
CHAPTER 1

Introduction

The **DB2 Universal Database** (DB2 UDB) refers to a family of database products:

- DB2 for Linux, UNIX, and Windows (distributed platforms, also referred to as **DB2 LUW**)
- DB2 for iSeries
- DB2 for z/OS (also referred to as **DB2 zSeries**)

Although each of these products have their own code base (which has been highly optimized for its intended hardware), application development and SQL support is fairly consistent across the platforms. All of these database products also support the DB2 SQL Procedural Language (SQL PL) but to varying degrees.

In the first edition of this book, we focused only on distributed platforms and have received very positive feedback on the approach we took to teach the SQL PL language. In this second edition, we expand our focus to include the rest of the DB2 family.

You can appreciate the difficulties of discussing a language in the context of three different development environments. Each platform has unique optimizations, and calling out the subtle (but important) differences can make it difficult to keep the discussion concise. We've worked hard to make this text an easy and enjoyable read. To that end, the approach we take is to describe each SQL PL feature as supported on DB2 for Linux, UNIX, and Windows platforms, and highlight the differences that may exist on the other platforms.

Before we begin, let's set up an SQL PL development environment.

Installing DB2

The CD included with this book includes a trial version of DB2 UDB Enterprise Server Edition version 8.2 for Windows.

The install program is located in <CDROM>:\db2\setup.exe.

If you are new to DB2, start by reading through Appendix A, “Getting Started with DB2.” After you install DB2, create the DB2 SAMPLE database (also explained in Appendix A) because the examples presented in this book will make use of it. The SAMPLE database is provided as part of the DB2 (Linux, UNIX, and Windows) installation. For DB2 iSeries and DB2 zSeries, the CD contains scripts to create the SAMPLE database so that all examples can be demonstrated in a consistent environment.

The CD also provides all code samples listed in this book in directories associated with each platform by chapter. For example, the code snippets for DB2 LUW for Chapter 4 are located in the file <CDROM>:\samples\LUW\LUWchpt4CD.sql.

Now that you’ve installed the necessary files, lean back, and read on to begin your journey...

History of Stored Procedures

The use of databases has progressed rapidly in the last 20 years. As the amount of data collected by companies has increased, so have the demands on application developers to make use of it.

When databases were originally used, all the processing was performed on large mainframes with the output being sent to dumb terminals. There was no concern about where the application processed the data, because the data always resided on the server. This process changed when databases began to appear on mid-range UNIX machines, where the client and server were often separate. **Stored procedures** were created to allow data processing to occur on the much faster servers, to reduce the workload and CPU bottlenecks on the slower clients, and to reduce the amount of data sent across the network.

In those days, DB2 stored procedures were primarily developed using the C programming language, which gave developers greater flexibility in how they could manipulate data. This flexibility, however, came with a price—writing C procedures was a complicated and error-prone process. Developers had to be highly knowledgeable in both C and embedded SQL, which was often a difficult combination to find.

This issue created a demand for an easier method to write stored procedures and led to the creation of a new third-generation (3GL) programming language. The language was based on the existing SQL syntax and used a simple structured programming language very similar to early BASIC. **DB2 SQL Procedural Language** (DB2 SQL PL) was born. This new language enabled programmers to quickly develop and build stored procedures without having to know any complex programming languages or data structures. The ease of development led to an explosion in the use of stored procedures, as both developers and database administrators quickly learned how to work with this new simplified programming language.

DB2 SQL PL for stored procedures was originally available on DB2 for AS/400, which is now referred to as **DB2 UDB for iSeries**. From there, DB2 for mainframe (now called **z/OS** or **zSeries**) and DB2 for Linux, UNIX, and Windows adopted the language as well.

A Brief Introduction to Stored Procedures, Triggers, and Functions

SQL PL is usually associated with stored procedures and incorrectly referred to as “DB2 Stored Procedure Language.” The proper meaning of DB2 SQL PL is “DB2 SQL Procedural Language.” Furthermore, SQL PL is not used just for stored procedures. Database triggers, user-defined functions (UDFs), and dynamic compound SQL are also developed using this language, and this book will show you how.

