IBM DB2 Universal Database

What's New

Version 5.2

S04L-6230-00

IBM DB2 Universal Database

What's New

Version 5.2

S04L-6230-00

Before using this information and the product it supports, be sure to read the general information under Appendix G, “Notices” on page 203.

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in U.S. or 1-800-IBM-4YOU in Canada.

Contents

Welcome to DB2 Universal Database and DB2 Connect Version 5.2

The need for better ways to access and manipulate data has driven the evolution of database management systems, from simple file processing systems to the newest generation of relational database management systems. In a world that seems to grow smaller every day, it's important to have a database that can embrace as much of that world as possible: a database that is truly universal. With DB2, IBM gives you a database that is:

- **Universally applicable:** to data warehousing, decision support, data mining, online transaction processing (OLTP), and online analytical processing (OLAP).
- **Universally scalable:** from laptops to desktops to workgroups to the largest enterprise systems.
- **Universally accessible:** from a wide range of clients, across computer platforms, and over the Internet.
- **Universally usable:** through graphical tools for controlling database functions.
- **Universally extensible:** with enhanced SQL and DB2 commands, in addition to DB2 Extenders for text, image, audio, video, and more.

DB2 has become synonymous with open, industrial-strength database management for business intelligence, transaction processing, and a broad range of applications for all types of businesses. Each new release of DB2 Universal Database builds on the strong foundation of the previous version. This book describes the new DB2 functions and enhancements available with DB2 Universal Database and DB2 Connect Version 5 and Version 5.2.

Note: Version 5 information is provided for customers who do not currently have DB2 Universal Database or DB2 Connect Version 5 installed. If you are already a Version 5 user, you can skip the Version 5 sections (see “Conventions” on page xiii for information on how to easily identify those sections).

Who Should Read This Book

This book is for current owners of DB2 who are upgrading or considering upgrading from one of these:

- DB2 Version 2 or Database Server Version 4.
- DB2 Parallel Edition Version 1.2 (which was available on AIX only).
- DB2 Universal Database Version 5.
- DDCS Version 2 or DB2 Connect Version 5.

If you are considering DB2 Universal Database or DB2 Connect for the first time, you should start by reading the Quick Beginnings book for your operating system to get basic DB2 information.

How This Book Is Structured

This book starts with an overview of some of the major DB2 enhancements for Version 5 and Version 5.2 and then describes these new features and enhancements.

Note: New features that were available as part of DB2 Universal Database Version 5 FixPaks and DB2 Connect Version 5 FixPaks are considered Version 5.2 enhancements.

Chapter 1, DB2 Universal Database and DB2 Connect Upgrades, describes the main DB2 product enhancements available with DB2 Universal Database and DB2 Connect Version 5 and Version 5.2.

Chapter 2, Accessibility, highlights how access to DB2 has been enhanced to support an expanding range of clients and platforms, including the Web.

Chapter 3, Extensibility, outlines improvements that extend DB2's capabilities as a database management system through SQL enhancements, new and improved commands and API's, and additional application development support.

Chapter 4, Performance, describes the DB2 enhancements that will your queries and applications run faster.

Chapter 5, Scalability, describes how DB2 continues to expand its support for users from standalone workstations through to multiprocessor environments.

Chapter 6, Usability, outlines the product enhancements that make DB2 Universal Database easier to use with each new release.

Chapter 7, Serviceability, describes improvements when it comes to solving DB2 Universal Database problems.

Appendix A, System Monitor Guide and Reference Updates, describes new information that is available from the database system monitor.

Appendix B, Call Level Interface Guide and Reference Updates, describes new application development information that is available using the DB2 Call Level Interface.

Appendix C, Embedded SQL Programming Guide Updates, describes new application development information that is available using embedded SQL.

Appendix D, Building Applications for Windows and OS/2 Environments Updates, describes new application development information for the Windows and OS/2 operating systems.

Appendix E, Web Control Center and NetQuestion, describes the new Web Control Center and the NetQuestion search facility.

Appendix F, How the DB2 Library Is Structured, describes the DB2 library; including books and online help.

Appendix G, Notices, contains notice and trademark information.

