LIST,TEST,LONGNAME';
    precomp_opts='SOURCE';
    prelink_opts='NOMAP,DEBUG (SHOWIO)';
    link_opts='AMODE=31,MAP,LIST=ALL';
    le_opts='TRAP (OFF) ,RPTOPTS (ON)';
    input_dsn=SRCDSN;
    action='BUILD';
    psm_program = '';
    bind_opts='PACKAGE (COLLID) ACT (REP) ISO (CS)';
    member_name=SPNAME;

end;

program2: procedure;

    comp_opts='LIST';
    precomp_opts='SOURCE';
    prelink_opts='NOMAP';
    link_opts='AMODE=31,MAP';
    le_opts='TRAP (OFF) ,RPTOPTS (ON)';
    input_dsn=SRCDSN;
    action='BUILD';
    bind_opts='PACKAGE (COLLID) ACT (REP) ISO (CS)';
    member_name=SPNAME;

end;

program3: procedure;

    action='DESTROY';
    member_name=SPNAME;

end;

program4: procedure;

    call program1;
    action='REBUILD';
end;

```

```

program6: procedure;

    action='REBIND';
    comp_opts='LIST,TEST,LONGNAME';
    precomp_opts='SOURCE';
    prelink_opts='NOMAP,DEBUG(SHOWIO)';
    link_opts='AMODE=31,MAP,LIST=ALL';
    le_opts='TRAP(OFF),RPTOPTS(ON)';
    input_dsn=SRCDSN;
    psm_program = '';
    bind_opts='PACKAGE(COLLID)';
    member_name=SPNAME;

end;

Print_SQLCA: procedure;
    PUT SKIP EDIT('DUMP SQLCA BY FIELDS') (A);
    PUT SKIP EDIT('SQLCAID=',SQLCAID) (A,A(8));
    PUT SKIP EDIT('SQLCABC=',SQLCABC) (A,F(11));
    PUT SKIP EDIT('SQLCODE=',SQLCODE) (A,F(11));
    PUT SKIP EDIT('SQLERRM=',SQLERRM) (A,A(LENGTH(SQLERRM)));
    PUT SKIP EDIT('SQLERRP=',SQLERRP) (A,A(8));
    PUT SKIP EDIT('SQLERRD(1)=' ,SQLERRD(1)) (A,F(11));
    PUT SKIP EDIT('SQLERRD(2)=' ,SQLERRD(2)) (A,F(11));
    PUT SKIP EDIT('SQLERRD(3)=' ,SQLERRD(3)) (A,F(11));
    PUT SKIP EDIT('SQLERRD(4)=' ,SQLERRD(4)) (A,F(11));
    PUT SKIP EDIT('SQLERRD(5)=' ,SQLERRD(5)) (A,F(11));
    PUT SKIP EDIT('SQLERRD(6)=' ,SQLERRD(6)) (A,F(11));
    PUT SKIP EDIT('SQLWARN0=' ,SQLWARN0) (A,A(1));
    PUT SKIP EDIT('SQLWARN1=' ,SQLWARN1) (A,A(1));
    PUT SKIP EDIT('SQLWARN2=' ,SQLWARN2) (A,A(1));
    PUT SKIP EDIT('SQLWARN3=' ,SQLWARN3) (A,A(1));
    PUT SKIP EDIT('SQLWARN4=' ,SQLWARN4) (A,A(1));
    PUT SKIP EDIT('SQLWARN5=' ,SQLWARN5) (A,A(1));
    PUT SKIP EDIT('SQLWARN6=' ,SQLWARN6) (A,A(1));
    PUT SKIP EDIT('SQLWARN7=' ,SQLWARN7) (A,A(1));
    PUT SKIP EDIT('SQLWARN8=' ,SQLWARN8) (A,A(1));
    PUT SKIP EDIT('SQLWARN9=' ,SQLWARN9) (A,A(1));
    PUT SKIP EDIT('SQLWARNA=' ,SQLWARNA) (A,A(1));
    PUT SKIP EDIT('SQLSTATE=' ,SQLSTATE) (A,A(5));
end;

END STUBPGM;
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNELI)
INCLUDE SYSLIB(DSNTIAR)
NAME STUBPGM(R)
//*****
//* BIND THE CALLING APPLICATION
//*****
//STEPBND EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN DD *
DSN SYSTEM(V51A)

FREE PACKAGE(COLLID.STUBPGM)
FREE PLAN(SPMAINT)

```

```

BIND PACKAGE(COLLID) MEMBER(STUBPGM)
BIND PLAN(SPMAINT) PKLIST(DSNREXCS.DSNREXX, COLLID.STUBPGM)

/*****
/* INVOKE THE CALLER
/*****
//STEPRUN EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN DD *
DSN SYSTEM(V51A)

RUN PROGRAM(STUBPGM) -
PLAN(SPMAINT) -
PARMS('/BUILD MEDIANSALARY SG245485.SAMPLES.SOURCE(MEDSAL)')
/*
/* PARM('/DESTROY MYTEST2 DUMMY')
/* PARM('/BUILD LT1PSM1 SG245485.SAMPLES.SOURCE(LT1PSM1)')
/* PARM('/REBUILD LT1PSM1 SG245485.SAMPLES.SOURCE(LT1PSM1)')

```

Once the calling program is compiled, the SQL stored procedure developer only needs to execute **STEPRUN** when they want to generate their own SQL stored procedure.

Below is the output received from the program above for a successful REBUILD of an SQL stored procedure:

```

#####
Function=                REBUILD
Programe=                MEDSAL
Dataset =                SG245485.SAMPLES.SOURCE(MEDSAL)
#####
*** SQLCODE from call ==                0
** STUBPGM Call return from REXX stored proc **
REXX output parm follows....
Returned outstring =0
Doing associate locators...
Doing allocate cursors...
** Result set follows **
** End of result set **

```

Now that your SQL stored procedure is built, you can test it by building the client application to call it, or testing it through the SPB Tool.

**Note:** DB2 delivers a sample C-language caller of DSNTPSMP called DSN8ED5. The sample job DSNTJ65 shows how to prepare and run DSN8ED4 to call DSNTPSMP to prepare a sample SQL stored procedure called DSN8ES2. DSNTJ65 also prepares and invokes a sample C-language caller of DSN8ES2 called DSN8ED5.

### 4.5.3 Using JCL

The SQL stored procedure can also be prepared by invoking the DB2 supplied JCL procedure DSNHSQL (4.5.3.2, "JCL for DSNHSQL procedure" on page 140).

See 4.5.3.1, "JCL for building an SQL stored procedure" on page 139 for a version 5 example of the steps mentioned here:



- DELCATG — Executes DSNTEP2. If definitions already exist for the stored procedure, that is, you have already built it, this step deletes it from SYSIBM.SYSPROCEDURES. If not, you will not need to execute this step.  
In version 6, a DROP PROCEDURE is executed instead of the delete. This removes the definition of your SQL stored procedure from SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS. An example of this statement follows:

```
DROP PROCEDURE SMP1LMS RESTRICT;
```

- PROCESS — Calls the DSNHSQL procedure which executes the following steps:

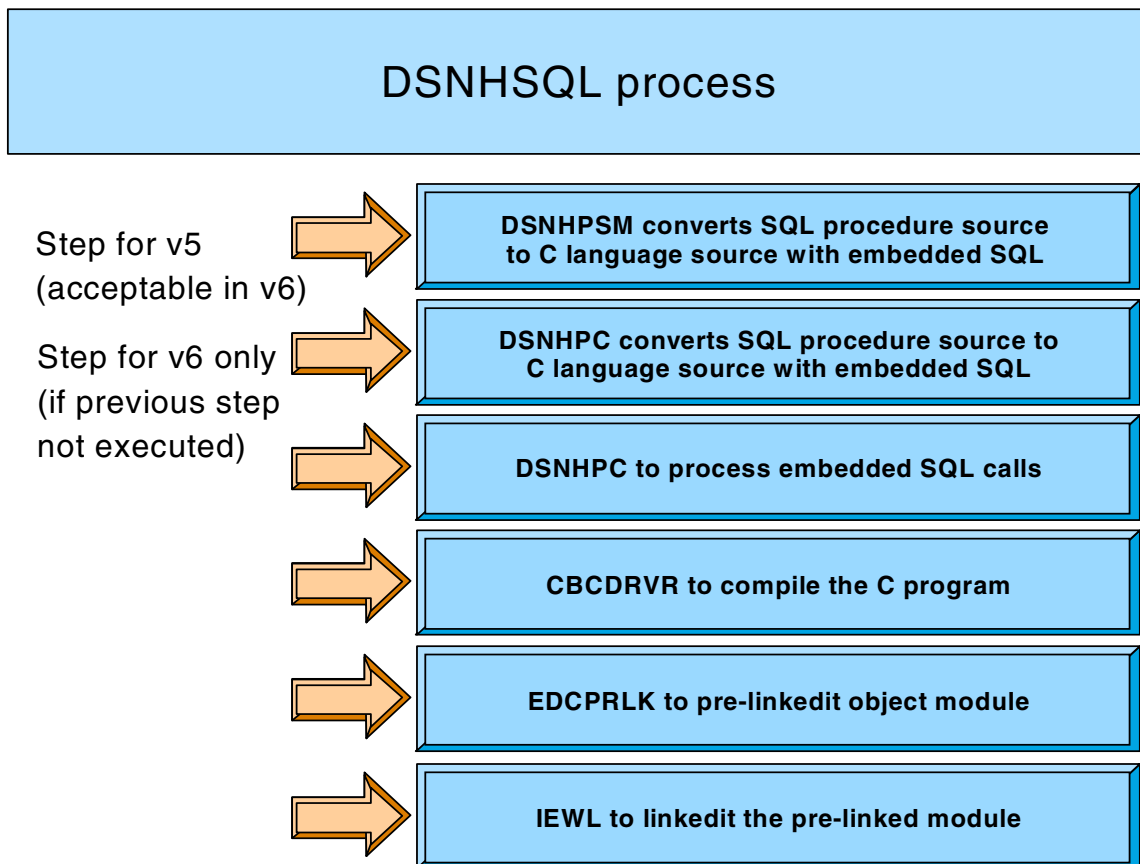


Figure 64. DSNHSQL process

1. For version 5, DSNHPSM is executed to convert your SQL stored procedure source module into an equivalent C language program which contains the embedded SQL calls. In version 6, normally the precompiler DSNHPC is executed twice instead. The parameter used when calling DSNHPC for the first run is `PARM='HOST(SQL)'`.

**Note:** In version 6 for compatibility purposes, it will still be possible to run DSNHPSM instead of DSNHPC with `PARM='HOST(SQL)'`

2. DSNHPC is called with `PARM='HOST(C)'` to precompile the C source generated by the previous step to process the embedded SQL calls.
3. C compiler CBCDRVR is called to perform a normal C compile.
4. EDCPRLK is called to perform a normal pre-linkedit.

5. IEWL is called to perform a normal linkedit.
- ISRTCATG — Executes DSNTIAD to register the definition for your SQL stored procedure to DB2. The statements generated are from the step which executes the DSNHPSM precompile.

For DB2 version 5, a DML INSERT statement is generated to update the SYSIBM.SYSPROCEDURES table. For example:

```
INSERT INTO SYSIBM.SYSPROCEDURES
  (PROCEDURE, AUTHID, LUNAME, LOADMOD, LINKAGE, COLLID,
   LANGUAGE, ASUTIME, STAYRESIDENT, IBMREQD,
   RUNOPTS, PARMLIST, RESULT_SETS, WLM_ENV,
   PGM_TYPE, EXTERNAL_SECURITY, COMMIT_ON_RETURN)
VALUES (
  'MEDSALRY', '', '', 'MEDSALRY', 'N', '',
  'SQL', 0, ' ', 'N',
  '',
  'DEPTNUM SMALLINT IN, MEDSALY SMALLINT OUT',
  0, 'WLMENV1', 'M', 'N', 'N')
```

For DB2 version 6, a DDL CREATE PROCEDURE statement is generated to define the procedure to SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS. For example:

```
CREATE PROCEDURE SMPOLMS6
  ( IN EMPLOYEE_NO CHAR ( 6 ) ,
  OUT EMP_FIRSTNAME VARCHAR ( 12 ) ,
  OUT EMP_LASTNAME VARCHAR ( 15 ) ,
  OUT SQLCPARM INTEGER )
FENCED RESULT SET 0 LANGUAGE SQL
DETERMINISTIC MODIFIES SQL DATA NO DBINFO COLLID SG245485
WLM ENVIRONMENT WLMENV1 ASUTIME NO LIMIT STAY RESIDENT NO
PROGRAM TYPE MAIN SECURITY DEFINER COMMIT ON RETURN NO
```

One row which defines your procedure is created in SYSIBM.SYSROUTINES.

The SYSIBM.SYSPARMS catalog table will contain one row for each input or output parameter required by your SQL stored procedure. In the example above, four rows will be added.

- BINDSP — Binds the DBRM for your SQL stored procedure.
 

**Note:** If you did not specify the collection id in the CREATE PROCEDURE statement, DB2 will not execute your SQL stored procedure even if you subsequently bind the DBRM to a specific collection id in this step. If you wish to use the collection id of the client, specify NO COLLID. Later when you bind the client package, you can then bind the SQL stored procedure package to the same collection id and your SQL stored procedure will be executed successfully.
- SELCATG — Executes DSNTEP2 to select from SYSIBM.SYSPROCEDURES to view the inserted values for the new procedure.

**Note:** DB2 delivers a sample job called DSNTEJ63 that demonstrates how to use DSNHSQL to prepare a sample SQL stored procedure called DSN8ES1. The sample job DSNTEJ64 prepares and calls a sample caller of DSN8ES1 called DSN8ED3.

### 4.5.3.1 JCL for building an SQL stored procedure

This is the JCL used in DB2 version 5 to manually create stored procedures using the SQL Procedures language. The SQL stored procedures source is input in ddname **PC.SYSIN**.

```
//BUILDER JOB 'USER=KATHLEEN', 'KATHLEEN', CLASS=A,
//          MSGCLASS=H, REGION=4096K
//*-----
//* BUILDING SOURCE -> BASECASE
//*-----
//JOBLIB DD DSN=CEE.SCEERUN, DISP=SHR <- IBM LE RUNTIME
//          DD DSN=DB2A.DSNEXIT, DISP=SHR <- DB2 USER EXITS
//          DD DSN=DB2A.DSNLOAD, DISP=SHR <- DB2 LOAD MODS
//*****
//* DELETE FROM CATALOG THIS PROCEDURE
//*****
//DELCATG EXEC TSOBATCH, DB2LEV=DB2A
//SYSTSIN DD *
DSN SYSTEM(V51A)
RUN PROGRAM(DSNTEP2)
//SYSIN DD *
DELETE FROM SYSIBM.SYSPROCEDURES WHERE
PROCEDURE LIKE 'BASECASE%';
//*
//PROCESS EXEC DSNHSQL, MEM=BASECASE,
//          COND=(4, LT),
//          PARM.PC='HOST (SQL)', SOURCE, XREF, MAR(1, 72), STDSQL(NO)',
//          PARM.PCC='HOST (C)', SOURCE, XREF, MAR(1, 80), STDSQL(NO)',
//          PARM.C='SOURCE LIST MAR(1, 80) NOSEQ LO RENT',
//          PARM.LKED='AMODE=31, RMODE=ANY, MAP, RENT'
//PC.SYSUT1 DD DSN=&&SPDML, DISP=(, PASS),
//          UNIT=SYSDA, SPACE=(TRK, 1),
//          DCB=(RECFM=FB, LRECL=80)
//PC.SYSIN DD DISP=SHR, DSN=USER.SAMPLES.SOURCE(&MEM.)
//LKED.SYSLMOD DD DSN=USER.RUNLIB.LOAD(&MEM.),
//          DISP=SHR
//LKED.SYSIN DD *
INCLUDE SYSLIB(DSNRLI)
NAME BASECASE(R)
//ISRTCATG EXEC PGM=IKJEFT01, DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(V51A)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIAD)
END
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD DSN=&&SPDML, DISP=(OLD, DELETE) <-FROM PRECEDING STEP
//*****
//* BIND STEP
//*****
//BINDSP EXEC TSOBATCH, DB2LEV=DB2A
//SYSTSIN DD *
DSN SYSTEM(V51A)
BIND PACKAGE(COLLID) MEMBER(BASECASE) ACT(REP) ISO(CS)
//*****
//* SELECT FROM CATALOG THIS PROCEDURE
```

```

//*****
//SELCATG EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN DD *
DSN SYSTEM(V51A)
RUN PROGRAM(DSNTEP2)
//SYSIN DD *
SELECT * FROM SYSIBM.SYSPROCEDURES WHERE
PROCEDURE LIKE 'BASECASE%';

```

#### 4.5.3.2 JCL for DSNHSQL procedure

A sample of the JCL for DSNHSQL which was executed against a DB2 version 5 environment is shown below. This JCL is supplied when you install the SQL Procedures support. It performs the steps described in Figure 64 on page 137.

```

//*****
//*
//DSNHSQL PROC WSPC=500,MEM=TEMPNAME,USDN=USER
//*
//*****
//* PC: Precompile the PSM source
//*****
//PC EXEC PGM=DSNHPSM, PARM='HOST (SQL) ', REGION=4096K
//STEPLIB DD DISP=SHR, DSN=DB2A.DSNEXIT
// DD DISP=SHR, DSN=DB2A.DSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSCIN DD DSN=&&DSNHSQL, DISP=(MOD, PASS), UNIT=SYSDA,
// SPACE=(800, (&WSPC, &WSPC))
//SYSLIB DD DISP=SHR, DSN=&USDN..SRCLIB.DATA
//SYSUT1 DD DUMMY <-- DML to register PSM SP (V5 only)
//SYSUT2 DD DUMMY <-- DDL to register PSM SP (V6 and subsequent)
//*
//*****
//* PCC: Precompile C source generated by the previous step
//*****
//PCC EXEC PGM=DSNHPC, REGION=4096K,
// PARM='HOST (C), MAR(1, 80) ',
// COND=(4, LT, PC)
//DBRMLIB DD DISP=SHR, DSN=&USDN..DBRMLIB.DATA (&MEM)
//STEPLIB DD DISP=SHR, DSN=DB2A.DSNEXIT
// DD DISP=SHR, DSN=DB2A.DSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD DSN=&&DSNHSQL, DISP=(OLD, DELETE)
//SYSCIN DD DSN=&&DSNHOUT, DISP=(MOD, PASS), UNIT=SYSDA,
// SPACE=(800, (&WSPC, &WSPC))
//SYSLIB DD DISP=SHR, DSN=&USDN..SRCLIB.DATA
//SYSUT1 DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT2 DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//*
//*****
//* C: Compile the output from the precompiler
//*****
//C EXEC PGM=CBCDRVR, REGION=4096K,
// PARM=('MAR(1, 80) NOSEQ LO RENT'),
// COND=(4, LT, PC), (4, LT, PCC)

```

```

//STEPLIB DD DSN=CBC.SCBCOMP,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
//SYSLIB DD DSN=CEE.SCEEH.H,DISP=SHR
// DD DSN=DB2A.SDSNC.H,DISP=SHR
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT DD SYSOUT=*
//SYSPRT DD SYSOUT=*
//SYSTEM DD DUMMY
//SYSIN DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//SYSUT1 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT2 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT3 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//SYSUT14 DD UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//*
//*****
//* PLKED: Pre-linkedit the object module from the C compiler
//*****
//PLKED EXEC PGM=EDCPRLK,REGION=2048K,
// COND=((4,LT,PC),(4,LT,PCC),(4,LT,C))
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//SYMSGSGS DD DSN=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB DD DUMMY
//SYSIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD DD DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//*

```

```

//*****
/* LKED: Linkedit the output from the pre-linkeditor
//*****
//LKED EXEC PGM=IEWL, PARM='MAP',
// COND=( (4,LT,PC), (4,LT,PCC), (4,LT,C), (4,LT,PLKED) )
//SYSLIB DD DSN=CEE.SCEELKED, DISP=SHR
// DD DSN=DB2A.DSNLOAD, DISP=SHR
//SYSLIN DD DSN=&&PLKSET, DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//SYSLMOD DD DSN=&USDN..RUNLIB.LOAD(&MEM), DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD SPACE=(32000, (30,30)), UNIT=SYSDA
//*
//DSNHSQ L PEND

```

Once the SQL stored procedure is created and its definition registered in DB2, you can test it by invoking it from a client application or by using the SPB Tool. Through the SPB Tool, all you need to do is to insert a connection to the OS/390 database, highlight the relevant procedure, and choose the RUN button.

**Note:** If your SQL stored procedure does not return any output, you must check if it was linked and bound correctly.

If the procedure is created manually (not using the OS/390 Procedures Processor), the definition for your SQL stored procedure will not be inserted into the tables SYSIBM.SYSPSM or SYSIBM.SYSPSMOPTS. Therefore, the source cannot be viewed through the SPB Tool automatically.

If you wish to view the source using the SPB Tool, you need to download the SQL stored procedure source to the workstation environment, where the SPB is installed, and save it in any directory. After that, when you choose the Get Source option, you can then point to where you have saved it. See Chapter 3, “The DB2 Stored Procedure Builder” on page 57 for more details on this option.

You can even choose to save all your SQL stored procedure source for your project in a shared drive. This could be important in an environment where the DB2 application developer or DB2 administrator wishes to keep track of the stored procedures for tuning or maintenance purposes through the SPB Tool.

---

## 4.6 Stored procedure debugging

Regardless of the method used to build your SQL stored procedure, you can invoke the Debug tool to debug it.

To debug your SQL stored procedure on OS/390, you need to install the Remote debugger client code on your workstation. Refer to 3.4.6, “Debugging stored procedures” on page 105, for detailed information about how to install and activate the client debugger.

## 4.6.1 Process

On the OS/390 side, you need to perform the following steps:

- Concatenate the debug load modules in the STEPLIB within your WLM region.
- Create your SQL stored procedure specifying the RUN OPTIONS keyword in your SQL source:

```
CREATE PROCEDURE MYPROC
  (INOUT P1 CHAR(6), IN P2 DATE, IN P3 TIME, IN P4 TIMESTAMP)
LANGUAGE SQL
  RUN OPTIONS 'POSIX(ON),TEST(ALL,*, ,VADTCP&9.112.111.13:*)'
LABEL1: BEGIN
  . . . . .
END LABEL1
```

In this example, 9.112.111.13 is the IP address of the workstation where you will be monitoring the debugger output of your SQL stored procedure. It will be the machine on which you have activated the listener. The debugger screens will be activated on this machine.

- Compile your SQL stored procedure with the TEST compile option and OPT(0). We recommend that you do not compile using options TEST and OPT(1) or OPT(2). Programs compiled with both the TEST option and either the OPT(1) or OPT(2) options do not have line hooks, block hooks, path hooks or a symbol table generated, regardless of the TEST suboptions specified. Refer to *Debug Tool: User's Guide and Reference*, SC09-2137.
- Ensure that your input C file (not output listing) to the C compile is created on a permanent data set. When using batch JCL to create your SQL stored procedure, this will be the dataset (in the sample JCL procedure DSNHSQL) called &&DSNHOUT which is normally allocated as a temporary dataset. When creating your SQL stored procedure using SPB or DSNTPSMP, this dataset is permanently allocated anyway.
- Run the SQL stored procedure and the debugger screens will appear on the machine identified by the TCPIP address selected above.

## 4.6.2 If the debugger does not start

If your debugger does not start, check that the following things have been done:

- The client listener has been started.
- The correct SPAS or WLM JCL is being used.
- IP address is correct.
- The RUN OPTIONS are specified correctly (for example, a wrong number of commas might have been used).
- The TEST option has been specified on the C compile.
- The input C file to the C compile has been made a permanent dataset (please note that it is the input file that is required — not the output listing).





---

## Chapter 5. SQL Procedures for DB2 UDB for UNIX, Windows, OS/2

This chapter describes the support for the new SQL Procedures language in DB2 UDB servers. During this project we use Windows NT and UNIX platforms.

---

### 5.1 General considerations

SQL Procedures is an easy-to-use, simple language, which provides a series of language elements that allow you to develop block-structured stored procedures, with exception handling, flow control, variable declarations, and so on. Using the SQL Procedures language, you have the same functions and performance as when using other supported languages, such as multiple parameters (input, output, input/output), returning multiple output result sets, and so on. However, the SQL Procedures language is easier for all developers to use, and is especially easy to learn for those developers familiar with Sybase, Oracle, Informix, and Microsoft SQL Server proprietary languages.

SQL stored procedures are created using the CREATE PROCEDURE statement and are registered in the DB2 catalog. There are four tables in the DB2 UDB catalog that contain information related to stored procedures:

- **SYSIBM.SYSPROCEDURES:** Contains a row for each stored procedure that is created
- **SYSIBM.SYSPROCPARMS:** Contains a row for each parameter of every stored procedure.
- **SYSIBM.SYSPROCOPTIONS:** Each row contains procedure-specific option values.
- **SYSIBM.SYSPROCPARMSOPTIONS:** Each row contains procedure parameter specific option values.

The source code of SQL stored procedures is stored in the DB2 catalog. The TEXT column of SYSIBM.SYSPROCEDURES table contains the source of your SQL stored procedure. You can easily access the source code using a SELECT statement.

There are no changes related to coding client programs to invoke SQL stored procedures. The syntax of the SQL CALL statement is the same regardless of the language being used at the server stored procedure.

The support for SQL stored procedures in DB2 UDB is implemented through the generation of an intermediary C code. This C code is precompiled, compiled, and link-edited automatically, and an executable file (.DLL in Windows NT) is generated for the stored procedure. For more details, refer to 5.5, "Stored procedures preparation" on page 159.

The IBM Distributed Debugger can be used to remotely debug SQL stored procedures executing on the DB2 UDB server. With the IBM Distributed Debugger you can follow the execution of your SQL stored procedure using the source code, verify and change values for variables, etc. For more information on debugging, refer to 5.6, "Stored procedure debugging" on page 166.

Stored procedures written in the SQL Procedures language are portable to DB2 servers in other platforms with no changes or minimal changes to the sources.

When other database management systems (DBMS) implement languages compatible with the SQL/PSM standard, it should be possible to port stored procedures from DB2 to other DBMSs and vice versa.

---

## 5.2 Supported platforms

SQL Procedures language will be supported in all DB2 UDB platforms: UNIX, Windows and OS/2. However, the first releases (beta code) of SQL Procedures are only supported on Windows NT, AIX, and Sun Solaris. The support for other platforms will follow shortly.

The remote debugging of DB2 UDB stored procedures is only available for servers on Windows NT, AIX, Sun Solaris, and OS/2. The IBM Distributed Debugger client executes only on Windows NT.

---

## 5.3 System requirements and planning

This section describes the requirements for using SQL Procedures with DB2 UDB servers.

Before creating SQL stored procedures, you must ensure that DB2 SDK is installed on your DB2 UDB server. This is required because the process that creates the SQL stored procedures on the server generates a C source that must be prepared using SDK functionality. You do not need to have DB2 SDK installed in the developer's client workstation, unless you plan to use SPB to build your stored procedures. It is recommended that you install SDK in the developer's workstation so they can benefit from the samples and manuals included in SDK, and also, to allow the developers to create client applications running on their workstations. Refer to Chapter 3, "The DB2 Stored Procedure Builder" on page 57 for more information about DB2 SPB.

Note that SPB is not a prerequisite to work with SQL stored procedures. The SQL Procedures language support is built into DB2 UDB base code, and you can create your SQL stored procedures using only the DB2 command line, or any other user interface that allows you to issue a CREATE PROCEDURE command.

### 5.3.1 Requirements for the Windows NT platform

To work with SQL Procedures, you must ensure that a C compiler supported by DB2 SDK is installed in your DB2 UDB server. For DB2 UDB servers executing on Windows NT platforms, initially the only C compiler supported is:

- Microsoft Visual C++ Version 5.0 and 6.0

The IBM VisualAge C compiler should be supported in future releases.

Some customization in the C environment is required to use SQL Procedures. On the Windows NT platform, there are two possible ways to customize your C environment: using DB2 registry variables, or using NT system variables.

#### 5.3.1.1 Customizing the C environment using the DB2 registry

DB2 UDB has a registry variable DB2UDP\_COMPILER\_PATH that can be set, when you plan to use SQL Procedures. This registry variable should contain the name of a script file that initializes the C compiler environment. DB2 UDB

executes the command stored in the DB2UDP\_COMPILER\_PATH automatically, when you issue a CREATE PROCEDURE statement for an SQL stored procedure.

You can set the DB2UDP\_COMPILER\_PATH DB2 registry variable using the db2set command, as follows:

```
db2set DB2UDP_COMPILER_PATH=initscript
```

In our project, we used the sample udpprof.bat file for initialization. This file is located in x:\sql1lib\samples\udp, where x: is the drive on which you have installed DB2 SDK. If you plan to use the sample script, you can issue the following:

```
db2set DB2UDP_COMPILER_PATH=c:\sql1lib\samples\udp\udpprof.bat
```

You must customize the sample udpprof.bat script according to your development environment. If the initialization script does not contain the correct settings, or the DB2UDP\_COMPILER\_PATH does not point to the right script, you will not be able to create SQL stored procedures. For example, the following shows the udpprof.bat file used in our project:

```
udpprof.bat
set VC_DRIVE=c:\progra~1\devstu~1
set include=%VC_DRIVE%\vc\include;%VC_DRIVE%\vc\atl\include;
%VC_DRIVE%\vc\mfc\include;%include%
set lib=%VC_DRIVE%\vc\lib;%lib%
set path=%VC_DRIVE%\sharedide\bin\ide;%VC_DRIVE%\sharedide\bin;
%VC_DRIVE%\vc\bin;%path%
```

### 5.3.1.2 Customizing the C environment using system variables

On the Windows NT environment, instead of using the DB2UDP\_COMPILER\_PATH DB2 registry variable, you have the option of using NT system environment variables. SQL Procedures support requires the following variables to be at system level for the DB2 server operating environment:

- INCLUDE
- LIB
- PATH

When you install Microsoft Visual C++, it defines these variables as user variables. For SQL Procedures, you must define these variables as system variables. To copy from a user variable into a system variable follow the steps below:

1. **Start -> Settings -> Control Panel -> System -> Environment tab**
2. Click on the required variable name in the **User Variables** list box
3. Highlight the entry in the **Value** entry field and click **Edit -> Copy**
4. Click on the required variable name in the **System Variables** list box
5. Check the differences between user and system variable values and modify the values in the system variable as appropriate in the **Value** entry field

Once this has been done for all the three variables required, reboot the machine.

### 5.3.2 Requirements for the UNIX platform

To work with SQL Procedures, you must ensure that a C compiler supported by DB2 SDK is installed in your DB2 UDB server. For DB2 UDB servers executing on the UNIX platform, to create SQL stored procedures, you must:

- Install the DB2 SDK on your DB2 server.  
Ensure that the C compiler supported by the DB2 SDK on your platform is installed and configured on your DB2 server.
- Install a supported C compiler.
  - SQL Procedures support is available for the following C compilers on AIX Version 4.2 and later:
    - IBM C for AIX Version 3.1.4 and 3.6.4
  - SQL Procedures support is available for the following C compilers on Sun Solaris Version 2.5.1, 2.6, and 2.7 (Solaris 7):
    - SPARCompiler C Version 4.2
    - SPARCompiler C Version 5.0
- Initialize the environment for SQL Procedures.

To enable the DB2 server to create SQL stored procedures, you must set the `DB2_SQLROUTINE_COMPILER_PATH` DB2 registry variable. When you issue a `CREATE PROCEDURE` statement for an SQL stored procedure, DB2 executes the command stored in the `DB2_SQLROUTINE_COMPILER_PATH` registry variable to initialize the C compiler environment. You can store the command to call an initialization script in the `DB2_SQLROUTINE_COMPILER_PATH` using the following syntax:

```
db2set DB2_SQLROUTINE_COMPILER_PATH="<initscript>"
```

where *<initscript>* represents the command you use to call an initialization script for the C compiler on your platform.

- Set up the environment for SQL Procedures

The Solaris and AIX platforms contain a sample initialization script called `udppro` in the `$DB2PATH/samples/sqlproc` directory, where `$DB2PATH` represents the directory in which you have created your DB2 instance. For example, if you create a DB2 instance in the `/home/db2inst1` directory, you can set the `DB2_SQLROUTINE_COMPILER_PATH` DB2 registry variable to call the sample initialization script with the following command:

```
db2set DB2_SQLROUTINE_COMPILER_PATH=". $DB2PATH/samples/sqlproc/udpprof"
```

where `$DB2PATH` represents the directory in which you have created your DB2 instance.

**Note:** You should use the `udpprof` sample script only as a basis for developing your own initialization script. You must customize the script to correspond to your own operating system and compiler environment. If the initialization script contains the wrong settings, does not have executable permissions, or if the `DB2_SQLROUTINE_COMPILER_PATH` DB2 registry variable is not set to run the script, you will not be able to create SQL stored procedures.

### 5.3.3 Changing compiler options

If you want to customize your C compiler options for SQL Procedures, you must store the entire command line, including all options, in the DB2 registry variable with the following command:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=<compiler_command>
```

where <compiler\_command> represents the C compiler command and all of the options and parameters required to create SQL stored procedures.

#### 5.3.3.1 Windows NT

On Windows NT, the default value for DB2\_SQLROUTINE\_COMPILE\_COMMAND is:

```
"cl -Od -W2 /TC -D_X86_=1  
-IE:\SQLLIB\include sqlroutine_filename.c /link -dll  
-def:sqlroutine_filename.def /out:sqlroutine_filename.dll  
E:\SQLLIB\lib\db2api.lib"
```

**Note:** To return debug information, you must set this variable using a command such as the following:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cl -Od -W2 /TC -D_X86_=1 -Z7  
-IE:\SQLLIB\include sqlroutine_filename.c /link -dll  
-def:sqlroutine_filename.def /out:sqlroutine_filename.dll  
-debug:full -pdb:none -debugtype:cv E:\SQLLIB\lib\db2api.lib"
```

where:

```
sqlroutine_filename represents the placeholder for the filename used in the  
generated files such as SQC, C, PDB DEF files, EXP  
files, messages log, and DLL files  
  
sqlroutine_entry represents the entry point name  
  
E:\ represents the location of the instance directory
```

To return to the default compiler options, clear the DB2 registry value for DB2\_SQLROUTINE\_COMPILE\_COMMAND with the following command:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND=
```

#### 5.3.3.2 AIX

On AIX, the default value for DB2\_SQLROUTINE\_COMPILE\_COMMAND is:

```
"xlc_r -+ -H512 -T512 -I/home/myusr/sqllib/include  
sqlroutine_filename.c -bE:sqlroutine_filename.exp -e  
sqlroutine_entry -o sqlroutine_filename -L/home/myusr/sqllib/lib -lc -ldb2"
```

**Note:** To return debug information, you must set this variable using a command such as the following:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="xlc_r -+ -H512 -T512 -g  
-I/home/myusr/sqllib/include sqlroutine_filename.c  
-bE:sqlroutine_filename.exp -e sqlroutine_entry  
-o sqlroutine_filename -L/home/myusr/sqllib/lib -lc -ldb2"
```

where:

`sqlroutine_filename` represents the placeholder for the filename used in the generated files such as SQC, C, PDB DEF files, EXP files, messages log, and shared library files

`sqlroutine_entry` represents the entry point name

`home/myusr/` represents the location of the instance directory

### 5.3.3.3 Solaris

On Solaris, the default value for `DB2_SQLROUTINE_COMPILE_COMMAND` is:

```
"cc -# -Kpic
-I/disks/home1/myusr/sqllib/include sqlroutine_filename.c
-G -o sqlroutine_filename -L/disks/home1/myusr/sqllib/lib
-R/disks/home1/myusr/sqllib/lib -ldb2"
```

**Note:** To return debug information, you must set this variable using a command such as the following:

```
db2set DB2_SQLROUTINE_COMPILE_COMMAND="cc -# -Kpic -g
-I/disks/home1/myusr/sqllib/include sqlroutine_filename.c -G -o
sqlroutine_filename -L/disks/home1/myusr/sqllib/lib
-R/disks/home1/myusr/sqllib/lib -ldb2"
```

where:

`sqlroutine_filename` represents the placeholder for the filename used in the generated files such as SQC, C, PDB DEF files, EXP files, messages log, and shared library files

`sqlroutine_entry` represents the entry point name

`/disks/home1/myusr/` represents the location of the instance directory

## 5.3.4 Retaining intermediate files

Issuing an SQL Procedures `CREATE PROCEDURE` statement, DB2 creates a number of intermediate files that are normally deleted if DB2 successfully completes the statement. If an SQL stored procedure does not perform as expected, you might find it useful to examine the SQC, C, PDB, and message log files created by DB2. To keep the files that DB2 creates during the successful execution of a `CREATE PROCEDURE` statement, you must set the value of the `DB2_SQLROUTINE_KEEP_FILES` DB2 registry variable to 1. This can be done through the following command:

```
db2set DB2_SQLROUTINE_KEEP_FILES=1
```

The intermediate files will be retained by DB2 in the following directories:

- **Windows NT**

```
%DB2PATH%\function\routine\udp\%$DATABASE%\%SCHEMA%
```

where:

`%DB2PATH%` represents the instance directory

`%DATABASE%` represents the database name

`%SCHEMA%` represents the schema name with which the SQL stored procedures were created

- **AIX, Solaris**

`$DB2PATH/function/routine/sqlproc/$DATABASE/$SCHEMA`

where:

`$DB2PATH` represents the instance directory

`$DATABASE` represents the database name

`$SCHEMA` represents the schema name with which the SQL stored procedure were created

---

## 5.4 Coding considerations

This section presents some considerations on coding SQL stored procedures for DB2 UDB. Most of the considerations and techniques described here are valid for Windows and UNIX servers, and any topic that applies only to one type of server will be clearly identified.

This section does not provide detailed information about the syntax of the SQL Procedures language. Some examples are provided, but for more information on the SQL Procedures language refer to Chapter 2, “The SQL Procedures language” on page 9.

### 5.4.1 Recommendations for writing portable stored procedures

Make sure that the builtin functions you use in your stored procedure are supported on all the target DB2 platforms you will use.

### 5.4.2 Structure of SQL stored procedures

An SQL stored procedure consists of two main blocks:

- A CREATE PROCEDURE statement to define the procedure
- A procedure body with SQL statements and SQL control statements

#### 5.4.2.1 The CREATE PROCEDURE statement

The CREATE PROCEDURE statement is used to register a stored procedure in the DB2 server. For SQL stored procedures, most of the options of the CREATE PROCEDURE statement are not valid, because these options are related to external stored procedures.

For SQL stored procedures, using the CREATE PROCEDURE statement, you can specify the list of parameters being passed to or from the stored procedure, an specific name for the procedure, the number of result sets being returned, and if the stored procedure reads or modifies SQL data. Following is a typical CREATE PROCEDURE statement for an SQL stored procedure:

```
CREATE PROCEDURE SQL1LNS (IN PARM1 CHAR(10), OUT PARM2 INTEGER) SPECIFIC
S4141979 RESULT SETS 1 READS SQL DATA LANGUAGE SQL BEGIN .... END
```

With DB2 UDB, you can have various procedures with the same procedure name, as long as they have different specific names, and a different number of parameters. DB2 UDB recognizes which of the procedures with the same name you are actually calling, by comparing the number of parameters being passed and the number of parameters defined in the catalog. Note that DB2 UDB only

counts the number of parameters; that is why you must have a different number of parameters for procedures with the same name.

For SQL stored procedures, you cannot specify the following options of the CREATE PROCEDURE statement:

- NO SQL
- FENCED/NOT FENCED
- DETERMINISTIC/NOT DETERMINISTIC
- CALLED ON NULL INPUT
- DBINFO/NO DBINFO
- EXTERNAL
- PARAMETER STYLE
- PROGRAM TYPE

The above options can only be used with external stored procedures. If you try to code your CREATE PROCEDURE statement for an SQL stored procedure using any of the above options, you will receive the following message:

```
SQL0628N Multiple or conflicting keywords involving the "<invalid attribute  
for SQL procedure>" clause are present.  SQLSTATE=42613
```

SQL stored procedures execute as NOT FENCED, unless they are returning result sets to the calling program. If the SQL stored procedure returns result sets, it executes as a FENCED stored procedure. This is done internally, and you cannot change the mode of execution of your SQL stored procedure.

#### 5.4.2.2 Defining parameters

If your SQL stored procedure needs to send/receive parameters from the client program, you must define them in the CREATE PROCEDURE statement.

You can define input, output, or input/output parameters of any supported SQL data types. The following data types are not supported:

- VARGRAPHIC
- LONG VARGRAPHIC
- BLOB
- CLOB
- DBCLOB
- DATALINK

User Defined Data types (UDTs) are not supported, even if they are based on supported SQL data types.

### 5.4.3 Coding the SQL stored procedures body

The stored procedures body in an SQL stored procedure contains the source statements for the stored procedure. The stored procedures body can contain a single SQL statement or a compound SQL statement. Following is an example of an SQL stored procedure with a single SQL statement:

```
CREATE PROCEDURE SQL2LNS (OUT PARM1 CHAR(10))
```



```
LANGUAGE SQL
  SET PARM1='FELIPE'
```

A stored procedures body with a compound SQL statement must be delimited by a BEGIN and an END statement. Following is an example of an SQL stored procedure with a compound SQL statement:

```
CREATE PROCEDURE SQL3LNS (OUT PARM1 CHAR(10), OUT PARM2 CHAR(10))
LANGUAGE SQL
P1:BEGIN
  SET PARM1='ALINE'
  SET PARM2='RICARDO'
END P1
```

A compound SQL statement may have a label associated to it. In the above example, the label P1 was set for the compound SQL statement. The labels in your SQL stored procedures body must be unique, and must not be the same as the name of the stored procedure. Any variables within a labeled compound SQL statement may be prefixed with the label, if necessary. See 5.4.3.2, “Defining variables in SQL stored procedures” on page 153 for more details.

#### 5.4.3.1 Statements in the stored procedures body

The SQL stored procedures body may contain SQL statements and SQL control statements. All executable SQL statements can be contained within an SQL stored procedures body, with the exception of the following:

- COMMIT
- CONNECT
- DISCONNECT
- RELEASE
- SET CONNECTION
- REVOKE

**Note:** The above restrictions are intended to be removed in future releases of SQL Procedures support in DB2 UDB. Check your DB2 UDB manuals, to verify if these restrictions still apply.

SQL control statements can be assignment statements, CASE statements, IF statements, LOOP statements, and others, as defined in SQL Procedures. For more information regarding the syntax of the different SQL Procedures control statements, please refer to Chapter 2, “The SQL Procedures language” on page 9.

#### 5.4.3.2 Defining variables in SQL stored procedures

Within a compound SQL statement you can declare SQL variables that can be referenced in other statements in the stored procedure using the DECLARE statement.

##### ***Recommended naming for parameters and variables:***

You do not need to declare your parameters as variables within your stored procedure. Your parameter description provides DB2 enough information to understand what the names and datatypes of the parameters are.

Name your local variables with different names than the parameters to your stored procedure. If you choose not to do this, then references to the ambiguous name will be interpreted as the local variable and not the parameter. To reference a parameter in this situation, you must qualify it with the stored procedure name. Figure 65 is an example of using variables and parameters with the same name.

**Avoid:**

Naming your parameters and your local variables with the same name.

```
CREATE PROCEDURE DRDARES1.SMP4LNS (out v_var1 integer, out
v_var2 double )
    SPECIFIC DRDARES1.S1473953
    LANGUAGE SQL

P1: BEGIN
    declare v_var1 integer;
    declare v_var2 smallint;

    set v_var1 = 10;
    set smp4lns.v_var1 = p1.v_var1;
    set v_var2 = 20;
    set smp4lns.v_var2 = p1.v_var2;

END P1
```

*Figure 65. Using the same name for variables and parameters*

Note that the variable `v_var2` does not have the same data type as the parameter, but DB2 handles data type conversions whenever possible.

You should avoid variables with the same name as DB2 table columns. If you have a variable with the same name as a table column, you must also qualify the variable in SQL statements with the compound statement label. If you do not qualify the variable, DB2 will interpret that variable as the column name. Figure 66 is an example of using a variable and a column with the same name.

```
CREATE PROCEDURE DRDARES1.SMP5LNS (out p_id integer )
    LANGUAGE SQL

P1: BEGIN
    declare id integer;
    set id = 100;
    select id into id from staff where id=p1.id;
    set p_id = id;

END P1
```

*Figure 66. Using variables with the same name as a column*

Note that it is not necessary to qualify the variable in the INTO clause, only in the WHERE condition.

In variable declarations you can specify a default value. If a default value is not specified, the variable is initialized to NULL.

### 5.4.3.3 Assigning values to variables

The SET statement is used to assign values to variables and parameters. With DB2 UDB, the value being assigned to the variable may be a constant, an expression, a DB2 special register, a result of a SELECT statement, and so on. Figure 67 is an example of assigning special register values to parameters in SQL stored procedures.

```
CREATE PROCEDURE DRDARES1.SMP2LNS ( out v_user char(8), out
v_date1 date,
out v_date2 date, out v_days1 integer, out v_time1 time,
out v_time2 time, out v_timest1 timestamp, out v_timest2
timestamp)
    SPECIFIC DRDARES1.S5336343
    LANGUAGE SQL
P1: BEGIN
    set v_user = user;
    set v_date1 = current date;
    set v_date2 = v_date1 - 10 days + 3 years;
    set v_days1 = days(v_date2);
    set v_time1 = current time;
    set v_time2 = v_time1 +1 hour - 30 minutes;
    set v_timest1 = current timestamp;
    set v_timest2 = v_timest1 - days(v_date1 - 10 days) days;

END P1
```

Figure 67. Assigning special registers to variables

You can also assign the result of DB2 UDB built-in functions or User Defined Functions(UDFs) to variables or parameters. However, you should keep in mind that using DB2 UDB functions, may limit the portability of your SQL stored procedures, since some of the built-in functions or UDFs might not be available in other DB2 servers, such as DB2 for OS/390 Version 5. Figure 68 is an example of assigning results of built-in functions to variables.

```

CREATE PROCEDURE DRDARES1.SMP8LNS (out v1 double, out v2 double,
out v3 integer, out v4 integer , out v5 integer,
out v6 integer , out v7 char (5),out v8 char(10), out v9
char(10), out v10 char (20), out v11 char(5), out v12 char(10))
    SPECIFIC DRDARES1.S8389109
LANGUAGE SQL
P1: BEGIN
    set v1 = tan(.5) - (sin(.5)/cos(.5));
    set v2 = exp(sin(.3)) + exp(cos(.3));
    set v3 = rand();
    set v4 = ceil(5.2) + floor(4.3);
    set v5 = quarter(current date);
    set v6 = week(current date);
    set v7 = repeat('*',5);
    set v8 = lcase('ALINE');
    set v9 = replace('alb1c1','1','2');
    set v10 = monthname(current date) || dayname(current date);
    set v11 = ltrim('felipe ');
    set v12 = substr('abcdefghijklmnopq',5,10);

END P1

```

Figure 68. Assigning results of built-in functions to variables

#### 5.4.3.4 Handling errors

In DB2 UDB SQL stored procedures, you can make references in your stored procedures body code to SQLCODE or SQLSTATE. All you need to do is to declare them previously in your stored procedures body.

We strongly recommend that you use handlers for finding and handling error conditions in your SQL stored procedure. This is the only choice you should make if you intend for your SQL stored procedure to be portable across DB2 platforms. Here is an example of how to do this:

```

CREATE PROCEDURE PROC1()
LANGUAGE SQL
BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    .....
END

```

Avoid checking SQLCA or SQLSTATE values. This is a tricky area and there are some platform differences. If you do use SQLSTATE or SQLCA checks explicitly in your SQL stored procedure, please note that you can only check one of these values.

You can declare program variables to hold the SQLCODE or the SQLSTATE for different tests. Keep in mind that you have to choose one of these variables, SQLCODE or SQLSTATE, because the second assignment statement will not get the return code of the original statement, but from the previous one. Figure 69 shows an example of trying to set both variables in an SQL stored procedure.

```

DECLARE SQLCODE INTEGER DEFAULT 0;
DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
DECLARE SQLC INTEGER DEFAULT 0;
DECLARE SQLST CHAR(5) DEFAULT '00000';
....

SELECT ID INTO VID FROM STAFF WHERE ID = 13; (1)
SET SQLC = SQLCODE; (2)
SET SQLST = SQLSTATE; (3)
....

```

Figure 69. Setting both SQLCODE and SQLSTATE variables to program variables

Note that in Figure 69, when you execute the SELECT statement: **(1)** if the row is not found, automatically DB2 UDB sets SQLCODE to 100 and SQLSTATE to '02000'. The SET statement **(2)** sets the program variable SQLC correctly to the value of SQLCODE, then SQLC is set to 100. However, the SET statement also resets the values of SQLCODE and SQLSTATE, and since the command executed successfully, they are both set to 0. When you execute the second SET statement **(3)**, the SQLSTATE you get no longer refers to the SQLSTATE of the SELECT statement **(1)**, but to the SQLSTATE of the SET statement **(2)**. This is the reason why you can only work with one of the variables.

The best way to write portable SQL stored procedures is to perform error handling using the DECLARE CONDITION and the DECLARE HANDLER statements in your SQL stored procedures. This will ensure portability of your SQL stored procedures among different DB2 platforms.

Nested condition handlers are not supported in the SQL stored procedures body. For more information on declaration of handlers, refer to Chapter 2, "The SQL Procedures language" on page 9.

#### 5.4.3.5 Nested compound statements

SQL Procedures support implemented by DB2 UDB allows you to have nested compound statements in your SQL stored procedures body. However, you cannot have ATOMIC compound statements nested. Figure 70 shows an example of a nested compound statement in an SQL stored procedure.

```

P1: BEGIN
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        BEGIN
            SET SQLC=SQLCODE;
            SET VAR1=1;
        END;

    SELECT id into vid FROM STAFF where id = vid;

END P1

```

Figure 70. Nested compound statement

Another example:

```

P1: BEGIN

```

```

DECLARE CONTINUE HANDLER FOR NOT FOUND
IF ( 1 = 1 ) THEN
    SET SQLC=SQLCODE;
    SET VAR1=1;
END IF;
SELECT id into vid FROM STAFF where id = vid;
END P1

```

In addition, the code generator supplies the following if SQLCODE and SQLSTATE radio buttons are selected in the wizard:

```

DECLARE EXIT HANDLER FOR SQLEXCEPTION
IF (1 = 1) THEN
    SET SQLSTATE_OUT = SQLSTATE;
    SET SQLCODE_OUT = SQLCODE;
END IF;

```

The "IF ( 1 = 1 )" gets around the compound statement limitation.

Since you cannot have **ATOMIC** nested compound statements, if you want to be able to undo the changes performed only in the nested compound statement, you can set a **SAVEPOINT** before the nested compound statement.

#### 5.4.3.6 Savepoints

Savepoints are points within an SQL stored procedure, that you can set to be able to rollback your transaction to that savepoint. A savepoint is set using the **SAVEPOINT** statement, and may have a name associated to it. (See Figure 71.)

```

P1: BEGIN
    UPDATE... ;
    INSERT... ;
    ...
    SAVEPOINT S1;
    BEGIN
        SET VAR1=1;
        DELETE... ;
        INSERT ... ;
    END;
    .....
END P1

```

*Figure 71. Setting a SAVEPOINT*

In your SQL stored procedures, if you detect an error, you can use the **ROLLBACK TO SAVEPOINT** statement to rollback changes performed after the savepoint was set. A rollback to a savepoint rolls back just the work done after the savepoint, the savepoint itself still exists. You can rollback to it again, if necessary. A rollback to a savepoint undoes any changes, and also closes all open cursors.

The first implementation of savepoints in DB2 UDB SQL Procedures does not allow nested savepoints. If you plan to build portable SQL stored procedures, you must be aware that DB2 for OS/390 V5 and DB2 for AS/400 do not support savepoints, but DB2 UDB for OS/390 V6 does.

#### 5.4.3.7 Considerations for comments and blank lines

Neither blank lines or comments are tolerated before the actual start of the stored procedure code, that is not in or before the CREATE PROCEDURE DDL statement. For example, the comments below are placed as early as they could possibly appear in the code.