Stored procedures, triggers, and functions are a class of objects called **database application objects**. Application objects encapsulate application logic at the database server rather than in application-level code. Use of application objects help reduce network traffic. In the case of stored procedures, for example, only the original request and the final output need to be transmitted between the client and the server.

Triggers are useful for enforcing business rules consistently and automatically across many applications that share a database.

Functions are useful for simplifying application development by encapsulating commonly used calculations or data lookups.

All of these objects help improve application runtime performance and can take advantage of the larger number of CPUs and disks that typically reside on a database server.

Use of database application objects is very popular because of their many benefits:

- **Improved manageability.** Application objects are modular and can be moved from database to database.
- **Clear division of roles between DBA and application developer.** The DBA knows the data model best. The application developer knows application interfaces best. Complex business logic can be encapsulated in database application objects. Application developers only need to know which procedures or functions to call.
- **Increased performance.** Keeping business logic in stored procedures residing on the server will help improve the performance of most data-driven applications. Instead of an application having to send data back and forth across a network for every SQL statement and cursor, a single call can be made with results returned at the end just once.

There are almost always opportunities to improve application performance using database application objects. Newer applications using three-tier or (n-tier) architectures were designed to reduce the cost of creating and maintaining applications. Practical experience, however, has shown that the performance of keeping all business logic in the application-server tier can have a large performance impact.

We have consistently witnessed that applications can benefit significantly by moving data-intensive logic from the middle tier back into the database tier. Application developers have discovered that while moving all business logic to the middle tier makes for very clean design, some classes of data processing should remain at the database tier for even reasonable performance. Use of SQL procedures, functions, and triggers has therefore continued to gain popularity in modern applications.

What's New in the Second Edition

If you are familiar with IBM Press books, your first question may have appropriately been, “Where was the first edition?” *DB2 SQL PL, Second Edition: Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS* is actually based on the book *DB2 SQL Procedural Language for Linux, UNIX, and Windows* (©2002). The change of book title was motivated by the need to create greater awareness of the DB2 SQL Procedural Language by using the term DB2 SQL PL, and deliver value to a broader audience by increasing the scope of the content to cover SQL PL for the entire DB2 family—not just Linux, UNIX, and Windows. Hence, the second edition builds on content from *DB2 SQL Procedural Language for Linux, UNIX, and Windows*.

Since the first edition, significant enhancements have been delivered in DB2 UDB version 8.2 for LUW (or DB2 UDB version 8.1 with FixPak 7) and have warranted an updated book. Following is a summary of the new features on DB2 UDB 8.2 for LUW:

- Native support for SQL procedures on Linux, UNIX, and Windows (the C compiler is no longer required)
- Enhanced SQL for greater application efficiency and performance
- Enhanced SQL `Table` function support to allow SQL PL logic as well as `UPDATE`, `INSERT`, `DELETE`, and `MERGE` statements
- Ability to call stored procedures from inline SQL PL blocks, functions, and triggers
- Support for nested save points
- Session-based locking (`SET LOCK WAIT` and `SET LOCK NO WAIT`)
- A new SQL procedure `REOPT` option, which causes a procedure to re-optimize with the latest available statistics at runtime
- Increased length of SQL procedure statements from 64KB to 2MB
- Ability to change procedure building prepare options using utility procedures `SET_ROUTINE_OPTS` and `GET_ROUTINE_OPTS`
- Enhanced `GET ROUTINE` and `PUT ROUTINE` commands for easier deployment

New versions of other DB2 products have also been released. Our previous edition of this book did not discuss these platforms.

NOTE

This book covers the latest features available in DB2 for iSeries V5R3 and DB2 for zSeries version 8.

New features in DB2 for iSeries V5R3 (as compared to V5R2) include

- Support for `SEQUENCE` objects
- Several new built-in functions for encryption, date/time manipulation, and string manipulation
- Enhancements to the `GET DIAGNOSTICS` statement
- Enhancements to the `DECLARE CURSOR` statement

New features in DB2 for zSeries version 8 (as compared to version 7):

- The length of SQL procedure statements have increased from 32KB to 2MB
- Handling of SQL conditions has been enhanced (for example, the `RESIGNAL` statement is now supported)
- The DB2 Development Center Integrated SQL Debugger can be used to debug DB2 for zSeries stored procedures

We updated all the chapters and added new ones to teach you how to leverage these new features of SQL PL.