Conventions

You will find this book easier to use if you look for these conventions:

	**Version 5**	This icon marks new functions and enhancements available with DB2 Universal Database Version 5.
	**Version 5.2**	This icon marks new functions and enhancements available with DB2 Universal Database Version 5.2.

Chapter 1. DB2 Universal Database and DB2 Connect Upgrades

This section provides a brief summary of the enhancements for the latest versions of DB2 Universal Database and DB2 Connect: Version 5 and Version 5.2. More detailed information is provided in the sections that follow, including references to associated items in other parts of the DB2 library.

Note: Some portions of the DB2 Universal Database library have not been updated for Version 5.2. These include:

- *Administration Getting Started*
- *Building Applications for Windows and OS/2 Environments*
- *CLI Guide and Reference*
- *Embedded SQL Programming Guide*
- *Master Index*
- *SQL Getting Started*
- *System Monitor Guide and Reference*
- *Troubleshooting Guide*

In situations when a document has not been reissued for DB2 Universal Database Version 5.2, then the details of any enhancements that might affect it are included in this document (*What's New*). You can then use the *What's New* in conjunction with the Version 5 documentation for DB2 Universal Database Version 5.2.

What's New in Version 5

Version 5

DB2 is a relational database management system that is Web-enabled with Java support; scalable from single processors to symmetric multiprocessors; and multimedia capable with image, audio, and text support. Version 5 of DB2 Universal Database is the follow-on product to DB2 Version 2 and Database Server Version 4. DB2 Universal Database Extended Enterprise Edition Version 5 is the follow-on product to DB2 Parallel Edition Version 1.2 (which was available on AIX only). It includes all the features of DB2 Version 2 and Database Server Version 4, in addition to the Version 5 enhancements. This section describes some of the major changes for this version.

Exploitation of symmetric multiprocessors (SMP) for improved scalability and performance
DB2 now exploits the ability of an SMP system to share resources across multiple

processors, performing your SQL queries more quickly. Commands like LOAD, BACKUP, and RESTORE also take advantage of the multiprocessor environments.

New graphical tools on OS/2 and Windows 32-bit operating systems that make it easy to install, configure, and administer DB2 databases
From the Control Center, you can accomplish just about any administrative task. A number of SmartGuides walk you through common tasks, such as configuring communications and creating tables. You can also use the graphical tools to administer any DB2 server on any of the available platforms.

From the Client Configuration Assistant, you can configure communications for clients to access remote or local DB2 servers.

Comprehensive online help and the Information Center help you easily find the information you need to use DB2.

Extensions to SQL, including support for Online Analytical Processing (OLAP)
With the new CUBE and ROLLUP aggregations, you can now create super groups, like overall total and cross tabulation, for improved online analytical processing. You can also perform outer joins, rename tables, define unique constraints on tables, and create your own table functions. DB2 has refined its authorization support, so you can grant authority at a more granular level; you can also create schemas to grant various attributes and privileges.

Performance, capacity, and memory improvements
DB2 now provides global caching for SQL statements, creating a public repository that improves performance. You can create multiple buffer pools of various sizes to better control the data in memory. Client/server communications are more efficient, crash and roll-forward recovery is faster, and LOAD features numerous performance enhancements.

Security enhancements, including support for Open Software Foundation's Distributed Computing Environment (DCE)
You can now use the DCE architecture to manage users, passwords, and groups more easily, and authenticate users more securely. DB2 also provides a *Trusted Clients* option so you can choose whether to trust all clients or only those that come from an operating system with inherent security.

Additional support for communicating with host databases using DB2 Connect and the Distributed Relational Database Architecture (DRDA)
In addition to communicating with host systems using SNA, you can now use TCP/IP to communicate with host systems that support this protocol. In addition, DB2 servers can accept requests from host systems using TCP/IP, allowing you to use your DB2 workstation server as an application server to a host application.

Enhancements to application programming for DB2
The DB2 Call Level Interface (DB2 CLI) now reflects the Microsoft Open Database Connectivity 3.0 specifications, allowing you to connect to DB2 databases from ODBC 3.0 applications. Additions to user-defined functions (UDFs) include scrollable cursors and the UCT_UNIQUE function to return a unique value to use in a table column. You can also invoke external UDFs that are methods of object linking and embedding (OLE) automation servers.