```
CREATE PROCEDURE TEAM.Proc1 ( OUT SQLSTATE_OUT char(5),
                             OUT SQLCODE_OUT int )
    SPECIFIC TEAM.S1036175
    RESULT SETS 1
    LANGUAGE SQL
-----
-- SQL stored procedure TEAM.Proc1
-----
P1: BEGIN
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE SQLCODE INT DEFAULT 0;

    -- Declare cursor
    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT * FROM SYSCAT.PROCEDURES;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        IF (1 = 1) THEN
            SET SQLSTATE_OUT = SQLSTATE;
            SET SQLCODE_OUT = SQLCODE;
        END IF;

    -- Cursor left open for client application
    OPEN cursor1;

    SET SQLSTATE_OUT = SQLSTATE;
    SET SQLCODE_OUT = SQLCODE;
END P1
```

---

## 5.5 Stored procedures preparation

In DB2 UDB, there are many different ways to build your SQL stored procedures into your server. You can use the Stored Procedures Builder tool, the DB2 Command Line Processor, the DB2 Command Center, or any application program issuing the CREATE PROCEDURE statement. Figure 72 shows different ways to create your SQL stored procedures.

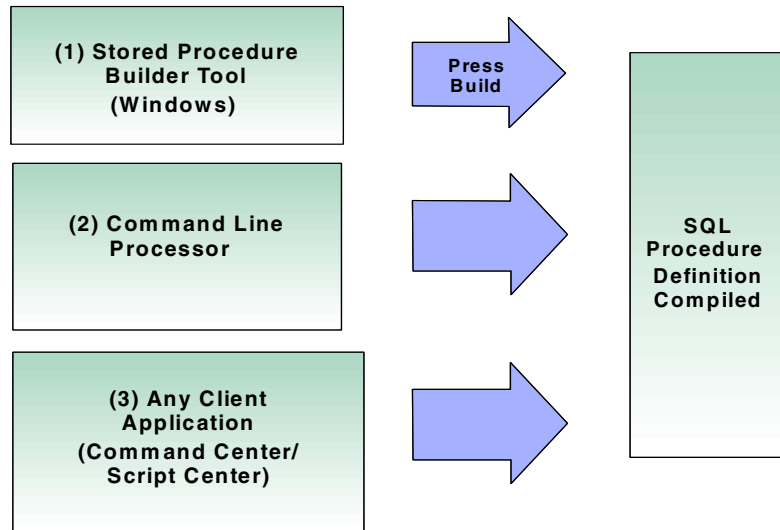


Figure 72. Ways to create SQL stored procedures in DB2 UDB

It is important to notice that, regardless of the method used to start the creation of the stored procedure, the process that is used by DB2 UDB internally is always the same, and the results obtained are equal.

When you submit a CREATE PROCEDURE statement, DB2 UDB performs a series of steps to build your procedure in the server. These steps are always the same, and involve the creation of a C source from your SQL stored procedures source. This C source is precompiled, compiled, and linked to generate a DLL for your SQL stored procedures. So, your SQL stored procedures are not interpreted during execution, which is important in terms of performance. The last step in the preparation of your SQL stored procedures is registering the procedure and parameters in the DB2 catalog tables SYSIBM.SYSPROCEDURES and SYSIBM.SYSPROCPARMS.

Figure 73 shows the steps involved in the preparation of an SQL stored procedure.



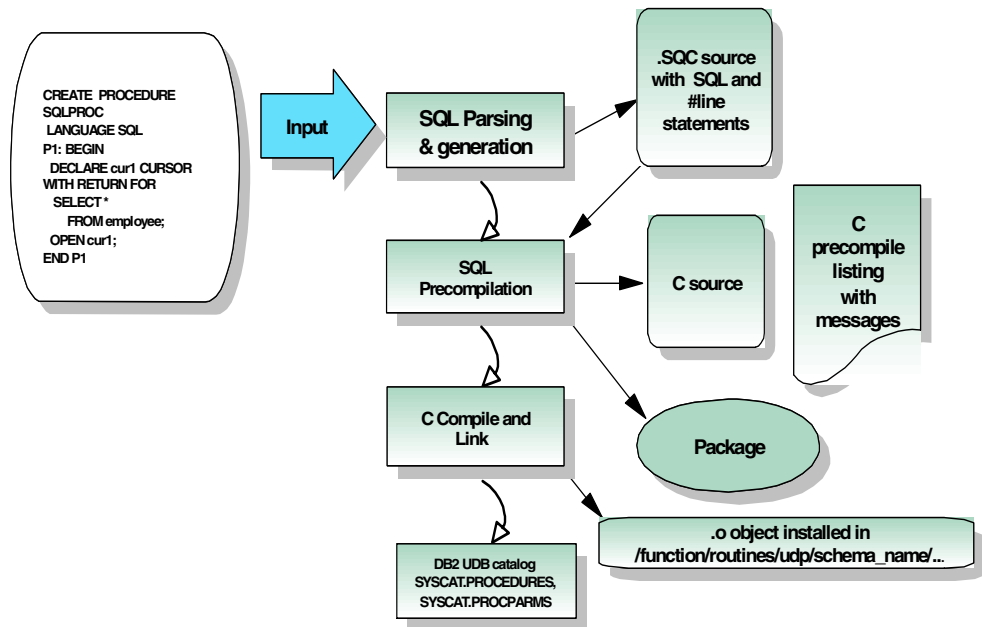


Figure 73. Preparation steps for SQL stored procedures in DB2 UDB

Note that as a result of the first step, SQL Parsing and Generation, an `sqc` file with a C source for your stored procedure is generated. This file also contains, as comments, your SQL stored procedures source. If you use the IBM Distribute Debugger for debugging your procedure, you will be able to debug using your SQL stored procedures source contained in the C source file. The `#lines` generated perform the mapping between the C statements being executed and the corresponding SQL Procedures statement. For more information on debugging your SQL stored procedures, refer to 5.6, “Stored procedure debugging” on page 166.

The executable files (DLLs in Windows NT) are created in the directory `/sqllib/function/routines/udp/schema_name`. The specific name of the stored procedure is used as the name for the executable file. As an example, if you have an SQL stored procedure named `PROC1` with an specific name `S4231567`, an executable named `S4231567` (`S4231567.DLL` in NT) will be created for your stored procedure.

An SQL stored procedure executes as a static application. A package is also bound as one of the steps of the preparation process. The specific name of the procedure is used as the package name.

### 5.5.1 Privileges required to prepare an SQL stored procedure

To issue a `CREATE PROCEDURE` statement that creates an SQL stored procedure, the authorization ID executing the statement must have at least one of the following privileges:

- `SYSADM` or `DBADM` authority
- `IMPLICIT_SCHEMA` authority on the database, if the implicit or explicit schema name of the procedure does not exist

- CREATIN privilege on the schema, if the schema name of the procedure refers to an existing schema

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

### 5.5.2 Preparing an SQL stored procedure from the DB2 CLP

To create an SQL stored procedure from the DB2 CLP, you must create the source of your stored procedure in a file.

The `CREATE PROCEDURE` statement must be interpreted by the DB2 CLP as a single SQL statement. However, DB2 CLP uses the semicolon (;) as the default delimiter for statements. When your stored procedures have compound statements, the statements within the compound statement are also terminated with semicolons, causing the DB2 CLP to interpret that as the end of the `CREATE PROCEDURE` statement, and a syntax error is raised.

To avoid this, you must use an alternative terminating character in your file, and change the DB2 CLP invocation command to identify this new character as the terminating character. The samples SQL stored procedures shipped with DB2 SDK use the \$ symbol as a terminating character.

Figure 74 shows a sample file, `STP.DB2`, containing a simple SQL stored procedure using the \$ symbol as terminating character.

```
STP.DB2

CONNECT TO SAMPRES1$

CREATE PROCEDURE DRDARES1.PROC1 ( )
    SPECIFIC DRDARES1.S3710781
    RESULT SETS 1
    LANGUAGE SQL
P1: BEGIN
    -- Declare cursor
    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT * FROM STAFF;

    -- Cursor left open for client application
    OPEN cursor1;

END P1 $

CONNECT RESET$
```

Figure 74. `STP.DB2` file containing SQL stored procedures using \$ as a delimiter

To execute the above file, you must invoke the DB2 CLP with the `-td` parameter to specify the \$ as the terminating character, as follows:

```
db2 -td$ -fSTP.DB2
```

### 5.5.3 Preparing an SQL stored procedure from the DB2 tools

If you plan to use DB2 tools such as the Command Center or the Script Center to submit your `CREATE PROCEDURE` statements, you must also change the

termination character for SQL statements, since the default for DB2 tools is also the semicolon (;) character.

To change the termination character for the DB2 Command Center, click on **Script->Options...** then check the **Use statement termination character** checkbox, and specify the character you want to use, for example, the \$.

Figure 75 shows the **Options** window of DB2 Command Center.

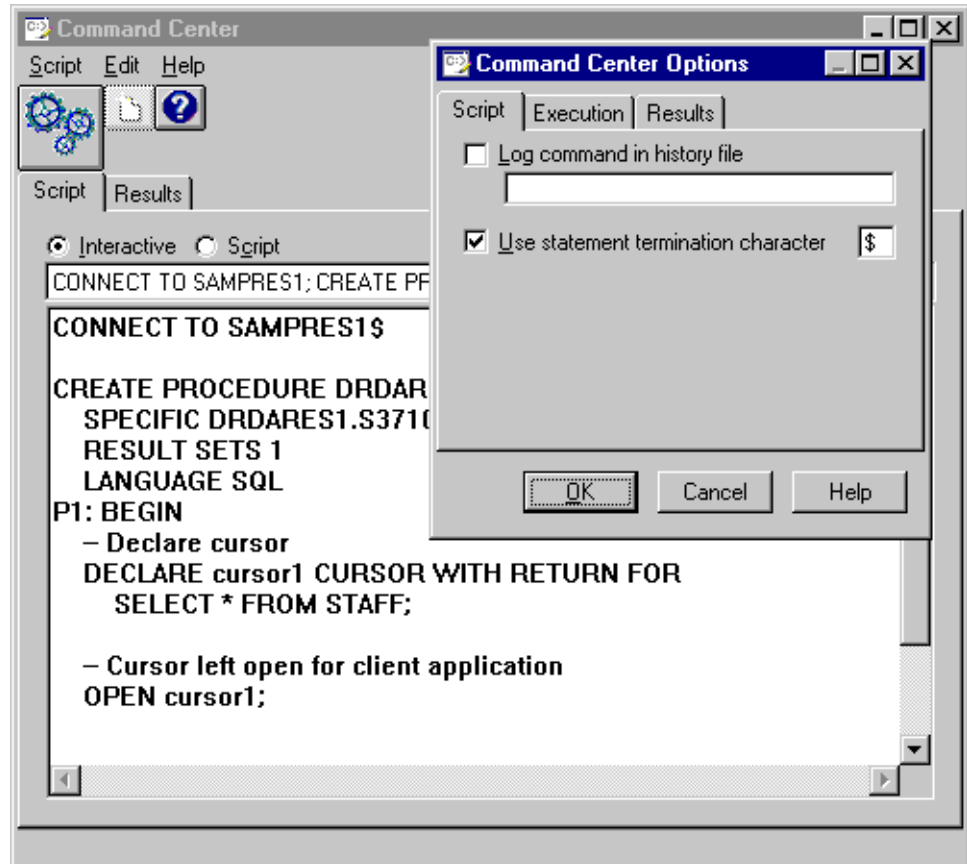


Figure 75. Changing the terminating character for the DB2 Command Center

To change the termination character for scripts submitted using the DB2 Script Center, click on **Tools->Tools Settings** then check the **Use statement termination character** checkbox, and specify the character you want to use, for example, the \$.

Figure 76 shows the **Tools Settings** window for DB2 tools.

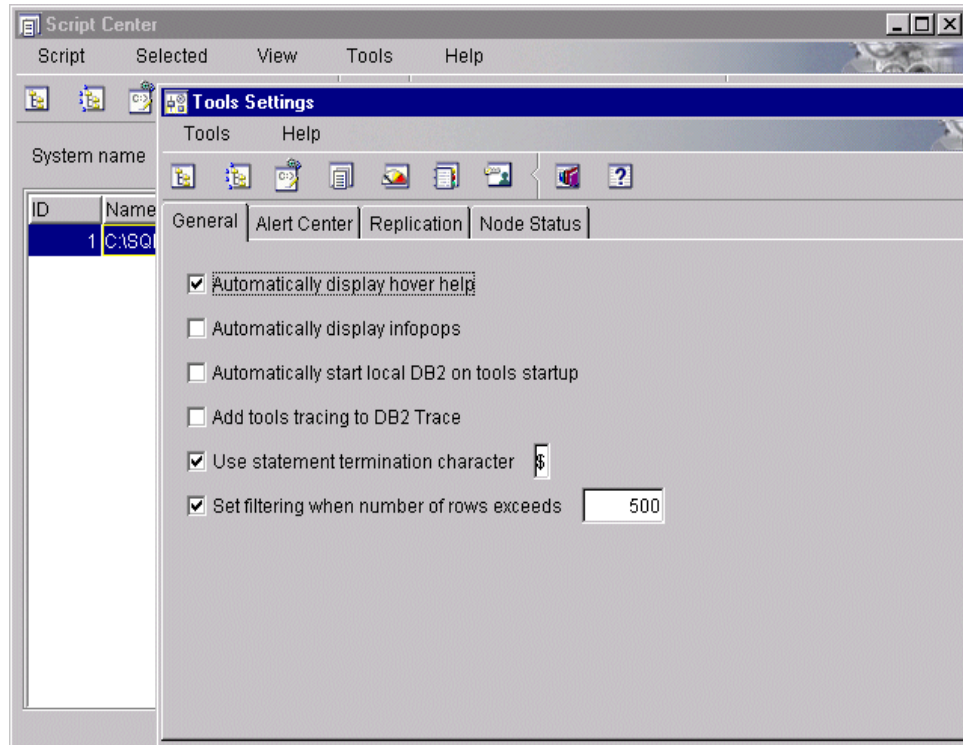


Figure 76. Changing the termination character for DB2 tools

#### 5.5.4 Preparing an SQL stored procedure from application programs

The CREATE PROCEDURE statement may be invoked from an application program written in CLI, ODBC, JDBC, or embedded SQL. It can be invoked in the application as a dynamic or a static SQL statement. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared.

#### 5.5.5 Preparing an SQL stored procedure from the SPB

If you create or change an SQL stored procedure with SPB, to prepare the stored procedure, all you have to do is right-click on the procedure name and then click on the **Build** option; or you can select the stored procedure and click on the **Build** icon. For more information about SPB, please refer to Chapter 3, “The DB2 Stored Procedure Builder” on page 57.

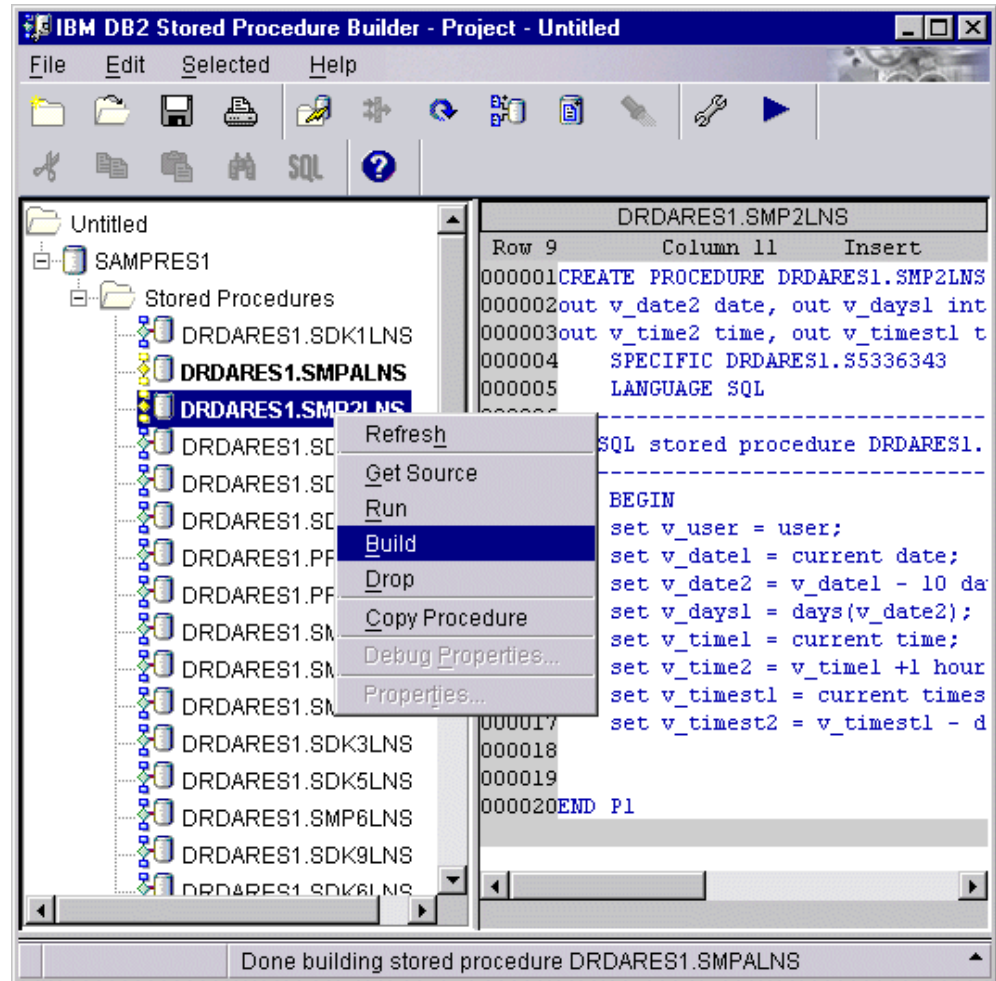


Figure 77. Using SPB to prepare SQL stored procedures

### 5.5.6 Copying SQL stored procedures between DB2 UDB servers

When the support for SQL Procedures become available for DB2 UDB, it is possible that a utility to generate the source CREATE PROCEDURE statements for the SQL stored procedures will be included. However, since the source is available in the DB2 UDB catalog, it is very simple to generate a sequential file with the CREATE PROCEDURE statements and then submit this file to another DB2 UDB server to replicate the stored procedures.

To generate a sequential file, after connecting to the DB2 server, issue the following command, from the **DB2 Command Window**:

```
db2 SELECT TEXT CONCAT '$' FROM SYSIBM.SYSPROCEDURES WHERE LANGUAGE = 'SQL'
> sqlp.ddl
```

The above command generates a file `sqlp.ddl`, that contains the CREATE PROCEDURE statements for all the SQL stored procedures in your database. You can include more conditions in the WHERE clause if you want to filter the procedures you want to copy. You can edit the file generated to remove the initial and final lines, and do any kind of changes you want, such as bulk changes to the schema name of the procedures, or table names.

After changing the file, you may submit it to another DB2 UDB server, by connecting to the server, and then issuing the following command:

```
db2 -td$ -fsqlp.ddl
```

The above command creates all the SQL stored procedures in the `sqlp.ddl` file at the remote DB2 UDB server.

---

## 5.6 Stored procedure debugging

The support for SQL Procedures implemented in DB2 UDB allows you to remotely debug stored procedures executing on the DB2 UDB server. The remote debug is already available for Java stored procedures. However, at the time of writing this redbook, remote debugging for SQL stored procedures (and C/C++) was still being developed, so we could not test this support.

In this section, we describe the steps required to remotely debug SQL stored procedures executing on a DB2 UDB server, based on the steps required for debugging Java stored procedures. Note that since we did not have the final version of the product, these steps may be different when the support for SQL Procedures is available.

### 5.6.1 Platforms supported for remote debugging

Although the support for SQL Procedures is implemented in all DB2 UDB server platforms, remote debugging of SQL stored procedures is only available for DB2 UDB servers executing on Windows NT, OS/2, AIX, and Sun platforms.

The IBM Distributed Debugger client must also be executing in a Windows NT system connected to the DB2 UDB server. The IBM Distributed Debugger client is included with DB2 UDB.

### 5.6.2 The DB2DBG.ROUTINE\_DEBUG debugger table

DB2 UDB holds information about the stored procedures you want to debug in a table named `DB2DBG.ROUTINE_DEBUG`. You must create this table in every DB2 UDB server database that you plan to debug stored procedures.

The file `db2debug.ddl` contains all the DDL to create the `DB2DBG.ROUTINE_DEBUG` table. This file is located in the `\SQLLIB\MISC` directory. To create the table, you must connect to the DB2 UDB server database and issue the following command:

```
db2 -tf x:\sqllib\misc\db2debug.ddl
```

The current version of the `db2debug.ddl` file creates the `DB2DBG.ROUTINE_DEBUG` table, a view named `DB2DBG.ROUTINE_DEBUG_USER`, and two triggers on the base table. The `DB2DBG.ROUTINE_DEBUG_USER` view, limits the access to the table only to rows belonging to the user connected to the database. The definition for the triggers is going to change, since the current triggers are used to ensure that the stored procedure being inserted in the table is a Java stored procedure.

The `DB2DBG.ROUTINE_DEBUG` table must be populated using `INSERT`, `UPDATE`, and `DELETE` SQL statements, or using the SPB Debug Properties dialog. Every stored procedure you want to debug must contain a row in the `DB2DBG.ROUTINE_DEBUG` table. Following is an example of an `INSERT` statement to include an entry in the debug table:

```
DB2 INSERT INTO db2dbg.routine_debug (AUTHID, TYPE, ROUTINE_SCHEMA,  
SPECIFICNAME, DEBUG ON, CLIENT_IPADDR, CLIENT_PORT) VALUES ('DRDARES1', 'S',  
'DRDARES1', 'S2351892', 'Y', '9.179.186.15',8000)
```

You must provide values for the columns of the table as follows:

- **AUTHID:** This contains the authorization id that is associated with the debugging of the stored procedure. DB2 UDB will search the debug table using the authorization id passed in the connect statement.
- **TYPE:** In the current version, the only value supported for this column is 'S' for stored procedures. In future versions, when the debugging facilities become available for other DB2 objects, such as functions, other values will be valid.
- **ROUTINE\_SCHEMA:** This is the schema associated with the stored procedure that you want to debug.
- **SPECIFICNAME:** This is the specific name of the stored procedure. If you do not know the specific name of the stored procedure you want to debug, you can check the SYSIBM.SYSPROCEDURES table to get the specific name.
- **DEBUG\_ON:** This column can contain a 'Y' to turn debugging for the stored procedure on, or a 'N' to turn off the debugging.
- **CLIENT\_IPADDR:** contains the IP address of the client workstation running the IBM Distributed Debugger client. When the stored procedure starts on the DB2 server, the debugger client will be started in the machine specified here. Note that this machine must be executing the IBM Distributed Debugger client daemon for receiving the debugger requests.
- **CLIENT\_PORT:** This contains the port number for the IBM Distributed Debugger client. This port number is specified when starting the debugger client in the client workstation.

There are two additional columns, DEBUG\_STARTN and DEBUG\_STOPN, that are not used in the current version.

### 5.6.3 DB2 environment variables for debugging

DB2 UDB has two environment variables that are used for the debugging of stored procedures.

The DB2ROUTINE\_DEBUG variable enables debugging for stored procedures in your DB2 UDB server instance. To debug your stored procedures, you must set this variable to ON, as follows:

```
db2set DB2ROUTINE_DEBUG=ON
```

To turn off debugging for your DB2 instance, reset the value of the DB2ROUTINE\_DEBUG variable, as follows:

```
db2set DB2ROUTINE_DEBUG=
```

The DER\_DBG\_PATH environment variable is used if the source code for the stored procedure resides on the client. This variable should be set at the client machine, and must provide the path where the source code resides on the client. In case of SQL stored procedures where the code resides in the DB2 tables, this variable is not used.

#### 5.6.4 Starting the debugger client

After you perform the previous steps in the DB2 UDB server, you are ready to debug your stored procedure. You must ensure that the IBM Distributed Debugger client is installed and started in the client workstation specified in the debugger table. To start the IBM Distributed Debugger client, you can use the following command:

```
idebug.exe -qdaemon -quiport=8000
```

You can invoke the stored procedure from any client workstation, using a client program, SPB, or the PCALL generic client program provided with DB2. When the stored procedure starts at the DB2 server, the debugging process begins in the client workstation.

For more information about the IBM Distributed Debugger client, refer to 3.4.6, "Debugging stored procedures" on page 105.

#### 5.6.5 Debugging stored procedures through SPB

Following is what needs to be done for debugging your stored procedure through the SPB, which provides an easy way to debug it:

- On the server side:
  1. Set the following: `db2set DB2ROUTINE_DEBUG=on`
- On the client side:
  1. Start the debugger daemon: `idebug.exe -qdaemon -quiport=8000`
  2. Start SPB, write your stored procedure, select the stored procedure, right-click -> "Debug Properties" -> "Add" (the values for the IP address are taken from the SPB machine) -> "OK"
  3. Run the stored procedure through the SPB.

The SPB will take care of creating the debugger table and inserting/deleting/updating the entries.



---

## Chapter 6. SQL Procedures for DB2 UDB for AS/400

This chapter describes the SQL Procedures language support available for DB2 UDB for AS/400. The SQL Procedures language was introduced in V4R2.

---

### 6.1 General Considerations

An SQL stored procedure is created with the CREATE PROCEDURE statement that includes a procedure body written in SQL or more precisely in SQL Procedures language. For SQL naming convention, the procedure will be created in the collection or library specified by the implicit or explicit qualifier. For system naming, the procedure will be created in the collection or library specified by the qualifier. If no qualifier is specified, the procedure will be created in the current library (\*CURLIB).

Stored procedures are automatically registered in the system catalog, when the procedure is created or restored onto another system. Client applications (ODBC, JDBC, ADO based) cannot invoke stored procedures unless they are registered in the database catalogs.

The DB2 UDB for AS/400 does not provide an SQL stored procedure statement debugger, so the ILE C program debugger must be used for any debug that is needed on the stored procedure program.

---

### 6.2 System requirements and planning

Before you start to develop the SQL stored procedures on the AS/400 system, make sure that you are running on V4R2 or a higher release of the OS/400 with the latest CUMPTF loaded.

When you execute the CREATE PROCEDURE statement for the SQL stored procedure, DB2 UDB for AS/400 walks through a multiphase process to create an ILE C program object (\*PGM). During this process DB2 UDB for AS/400 generates an intermediary ILE C code with embedded SQL statements. This ILE C code is then precompiled, compiled, and linked automatically. This means that the SQL Development Kit for AS/400, and the ILE C compiler, need to be installed on the system where you plan to develop SQL stored procedures. Once the ILE C object is created, it can be restored onto any V4R2 or higher system and run without the SQL Development Kit and ILE C compiler.

Please note that, for performance reasons, the ILE C program object is created with Activation Group parameter set to \*CALLER.

---

### 6.3 System Catalog Tables

The database catalog tables contain information about tables, parameters, procedures, packages, views, indexes, and constraints on the AS/400 system.

The database manager provides views over the catalog tables. The views provide more consistency with the catalog views of other IBM SQL products and with the catalog views of the ANSI and ISO standard. Tables and views in the catalog are the same as any other database tables and views. If you have the authorization,

you can use SQL statements to look at data in the catalog views in the same way that you retrieve data from any other table in the AS/400 system. The database manager ensures that the catalog contains accurate descriptions of the objects in the database at all times.

When you create an SQL stored procedure or an external procedure, there are two catalog tables in QSYS2 that are updated: SYSROUTINES and SYSPROCS.

The SYSPROCS table contains one row for each procedure created by the CREATE PROCEDURE statement. Some of the fields are:

- Name of the collection or library where the procedure is created
- Name of the procedure
- Type of routine body (External or SQL)
- Language of the procedure (SQL, C, CL, RPG...)
- Number of input parameters
- Number of output parameters
- Number of input-output parameters
- The source code of an SQL stored procedure

**Note:** If the source of the SQL stored procedure is more than 18K, the source code is not stored in SYSPROCS.

The SYSPARMS table contains one row for each parameter of a procedure created by the CREATE PROCEDURE statement. Some of the fields of this table are:

- Name of the collection or library where its created
- Name of the procedure
- Type of parameter (IN, OUT, INOUT)
- Name of the parameter
- Data type of the parameter
- Data scale of the parameter
- Data precision of the parameter

---

## 6.4 Creating an SQL stored procedure

In this section, we document the steps required to edit and compile an SQL stored procedure (SP). On the AS/400 system there are many different ways to build your SQL stored procedure. You can use following methods:

- Traditional 5250 programming using Source Entry Utility (SEU) and RUNSQLSTM utilities. This gives the best control over the compiler parameters and allows you to debug the ILE C program object.
- Operations Navigator GUI
- Operations Navigator SQL script utility

### 6.4.1 Creating an SQL SP with traditional tools

The steps required to create your SQL stored procedures with traditional 5250 tools are outlined in the following list:

- Create a library if you do not have one already.
- Create a source physical file; this is the file where all the SQL source members are going to be stored.

- Start a Source Entry Utility (SEU) editing session.
- Enter the SQL stored procedure source code.
- Create the SQL stored procedure using the RUNSQLSTM command to issue a CREATE PROCEDURE command. This creates a C program object that runs when the procedure is called. If there are problems generating the procedure, there is a listing that shows the syntax errors of the source.
- Invoke the stored procedure through the SQL CALL statement passing the parameter list.
- Check for the completion status of the SQL stored procedure.

Let's see how to implement this scenario. First, create a library, a source file, and start an editing session.

1. To create a library called SQLPROCS, type the following CL command at the 5250 emulation prompt:

```
CRTLIB LIB (SQLPROCS)
```

2. To create a source physical file called QSQLSRC, type the following command:

```
CRTSRCPF FILE(SQLPROCS/QSQLSRC) RCDLEN(112) TEXT(' Source physical file for SQL Procedures' )
```

The CRTSRCPF command creates a source physical file QSQLSRC in SQLPROCS library.

3. To start an editing session and create a source member, named SDK2LMS, type the following command:

```
STRSEU SRCFILE(ORDAPPLIB/QSQLSRC) SRCMBR(SDK2LMS) TYPE(TXT) OPTION(2)
```

Entering OPTION(2) indicates that you want to start a session for a new member. The STRSEU command creates a new member, SDK2LMS, in the QSQLSRC file in the SQLPROCS library and starts an edit session.

4. Use SEU to enter the procedure's source code as shown in Figure 78.

```

CREATE PROCEDURE SDK2LMS
  ( IN   empnum CHAR(6) 1
    INOUT rating SMALLINT )
LANGUAGE SQL
- The procedure's body begins here
BEGIN
  DECLARE not_found CONDITION FOR '02000';
  DECLARE EXIT HANDLER FOR not_found
    SET rating = -1;
  IF rating = 1
    THEN UPDATE employee
      SET salary = salary * 1.10, bonus = 1000
      WHERE empno = empnum;
  ELSEIF rating = 2
    THEN UPDATE employee
      SET salary = salary * 1.05, bonus = 500
      WHERE empno = empnum;
  ELSE UPDATE employee
      SET salary = salary * 1.03, bonus = 0
      WHERE empno = empnum;
  END IF;
END;

```

Figure 78. Entering source code

**Note: 1** We intentionally omitted a comma, which should separate the parameters to produce an error listing in the next step.

5. Run the RUNSQLSTM command to create the procedure. We recommend using the Debugging view option \*LIST and Listing output \*PRINT. It is useful for debugging and testing purposes. Refer to section 6.6.2, “Preparing the SQL stored procedure for debugging” on page 182 for more details.

```

                                Run SQL Statements (RUNSQLSTM)

Type choices, press Enter.

Source file . . . . . > QSQLSRC      Name
Library . . . . . > SQLPROCS      Name, *LIBL, *CURLIB
Source member . . . . . > SDK2LMS   Name
Commitment control . . . . . > *NONE *CHG, *ALL, *CS, *NONE...
Naming . . . . . > *SYS           *SYS, *SQL

                                Additional Parameters

Debugging view . . . . . > *LIST     *STMT, *LIST, *NONE
Listing output . . . . . > *PRINT    *NONE, *PRINT

                                                                Bottom
F3=Exit   F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display  F24=More keys

```

Figure 79. Creating the SQL stored procedure

6. If there are syntax errors in your source code, the SQL9010 'RUNSQLSTM command failed' message appears on your screen. To check for possible errors, you need to look at the spool file created by the precompiler. Type following CL command at the command prompt:

WRKSPLF

The list of your spool files appears. Find the spool file named SDK2LMS and User Data value SQL. To display the spool file contents, use option 5 as shown in Figure 80.

```
Work with All Spooled Files

Type options, press Enter.
 1=Send  2=Change  3=Hold  4=Delete  5=Display  6=Release  7=Messages
 8=Attributes  9=Work with printing status

Opt  File      User      Device or  User Data  Sts  Total  Cur
   5  SDK2LMS    JAREK    QPRINT    SQL        RDY   3     Page  Copy
                                           1

Parameters for options 1, 2, 3 or command
====>
F3=Exit  F10=View 4  F11=View 2  F12=Cancel  F22=Printers  F24=More keys

Bottom
```

Figure 80. Working with spool files

7. The SQL stored procedure listing appears. Scroll down and find the SQL messages section as shown in Figure 81.

```

Display Spooled File
File . . . . . : SDK2LMS
Page/Line 2/17
Control . . . . .
Columns 1 - 130
Find . . . . .

*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8..
..+...0...+...1...+...2...+...3
13 WHERE empno = empnum;
14 ELSEIF rating = 2
15 THEN UPDATE employee
16 SET salary = salary * 1.05, bonus = 500
17 WHERE empno = empnum;
18 ELSE UPDATE employee
19 SET salary = salary * 1.03, bonus = 0
20 WHERE empno = empnum;
21 END IF;
22 END;

***** END OF SOURCE *****
RCHASM20 - V04R04M00 - 471125
5769ST1 V4R4M0 990521 Run SQL Statements SDK2LMS
09/14/99 13:52:46 Page
Record *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..
SEQNBR Last change
MSG ID SEV RECORD TEXT
SQL0199 30 3 Position 11 Keyword INOUT not expected. Valid tokens: ) ,
.
SQL0104 30 8 Position 20 Token HANDLER was not valid. Valid tokens:
SCROLL.

More...

```

Figure 81. Displaying SQL precompiler error messages

8. In the preceding listing, there is a syntax error that probably generated the other ones. Correct the syntax error using the SEU utility and execute the RUNSQLSTM command again. This time the command should complete successfully.

After the procedure has been successfully created, two system catalog tables are updated: SYSROUTINES and SYSPARMS. The SYSROUTINES view contains one row for each procedure or User Defined Function. The SYSPARMS table contains one row for each parameter of a procedure or UDF. If you intend to work only with stored procedures, you can also use a catalog view called SYSPROCS, which presents information pertaining to the stored procedures. The system catalog includes also a view called SYSFUNCS, which shows the information for the UDFs.

Once the procedure has been created, it can be invoked with the SQL call statement using any interface that supports SQL (embedded SQL, ODBC, JDBC, SQLJ, CLI, and so on).

## 6.4.2 Creating an SQL SP with Operations Navigator GUI

The Operations Navigator provides an attractive graphical interface that allows you to perform typical database administration tasks. It allows easy access to all server administration tools, gives a clear overview of the entire database system,

enables remote database management, and provides assistance for complex tasks.

In this section, you will learn how to efficiently use the GUI administration tools offered by Client Access Express to work with SQL stored procedures on the AS/400 system. We assume that you already know how to set up the Operations Navigator connection to your AS/400.

The steps below show you how to create an SQL stored procedure using the Create New SQL Procedure dialog:

1. Double click the **Operations Navigator** icon on your desktop. In the main panel right-click on the library, which contains your database. In our case the name of the library is *SAMPLE*. Select **New ->Procedure-> SQL**. The New SQL Procedure dialog appears.
2. Enter the following for the stored procedure name: *SDK2LMS*.
3. For the description, type the following: *Increase salary depending on rating*.
4. Click on the **Parameters** tab.
5. Click the **Insert** button. For the first parameter name, type the following: *empnum*. From the type drop down list, select **CHARACTER**. In the parameter length box, enter the number '6'. Change the parameter style to **IN/OUT**.
6. Insert the second parameter as shown in Figure 82.

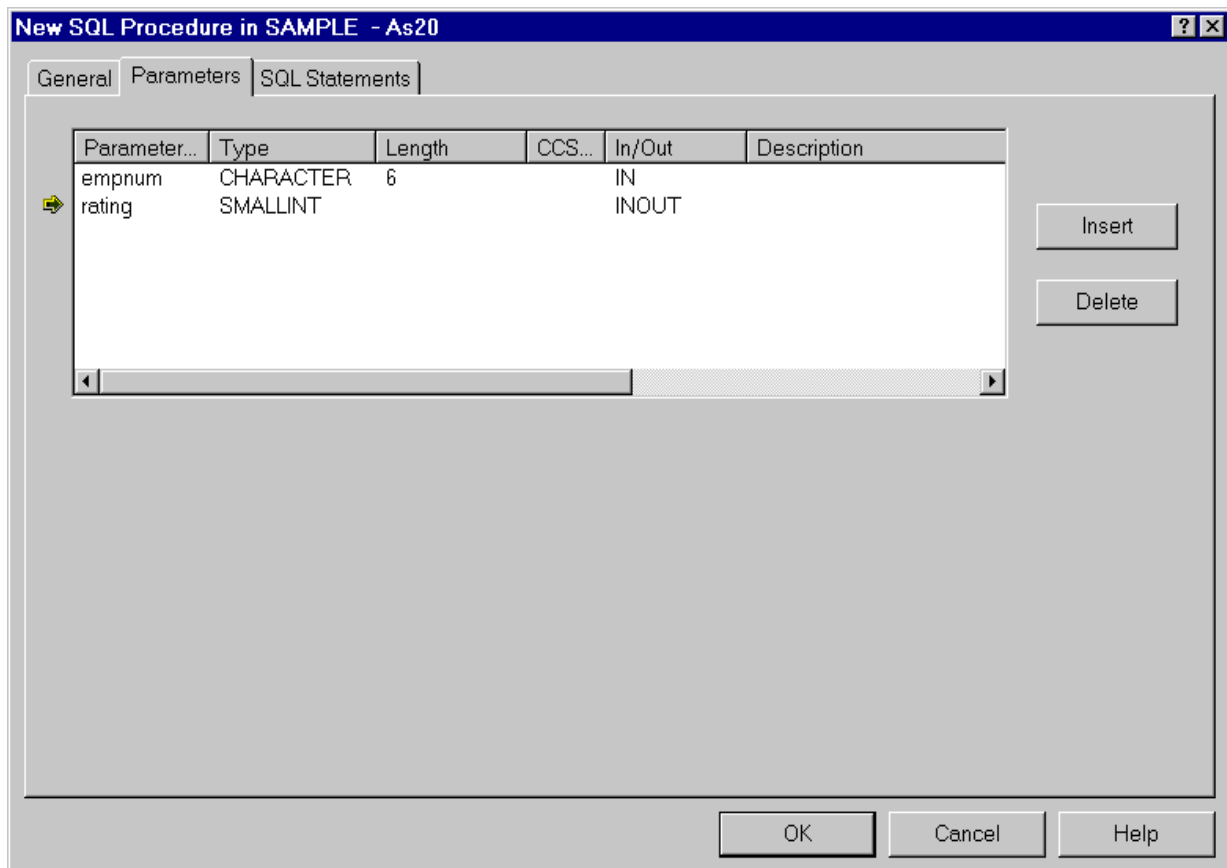


Figure 82. Parameters definition for SQL stored procedure

7. Click on SQL Statements tab. Type the procedure body as shown in Figure 83.

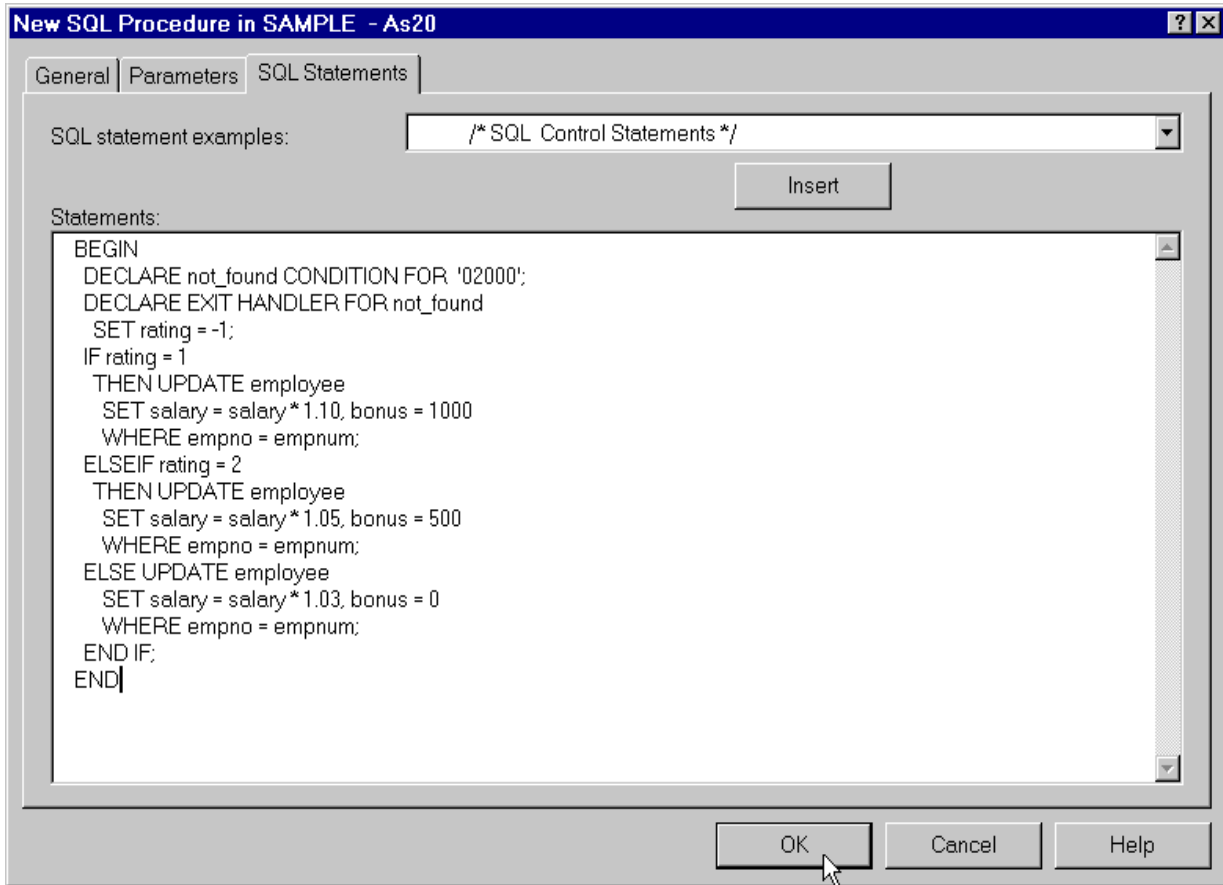


Figure 83. Entering SQL statements

8. Click the OK button. The stored procedure is now created.

### 6.4.3 Creating an SQL SP with the Run SQL Scripts utility

The Run SQL Script utility is yet another interface that you can use on the AS/400 system to create a stored procedure. The script utility is available through the Operations Navigator GUI. It allows you to you create, edit, run, and troubleshoot scripts of SQL statements. You can also save the scripts with which you work on your PC.

The steps below show you how to create an SQL stored procedure using the SQL Script utility.

1. Double click the **Operations Navigator** icon on your desktop. In the main panel right-click the Database object and select Run SQL Script. The Run SQL Scripts windows appears.
2. Type the procedure body as shown in Figure 84.



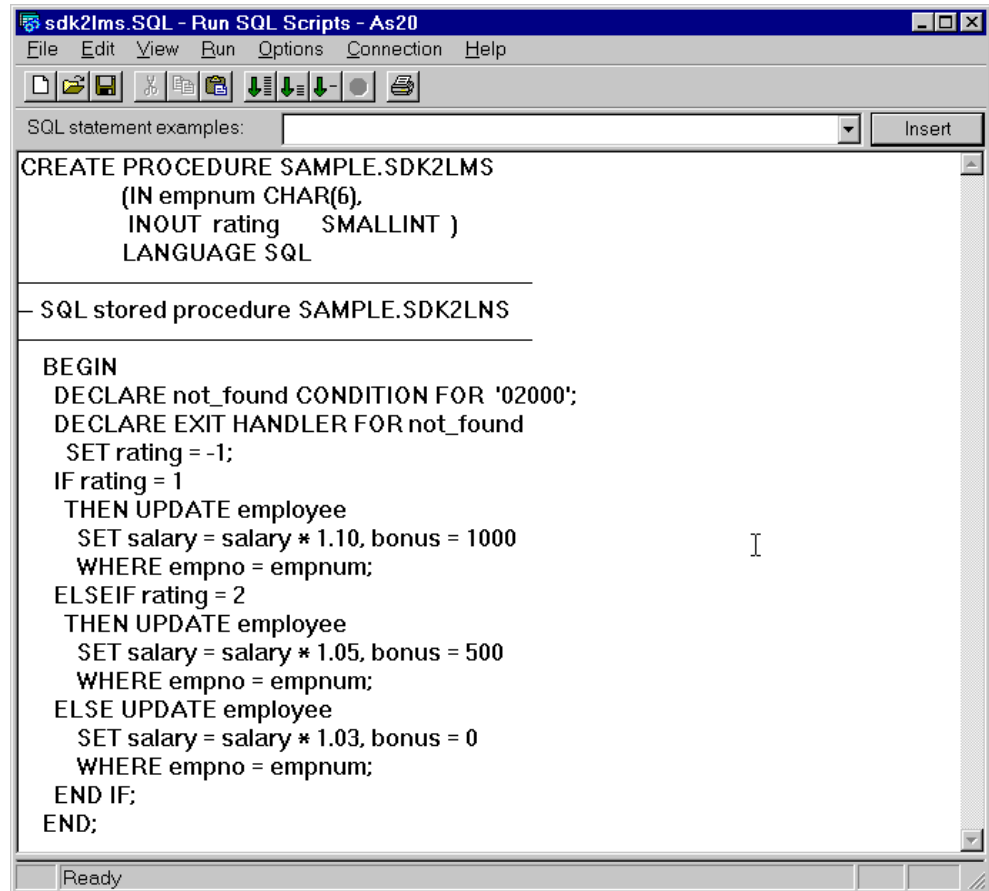


Figure 84. Creating SQL SP with script utility

- To run the CREATE PROCEDURE statement, select **Run->All** from the Run pull down menu. If the syntax of your SQL statement is correct, the SQL stored procedure is created in the SAMPLE library on your AS/400 system. Check for the completion status in the run history panel of the Run SQL Script window. The last message displayed in this panel should read:

**Statement ran successfully**

If the run history panel does not supply sufficient information about the execution of the SQL statements, you can view the AS/400 job log to get additional, more specific information. From the **View** drop down menu, select **Job Log**. A job log window appears as shown in Figure 85.

Message ID	Message	Date sent	Time sent
CPF9862	Member QAUGDBJOB added to output file QAUGDBJOB in library QTEMP.	09/25/99	11:03:59
CPF9861	Output file QAUGDBJOB created in library QTEMP.	09/25/99	11:03:58
CPC5D07	Program SDK2LMS created in library SAMPLE.	09/25/99	10:19:58
CQM0607	Module SDK2LMS was created in library SAMPLE on 09/25/99 at 10:19:56.	09/25/99	10:19:56
CQM0614	Warnings were issued during compilation.	09/25/99	10:19:54
CPD4090	Printer device QPRINT not found. Output queue changed to QPRINT in library QGPL.	09/25/99	10:19:51
CPC7305	Member SDK2LMS added to file QSQLTEMP in QTEMP.	09/25/99	10:19:50
CPC7301	File QSQLTEMP created in library QTEMP.	09/25/99	10:19:50
CPC7305	Member SDK2LMS added to file QSQLSRC in QTEMP.	09/25/99	10:19:49
CPC7303	File QSQLSRC in library QTEMP changed.	09/25/99	10:19:49
CPC7301	File QSQLSRC created in library QTEMP.	09/25/99	10:19:49
CPIAD12	Servicing user profile JAREK from client 9.5.62.51.	09/25/99	10:07:26
CPIAD02	Servicing user profile JAREK.	09/25/99	10:07:26
CPD0912	Printer device QPRINT not found.	09/25/99	10:07:26
CPF1124	Job 064153/QUUSER/QZDASOINIT started on 09/25/99 at 09:35:09 in subsystem QSERVER in QSYS. Jo	09/25/99	09:35:09

Figure 85. Job log window

**Note:** You may not have the same messages in the job log.

To view a second level message in the job log, double click on the item you wish to view. A dialog window appears with all of the information for that message.

To save the script that contains the source code for the SDK2LMS stored procedure, select **File->Save As** from the script utility menu bar. The Save As dialog is displayed. In the Save in list combo, open the directory you wish to use as your SQL script repository. In our case we use *d:\sg24\_5485\work\_in\_progress* directory. Enter *sdk2lms* in the file name input field.

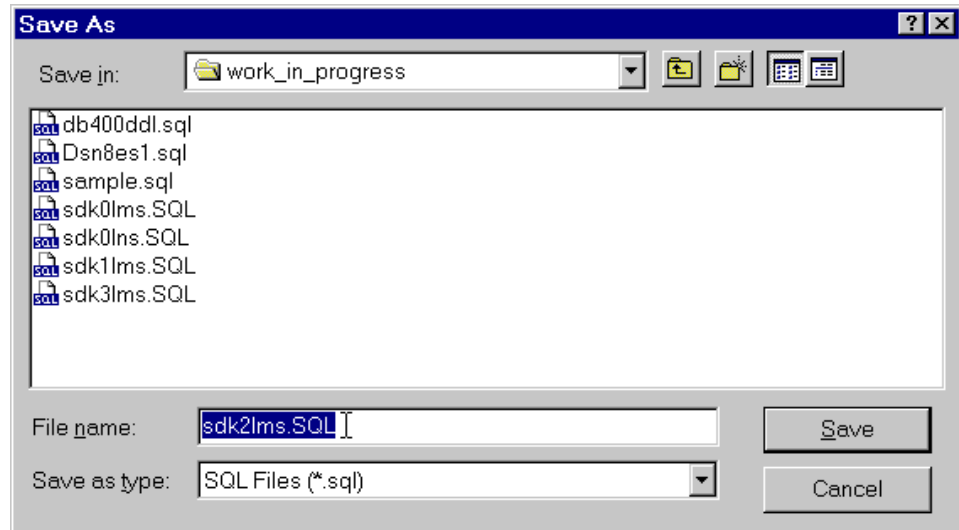


Figure 86. Saving SQL Script File

Click **Save** to return to the Run SQL Script dialog.

The Run SQL Script utility proved to be very useful when we ported the SQL stored procedures from other DB2 UDB platforms to the AS/400. All we had to do was to copy the scripts to our working directory, and change the file extension from .stp to .sql. Then we could double click a stored procedure file from the Windows Explorer window to load the script into the Run SQL Script utility.

#### 6.4.4 Verifying the stored procedure properties

Once the stored procedure has been successfully created, you can verify its properties by using the Operations Navigator interface:

1. In the main Operations Navigator window double click the SAMPLE library icon. The right-hand panel displays now all DB2 UDB for AS/400 objects in this library.
2. Find the SDK2LMS stored procedure icon and right-click it. The context menu for this object appears. Select Properties. The SDK2LMS Properties window shows up. It has three tabs:
  - The **General** page specifies the name by which the procedure is known to SQL programs and the number of result sets it should return. If you want to call an external program as a procedure, you need to define the program as a procedure before you can call it from an SQL program.
  - The **Parameters** page specifies the parameters that the procedure uses.
  - The **SQL Statements** page contains the code for the external SQL program that you are defining as a procedure. You can use the SQL statement examples and fill in the necessary information to make coding SQL easier. After an SQL stored procedure has been created, the SQL statements cannot be changed.

The Parameters page for the SDK2LMS stored procedure is shown in Figure 87.

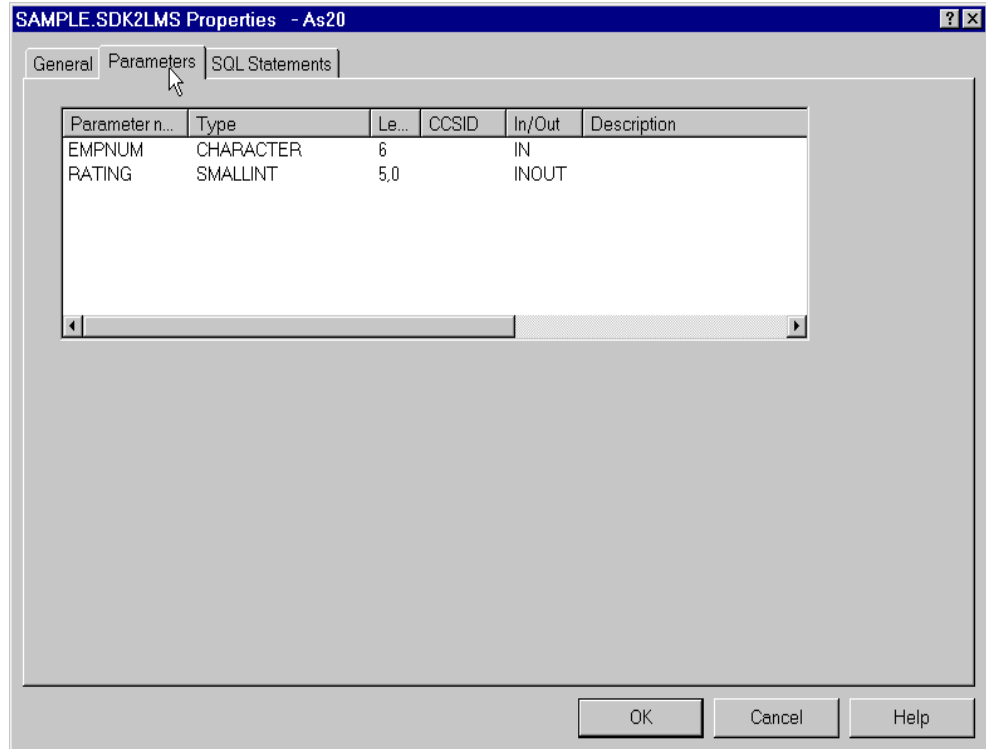


Figure 87. Displaying the stored procedure properties

## 6.5 Deleting or replacing the SQL stored procedure

When you create a procedure, its library and name must be unique to register the SQL stored procedure in the catalogs. However, the CREATE PROCEDURE statement does not have a replace option. For this reason, if you want to re-create or delete an existing procedure, use the DROP PROCEDURE statement. If you try to create a stored procedure that already exists in a given library, you will receive an error return code SQL0454.

For example, when we tried to re-run the CREATE PROCEDURE statement for the SDK2LMS stored procedure the following error message was displayed in the run history panel of the SQL Script utility:

**SQL0454 - 'Function SDK2LMS in SAMPLE with the same signature already exists'.**

There are several ways to drop a stored procedure from the AS/400 system:

- In the traditional "green screen" environment, start the interactive SQL session with the STRSQL command, and at the ISQL prompt, type the following SQL statement:  
`DROP PROCEDURE library/procedure-name`
- In the Operations Navigator environment, in the right panel of the main Operations Navigator window, right-click the procedure you want to drop and select the **Delete** option. A window appears with the stored procedure object selected for deletion. Confirm that this is the procedure you want to delete, and click the **Delete** button, as shown in Figure 88.

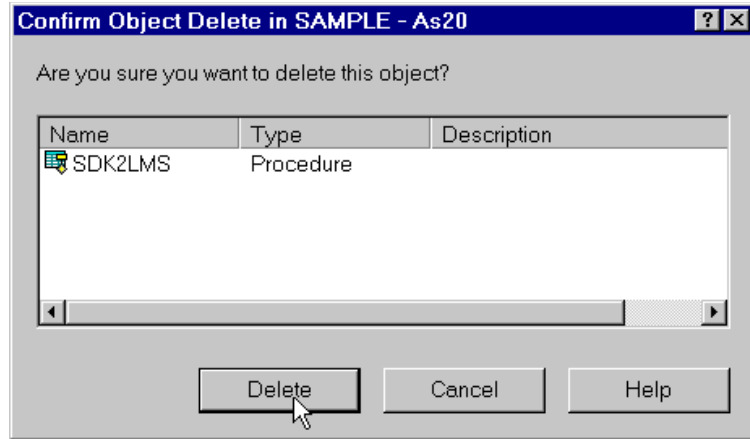


Figure 88. Deleting a stored procedure

- In the Run SQL Script utility, insert the `DROP PROCEDURE procedure-name` statement in the workable area and then select **Run->All** from the menu bar.

The system catalog tables, SYSROUTINES and SYSPARMS, are updated when a `DROP PROCEDURE` statement is executed. In the SYSROUTINES table, a row is deleted corresponding to the information of the deleted procedure. In SYSPARMS table, the number of rows deleted depends on the number of parameters defined in the procedure.

## 6.6 Debugging SQL stored procedures

In this section we show how to debug an SQL stored procedure on the AS/400 system. DB2 UDB for AS/400 does not provide a native SQL debugger, so the processes of eliminating run-time errors requires a certain level of programming skills in the AS/400 Integrated Language Environment (ILE). As discussed in 6.2, “System requirements and planning” on page 169, when you create an SQL stored procedure, under the covers, the system is creating an ILE C program object, which implements the procedure. The ILE C programs, in turn can be debugged with the ILE Source Debugger. We start this section with a brief description of the basic ILE Source Debugger functions.

### 6.6.1 The ILE Source Debugger

The ILE source debugger is used to detect errors in and eliminate errors from program objects and service programs. By using debug commands with any ILE program, you can:

- View the program source or change the debug view
- Set and remove conditional and unconditional breakpoints
- Step through a specified number of statements
- Display or change the value of fields, structures, and arrays
- Equate a shorthand name with a field, expression, or debug command

Many debug commands are available for use with the ILE source debugger. These debug commands and their parameters are entered on the debug command line displayed in the bottom of the Display Module Source display and the Evaluate Expression display. These commands can be entered in uppercase, lowercase, or mixed case.

**Note:** The debug commands on the debug command line are not CL commands.

The most important debug commands are briefly described in the following list:

<b>Command</b>	<b>Description</b>
<b>ATTR</b>	Permits you to display the attributes of a variable. The attributes are the size and type of the variable.
<b>BREAK</b>	Permits you to enter either an unconditional or conditional breakpoint at a position in the program being tested. Use BREAK line-number WHEN expression to enter a conditional breakpoint.
<b>CLEAR</b>	Permits you to remove conditional and unconditional breakpoints.
<b>DISPLAY</b>	Allows you to display the names and definitions assigned by using the EQUATE command.
<b>EQUATE</b>	Allows you to assign an expression, variable, or debug command to a name for shorthand use.
<b>EVAL</b>	Allows you to display or change the value of a variable or to display the value of expressions, records, structures, or arrays.
<b>QUAL</b>	Allows you to define the scope of variables that appear in subsequent EVAL commands.
<b>STEP</b>	Allows you to run one or more statements of the procedure being debugged.
<b>FIND</b>	Searches forwards or backwards in the module currently displayed for a specified line number or string or text.
<b>UP</b>	Moves the displayed window of source towards the beginning of the view number of lines entered.
<b>DOWN</b>	Moves the displayed window of source towards the end of the view number of lines entered.
<b>LEFT</b>	Moves the displayed window of source to the left.
<b>RIGHT</b>	Moves the displayed window of source to the right by the number of characters entered.
<b>TOP</b>	Positions the view to show the first line.
<b>BOTTOM</b>	Positions the view to show the last line.
<b>NEXT</b>	Positions the view to the next breakpoint in the source currently displayed.
<b>PREVIOUS</b>	Positions the view to the previous breakpoint in the source displayed.
<b>HELP</b>	Shows the online help information for the available source debugger commands.

### 6.6.2 Preparing the SQL stored procedure for debugging

A program or module must have debug data available if you are to debug it. Since debug data is created during compilation, you need to specify the DBGVIEW parameter on the RUNSQLSTM command. The DBVIEW parameter specifies the type of source debug information to be provided by the SQL precompiler. The default value for this parameter is \*NONE, so no debugging information is included in the program object.

To create the SQL stored procedure with the debug data follow the steps outlined below:

1. At the CL command prompt type RUNSQLSTM and press F4 for prompting. Provide the source file name, library, and source member as shown in Figure 89 for our SDK2LMS example.