DB2's SQL Procedural Language (SQL PL)

SQL PL is a subset of the SQL Persistent Stored Modules (SQL/PSM) language standard. The specification of the current SQL/PSM standard can be found in ANSI/ISO/IEC 9075-4:1999 Information Technology, Database Language SQL, Part 4: Persistent Stored Modules (SQL/PSM).

This standard is the basis for the structured programming languages used with SQL to write stored procedures and functions. The standard combines the ease of data access of SQL with the flow control structures of a simple programming language. It gives developers the ability to create compound SQL statements and procedures that only need to be coded once for multiple platforms.

Other Stored Procedure Languages

All the major database vendors support their own versions of an SQL procedural language. Each language supports the same core SQL commands, but each has its own unique implementation. Microsoft's SQL Server and Sybase's procedural languages are quite similar and are called **T-SQL** (for Transact-SQL). Informix uses **Informix SPL** (a stored procedure language), which is a 3GL similar to DB2 SQL PL. Oracle's procedural language is called **PL/SQL**.

A number of references on the Web can guide you in converting your stored procedures from other languages to stored procedures for DB2. A list of these references can be found in Appendix G, “Additional Resources.”

SQL PL Development Tools—DB2 Development Center

The **DB2 Development Center** (see Figure 1-1) is a unified development environment for creating stored procedures using SQL PL for the entire DB2 family. SQL functions can also be written using the Development Center for Linux, UNIX, and Windows platforms.

You can find a tutorial on how to use the Development Center in Appendix D, “Using the DB2 Development Center.”

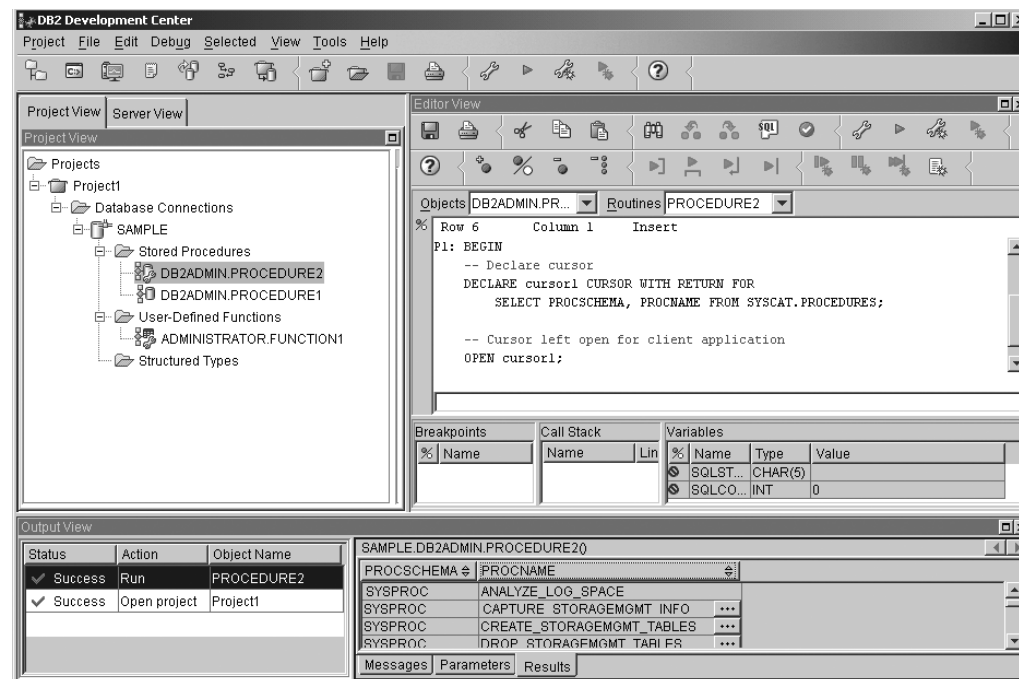


Figure 1.1 The Development Center.

The **iSeries Navigator for Windows** is a common development tool for the iSeries platform. Appendix A, “Getting Started with DB2,” provides a brief introduction to this tool.