Other new features
The DB2 Governor can be used to control application behavior.

New table space features include point-in-time roll-forward recovery and selective restore.

Error messages have been improved.

What's New in Version 5.2

Version 5.2

With Version 5.2, DB2 Universal Database continues the evolution of advanced database technology begun in Version 5. It delivers more client/server functions, more support for open industry standards, and improved performance and availability.

Expanded data type support
DB2 has several new features in the area of data types. To provide referential integrity and coordinated backup/recovery for data stored in files outside the database, DB2 provides the new data type DATALINK. To support larger integer values, DB2 now supports the data type BIGINT. DB2's object-relational functionality has been extended with new support for user-defined structured types, with subtyping and references, that can be used as the basis for defining tables of objects and object views. In addition, DB2 extender support is now available on all DB2 platforms.

Extensions to SQL
DB2 now supports data storage on 4 KB and 8 KB pages. You can now leave free space on pages, alter the length of a VARCHAR column, specify the qualifier for unqualified object references, and control the number of lines returned by a fetch.

Support for Java
You can use JDBC, Java, and SQLJ to access data from your DB2 databases. DB2's JDBC support lets you build Java applications and applets in a manner that is very similar to writing a C application to access the database using DB2 CLI or ODBC. SQLJ provides embedded static SQL for Java programmers and complements the dynamic SQL support provided with JDBC.

Administering databases over the Web
The DB2 Control Center now lets you perform your administration tasks over the Web. DB2's comprehensive documentation can also be accessed from the Web.

Improved system monitoring support for DB2 Connect
The LIST DCS APPLICATIONS and GET SNAPSHOT commands have new parameters for collecting information on DB2 Connect applications. These new options return additional information about each user of DB Connect Enterprise Edition, including such important items as the current state of the user connection and the time that state was entered. In addition new data elements have been added that monitor DB2 Connect

application activity. Now administrators can get an instant view of what each application user is doing.

International Support
DB2 provides support for bidirectional languages and the euro.

Year 2000 Ready

DB2 Universal Database Version 5 and Version 5.2 are Year 2000 ready. This means that when used in accordance with its associated documentation, DB2 Universal Database is capable of correctly processing, providing and/or receiving date data within and between the 20th and 21st centuries, provided that all products (for example, hardware, software and firmware) used with the product properly exchange accurate date data with it. More information about IBM and Year 2000 readiness can be found at `http://wwwyr2k.raleigh.ibm.com`.

Books Updated for Version 5.2

The table that follows lists the DB2 books that have been updated for Version 5.2 You can order them individually using the form number next to each book title or you can order them as a set by using the form number SBOF-8921-00. See Appendix F, “How the DB2 Library Is Structured” on page 193 for information on how to view, search, print, and order them.

Book	Form Number
Administration Guide	S10J-8154
API Reference	S10J-8167
Building Applications for UNIX Environments	S10J-8161
Command Reference	S10J-8166
DB2 Connect Enterprise Edition Quick Beginnings	S10J-7888
DB2 Connect Personal Edition Quick Beginnings	S10J-8162
DB2 Connect User's Guide	S10J-8163
DB2 Connectivity Supplement	No form number. Available in HTML and PostScript formats.
DB2 Extended Enterprise Edition for Windows NT Quick Beginnings	S10J-6713
DB2 File Manager Quick Beginnings	S04L-6231
DB2 Replication Guide and Reference	S95H-0999
Installing and Configuring DB2 Clients	No form number. Available in HTML and PostScript formats.
Messages Reference	S10J-8168

Book	Form Number
Quick Beginnings for UNIX	S10J-8148
What's New	S04L-6230

Chapter 2. Accessibility

Whether it's running on OS/2, Windows NT, or a UNIX system, DB2 Universal Database gives you the reliability and availability that you require. Client access is available from such popular operating systems as: Windows 95, Windows 98, Windows NT, OS/2, AIX, HP-UX, SCO UnixWare 7 and Solaris. Legacy data stored on host and AS/400 databases can be accessed with DB2 Connect.