```

Run SQL Statements (RUNSQLSTM)

Type choices, press Enter.

Source file . . . . . > QSQLSRC      Name
Library . . . . . > SQLPROCS      Name, *LIBL, *CURLIB
Source member . . . . . > SDK2LMS  Name
Commitment control . . . . . *CHG   *CHG, *ALL, *CS, *NONE...
Naming . . . . . *SYS              *SYS, *SQL

Additional Parameters

Severity level . . . . . 10         0-40
Date format . . . . . *JOB         *JOB, *USA, *ISO, *EUR...
Date separator character . . . . *JOB   *JOB, /, ., ,, -, ' ', *BLANK
Time format . . . . . *HMS        *HMS, *USA, *ISO, *EUR, *JIS
Time separator character . . . . *JOB   *JOB, :, ., ,, ' ', *BLANK
Default collection . . . . . *NONE   Name, *NONE
IBM SQL flagging . . . . . *NOFLAG  *NOFLAG, *FLAG
ANS flagging . . . . . *NONE       *NONE, *ANS

More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 89. RUNSQLSTM command

2. Press the Page Down key to scroll to the DBGVIEW parameter. Set the parameter value to \*LIST as shown in Figure 90. We also recommend that you set the OUTPUT parameter to \*PRINT. The OUTPUT parameter specifies whether the precompiler listing is generated.

**Note:** In our example we use the system naming convention, which gives much more naming flexibility on the AS/400 system than the SQL naming convention. In the SDK2LMS SQL source, we did not qualify the procedure name with a library name, so it is going to be created in the current library. If you want the stored procedure to be created in the SAMPLE library, make sure it is your current library at the time you run the RUNSQLSTM command. You can use the Display Library List (DSPLIBL) command to display your library list, and the Change Current Library (CHGCURLIB) command to change the current library for your AS/400 job.

```

Run SQL Statements (RUNSQLSTM)

Type choices, press Enter.

Decimal Point . . . . . *JOB      *JOB, *SYSVAL, *PERIOD...
Sort sequence . . . . . *JOB      Name, *HEX, *JOB...
  Library . . . . .      Name, *LIBL, *CURLIB
Language id . . . . . *JOB      *JOB, *JOBRUN...
Print file . . . . . QSYSVRT   Name
  Library . . . . .      *LIBL   Name, *LIBL, *CURLIB
Statement processing . . . . . *RUN    *RUN, *SYN
Allow copy of data . . . . . *OPTIMIZE *OPTIMIZE, *YES, *NO
Close SQL cursor . . . . . *ENDACTGRP *ENDMOD, *ENDACTGRP
Allow blocking . . . . . *ALLREAD  *ALLREAD, *NONE, *READ
Delay PREPARE . . . . . *NO       *YES, *NO
Debugging view . . . . . > *LIST    *STMT, *LIST, *NONE
User profile . . . . . *NAMING  *NAMING, *USER, *OWNER
Dynamic user profile . . . . . *USER    *USER, *OWNER
Listing output . . . . . > *PRINT   *NONE, *PRINT
Target release . . . . . *CURRENT *CURRENT, VxRxMx

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 90. Specifying the DBGVIEW and OUTPUT parameters

The possible values for the *Debugging view* parameter are:

- \*STMT**        Allows the compiled module object to be debugged using program statement numbers and symbolic identifiers.
- \*NONE**        The debug view is not be generated.
- \*LIST**        Generates the listing view for debugging the compiled module object.

The possible values for the Listing output parameter are:

- \*PRINT**        The precompiler listing is generated.
- \*NONE**        The precompiler listing is not generated.

You must specify *\*STMT* or *\*LIST* if you want debugging data to be saved in the program. After the RUNSQLSTM command has successfully created the procedure, we are ready to test it.

### 6.6.3 Testing the SQL stored procedure in traditional environment

As you have probably realized by now, the SQL control statements do not include the PRINT or DISPLAY statements. Therefore, the easiest way to test the execution of the procedure is to use ILE C source code debugging.

While debugging and testing your program, ensure that your library list is changed to direct the programs to a test library containing the test data so that any existing real data is not affected.

To start a debugging session, type the SRTDBG command at the CL prompt and press F4 for prompting. Provide the program name and the library. Make sure that you change the Update production files parameter to *\*YES*. Even if you work with the test data the library attribute is set to PROD, and your procedure will fail miserably the first time you try to access the data. An example of the STRDBG command is shown in Figure 91.



```

                                Start Debug (STRDBG)

Type choices, press Enter.

Program . . . . . > SDK2LMS      Name, *NONE
Library . . . . . >  SAMPLE      Name, *LIBL, *CURLIB
      + for more values
                                *LIBL
Default program . . . . . *PGM    Name, *PGM, *NONE
Maximum trace statements . . . . . 200      Number
Trace full . . . . . *STOPTRC    *STOPTRC, *WRAP
Update production files . . . . . > *YES    *NO, *YES
OPM source level debug . . . . . *NO     *NO, *YES
Service program . . . . . *NONE      Name, *NONE
  Library . . . . .              Name, *LIBL, *CURLIB
      + for more values

More...

F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display  F24=More keys

```

Figure 91. Starting a debug session

**Note:** When your session is in debug mode, the job log of the session saves a lot of information related to the SQL statements being executed. The application developer can use this information for problem detection and performance tuning.

Once you have filled all the required parameters, press Enter to initialize the debug session. The ILE Source Debugger loads the ILE C source created for your SQL stored procedure. At the debug prompt, type the following command: `find main` and hit ENTER. This positions you at the main function as shown in Figure 92. Now you can set a breakpoint. It always a good idea to check, at the beginning of a stored procedure execution, whether the parameters were passed correctly, so set the breakpoint at line 110. You are all set now — just press F12 to return to the command line prompt.

```

Display Module Source
Program:  SDK2LMS      Library:  SAMPLE      Module:  SDK2LMS
101      void main(int argc, char* argv[]) {
102          1 SQLP_IND = (short int*) argv[3];
103          2 sqlcap = (SQLCA*) argv[4];
104          3 SQLInitSQLCA((SQLCA*)&sqlca);
105          4 SDK2LMS.SQLP_I1 = *(SQLP_IND+0);
106          5 if (SDK2LMS.SQLP_I1 != SQLP_NULLIND)
107          6 strcpy(SDK2LMS.EMPNUM, argv[1]);
108          7 SDK2LMS.SQLP_I2 = *(SQLP_IND+1);
109          8 if (SDK2LMS.SQLP_I2 != SQLP_NULLIND)
110          9 SDK2LMS.RATING = * (short *) argv[2];
111      10 for ( ; ; ) {
112          11 sqlca.sqlcaid[6] = 0x00;
113          12 SQLP_RC1 = 0;
114          13 if (SQLP_RC1 != -1 &&
115          14 if (SQLP_RC1 != -1 &&
More...
Debug . . .

F3=End program  F6=Add/Clear breakpoint  F10=Step  F11=Display variable
F12=Resume      F17=Watch variable  F18=Work with watch  F24=More keys

```

Figure 92. Debug session

The next step in the stored procedure testing is to actually invoke the procedure. As mentioned earlier, you cannot invoke the stored procedure from the command prompt; you need to use the SQL call.

To test the SDK2LMS stored procedure, we coded a small embedded SQL ILE C program that calls the procedure and displays the results. The source code for the INVSDK2LMS is shown in Figure 93. You can compile this program with the following CL command:

```

CRTSQLCI OBJ (SQLPROCS/INVSDK2LMS) SRCFILE (SQLPROCS/QCSRC)
OBJTYPE (*PGM) OUTPUT (*PRINT) DBGVIEW (*SOURCE)

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
    char Employee_Number??( 6 ??);
    short Rating;
EXEC SQL END DECLARE SECTION;

void main( int argc, char **argv )
{
    /* copy the parameters to the host variables */
    strcpy(Employee_Number, argv??(1??));
    Rating = (short)*argv??( 2 ??);

    /* any sql errors at the call time? */
    EXEC SQL WHENEVER SQLERROR GOTO badnews;

    EXEC SQL CALL SAMPLE/SDK2LMS( :Employee_Number,
                                   :Rating );

    if( Rating != -1)
        printf("Stored Procedure ran successfully...\n");
    else
        printf("Error in Stored Procedure.\n");
        exit(0);

    badnews:
    printf( "Error occurred in invoking program. SQLCODE = %5d\n", SQLCODE);
    exit(1);
}

```

Figure 93. INVSDK2LMS source code

To run the INVSDK2LMS program, call the program from the command prompt, passing two required parameters as shown below:

```
CALL PGM(SQLPROCS/INVSDK2LMS) PARM('000010' 1)
```

The INVSDK2LMS program, in turn invokes the stored procedure and passes the control to it. The stored procedure hits the breakpoint, and the debugger session is activated. On the debugging line, you can enter any of the debug commands. In this way, you can display the content of any variable, check the SQL return code and so on. You can also step through the program using the F10 key.

Since your session is in debug mode, the job log has all the messages related to the execution of the procedure. We highly recommend that, while developing stored procedures, you always check the joblog messages inserted by the DB2 UDB for AS/400 optimizer.

**Note:** If your stored procedure is defined with only IN parameters and it does not return any results sets, you can test it very easily using the Interactive SQL. Let's suppose you created a stored procedure called *setSalary*, which takes two input parameters: *employee\_number* of type char(6) and *salary* of type decimal(11,2). You could test this procedure from the ISQL session by typing the following statement:

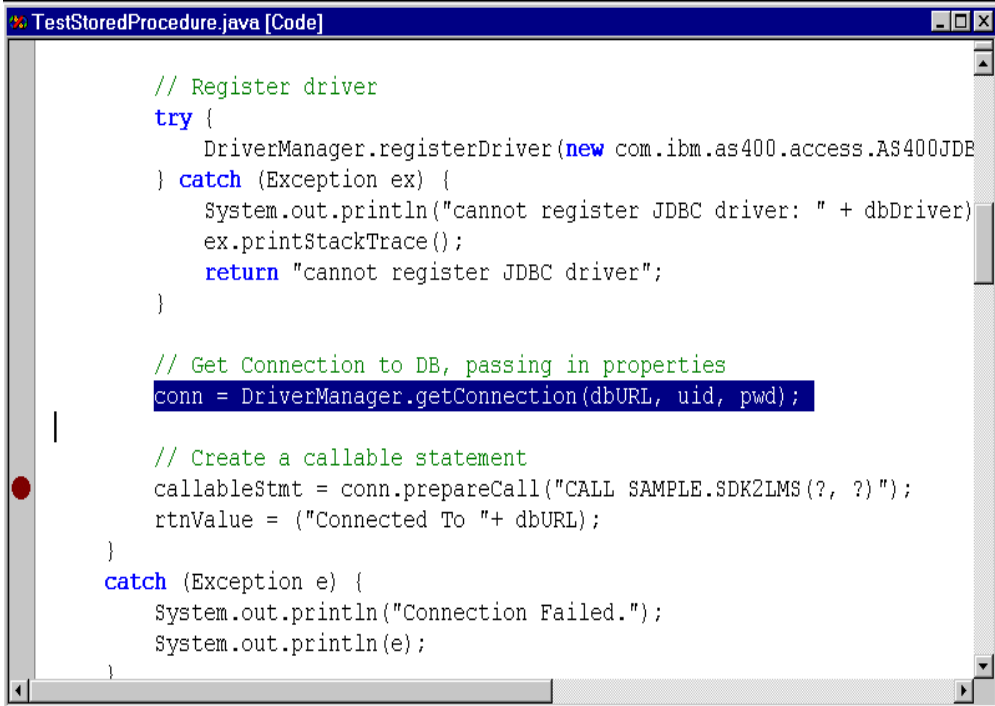
```
CALL setSalary ('000010', 65000.00)
```

## 6.6.4 Testing the SQL stored procedure in client/server environment

Testing and debugging of SQL stored procedures in the client/server environment maybe a little bit more tricky than in the traditional AS/400 environment. In this section we will outline the steps required to debug an SQL stored procedure called from the Java client. The combination of Java running on the client and SQL running on a powerful database server like the AS/400 can result in a highly scalable and robust software solution.

In our test scenario, we coded a Java client, which uses the AS/400 JDBC driver, to send the SQL request to DB2 UDB for AS/400. In the AS/400 client/server architecture, a JDBC client communicates with a corresponding AS/400 server job, which runs the SQL requests on behalf of this client. In other words, when we call a stored procedure from the Java client, there is an AS/400 server job that actually invokes the stored procedure on the server and then passes back the results to the client. The AS/400 server jobs associated with the database access are named QZDASOINIT, and run in the QSERVER subsystem. At any given time, there maybe a large number of database server jobs active in the QSERVER subsystem, so the first step in our debug procedure is to find the server job, which serves our client. The Java client code we used to debug the SDK2LMS stored procedure is listed in section 6.6.4.1, "Java client calling the stored procedure on the AS/400 server" on page 190.

1. Start the Java code in debug mode and set the breakpoint at the line just below the invocation of the *getConnection* method, as shown in Figure 94.



```
TestStoredProcedure.java [Code]

// Register driver
try {
    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDB
} catch (Exception ex) {
    System.out.println("cannot register JDBC driver: " + dbDriver)
    ex.printStackTrace();
    return "cannot register JDBC driver";
}

// Get Connection to DB, passing in properties
conn = DriverManager.getConnection(dbURL, uid, pwd);

// Create a callable statement
callableStmt = conn.prepareCall("CALL SAMPLE.SDK2LMS(?, ?)");
rtnValue = ("Connected To " + dbURL);
}
catch (Exception e) {
    System.out.println("Connection Failed.");
    System.out.println(e);
}
```

Figure 94. Running Java client

**Note:** The AS/400 server job is assigned to your client *after* the connection was established; that is why you need to set the breakpoint below the *getConnection* method invocation.

- Switch to the AS/400 session. To find the QZDASOINIT job serving your Java client, run the following CL command:

```
WRKOBJLCK OBJ(TEAMXX) OBJTYPE(*USRPRF)
```

where `TEAMXX` is the user profile you use to log into the AS/400 system.

The Work with Object Locks dialog appears. There should be one job named QZDASOINIT listed. Type 5 in the Option field next to this job, as shown in Figure 95, and hit Enter.

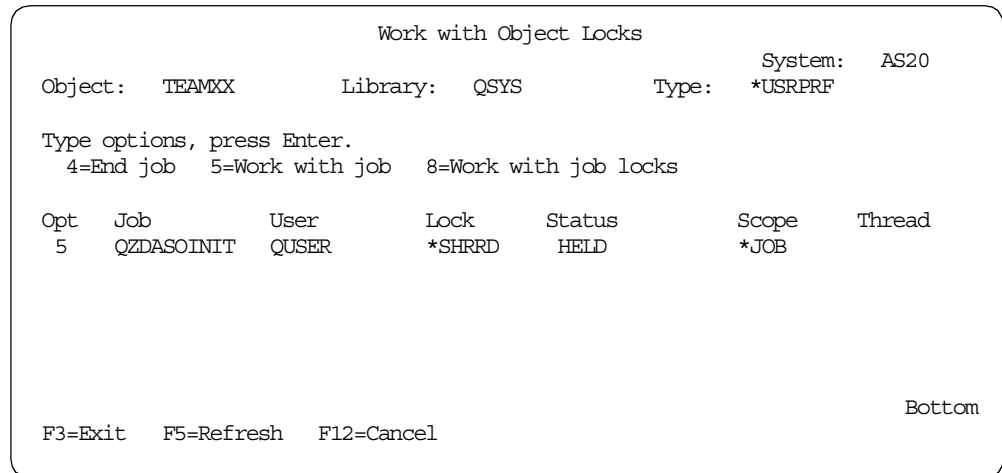


Figure 95. Finding the database server job

On the Work with Job dialog, select Option 10 'Display job log, if active or on job queue'. The Display Job Log screen appears. Find the first message in the joblog and write down the fully qualified job name for your database server job, as shown in Figure 96.

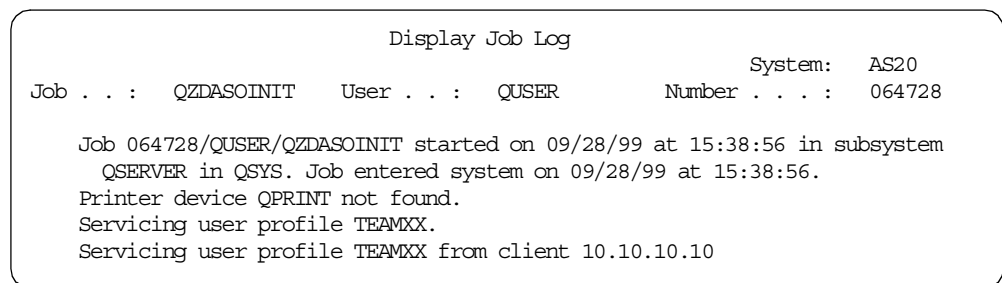


Figure 96. Job log for a database server job

In our case, the fully qualified name is: `064728/QUSER/QZDASOINIT`.

- Return to the command prompt and run following CL command:

```
STRSRVJOB JOB(064728/QUSER/QZDASOINIT )
```

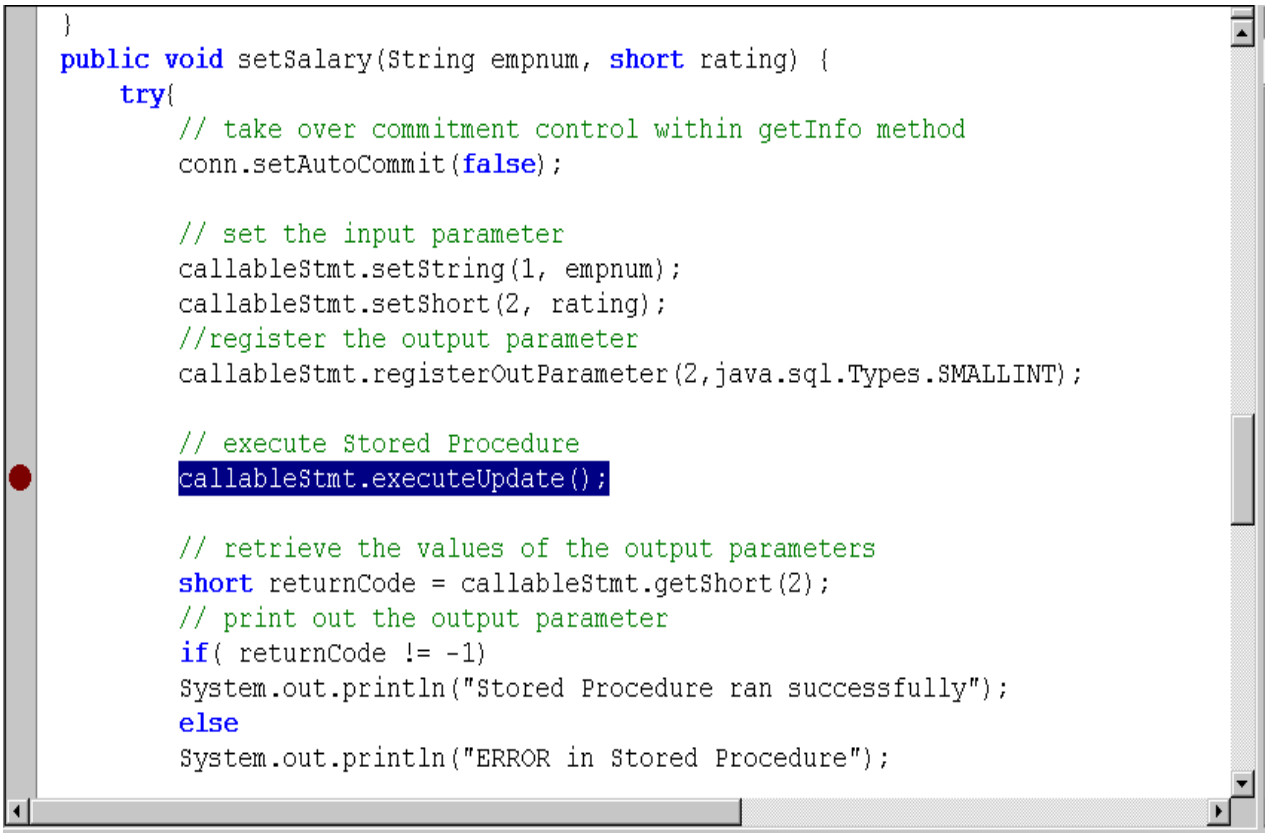
**Note:** The Start Service Job (STRSRVJOB) command starts the remote service operation for a specified job so that other service commands can be entered to service the specified job. Any dump, debug, and trace commands can be run in that job until service operation ends.

4. Start the ILE C Source Debugger for your server job with the following CL command:

```
STRDBG PGM(SAMPLE/SDK2LMS) UPDPROD(*YES)
```

The ILE Source Debugger loads the ILE C source created for your SQL stored procedure. Set the breakpoint and return to the command line.

5. Switch back to the Java client session and set the breakpoint at the statement that calls the stored procedure on the AS/400 system, as shown in Figure 97.

A screenshot of a Java IDE window showing a code editor. The code is a Java method named setSalary. A red dot on the left margin indicates a breakpoint is set on the line callableStmt.executeUpdate();. The code includes comments and uses JDBC CallableStatement for database interaction.

```
}  
public void setSalary(String empnum, short rating) {  
    try{  
        // take over commitment control within getInfo method  
        conn.setAutoCommit(false);  
  
        // set the input parameter  
        callableStmt.setString(1, empnum);  
        callableStmt.setShort(2, rating);  
        //register the output parameter  
        callableStmt.registerOutParameter(2, java.sql.Types.SMALLINT);  
  
        // execute Stored Procedure  
        callableStmt.executeUpdate();  
  
        // retrieve the values of the output parameters  
        short returnCode = callableStmt.getShort(2);  
        // print out the output parameter  
        if( returnCode != -1)  
            System.out.println("Stored Procedure ran successfully");  
        else  
            System.out.println("ERROR in Stored Procedure");  
    }  
}
```

Figure 97. Calling the stored procedure from Java

Run the statement at the breakpoint. The execution of the client code is now suspended, since the control was passed to the stored procedure on the AS/400 system.

6. Switch to the AS/400 session. The ILE C Source Debugger was activated, and you can step through your stored procedure on the server. Run the procedure to the completion. The control returns to the client, and you can continue to work with the Java code.

#### 6.6.4.1 Java client calling the stored procedure on the AS/400 server

The following example Java program shows you how to use the AS/400 JDBC driver to connect to the AS/400 system and call a stored procedure. It also teaches you how to bind IN and INOUT parameters to the *CallableStatement* class. You need to pass two arguments: employee's number and rating, to the program, when calling it from the command line, as shown in the example below:

```
java TestStoredProcedure 000010 1
```

**Note:** Make sure the host server is up and running on the AS/400 system and that jt400.zip is in your classpath on the client. Refer to "AS/400 Toolbox for Java Setup Guide", SC41-5438 for more details.

```
import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.as400.access.*;
import java.math.*;

class TestStoredProcedure {

    // declaration of instance vars
    private Connection conn;
    private CallableStatement callableStmt;

public String connectToDB( ) {

    String dbDriver = null;
    String dbURL = null;
    String rtnValue = null;
    String uid = null;
    String pwd = null;

    try {
        //Retrieve driver name and url from config.properties file

        dbDriver = "com.ibm.as400.access.AS400JDBCdriver";
        dbURL = "jdbc:as400://as20";
        uid = "TEAMXX";
        pwd = "TEST26T";

        // Register driver
        try {
            DriverManager.registerDriver(new
                com.ibm.as400.access.AS400JDBCdriver());
        } catch (Exception ex) {
            System.out.println("cannot register JDBC driver: " + dbDriver);
            ex.printStackTrace();
            return "cannot register JDBC driver";
        }

        // Get Connection to DB, passing in properties
        conn = DriverManager.getConnection(dbURL, uid, pwd);

        // Create a callable statement
        callableStmt = conn.prepareCall("CALL SAMPLE.SDK2LMS(?, ?)");
        rtnValue = ("Connected To "+ dbURL);
    }
    catch (Exception e) {
        System.out.println("Connection Failed.");
        System.out.println(e);
    }

    return (rtnValue);
}

public void dispose() {
```

```

try {
    // close the the statement, the lastly the connection

    if (null != callableStmt) {
        callableStmt.close();
    }
    if (null != conn) {
        conn.close();
    }
    System.exit(0);
}
catch (Exception e) {
    System.out.println("Error while closing...");
    System.out.println(e);
}
}

public void setSalary(String empnum, short rating) {
    try{
        // take over commitment control within getInfo method
        conn.setAutoCommit(false);

        // set the input parameters
        callableStmt.setString(1, empnum);
        callableStmt.setShort(2, rating);
        //register the output parameter
        callableStmt.registerOutParameter(2, java.sql.Types.SMALLINT);

        // execute Stored Procedure
        callableStmt.executeUpdate();

        // retrieve the values of the output parameter
        short returnCode = callableStmt.getShort(2);
        // print out the result of the call
        if( returnCode != -1)
            System.out.println("Stored Procedure ran successfully");
        else
            System.out.println("ERROR in Stored Procedure");

        // commit and give back commitment control
        conn.commit();
        conn.setAutoCommit(true);
    }
    catch (Exception e) {
        System.out.println("Error while retrieving information from " +
            "Database.");
        System.out.println(e);
        try {
            // error -> rollback and give back commitment control
            conn.rollback();
            conn.setAutoCommit(true);
        } catch (Exception e2) {
            System.out.println("Error in rollback");
            System.out.println(e2);
        }
    }
}
}

```



```

        return;
    }
    /**
     * Starts the application.
     * @param args an array of command-line arguments
     */
    public static void main(java.lang.String[] args) {

        String empnum = args[0];
        short rating = new Short(args[1]).shortValue();
        String connect = null;
        TestStoredProcedure jsp = new TestStoredProcedure();

        try {
            //connect to Database
            connect = jsp.connectToDB();
            System.out.println(connect);

            //Retrieve information from Database
            jsp.setSalary(empnum, rating);

            //clean up
            jsp.dispose();
        }
        catch (Exception e){
            System.out.println(e);
        }
    }
}

```



---

## Appendix A. Sample SQL stored procedure programs

This appendix shows examples illustrating SQL stored procedures for each DB2 release. The same samples were executed against OS/390, NT, and AIX. The sample SQL stored procedure programs illustrate the theory discussed in this redbook and are useful for getting started with the SQL Procedures language in your own environment and gaining some hands-on experience, whatever platform you have.

---

### A.1 Naming convention

We used the following naming convention shown in Figure 98 for these sample SQL stored procedures:

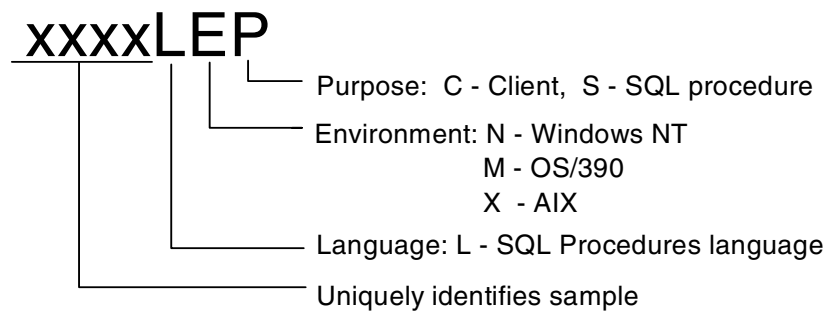


Figure 98. Naming convention for samples

The only exception is the sample called DSN8ES1, which will be a sample included in the delivery of the PTF for SQL Procedures language support for OS/390.

The samples beginning with SDK are included when you install the DB2 Software Developer's Kit (SDK) on the workstation platform. They have been customized for use in the OS/390 platform.

---

### A.2 OS/390 samples

#### A.2.1 DSN8ES1

This sample shows the use of: cursors, handlers, WHILE, IF (nested within the WHILE).

```
CREATE PROCEDURE DSN8ES1
    ( IN DEPTNO          CHAR(3) ,
      OUT DEPTSAL        DECIMAL(15,2) ,
      OUT BONUSCNT       INT )
    FENCED
    RESULT SET 1
    LANGUAGE SQL
    NOT DETERMINISTIC
    MODIFIES SQL DATA
    NO DBINFO
    COLLID DSN8ES51
    NO WLM ENVIRONMENT
```

```

ASUTIME NO LIMIT
STAY RESIDENT NO
PROGRAM TYPE MAIN
SECURITY DB2
COMMIT ON RETURN NO

P1: BEGIN NOT ATOMIC
  DECLARE EMPLOYEE_NUMBER      CHAR(6);
  DECLARE EMPLOYEE_FIRSTNME    CHAR(12);
  DECLARE EMPLOYEE_LASTNAME    CHAR(15);
  DECLARE EMPLOYEE_SALARY      DECIMAL(15,2) DEFAULT 0;
  DECLARE EMPLOYEE_BONUS       DECIMAL(15,2) DEFAULT 0;
  DECLARE TOTAL_SALARY         DECIMAL(15,2) DEFAULT 0;
  DECLARE BONUS_COUNTER        INT          DEFAULT 0;
  DECLARE ENDTABLE             INT          DEFAULT 0;

  -- Cursor for result set of employees who got a bonus
  DECLARE DSN8ES1_RS_CSR CURSOR WITH RETURN WITH HOLD FOR
    SELECT RS_SEQUENCE,
           RS_EMPNO,
           RS_FIRSTNME,
           RS_LASTNAME,
           RS_BONUS
    FROM DSN8ES1_RS_TBL
    ORDER BY RS_SEQUENCE;

  -- Cursor to fetch department employees
  DECLARE C1 CURSOR FOR
    SELECT EMPNO,
           FIRSTNME,
           LASTNAME,
           SALARY,
           BONUS
    FROM EMP
    WHERE WORKDEPT = DEPTNO;

  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET ENDTABLE = 1;

  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SET DEPTSAL = NULL;

  -- Clean residual from the result set table
  DELETE FROM db2res1.DSN8ES1_RS_TBL;

  OPEN C1;

  FETCH C1
  INTO EMPLOYEE_NUMBER,
       EMPLOYEE_FIRSTNME,
       EMPLOYEE_LASTNAME,
       EMPLOYEE_SALARY,
       EMPLOYEE_BONUS;

  WHILE ENDTABLE = 0 DO
    SET TOTAL_SALARY = TOTAL_SALARY
      + EMPLOYEE_SALARY
      + EMPLOYEE_BONUS;

```

```

        IF EMPLOYEE_BONUS > 0.00 THEN
            SET BONUS_COUNTER = BONUS_COUNTER + 1;

        -- Add the employee's data to the result set
        INSERT INTO db2res1.DSN8ES1_RS_TBL
            ( RS_SEQUENCE,
              RS_EMPNO,
              RS_FIRSTNME,
              RS_LASTNAME,
              RS_SALARY,
              RS_BONUS )
        VALUES( P1.BONUS_COUNTER,
                 P1.EMPLOYEE_NUMBER,
                 P1.EMPLOYEE_FIRSTNME,
                 P1.EMPLOYEE_LASTNAME,
                 P1.EMPLOYEE_SALARY,
                 P1.EMPLOYEE_BONUS );

        END IF;

    FETCH C1
    INTO EMPLOYEE_NUMBER,
        EMPLOYEE_FIRSTNME,
        EMPLOYEE_LASTNAME,
        EMPLOYEE_SALARY,
        EMPLOYEE_BONUS;

    END WHILE;

    CLOSE C1;
    SET DEPTSAL = TOTAL_SALARY;
    SET BONUSCNT = BONUS_COUNTER;

    -- Open the cursor to the result set
    OPEN DSN8ES1_RS_CSR;
END P1

```

## A.2.2 SDK0LMS

This sample shows the use of result sets.

```

CREATE PROCEDURE SDK0LMS      (OUT MEDSAL DOUBLE)
    RESULT SETS 2
    LANGUAGE SQL
    COLLID SG245485
    WLM ENVIRONMENT WLMENV1
-----
-- SQL STORED PROCEDURE SDK0LMS
-----
BEGIN
    DECLARE V_NUMRECORDS INT DEFAULT 1;
    DECLARE V_COUNTER INT DEFAULT 0;
    DECLARE C1 CURSOR FOR
        SELECT INTEGER(SALARY) FROM DB2RES1.STAFF
        ORDER BY SALARY;
    -- USE WITH RETURN IN DECLARE CURSOR TO RETURN A RESULT SET
    DECLARE C2 CURSOR WITH RETURN FOR

```

```

SELECT NAME, JOB, INTEGER(SALARY)
FROM DB2RES1.STAFF
WHERE SALARY > MEDSAL
ORDER BY SALARY;
-- YOU CAN RETURN AS MANY RESULT SETS AS YOU LIKE, JUST
-- ENSURE THAT THE EXACT NUMBER IS DECLARED IN THE RESULT SETS
-- CLAUSE OF THE CREATE PROCEDURE STATEMENT
-- USE WITH RETURN IN DECLARE CURSOR TO RETURN ANOTHER RESULT SET
DECLARE C3 CURSOR WITH RETURN FOR
SELECT NAME, JOB, INTEGER(SALARY)
FROM DB2RES1.STAFF
WHERE SALARY < MEDSAL
ORDER BY SALARY DESC;
DECLARE EXIT HANDLER FOR NOT FOUND
SET MEDSAL = 6666;
-- INITIALIZE OUT PARAMETER
SET MEDSAL = 0;
SELECT COUNT(*) INTO V_NUMRECORDS FROM DB2RES1.STAFF;
OPEN C1;
WHILE V_COUNTER < (V_NUMRECORDS / 2 + 1) DO
    FETCH C1 INTO MEDSAL;
    SET V_COUNTER = V_COUNTER + 1;
END WHILE;
CLOSE C1;
-- RETURN 1ST RESULT SET, DO NOT CLOSE CURSOR
OPEN C2;

-- RETURN 2ND RESULT SET, DO NOT CLOSE CURSOR
OPEN C3;
END

```

### A.2.3 SDK1LMS

This sample shows the use of the CASE statement. Note that the SPECIFIC keyword used for the other samples is not valid in OS/390.

```

CREATE PROCEDURE SDK1LMS (IN empnum char(6),
                        IN rating INT, OUT msg char(20))
-- SPECIFIC DRDARES1.S1156162
LANGUAGE SQL
COLLID SG245485
WLM ENVIRONMENT WLMENV1
-----
BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
        SET msg = 'not found';
    CASE rating
    WHEN 1 THEN
        UPDATE db2res1.employee
        SET salary = salary * 1.10, bonus = 1000
        WHERE empno = empnum;
    WHEN 2 THEN
        UPDATE db2res1.employee
        SET salary = salary * 1.05, bonus = 500
        WHERE empno = empnum;
    ELSE

```

```

        UPDATE db2res1.employee
        SET salary = salary * 1.03, bonus = 0
        WHERE empno = empnum;
    END CASE;
END

```

## A.2.4 SDK2LMS

This sample shows the IF statement.

```

CREATE PROCEDURE SDK2LMS (IN empnum CHAR(6),
                        IN rating SMALLINT )
    COLLID SG245485
    LANGUAGE SQL
    WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK2LMS
-----
BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
        SET rating = -1;
    IF rating = 1
        THEN UPDATE db2res1.employee
            SET salary = salary * 1.10, bonus = 1000
            WHERE empno = empnum;
        ELSEIF rating = 2
            THEN UPDATE db2res1.employee
                SET salary = salary * 1.05, bonus = 500
                WHERE empno = empnum;
        ELSE UPDATE db2res1.employee
            SET salary = salary * 1.03, bonus = 0
            WHERE empno = empnum;
    END IF;
END

```

## A.2.5 SDK3LMS

This sample shows the use of the RUN OPTIONS debug line and DYNAMIC SQL statements.

```

CREATE PROCEDURE SDK3LMS
    (IN deptnum CHAR(3), OUT tabname CHAR(31) )
    LANGUAGE SQL
    COLLID SG245485
    WLM ENVIRONMENT WLMENV1
    RUN OPTIONS 'POSIX(ON),TEST(ALL,*,*,VADTCPIP&9.112.16.127:*)'
-----
-- SQL stored procedure SDK3LMS
-----
BEGIN
    DECLARE stmt VARCHAR(1000);
    -- continue if sqlstate 42704 ('undefined object name')
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
        SET stmt = '';
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        SET tabname = 'PROCEDURE_FAILED';

```

```

SET tabname = 'DEPT_' || deptnum || '_T';
SET stmt = 'CREATE TABLE ' || tabname ||
'( empno CHAR(6) NOT NULL, ' ||
'firstnme VARCHAR(12) NOT NULL, ' ||
'midinit CHAR(1) NOT NULL, ' ||
'lastname CHAR(15) NOT NULL, ' ||
'salary DECIMAL(9,2))' ||
'IN DATABASE RUNNING' ;
PREPARE s2 FROM STMT;
EXECUTE s2;
SET stmt = 'INSERT INTO ' || tabname ||
' SELECT empno, firstnme, midinit, lastname, salary ' ||
' FROM db2res1.employee e ' ||
' WHERE workdept = ?';
PREPARE s3 FROM stmt;
EXECUTE s3 USING deptnum;
END

```

## A.2.6 SDK4LMS

This sample shows use of the LOOP and LEAVE statements. Note that the ITERATE keyword is not valid for OS/390.

```

CREATE PROCEDURE SDK4LMS ( )
LANGUAGE SQL
COLLID SG245485
WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK4LMS
-----
BEGIN
DECLARE v_dept CHAR(3);
DECLARE v_deptname VARCHAR(29);
DECLARE v_admdept CHAR(3);
DECLARE at_end INT DEFAULT 0;
DECLARE not_found CONDITION FOR SQLSTATE '02000';
DECLARE c1 CURSOR FOR
SELECT deptno, deptname, admrdept
FROM db2res1.department
ORDER BY deptno;
DECLARE CONTINUE HANDLER FOR not_found
SET at_end = 1;

OPEN c1;
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 db2res1.department (deptno, deptname, admrdept)
VALUES ('NEW', v_deptname, v_admdept);
END LOOP;
CLOSE c1;
END

```



## A.2.7 SDK5LMS

This sample shows use of the IF statement nested within the LOOP statement.

```
CREATE PROCEDURE SDK5LMS (OUT counter INT )
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK5LMS
-----
BEGIN
  DECLARE v_firstnme VARCHAR(12);
  DECLARE v_midinit CHAR(1);
  DECLARE v_lastname VARCHAR(15);
  DECLARE at_end SMALLINT DEFAULT 0;
  DECLARE not_found
    CONDITION for SQLSTATE '02000';
  DECLARE c1 CURSOR FOR
    SELECT firstnme, midinit, lastname
    FROM db2res1.employee;
  DECLARE CONTINUE HANDLER for not_found
    SET at_end = 1;
  -- initialize OUT parameter
  SET counter = 0;
  OPEN c1;
  fetch_loop:
  LOOP
    FETCH c1 INTO
      v_firstnme, v_midinit, v_lastname;
    IF at_end <> 0 THEN LEAVE fetch_loop;
    END IF;
    SET counter = counter + 1;
  END LOOP fetch_loop;
  CLOSE c1;
END
```

## A.2.8 SDK6LMS

This sample also shows LEAVE and IF statements within a loop.

```
CREATE PROCEDURE SDK6LMS (OUT counter INT )
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK6LMS
-----
BEGIN
  DECLARE v_firstnme VARCHAR(12);
  DECLARE v_midinit CHAR(1);
  DECLARE v_lastname VARCHAR(15);
  DECLARE c1 CURSOR FOR
    SELECT firstnme, midinit, lastname
    FROM db2res1.employee;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET counter = -1;

  -- initialize OUT parameter
```

```

SET counter = 0;
OPEN c1;
fetch_loop:
LOOP
  FETCH c1 INTO
    v_firstnme, v_midinit, v_lastname;
  IF v_midinit = ' ' THEN
    LEAVE fetch_loop;
  END IF;
  SET counter = counter + 1;
END LOOP fetch_loop;
CLOSE c1;
END

```

## A.2.9 SDK7LMS

This sample shows nested CASE statements.

```

CREATE PROCEDURE SDK7LMS (IN deptnum SMALLINT )
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK7LMS
-----
BEGIN
  DECLARE v_salary DOUBLE;
  DECLARE v_id SMALLINT;
  DECLARE v_years SMALLINT;
  DECLARE at_end INT DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';
  DECLARE C1 CURSOR FOR
    SELECT id, FLOAT(salary), years
    FROM db2res1.staff
    WHERE dept = deptnum;
  DECLARE CONTINUE HANDLER FOR not_found
    SET at_end = 1;
  OPEN C1;
  FETCH C1 INTO v_id, v_salary, v_years;
  WHILE at_end = 0 DO
    CASE
      WHEN (v_salary < 2000 * v_years)
        THEN UPDATE db2res1.staff
          SET salary = 2150 * v_years
          WHERE id = v_id;
      WHEN (v_salary < 5000 * v_years)
        THEN CASE
          WHEN (v_salary < 3000 * v_years)
            THEN UPDATE db2res1.staff
              SET salary = 4000 * v_years
              WHERE id = v_id;
          ELSE UPDATE db2res1.staff
            SET salary = v_salary * 1.10
            WHERE id = v_id;
        END CASE;
      ELSE UPDATE db2res1.staff
        SET job = 'PREZ'
    END CASE;
  END WHILE;
END

```

```

        WHERE id = v_id;
    END CASE;
    FETCH C1 INTO v_id, v_salary, v_years;
END WHILE;
CLOSE C1;
END

```

## A.2.10 SDK8LMS

This sample shows use of nested WHILE and IF statements.

```

CREATE PROCEDURE SDK8LMS (IN deptnum SMALLINT )
LANGUAGE SQL
COLLID SG245485
WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK8LMS
-----
BEGIN
    DECLARE v_salary DOUBLE;
    DECLARE v_id SMALLINT;
    DECLARE v_years SMALLINT;
    DECLARE at_end INT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    -- CAST salary as DOUBLE because SQL procedures do not support DECIMA
    DECLARE C1 CURSOR FOR
        SELECT id, FLOAT(salary), years
        FROM db2res1.staff
        WHERE dept = deptnum;
    DECLARE CONTINUE HANDLER FOR not_found
        SET at_end = 1;
    OPEN C1;
    FETCH C1 INTO v_id, v_salary, v_years;
    WHILE at_end = 0 DO
        IF (v_salary < 2000 * v_years)
            THEN UPDATE db2res1.staff
                SET salary = 2150 * v_years
                WHERE id = v_id;
        ELSEIF (v_salary < 5000 * v_years)
            THEN IF (v_salary < 3000 * v_years)
                THEN UPDATE db2res1.staff
                    SET salary = 3000 * v_years
                    WHERE id = v_id;
                ELSE UPDATE db2res1.staff
                    SET salary = v_salary * 1.10
                    WHERE id = v_id;
            END IF;
        ELSE UPDATE db2res1.staff
            SET job = 'PREZ'
            WHERE id = v_id;
        END IF;
        FETCH C1 INTO v_id, v_salary, v_years;
    END WHILE;
    CLOSE C1;
END

```

## A.2.11 SDK9LMS

Sample which uses the REPEAT statement.