Other development tools can also be used with DB2, but are beyond the scope of this book. We still encourage you to explore these other tools:

- DB2 Universal Database Add-in for Microsoft Visual Studio.NET
- DB2 Universal Database development plug-in for Eclipse and IBM WebSphere Studio Application Developer (WSAD)

Book Conventions

As you read this book, it will be useful to be aware of the conventions, styles, and structure used throughout.

Syntax Description

All SQL syntax diagrams throughout this book follow style that is consistent with DB2's official product documentation. The syntax diagrams are all read from left to right and top to bottom following the path of each line.

The >>--- symbol indicates the beginning of a syntax diagram.

The ---> symbol indicates that the syntax is continued on the next line.

The >--- symbol indicates that the syntax is continued from the previous line.

The ---< symbol indicates the end of a syntax diagram.

Syntax fragments start with the |--- symbol and end with the ---| symbol.

Required items appear on the horizontal line (the main path).

```
>>required_item-----><
```

Optional items appear below the main path.

```
>>required_item--+-+-----+-----><
                    '-optional_item-'
```

If an optional item appears above the main path, that item has no effect on execution, and is used only for readability and or backward-version compatibility.

```
                    .-optional_item-.
>>required_item--+-+-----+-----><
```

If you can choose from two or more items, they appear in a stack.

If you must choose one of the items, one item of the stack appears on the main path.

```
>>required_item--+-+required_choice1+-----><
                    '-required_choice2-'
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
>>required_item--+-+-----+-----><
                    +-optional_choice1+
                    '-optional_choice2-'
```

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
                    .-default_choice--.
>>required_item--+-+-----+-----><
                    +-optional_choice+
                    '-optional_choice-'
```

An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

```

      .-----
      v           |
>>-required_item---repeatable_item+-----><

```

If the repeat arrow contains a comma, you must separate repeated items with a comma.

```

      .-,-----
      v           |
>>-required_item---repeatable_item+-----><

```

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

Keywords appear in uppercase (for example, *FROM*). They must be spelled exactly as shown. Variables appear in lowercase (for example, *column-name*). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sometimes a single variable represents a larger fragment of the syntax. For example, in the following diagram, the variable *parameter-block* represents the whole syntax fragment that is labeled *parameter-block*:

```

>>-required_item--| parameter-block |-----><
parameter-block:
|---parameter1-----+-----|
' -parameter2---parameter3--- '
' -parameter4- '

```

Adjacent segments occurring between large bullets (*) may be specified in any sequence.

```

>>-required_item--item1--*--item2--*--item3--*--item4-----><

```

The previous diagram shows that *item2* and *item3* may be specified in either order. Both of the following are valid:

```

required_item item1 item2 item3 item4
required_item item1 item3 item2 item4

```

All DB2 tokens appear in uppercase and variable names in italic.

A common naming convention is used throughout the guide for all the examples. You are not required to label your variables this way, but it will make debugging and maintaining your code easier.

Book Conventions

9

p_parameterName: SQL procedure input and output parameters
v_parameterName: SQL variables
c_cursorName: Cursors

Meanings of Style

A number of different styles are used to indicate items of interest throughout the guide.

Code

```
CREATE PROCEDURE intro1 (OUT p_output INT)
LANGUAGE SQL
BEGIN
    SET p_output = 1;
END
```

TIP

This sidebar indicates a useful tip in the context of the current topic.

NOTE

This sidebar indicates a note to draw attention to important information.

Book Structure

Each successive chapter in the book is designed to be a prerequisite for the chapters that follow it. At the beginning of every chapter is a summary of the terms and subjects that will be covered. The summary allows experienced users to check whether they already understand the material that will be covered in the chapter.

Contacting the Authors

We are always looking for any feedback that you have both about this book and about DB2 SQL procedures. Please contact us with your opinions and inquiries at

db2sqlpl@ca.ibm.com

Depending on the volume of inquiries, we may be unable to respond to every technical question but we'll do our best.

The DB2 newsgroup at `comp.databases.ibm-db2` is another great way to get assistance from IBM employees and the DB2 user community.

Finally, for the latest information on this book, including updates and errata, be sure to visit

www.chak.ca/publications/sqlpl

Have fun!