DB2 Connect Enhancements

DB2 Connect provides a managed method to access databases stored on the following systems:

- DRDA server: DB2 for MVS/ESA, DB2 for AS/400, DB2 for OS/390, DB2 for VSE and VM systems
- DB2 Universal Database servers running on OS/2, Windows NT, and several UNIX systems.

DB2 Connect provides access to those databases in a cost-effective way by implementing a standard architecture for managing distributed data, known as Distributed Relational Database Architecture (DRDA). Use of DRDA allows applications to establish a fast connection to host and AS/400 databases without expensive host components or proprietary gateways.

Version 5

Using the TCP/IP Communications Protocol

TCP/IP has been added as a second protocol (the other being SNA) over which DB2 Connect can communicate to host systems. Also, DB2 servers are enabled to accept incoming DRDA connections from the host using TCP/IP. In other words, DRDA Application Requester (AR) and DRDA Application Server (AS) functionality has been implemented using TCP/IP in addition to SNA. See the *DB2 Connect Enterprise Edition Quick Beginnings* and the *DB2 Connect User's Guide* for details.

Two-Phase Commit

DRDA two-phase commit for the DB2 DRDA Application Requester (DB2 Connect) over the TCP/IP communications protocol has been implemented.

In addition, applications running on the host (for example, DB2 for MVS/ESA applications) using SNA can invoke the two-phase commit processing involving both the host and DB2 Version 5 databases. See the *DB2 Connect Enterprise Edition Quick Beginnings* and the *DB2 Connect User's Guide* for details.

See “Two-Phase Commit Enhancements for Version 5.2” on page 10 for additional enhancements to two-phase commit in Version 5.2.

Multi-Row Stored Procedures

The ability to execute stored procedures on DB2 for OS/390 is greatly improved. Applications can now receive multiple open cursors from stored procedures executing on DB2 for OS/390. Each cursor can have multiple rows. See the *DB2 Connect Enterprise Edition Quick Beginnings* and the *DB2 Connect User's Guide* for details.

DCE Security

Support is added for the Open Software Foundation's (OSF) Distributed Computing Environment (DCE) Security component for use in authentication of database users. This provides both a more secure authentication mechanism and more central management of users, passwords, and groups using the DCE architecture.

In addition, DCE Cell Directory support is available for DB2 for OS/2. The supported protocols are APPC, TCP/IP, NetBIOS, and IPX/SPX.

The restriction on AIX and other UNIX platforms that the SYSADM_GROUP must be the primary group of the instance owner is removed. See the *Administration Guide* for details.

Version 5.2

DCE Cell Directory Support and Host Systems

Users working with host and AS/400 databases servers now have additional options for providing database location information when using DCE Cell Directory support for implementations from IBM and Gradient. See the *Administration Guide* for details.

Enhanced Security Failure Notification

Users connecting to host and AS/400 databases can now get additional information on the cause of security failures when they occur, for example as the result of an expired password. See the *DB2 Connect Quick Beginnings* manuals for details.

Enhanced System/390 SYSPLEX Exploitation

DB2 Connect Enterprise Edition (EE), and the DB2 Connect component that is included in both DB2 UDB EE and DB2 UDB Extended - Enterprise Edition (EEE), can now provide enhanced load balancing and fault tolerance by routing connections to different nodes on a System/390 SYSPLEX. Some additional configuration considerations apply, and these are documented in the *DB2 Connect Enterprise Edition Quick Beginnings* manual.

Optimized Catalog Access for ODBC and JDBC Applications

A new tool db2ocat - DB2 ODBC Catalog Optimizer is provided on Windows 32-bit operating systems in order to help you optimize system catalog searches for ODBC applications. DB2 Connect now offers a way to dramatically improve the performance of ODBC and JDBC applications that make extensive use of the system catalog. This improvement is provided using the CLISCHEMA parameter in the `db2cli.ini` file, which allows applications to use an ODBC-optimized catalog instead of the regular system catalog tables. A point-and-click application that simplifies the creation and maintenance of ODBC-optimized catalogs can be obtained by downloading `db2ocat.zip` from `ftp://ftp.software.ibm.com/ps/products/db2/tools`.

New BIND Options

There are two new enumerated values for the DYNAMICRULES parameter of the BIND command. DEFINE and INVOKE are used to specify the authorization identity to be used for the execution of a dynamic SQL statement in a user-defined function (UDF) or stored procedure. See the *Command Reference* for details.