```
CREATE PROCEDURE SDK9LMS (OUT counter INT )
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SDK9LMS
-----
BEGIN
  DECLARE v_firstnme VARCHAR(12);
  DECLARE v_midinit CHAR(1);
  DECLARE v_lastname VARCHAR(15);
  DECLARE at_end SMALLINT DEFAULT 0;
  DECLARE not_found
    CONDITION FOR SQLSTATE '02000';
  DECLARE c1 CURSOR FOR
    SELECT firstnme, midinit, lastname
    FROM db2res1.employee;
  DECLARE CONTINUE HANDLER FOR not_found
    SET at_end = 1;
  -- initialize OUT parameter
  SET counter = 0;
  OPEN c1;
  fetch_loop:
    REPEAT
      FETCH c1 INTO
        v_firstnme, v_midinit, v_lastname;
      SET counter = counter + 1;
    UNTIL at_end <> 0
  END REPEAT fetch_loop;
  SET counter = counter - 1; -- count is 1 more than actual for repeat
  CLOSE c1;
END
```

## A.2.12 SMP0LMS

This sample returns RESULT SETS using CURSOR WITH RETURN.

```
CREATE PROCEDURE SMP0LMS ( IN PID INT )
  RESULT SETS 1
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL STORED PROCEDURE SMP0LMS
-----
P1: BEGIN
  -- DECLARE CURSOR
  DECLARE CURSOR1 CURSOR WITH RETURN FOR
    SELECT * FROM DB2RES1.STAFF WHERE ID > PID;
  OPEN CURSOR1;
END P1
```

### A.2.13 SMP1LMS

This sample shows use of the IF statement nested within LOOP statement.

```
CREATE PROCEDURE SMP1LMS ( )
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL STORED PROCEDURE SMP1LMS
-----
P1: BEGIN
  DECLARE V_DEPT CHAR(3);
  DECLARE V_DEPTNAME VARCHAR(29);
  DECLARE V_MGRNO CHAR(6);
  DECLARE V_ADMDEPT CHAR(3);
  DECLARE AT_END SMALLINT DEFAULT 0;
  DECLARE NOT_FOUND CONDITION FOR SQLSTATE '02000';
  -- DECLARE CURSOR
  DECLARE C1 CURSOR FOR
    SELECT
      db2res1.DEPARTMENT.DEPTNO,
      db2res1.DEPARTMENT.DEPTNAME,
      db2res1.DEPARTMENT.MGRNO,
      db2res1.DEPARTMENT.ADMRDEPT
    FROM
      db2res1.DEPARTMENT;
  DECLARE CONTINUE HANDLER FOR NOT_FOUND
    SET AT_END = 1;
  OPEN c1;
  loop1:
  LOOP
    FETCH c1 into v_dept, v_deptname, v_mgrno, v_admdept;
    IF at_end = 1 THEN
      LEAVE loop1;
    ELSEIF v_mgrno is null THEN
      UPDATE db2res1.department
        set mgrno = '000000' where deptno = v_dept;
    ELSEIF v_mgrno = '000000' THEN
      UPDATE db2res1.department
        set mgrno = null where deptno = v_dept;
    END IF;
  END LOOP;
END P1
```

### A.2.14 SMP2LMS

This sample was created to test various date and time values being returned.

```
CREATE PROCEDURE SMP2LMS (out vuser char(8), out vdate1 date,
  out vdate2 date, out vdays1 integer, out vtime1 time,
  out vtime2 time, out vtimest1 timestamp, out vtimest2 timestamp)
  LANGUAGE SQL
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SMP2LMS
-----
P1: BEGIN
```

```

set vuser = user;
set vdate1 = current date;
set vdate2 = vdate1 - 10 days + 3 years;
set vdays1 = days(vdate2);
set vtime1 = current time;
set vtime2 = vtime1 +1 hour - 30 minutes;
set vtimest1 = current timestamp;
set vtimest2 = vtimest1 - days(vdate1 - 10 days) days;
END p1

```

## A.2.15 SMP3LMS

This sample was created to test the settings of null and zero.

```

CREATE PROCEDURE SMP3LMS
(out msg1 char(20), out msg2 char(20), out msg3 char(20) )
LANGUAGE SQL
COLLID SG245485
WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SMP3LMS
-----
P1: BEGIN
declare v_var1 integer;
if v_var1 is null then
    set msg1 = 'is null';
elseif v_var1 = 0 then
    set msg1 = 'is zero';
end if;

set v_var1 = null;

if v_var1 is null then
    set msg2 = 'is null';
end if;
set v_var1 = 0;
if v_var1 is not null then
    set msg3 = 'not null';
end if;
END p1

```

## A.2.16 SMP4LMS

This is the equivalent sample when run on NT, which allows parameter and variable names to be the same. On OS/390 it was invalid, even when we prefixed the parameter name with the procedure name, and the variable name with the label name. To make it work, we had to have unique parameter and variable names.

```

CREATE PROCEDURE SMP4LMS
(out pvar1 integer, out pvar2 double )
LANGUAGE SQL
COLLID SG245485
WLM ENVIRONMENT WLMENV1
-----
-- SQL stored procedure SMP4LMS
-----

```

```

BEGIN
declare vvar1 integer;
declare vvar2 smallint;

set vvar1 = 10;
set pvar1 = vvar1;
set vvar2 = 20;
set pvar2 = vvar2;

END

```

### A.2.17 SMP5LMS

This sample shows that variable and column names could be the same, but you need to qualify the variable name with the label name (`p1`) and the column name with the table name (`staff`).

```

CREATE PROCEDURE SMP5LMS
(out pid integer )
LANGUAGE SQL
COLLID SG245485
-----
-- SQL stored procedure smp5LMS
-----

P1: BEGIN
declare id integer;
set id = 100;
select staff.id into p1.id from db2res1.staff where staff.id=p1.id;
set pid = p1.id;

END P1

```

### A.2.18 SMP5LMS2

This sample shows that parameter and column names could be the same, but you need to qualify the column name with the table name (`staff`).

```

CREATE PROCEDURE SMP5LMS
(inout id integer )
LANGUAGE SQL
COLLID SG245485
-----
-- SQL stored procedure SMP5LMS
-----

L1: BEGIN
SELECT staff.id INTO id
FROM db2res1.staff WHERE staff.id=id;

END L1

```

### A.2.19 SMP7LMS

This sample shows how to return the `SQLSTATE` and `SQLCODE` as parameters from the SQL procedure. Note the use of `IF (1=1)` within the handler declaration.

```

CREATE PROCEDURE SMP7LMS ( OUT PSQLST char(5) ,

```

```

                                OUT PSQLCO int )
RESULT SETS 1
LANGUAGE SQL
COLLID SG245485
ASUTIME NO LIMIT
-----
-- SQL stored procedure SYSPROC.SMP7LMS
-----
P1: BEGIN
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLCODE INT DEFAULT 0;

  -- Declare cursor
  DECLARE cursor1 CURSOR WITH RETURN FOR
    SELECT * FROM db2res1.STAFF;

  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    IF (1 = 1) THEN
      SET PSQLST = SQLSTATE;
      SET PSQLCO = SQLCODE;
    END IF;

  -- Cursor left open for client application
  OPEN cursor1;
  SET PSQLST = SQLSTATE;
  SET PSQLCO = SQLCODE;
END P1

```

## A.2.20 SMP8LMS

This sample was used to test most column and scalar functions for version 5.

```

CREATE PROCEDURE DRDARES1.SMP8LMS (out v1 double,out v2 integer,
out v3 integer,
out v4 integer ,out v5 integer,out v6 char(15),out v7 char (5),
out v8 decimal(15),out v9 char(10), out v10 double,
out v11 char(5), out v12 integer, out v13 integer, out v14 char(5))
COLLID SG245485
WLM ENVIRONMENT WLMENV1
LANGUAGE SQL
-----
-- SQL stored procedure SMP8LMS
-----
P1: BEGIN
  SELECT AVG(SALARY) ,COUNT(*) ,MAX(SALARY) ,MIN(SALARY) ,SUM(SALARY)
  INTO v1,v2,v3,v4,v5
  FROM DB2RES1.STAFF;
  set v6 = char(current date);
  set v7 = coalesce('hello','');
  set v8 = decimal(10);
  set v9 = digits(12345);
  set v10 = float(123);
  SELECT HEX(ID) ,INTEGER(SALARY) , LENGTH(NAME)
  INTO v11,v12,v13
  FROM DB2RES1.STAFF WHERE ID = 100;
  set v14 = nullif('hello','HELLO');
END P1

```



## A.2.21 SMP8LMS2

This sample was created to test the column and scalar functions for version 6.

```
CREATE PROCEDURE ADMF001.SMP8LMS2 (out v1 double,out v2 double,
out v3 integer, out v5 decimal(10), out v7 varchar (27),
out v8 integer,out v9 integer, out v10 integer,
out v12 char(8), out v13 char(3), out v14 integer,
out v15 integer, out v16 integer, out v17 char(12), out v18 integer)
  COLLID SG245485
  WLM ENVIRONMENT WLMENV1
  LANGUAGE SQL
-----
-- SQL STORED PROCEDURE ADMF001.SMP8LMS2
-----
P1: BEGIN
  SELECT STDDEV(SALARY),VAR(SALARY) INTO v1,v2 FROM DB2RES1.STAFF;
  SET v3 = ABS(-4356);
  SELECT CEIL(MAX(SALARY)/12) INTO v5 FROM DSN8710.EMP;
  SELECT CONCAT(FIRSTNAME, LASTNAME) INTO v7 FROM DSN8710.EMP
    WHERE EMPNO = '000140';
  SELECT DAYOFWEEK(HIREDATE) INTO v8 FROM DSN8710.EMP
    WHERE EMPNO = '000140';
  SELECT AVG(DAYOFYEAR(HIREDATE)) INTO v9 FROM DSN8710.EMP;
  SELECT FLOOR(MAX(SALARY)/12) INTO v10 FROM DSN8710.EMP;
  SET v12 = LCASE('KATHLEEN');
  SET v13 = LEFT('JONATHAN',3);
  SELECT MIDNIGHT_SECONDS('24:00:00'),MIDNIGHT_SECONDS('00:00:00')
    INTO v14,v15 FROM SYSIBM.SYSDUMMY1;
  SET v16 = QUARTER('1999-09-09');
  SET v17 = REPEAT('KATH',3);
  SET v18 = SIGN(-1000);
END P1
```

---

## A.3 NT and AIX samples

### A.3.1 SDK0LNS

This sample shows how RESULT SETS can be returned to the client.

```
CREATE PROCEDURE DRDARES1.SDK0LNS (OUT medianSalary DOUBLE)
  RESULT SETS 2
  SPECIFIC DRDARES1.S0265192
  LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK0LNS
-----
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;

  -- use WITH RETURN in DECLARE CURSOR to return a result set
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, CAST(salary AS INTEGER)
    FROM staff
```

```

WHERE salary > medianSalary
ORDER BY salary;

-- you can return as many result sets as you like, just
-- ensure that the exact number is declared in the RESULT SETS
-- clause of the CREATE PROCEDURE statement

-- use WITH RETURN in DECLARE CURSOR to return another result set
DECLARE c3 CURSOR WITH RETURN FOR
  SELECT name, job, CAST(salary AS INTEGER)
  FROM staff
  WHERE salary < medianSalary
  ORDER BY SALARY DESC;

DECLARE EXIT HANDLER FOR NOT FOUND
  SET medianSalary = 6666;

-- initialize OUT parameter
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;

-- return 1st result set, do not CLOSE cursor
OPEN c2;

-- return 2nd result set, do not CLOSE cursor
OPEN c3;
END

```

### A.3.2 SDK1LNS

This sample shows the CASE statement.

```

CREATE PROCEDURE DRDARES1.SDK1LNS (IN employee_number CHAR(6),
                                  IN rating INT)
  SPECIFIC DRDARES1.S1156162
  RESULT SETS 1
  LANGUAGE SQL
  BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
      SIGNAL SQLSTATE '02444';

    CASE rating
      WHEN 1 THEN
        UPDATE employee
        SET salary = salary * 1.10, bonus = 1000
        WHERE empno = employee_number;
      WHEN 2 THEN
        UPDATE employee
        SET salary = salary * 1.05, bonus = 500

```

```

        WHERE empno = employee_number;
    ELSE
        UPDATE employee
        SET salary = salary * 1.03, bonus = 0
        WHERE empno = employee_number;
    END CASE;
END

```

### A.3.3 SDK2LNS

This sample is similar to the one above, but it is implemented using the IF statement.

```

CREATE PROCEDURE DRDARES1.SDK2LNS (IN employee_number CHAR(6), IN rating
SMALLINT )
    SPECIFIC DRDARES1.S030109
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK2LNS
-----
BEGIN
    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE EXIT HANDLER FOR not_found
        SET rating = -1;

    IF rating = 1
    THEN UPDATE employee
        SET salary = salary * 1.10, bonus = 1000
        WHERE empno = employee_number;
    ELSEIF rating = 2
    THEN UPDATE employee
        SET salary = salary * 1.05, bonus = 500
        WHERE empno = employee_number;
    ELSE UPDATE employee
        SET salary = salary * 1.03, bonus = 0
        WHERE empno = employee_number;
    END IF;
END

```

### A.3.4 SDK3LNS

This sample shows use of DYNAMIC SQL statements.

```

CREATE PROCEDURE DRDARES1.SDK3LNS (IN deptNumber CHAR(4), OUT table_name
CHAR(31) )
    SPECIFIC DRDARES1.S041662
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK3LNS
-----
BEGIN
    DECLARE stmt VARCHAR(1000);

    -- continue if sqlstate 42704 ('undefined object name')
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42704'
        SET stmt = '';
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION

```

```

        SET table_name = 'PROCEDURE_FAILED';

SET table_name = 'DEPT_' || deptNumber || '_T';
SET stmt = 'DROP TABLE ' || table_name;
PREPARE s1 FROM stmt;
EXECUTE s1;
SET stmt = 'CREATE TABLE ' || table_name ||
'( empno CHAR(6) NOT NULL, ' ||
'firstnme VARCHAR(12) NOT NULL, ' ||
'midinit CHAR(1) NOT NULL, ' ||
'lastname CHAR(15) NOT NULL, ' ||
'salary DECIMAL(9,2))';
PREPARE s2 FROM STMT;
EXECUTE s2;
SET stmt = 'INSERT INTO ' || table_name ||
'SELECT empno, firstnme, midinit, lastname, salary ' ||
'FROM employee ' ||
'WHERE workdept = ?';
PREPARE s3 FROM stmt;
EXECUTE s3 USING deptNumber;
END

```

### A.3.5 SDK4LNS

This sample shows the LOOP, LEAVE and ITERATE statements.

```

CREATE PROCEDURE DRDARES1.SDK4LMS ( )
    SPECIFIC DRDARES1.S0202895
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK4LMS
-----

BEGIN
    DECLARE v_dept CHAR(3);
    DECLARE v_deptname VARCHAR(29);
    DECLARE v_admdept CHAR(3);
    DECLARE at_end INT DEFAULT 0;

    DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
        SELECT deptno, deptname, admrdept
        FROM department
        ORDER BY deptno;
    DECLARE CONTINUE HANDLER FOR not_found
        SET at_end = 1;

    OPEN c1;
    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;
CLOSE c1;
END

```

### A.3.6 SDK5LNS

This sample shows the LOOP and IF statements.

```

CREATE PROCEDURE DRDARES1.SDK5LNS (OUT counter INT )
SPECIFIC DRDARES1.S0214278
LANGUAGE SQL

```

```

-----
-- SQL stored procedure DRDARES1.SDK5LNS
-----

```

```

BEGIN
DECLARE v_firstnme VARCHAR(12);
DECLARE v_midinit CHAR(1);
DECLARE v_lastname VARCHAR(15);
DECLARE at_end SMALLINT DEFAULT 0;
DECLARE not_found
CONDITION for SQLSTATE '02000';
DECLARE c1 CURSOR FOR
SELECT firstnme, midinit, lastname
FROM employee;
DECLARE CONTINUE HANDLER for not_found
SET at_end = 1;

-- initialize OUT parameter
SET counter = 0;
OPEN c1;
fetch_loop:
LOOP
FETCH c1 INTO
v_firstnme, v_midinit, v_lastname;
SET counter = counter + 1;
IF at_end <> 0 THEN LEAVE fetch_loop;
END IF;
END LOOP fetch_loop;
CLOSE c1;
END

```

### A.3.7 SDK6LNS

This sample also shows the LOOP and LEAVE statements.

```

CREATE PROCEDURE DRDARES1.SDK6LNS (OUT counter INT )
SPECIFIC DRDARES1.S0225562
LANGUAGE SQL

```

```

-----
-- SQL stored procedure DRDARES1.SDK6LNS
-----

```

```

BEGIN
DECLARE v_firstnme VARCHAR(12);
DECLARE v_midinit CHAR(1);
DECLARE v_lastname VARCHAR(15);
DECLARE c1 CURSOR FOR

```

```

        SELECT firstnme, midinit, lastname
        FROM employee;
DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET counter = -1;

-- initialize OUT parameter
SET counter = 0;
OPEN c1;
fetch_loop:
LOOP
        FETCH c1 INTO
                v_firstnme, v_midinit, v_lastname;
        SET counter = counter + 1;
        IF v_midinit = ' ' THEN
                LEAVE fetch_loop;
        END IF;
END LOOP fetch_loop;
CLOSE c1;
END

```

### A.3.8 SDK7LNS

This sample shows nested CASE statements.

```

CREATE PROCEDURE DRDARES1.SDK7LNS (IN deptnumber SMALLINT )
        SPECIFIC DRDARES1.S0235345
        LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK7LNS
-----
BEGIN
        DECLARE v_salary DOUBLE;
        DECLARE v_id SMALLINT;
        DECLARE v_years SMALLINT;
        DECLARE at_end INT DEFAULT 0;
        DECLARE not_found CONDITION FOR SQLSTATE '02000';

-- CAST salary as DOUBLE because SQL procedures do not support DECIMAL
DECLARE C1 CURSOR FOR
        SELECT id, CAST(salary AS DOUBLE), years
        FROM staff
        WHERE dept = deptnumber;
DECLARE CONTINUE HANDLER FOR not_found
        SET at_end = 1;

OPEN C1;
FETCH C1 INTO v_id, v_salary, v_years;
WHILE at_end = 0 DO
        CASE
                WHEN (v_salary < 2000 * v_years)
                        THEN UPDATE staff
                                SET salary = 2150 * v_years
                                WHERE id = v_id;
                WHEN (v_salary < 5000 * v_years)
                        THEN CASE
                                WHEN (v_salary < 3000 * v_years)
                                        THEN UPDATE staff
                                                SET salary = 3000 * v_years

```

```

        WHERE id = v_id;
    ELSE UPDATE staff
        SET salary = v_salary * 1.10
        WHERE id = v_id;
    END CASE;
    ELSE UPDATE staff
        SET job = 'PREZ'
        WHERE id = v_id;
    END CASE;
    FETCH C1 INTO v_id, v_salary, v_years;
END WHILE;
CLOSE C1;
END

```

### A.3.9 SDK8LNS

This sample shows nested IF statement within a WHILE statement.

```

CREATE PROCEDURE DRDARES1.SDK8LNS (IN deptnumber SMALLINT )
    SPECIFIC DRDARES1.S0244956
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK8LNS
-----
BEGIN
    DECLARE v_salary DOUBLE;
    DECLARE v_years SMALLINT;
    DECLARE v_id SMALLINT;
    DECLARE at_end INT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';

    -- CAST salary as DOUBLE because SQL procedures do not support DECIMAL
    DECLARE C1 CURSOR FOR
        SELECT id, CAST(salary AS DOUBLE), years
        FROM staff;
    DECLARE CONTINUE HANDLER FOR not_found
        SET at_end = 1;

    OPEN C1;
    FETCH C1 INTO v_id, v_salary, v_years;
    WHILE at_end = 0 DO
        IF (v_salary < 2000 * v_years)
            THEN UPDATE staff
                SET salary = 2150 * v_years
                WHERE id = v_id;
            ELSEIF (v_salary < 5000 * v_years)
                THEN IF (v_salary < 3000 * v_years)
                    THEN UPDATE staff
                        SET salary = 3000 * v_years
                        WHERE id = v_id;
                    ELSE UPDATE staff
                        SET salary = v_salary * 1.10
                        WHERE id = v_id;
                    END IF;
            ELSE UPDATE staff
                SET job = 'PREZ'
                WHERE id = v_id;
            END IF;
        END IF;
    END WHILE;
END

```

```

        FETCH C1 INTO v_id, v_salary, v_years;
    END WHILE;
    CLOSE C1;
END

```

### A.3.10 SDK9LNS

This sample shows the REPEAT statement.

```

CREATE PROCEDURE DRDARES1.SDK9LNS (OUT counter INT )
    SPECIFIC DRDARES1.S0254846
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDK9LNS
-----
BEGIN
    DECLARE v_firstnme VARCHAR(12);
    DECLARE v_midinit CHAR(1);
    DECLARE v_lastname VARCHAR(15);
    DECLARE at_end SMALLINT DEFAULT 0;
    DECLARE not_found
        CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR
        SELECT firstnme, midinit, lastname
        FROM employee;
    DECLARE CONTINUE HANDLER FOR not_found
        SET at_end = 1;

    -- initialize OUT parameter
    SET counter = 0;
    OPEN c1;
    fetch_loop:
        REPEAT
            FETCH c1 INTO
                v_firstnme, v_midinit, v_lastname;
            SET counter = counter + 1;
        UNTIL at_end <> 0
        END REPEAT fetch_loop;
    CLOSE c1;
END

```

### A.3.11 SDKALNS

This sample is similar to the previous one, but it is implemented using the WHILE statement.

```

CREATE PROCEDURE DRDARES1.SDKALNS (IN deptNumber SMALLINT,
                                   OUT medianSalary DOUBLE )
    SPECIFIC DRDARES1.S0304753
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SDKALNS
-----
BEGIN
    DECLARE v_numRecords INT DEFAULT 1;
    DECLARE v_counter INT DEFAULT 0;

```



```

DECLARE c1 CURSOR FOR
  SELECT CAST(salary AS DOUBLE) FROM staff
  WHERE DEPT = deptNumber
  ORDER BY salary;
DECLARE EXIT HANDLER FOR NOT FOUND
  SET medianSalary = 6666;

-- initialize OUT parameter
SET medianSalary = 0;

SELECT COUNT(*) INTO v_numRecords FROM staff
  WHERE DEPT = deptNumber;

OPEN c1;
WHILE v_counter < (v_numRecords / 2 + 1) DO
  FETCH c1 INTO medianSalary;
  SET v_counter = v_counter + 1;
END WHILE;
CLOSE c1;
END

```

### A.3.12 SMP1LNS

This sample shows an IF statement within a LOOP statement.

```

CREATE PROCEDURE DRDARES1.SMP1LNS ( )
  SPECIFIC DRDARES1.S4141979
  LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP1LNS
-----
P1: BEGIN
  DECLARE v_dept CHAR(3);
  DECLARE v_deptname VARCHAR(29);
  declare v_mgrno char(6);
  DECLARE v_admdept CHAR(3);
  declare at_end smallint default 0;
  DECLARE not_found CONDITION FOR SQLstate '02000';

  -- Declare cursor
  DECLARE c1 CURSOR FOR
    SELECT
      DRDARES1.DEPARTMENT.DEPTNO,
      DRDARES1.DEPARTMENT.DEPTNAME,
      DRDARES1.DEPARTMENT.MGRNO,
      DRDARES1.DEPARTMENT.ADMRDEPT
    FROM
      DRDARES1.DEPARTMENT;
  DECLARE CONTINUE HANDLER FOR not_found
    SET at_end = 1;

  OPEN c1;
loop1:
  LOOP
    FETCH c1 into v_dept, v_deptname, v_mgrno, v_admdept;
    IF at_end = 1 THEN
      LEAVE loop1;
    ELSEIF v_mgrno is null THEN

```

```

        UPDATE department set mgrno = '000000' where deptno = v_dept;
    ELSEIF v_mgrno = '000000' THEN
        UPDATE department set mgrno = null where deptno = v_dept;
    END IF;
    ITERATE loop1;
END LOOP;
END P1

```

### A.3.13 SMP2LNS

This sample shows testing of various date and time functions.

```

CREATE PROCEDURE DRDARES1.SMP2LNS ( out v_user char(8), out v_date1 date,
out v_date2 date, out v_days1 integer, out v_time1 time,
out v_time2 time, out v_timest1 timestamp, out v_timest2 timestamp)
    SPECIFIC DRDARES1.S5336343
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP2LNS
-----
P1: BEGIN
    set v_user = user;
    set v_date1 = current date;
    set v_date2 = v_date1 - 10 days + 3 years;
    set v_days1 = days(v_date2);
    set v_time1 = current time;
    set v_time2 = v_time1 +1 hour - 30 minutes;
    set v_timest1 = current timestamp;
    set v_timest2 = v_timest1 - days(v_date1 - 10 days) days;

END P1

```

### A.3.14 SMP3LNS

This sample shows testing of null and zero indicators.

```

CREATE PROCEDURE DRDARES1.SMP3LNS (out msg1 char(20), out msg2 char(20), out
msg3 char(20) )
    SPECIFIC DRDARES1.S0535160
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP3LNS
-----
P1: BEGIN
    declare v_var1 integer;

    if v_var1 is null then
        set msg1 = 'is null';
    elseif v_var1 = 0 then
        set msg1 = 'is zero';
    end if;

    set v_var1 = null;

    if v_var1 is null then
        set msg2 = 'is null';
    end if;

```

```

set v_var1 = 0;

if v_var1 is not null then
    set msg3 = 'not null';
end if;
END P1

```

### A.3.15 SMP4LNS

This sample shows the same parameter and variable names, but they need to be qualified.

```

CREATE PROCEDURE DRDARES1.SMP4LNS (out v_var1 integer, out v_var2 double )
    SPECIFIC DRDARES1.S1473953
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP4LNS
-----
P1: BEGIN
    declare v_var1 integer;
    declare v_var2 smallint;

    set v_var1 = 10;
    set smp4lns.v_var1 = p1.v_var1;
    set v_var2 = 20;
    set smp4lns.v_var2 = p1.v_var2;

END P1

```

### A.3.16 SMP5LNS

This sample shows the same variable and column names. Only the variable needs to be qualified in the WHERE clause.

```

CREATE PROCEDURE DRDARES1.SMP5LNS (out p_id integer )
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP5LNS
-----
P1: BEGIN
    declare id integer;
    set id = 100;
    select id into id from staff where id=p1.id;
    set p_id = id;

END P1

```

### A.3.17 SMP7LNS

This sample tests SQLSTATE and SQLCODE. When run for a row that was not found, only MSG3 was set to 'NOT FOUND'.

```

CREATE PROCEDURE DRDARES1.SMP7LNS (OUT PARM1 INTEGER, OUT MSG CHAR(10), OUT
MSG1 CHAR(10), out msg2 char(10),
    OUT MSG3 CHAR(10), IN P_ID INTEGER)
    SPECIFIC S1140197
    LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP7LNS

```

```

-----
P1: BEGIN
  DECLARE VAR1 CHAR(20);
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLCODE INT DEFAULT 0;

  DECLARE NOT_FOUND CONDITION FOR SQLSTATE '02000';
  DECLARE CONTINUE HANDLER FOR NOT_FOUND SET MSG3='NOT FOUND';

  SELECT NAME INTO VAR1 FROM STAFF WHERE ID=P_ID;

  SET PARM1 = SQLCODE;
  IF SQLCODE = 100 THEN
    SET MSG = 'NOT FOUND';
  END IF;
  IF PARM1 = 100 THEN
    SET MSG1 = 'NOT FOUND';
  END IF;
  IF SQLSTATE = '02000' THEN
    SET MSG2 = 'NOT FOUND';
  END IF;

END P1

```

### A.3.18 SMP8LNS

This sample tests various scalar functions.

```

CREATE PROCEDURE DRDARES1.SMP8LNS (out v1 double, out v2 double, out v3
integer,
out v4 integer , out v5 integer, out v6 integer , out v7 char (5),
out v8 char(10), out v9 char(10), out v10 char (20), out v11 char(5), out v12
char(10))
  SPECIFIC DRDARES1.S8389109
  LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP8LNS
-----
P1: BEGIN
  set v1 = tan(.5) - (sin(.5)/cos(.5));
  set v2 = exp(sin(.3)) + exp(cos(.3));
  set v3 = rand();
  set v4 = ceil(5.2) + floor(4.3);
  set v5 = quarter(current date);
  set v6 = week(current date);
  set v7 = repeat('*',5);
  set v8 = lcase('ALINE');
  set v9 = replace('a1b1c1','1','2');
  set v10 = monthname(current date) || dayname(current date);
  set v11 = ltrim('felipe ');
  set v12 = substr('abcdefghijklmnopq',5,10);

END P1

```

### A.3.19 SMP9LNS

This sample tests the SQLCODE. It has an interesting result, in that only PARM1 and MSG1 were set correctly.

```

CREATE PROCEDURE DRDARES1.SMP9LNS (OUT PARM1 INTEGER, OUT MSG CHAR(10), OUT
MSG1 CHAR(10), out msg2 char(10),
  IN P_ID INTEGER)
  SPECIFIC S1230197
  LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMP7LNS
-----
P1: BEGIN
  DECLARE VAR1 CHAR(20);
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLCODE INT DEFAULT 0;

  SELECT NAME INTO VAR1 FROM STAFF WHERE ID=P_ID;

  SET PARM1 = SQLCODE;
  IF SQLCODE = 100 THEN
    SET MSG = 'NOT FOUND';
  END IF;
  IF PARM1 = 100 THEN
    SET MSG1 = 'NOT FOUND';
  END IF;
  IF SQLSTATE = '02000' THEN
    SET MSG2 = 'NOT FOUND';
  END IF;

END P1

```

### A.3.20 SMPALNS

This sample shows the SQLCODE being passed as an output parameter.

```

CREATE PROCEDURE DRDARES1.SMPALNS (in vid integer, out sqlc integer, out var1
integer)
  SPECIFIC DRDARES1.S1140337
  LANGUAGE SQL
-----
-- SQL stored procedure DRDARES1.SMPALNS
-----
P1: BEGIN
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    SET SQLC=SQLCODE;
    SET VAR1=1;
  END;

  SELECT id into vid FROM STAFF where id = vid;

END P1

```



---

## Appendix B. Special notices

This publication is intended to help database administrators implement DB2 SQL Stored Procedures and the Stored Procedures Builder in a client/server environment. The information in this publication is not intended as the specification of any programming interfaces that are provided by the DB2 family of products. See the PUBLICATIONS section of the IBM Programming Announcement for Stored Procedure Builder and the DB2 UDB SQL Procedures support for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. 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 be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating

environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

ACF/VTAM	AD/Cycle
AIX	AS/400
AT	CT
CICS	CICS/ESA
C/370	DATABASE 2
DataGuide	DataJoiner
DataPropagator	DB2
DFSMS	DFSMS/MVS
DFSORT	GDDM
ESCON	ES/9000
Hiperspace	IBM
BMLink	IMS
Information Warehouse	Integrated Language Environment
Intelligent Miner	Language Environment
Multiprise	MVS/ESA
Net.Data	Netfinity
OS/2	OS/390
OS/400	Parallel Sysplex
PR/SM	QMF
RACF	RAMAC
RETAIN	RMF
RS/6000	SP
SP1	SP2
S/390	S/390 Parallel Enterprise Server
System/390	VisualAge
VisualGen	Visual Warehouse
VM/ESA	VSE/ESA
VTAM	WebSphere
XT	400

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

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



UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

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



---

## Appendix C. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

### C.1 International Technical Support Organization publications

For information on ordering these ITSO publications see “How to get ITSO redbooks” on page 229.

- *Getting Started with DB2 Stored Procedures: Give Them a Call through the Network*, SG24-4693
- *DB2 Server for OS/390 Version 5 Recent Enhancements - Reference Guide*, SG24-5421
- *DB2/400 Advanced Database Functions*, SG24-4249
- *DB2 DRDA Supports TCP/IP*, SG24-2212

---

### C.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

---

### C.3 Other publications

These publications are also relevant as further information sources:

- *DB2 for OS/390 V5 Preview of SQL Procedures*, (\*)
- *DB2 for OS/390 V5 Application Programming and SQL Guide*, SC26-8958
- *DB2 for OS/390 V5 SQL Reference*, SC26-8966
- *DB2 UDB for OS/390 V6 Preview of SQL Procedures*, (\*)
- *DB2 UDB for OS/390 V6 Application Programming and SQL Guide*, SC26-9004
- *DB2 UDB for OS/390 V6 SQL Reference*, SC26-9014
- *DB2 UDB Version 6 SQL Reference, Volume 1 and Volume 2*, SBOF-8923
- *DB2 UDB Version 6 Application Development*, SC09-2845

- *Debug Tool: User's Guide and Reference*, SC09-2137
- *AS/400 Toolbox for Java Setup Guide*", SC41-5438
- *Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM*, Jim Melton, Norgan Kaufmann Publishers, Inc., ISBN 1-55860-461-8.

(\*) download through the Web:

<http://www.software.ibm.com/db2/os390/sqlproc>

---

## How to get ITSO redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the redbooks fax order form to:

	<b>e-mail address</b>
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the "How to Order" section at this site: <a href="http://www.elink.ibm.ibm.com/pbl/pbl/">http://www.elink.ibm.ibm.com/pbl/pbl/</a>

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.ibm.com/pbl/pbl/">http://www.elink.ibm.ibm.com/pbl/pbl/</a>

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.ibm.com/pbl/pbl/">http://www.elink.ibm.ibm.com/pbl/pbl/</a>

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.



---

## List of abbreviations

<b>ACEE</b>	access control environment element	<b>DLL</b>	dynamic link library
<b>ADK</b>	application development toolkit	<b>DML</b>	data manipulation language
<b>ADO</b>	ActiveX Data Objects	<b>DMS</b>	database managed storagespace
<b>ANN</b>	artificial neural network	<b>DPropR</b>	DataPropagator Relational
<b>ANSI</b>	American National Standards Institute	<b>DRDA</b>	distributed relational database architecture
<b>APAR</b>	authorized program analysis report	<b>DUW</b>	distributed unit of work
<b>APPC</b>	advanced program to program communication	<b>DW</b>	data warehouse
<b>API</b>	application programming interface	<b>DSS</b>	decision support system
<b>AR</b>	application requester	<b>EIS</b>	executive information system
<b>ARM</b>	automatic restart manager	<b>ESO</b>	expanded service option
<b>ASCII</b>	American National Standard Code for Information Interchange	<b>ERP</b>	enterprise resource planning
<b>BV</b>	Business View	<b>FTP</b>	File Transfer Protocol
<b>CAE</b>	client application enabler	<b>GBP</b>	group buffer pool
<b>CAF</b>	call attachment facility	<b>GID</b>	group ID
<b>CBIPO</b>	custom-build installation process offering	<b>GUI</b>	graphical user interface
<b>CBPDO</b>	custom-build product delivery offering	<b>GWAPI</b>	Domino Go Web server application programming interface
<b>CCSID</b>	coded character set identifier	<b>HLQ</b>	high level qualifier
<b>CFRM</b>	coupling facility resource management	<b>H-OLAP</b>	hybrid OLAP
<b>CLIST</b>	command list	<b>HTML</b>	hypertext markup language
<b>DAM</b>	data access module	<b>IBM</b>	International Business Machines Corporation
<b>DARM</b>	data archive retrieval manager	<b>ICAPI</b>	internet connection application programming interface
<b>DASD</b>	direct access storage device	<b>ICF</b>	integrated coupling facility
<b>DBMS</b>	database management system	<b>IDS</b>	intelligent decision support
<b>DBRM</b>	database request module	<b>IFI</b>	instrumentation facility interface
<b>DB2PM</b>	DB2 performance monitor	<b>IRLM</b>	internal resource lock manager
<b>DCE</b>	Distributed Computing Environment	<b>ISO</b>	International Organization for Standardization
<b>DCL</b>	data control language	<b>I/O</b>	input/output
<b>DDF</b>	distributed data facility	<b>IM</b>	Intelligent Miner
<b>DDL</b>	data definition language	<b>IMS</b>	Information Management System
<b>DES</b>	Data Encryption Standard	<b>IT</b>	information technology
		<b>ITSO</b>	International Technical Support Organization

<b>JCL</b>	job control language	<b>RRSAF</b>	recoverable resource manager services attachment facility
<b>JDBC</b>	Java Database Connectivity		
<b>JDK</b>	Java Developers Kit	<b>RUW</b>	remote unit of work
<b>JIT</b>	just in time compiler	<b>SCA</b>	shared communication area
<b>JRE</b>	java runtime environment	<b>SDSF</b>	system display and search facility
<b>JVM</b>	Java Virtual Machine		
<b>LE</b>	Language Environment	<b>SEU</b>	source entry utility
<b>LIS</b>	large item set	<b>SMP</b>	symmetrical multiprocessor
<b>LOB</b>	large object	<b>SMP/E</b>	system modification program/enhanced
<b>LPP</b>	licensed program product		
<b>LRO</b>	linked reporting object	<b>SMS</b>	storage management system
<b>LRU</b>	last recently used	<b>SNA</b>	systems network architecture
<b>MDIS</b>	Metadata Interchange Specification	<b>SQL</b>	Structured query language
		<b>SQLDA</b>	SQL descriptor area
<b>MLP</b>	multilayer perceptron	<b>SWA</b>	scheduler work area
<b>M-OLAP</b>	multidimensional OLAP	<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>MPP</b>	massive parallel processing		
<b>NCF</b>	IBM Network Computing Framework	<b>UDA</b>	user defined attribute
		<b>UDB</b>	Universal Database
<b>ODBA</b>	IMS open database access	<b>UDF</b>	user defined function
<b>ODBC</b>	open database connectivity	<b>UDT</b>	user-defined type
<b>OEM</b>	original equipment manufacturer	<b>URL</b>	Universal Resource Locator
<b>OLAP</b>	online analytical processing	<b>VSAM</b>	Virtual Storage Access Method
<b>OLE</b>	object linking and embedding	<b>VWP</b>	Visual Warehouse Program
<b>OLTP</b>	online transaction processing	<b>XES</b>	MVS Cross-system extended services
<b>OMG</b>	Object Management Group		
<b>OSA</b>	open systems adapter	<b>XMI</b>	XML Metadata Interchange
<b>PSM</b>	persistent stored module	<b>XML</b>	extended markup language
<b>PSP</b>	preventive service planning		
<b>RACF</b>	OS/VS2 MVS Resource Access Control Facility		
<b>RAM</b>	random access memory		
<b>RBA</b>	relative byte address		
<b>RBF</b>	radial basis function		
<b>RBFN</b>	radial basis function network		
<b>RDBMS</b>	relational database management system		
<b>ROI</b>	return on investment		
<b>R-OLAP</b>	relational OLAP		
<b>RDS</b>	relational data system		
<b>RRS</b>	recoverable resource manager services		



---

# Index

## Numerics

5250 emulation 171  
5250 tools 170

## A

ADO 169  
ALLOCATE statement 28, 30  
    definition 31  
    sample 32  
ALTER 3  
Ambiguous names 38  
Assignment statement 18  
    definition 20  
    sample 20  
    SET 18  
ASSOCIATE statement 28, 30  
    definition 31  
    sample 31  
ATOMIC compound statement 27  
    Nested 37  
    sample 27  
Authorization behavior 37  
    OS/390 120

## B

BASIC 16  
Bind  
    dynamic invocation 110, 111  
bind option  
    PATH 3  
Build 76

## C

C 3, 7, 10, 11, 14, 15, 16, 17, 31, 42, 43  
    comment 38  
C++ 3, 7, 16, 18  
CALL statement 1, 17, 28, 30  
    definition 31  
    sample 31  
CASE statement 18  
    definition 21  
    sample 21  
character variable  
    length 38  
CLI 3, 4, 7, 11, 30, 37, 42  
CLOSE 28, 30  
COBOL 3, 6, 7, 10, 14, 15, 16, 17, 31, 43  
COMMENT ON 28  
COMMIT 37  
Compound statement 18  
    ATOMIC  
        definition 27  
        sample 27  
    BEGIN 18  
        definition 27

    END 18  
    NOT ATOMIC  
        definition 27  
        sample 27  
    order of statements 27  
condition handler 27, 33, 36  
CONTINUE  
    definition 34  
EXIT  
    definition 34  
RESIGNAL statement 26  
SIGNAL statement 25  
    specification 18  
UNDO  
    definition 34  
CONNECT 37  
CREATE 3, 28  
    Create 75  
CREATE MODULE 9  
CREATE PROCEDURE 5, 38, 116, 117, 119, 127, 138  
    AS/400 169, 170, 171, 177, 180  
CURRENT PATH 3

## D

Database Connection 77  
DataJunction 40, 41  
DATE arithmetic 38  
DB2 Connect 60, 61  
DB2 SDK 61, 64, 65, 81, 105  
DB2CLI.PROCEDURES 3  
DB2DARI 5  
DB2SPB.INI file 66  
DBRM 119, 122, 127, 128, 130, 131, 138  
DECIMAL data types 38  
DECLARE CURSOR 28  
DECLARE GLOBAL TEMPORARY TABLE 28  
DECLARE PROCEDURE 5  
DECLARE RESULT SET LOCATOR 30  
DELETE 28  
Dirty Procedures 77  
DRDA 2, 3, 60  
DROP 3, 28  
DROP PROCEDURE  
    AS/400 180, 181  
DSNDPSM 112  
DSNHSQL 110, 118, 136, 137, 140, 143  
DSNPSMX1 112  
DSNPSMX2 112  
DSNPXMOX1 112  
DSNPSM 112  
DSNTIJSJ 113  
DSNTIJSQ 110, 111, 112, 113  
DSNTPSMP  
    BUILD 129  
    DESTROY 130  
    input parameters 126  
    PL/1 client program 131

- sample 131
- REBIND 131
- REBUILD 131
- REXX 110, 120
- DSNWLMP 110
- DSNWSPM 124, 125
- dynamic calls 37
- Dynamic SQL
  - definition 25
  - EXECUTE 25
  - EXECUTE IMMEDIATE 25
  - PREPARE 25
  - sample 25

## E

- error code
  - 060 39
  - 061 39
  - 775 39
  - 776 39
  - 777 39
  - 778 39
  - 779 39
  - 780 39
  - 781 39
  - 782 39
  - 783 39
  - 785 39
- error messages 39
- EXECUTE 25, 28
- EXECUTE IMMEDIATE 25, 28
- external stored procedure 7, 10, 11, 14, 16, 17, 42

## F

- FETCH 28, 30
- Filter 82, 84
- FOR statement 18, 37
  - definition 23
  - sample 23
- Fortran 3, 7, 16

## G

- Generate 75
- Get Source 76
- GOTO 38
- GRANT 3, 28, 37

## H

- Handling errors
  - definition 33
  - NOT FOUND 33
  - sample 35
  - SQL EXCEPTION 33
  - SQLSTATE 33
  - SQLWARNING 33
  - WHENEVER statement 33
- Handling result sets 29
  - DECLARE CURSOR 29

- sample 29
  - WITH RETURN TO CALLER clause 29
  - WITH RETURN TO CLIENT clause 29
- host variables 19

## I

- IBM DataJoiner 41
- IBM VisualAge for Java 62
- IF statement 18
  - definition 21
  - sample 21, 22
- IMS
  - DBCTL 3
    - Open Database Access 3
  - InfoModelers InfoModeler 40
  - Informix 40, 41, 43, 55
  - INSERT 28
  - INVSDK2LMS 186, 187
  - ISO/ANSI 1, 9, 10
  - ITERATE statement 18, 24, 38
    - definition 22
    - sample 22

## J

- Java 7, 10, 16, 31, 37, 42, 43
  - stored procedure 17
- JDBC 4, 7, 37, 169, 174, 188, 190

## K

- KEEPDARI 61

## L

- LABEL ON 28
- LEAVE statement 18
  - definition 22
  - sample 22
- LOCK TABLE 28
- LOOP statement 18
  - definition 24
  - sample 24

## M

- Microsoft 6
- Microsoft SQL Server 40, 43, 45
- Microsoft Visual Basic 62, 65
- Microsoft Visual Studio 62, 64
- Migrating
  - business logic 41
  - control statements 43
  - database data 41
  - database structure 40
  - tools 40, 41
- Module 9
- module 2, 7, 10, 18, 36, 49
- multi-rowed result sets 37

## N

Nested  
  ATOMIC compound statement 37  
  NOT ATOMIC compound statement 37  
  stored procedure 37  
New line markers 38  
NOT ATOMIC compound statement 27  
  Nested 37  
  sample 27

## O

ODBC 3, 4, 37, 42, 169, 174  
OPEN 28  
Operations Navigator 6  
Oracle 6, 9, 40, 41, 42, 48, 49, 50  
  FORMS 41  
OS/390 Procedures Processor 112  
OS/390 SQL Procedure Processor 109

## P

parameter names  
  length 38  
PASCAL 16  
PATH 3  
Persistent Stored Module 6, 9  
PL/SQL 6, 9, 15, 40, 42, 48  
Platinum ERwin ERX 40  
PowerBuilder 41  
PREPARE 25  
PREPARE FROM 28  
Processing result sets 32  
  sample 32  
Project 75  
  sharing SPB projects 82

## Q

QSQLSRC 171

## R

readme file 110  
Register 76  
RELEASE 29  
RELEASE SAVEPOINT 29  
RENAME 29  
REPEAT statement 18  
  definition 24  
  sample 24  
RESIGNAL statement 19, 37  
  definition 26  
  sample 26  
result sets 2, 37  
Retrieving result sets 30  
  sample 30  
REVOKE 3, 29, 37  
REXX 3, 7, 16  
  DSNTPSMP 110, 120  
  language support 110, 111

  stored procedure 116  
  stored procedure support 110, 111  
REXX stored procedure 5  
ROLLBACK 29, 37  
ROLLBACK TO SAVEPOINT 29  
RPG 6  
Run 76  
RUNSQLSTM 6, 170, 171, 172, 174, 182, 183

## S

SAVEPOINT 29  
Savepoint 36  
SDK2LMS 171, 173, 175, 178, 179, 180, 183, 186, 188  
SELECT INTO 29  
SET statement 38  
SIGNAL statement 19, 37  
  definition 25  
  sample 26  
Single statement procedure 38  
SmartGuide 78, 79, 85, 86  
sored procedure  
  processing return sets 37  
SPB  
  concepts and terminology  
  Build 76  
  Create 75  
  Database Connection 77  
  Dirty Procedures 77  
  Generate 75  
  Get Source 76  
  Modify 76  
  Project 75  
  Register 76  
  Run 76  
  configuring your environment 65  
  CREATE PROCEDURE 83, 88, 92  
  DB2 parameter  
  KEEPDARI 61  
  DB2SPB.INI file 66  
  entries 72  
  sample 66  
DRDA 60  
DSNTPSMP 93, 94, 102  
IBM VisualAge for Java 62  
JDBC stored procedure  
  sample 59  
  managing projects 82  
  Microsoft Visual Basic 65  
  Microsoft Visual Studio 64  
  pre-requisites 60  
  programming languages supported 58  
  sharing projects 82  
  SQL Costing Information 111  
  SQL stored procedure  
  sample 58  
  SQLJ stored procedure  
  sample 59  
  supported tasks 7  
  tasks  
  actual costs 90

- building stored procedures 101
- copying and pasting stored procedures 104
- creating new stored procedures 85
- debugging stored procedures 105
- modifying existing stored procedures 102
- viewing existing stored procedures 83
- SQL Assistant 74, 76, 78, 79, 86, 94, 96, 97, 99, 100, 101
- SQL Conversion Workbench 42
- SQL Conversion Workbench 41
- SQL Costing Information 111, 124
- SQL function 7
- SQL local variables
  - declaration 19
  - sample 19
- SQL Procedures
  - OS/390
    - Declared Temporary Table 111
    - External Savepoint 111
    - Identity Columns 111
  - portability across DB2 platforms 36
  - restrictions 36
  - SPB 38
  - statements supported 18
  - stored procedure builder 17, 18
- SQL Script 176, 177, 179, 180, 181
- SQL script 170, 178
- SQL stored procedure 7
  - AS/400
    - Operations Navigator GUI 170
    - Operations Navigator SQL 170
    - Traditional 5250 170
  - creating result sets 29
  - declaring SQL local variables 19
  - handling errors 33
  - migrating from OEM DBMSs 42
  - OS/390
    - Method 1 109
    - Method 2 109
    - Method 3 109
    - SPB 109
  - processing result sets 32
  - retrieving result sets 30
  - source code size limit 36
- SQL Windows 41
- SQL/PSM 1, 9, 14, 19, 36, 44, 49, 55, 7
- SQL\_API\_FN 5
- SQL3 7, 9, 10
- SQLCODE 37
- SQLDA 3, 5
- SQLJ 4, 7
- SQLPROCS 171
- SQLSTATE 26, 33, 37, 39, 40, 49
- SQLVARs 3, 4, 5
- Static DDL 38
- stored procedure
  - address spaces 2
  - evolution
    - DB2 for Distributed Platforms 3

- DB2 for OS/390 2
- DB2 UDB for AS/400 5
- parameter
  - IN 5
  - INOUT 5
  - OUT 5
- parameter style
  - DB2DARI 4
  - DB2GENERAL 4
  - DB2SQL 4
  - GENERAL 4
  - GENERAL WITH NULLS 4
  - JAVA 4
- returning result sets 37
- stored procedure monitor program 124
- STRSEU 171
- Sybase 9, 40, 41, 43, 45
  - APIs 42
- SYSCAT.PROCEDURES 4, 5
- SYSFUNCS 174
- SYSIBM.SYSPARMS 119, 129, 130, 137, 138
- SYSIBM.SYSPROCEDURES 116, 117, 119, 120, 129, 130, 137, 138, 145, 160, 167
- SYSIBM.SYSPROCOPTIONS 145
- SYSIBM.SYSPROCPARMS 145, 160
- SYSIBM.SYSPROCPARMSOPTIONS 145
- SYSIBM.SYSPSM 112, 113
- SYSIBM.SYSPSMOPTS 112, 113, 119, 120, 129, 130, 142
- SYSIBM.SYSPSMOUT 112, 114
- SYSIBM.SYSROUTINES 119, 129, 130, 137, 138
- SYSPARMS 170, 174, 181
- SYSPROCS 170, 174
- SYSROUTINES 170, 174, 181

## T

- T/SQL 6, 9, 15, 40, 42, 43, 48
- TCP/IP 2

## U

- UPDATE 29

## V

- VALUES INTO 29
- variables
  - declaration order 19
- Visual Basic 15
- VisualAge for Java 62
- VisualAge Remote Debugger 79

## W

- WHILE statement 18
  - definition 24
  - sample 24
- WLM
  - DDNAMES 127
  - stored procedure address space 2

---

## ITSO rebook evaluation

Developing Cross-Platform DB2 Stored Procedures: SQL Procedures and the DB2 Stored Procedure Builder  
SG24-5485-00

Your feedback is very important to help us maintain the quality of ITSO rebooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

Which of the following best describes you?

**Customer**    **Business Partner**    **Solution Developer**    **IBM employee**  
 **None of the above**

**Please rate your overall satisfaction** with this book using the scale:  
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction \_\_\_\_\_

**Please answer the following questions:**

Was this rebook published in time for your needs?      Yes\_\_\_ No\_\_\_

If no, please explain:

---

---

---

---

What other rebooks would you like to see published?

---

---

---

**Comments/Suggestions:      (THANK YOU FOR YOUR FEEDBACK!)**

---

---

---

---

---

**SG24-5485-00**

**Printed in the U.S.A.**

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

**SG24-5485-00**