Microsoft Transaction Server Support

DB2 family databases (including host and AS/400 databases) can now fully participate in distributed transactions managed by the Microsoft Transaction Server (MTS). See "Microsoft Transaction Server" on page 24 and the *DB2 Connect Enterprise Edition Quick Beginnings* manual for details.

Simplified Password Management

DB2 Universal Database and DB2 Connect now provide the ability to manage passwords without requiring users to log on to database server machines. Passwords can be changed using the SQL CONNECT statement, by requesting a password change from the ODBC login dialog, or by using the ATTACH command. The ability to change user passwords is provided for Embedded SQL, ODBC, DB2 CLI, and Java (using JDBC and SQLJ).

For OS/390 users, if an SQL CONNECT statement returns a message indicating that the user ID's password has expired, it is now possible to change the password without signing on to TSO. Through DRDA, DB2 for OS/390 can change the password for you. An additional benefit is that with TCP/IP connections to the host, a separate LU definition is no longer required, as was the case with DB2 Connect Version 5.0. See the *DB2 Connect Enterprise Edition Quick Beginnings* manual for details.

Client Information Enhancements

The new sqleseti - Set Client Information API allows three-tier client/server or TP monitor applications to pass more specific information about the application end user to DB2 for OS/390 (see "Client Information" on page 21). The new information includes the end user name supplied by the server application, the workstation name, the application name, and the accounting string. This information can now be reported by the DB2 for OS/390 DISPLAY THREAD command and it is available in DB2 for OS/390 accounting records. Previously, in three-tier environments, DB2 for OS/390 could only provide information about the server application and the individual authentication user

ID, and not about the numerous end users who multiplex SQL queries on long-running connections. See the *API Reference* and “Transaction Processor Monitoring” on page 67 for details.

Bidirectional Language Support

DB2 Connect now provides complete support for working with host and AS/400 databases configured for bidirectional languages such as Arabic and Hebrew. See “Bidirectional CCSID Support” on page 26 and the *Administration Guide* for details.

Monitoring DB2 Connect Applications

The capabilities of the database system monitor have been expanded to collect more information on DB2 Connect applications. These enhancements include the ability to collect information on DCS applications running at the gateway, SQL statements being executed, and database connections. See “System Monitoring for DB2 Connect” on page 51 for more information.

Two-Phase Commit Enhancements for Version 5.2

In DB2 Connect Enterprise Edition Version 5.0, two-phase commit support over SNA connections using the DB2 Syncpoint Manager (SPM) was only available on AIX and OS/2 (see “Two-Phase Commit” on page 7). With DB2 Connect Enterprise Edition Version 5.2, this support is now extended to Windows NT. This support requires IBM eNetwork Communications Server for Windows NT Version 5.01 or higher (see “DB2 Connect for Personal Communications” on page 11).

Two-phase commit for XA applications was previously only supported over SNA connections, using the SPM. It is now also supported over TCP/IP connections using the SPM.

Applications executed by Transaction Processing Monitors such as IBM TXSeries, CICS for Open Systems, Encina Monitor, and Microsoft Transaction Server previously had to access host systems such as DB2 for OS/390 using SNA. With DB2 Connect Version 5.2, TCP/IP can now be used by these same applications. The DB2 Syncpoint Manager must be used to enable this new feature.

See the *DB2 Connect Quick Beginnings* manuals for details.

Simplified DB2 Syncpoint Manager Configuration

DB2 Syncpoint Manager configuration has been simplified. Many steps are now automated or eliminated compared to previous releases. See the *DB2 Connect Quick Beginnings* manuals for details.

Additional Data Objects and Types Supported

DB2 Connect Version 5.2 now provides support for big integer (BIGINT), large object (LOBs), and user-defined distinct data types (UDT). See the *SQL Reference* for details.

DB2 Connect for Personal Communications

DB2 Connect for Personal Communications is a component of IBM eNetwork Personal Communications for Windows 95 and Windows NT. It provides many of the functions available with the DB2 Connect Personal Edition with the following limitations:

- Native TCP/IP connectivity to AS/400 and host DB2 database is not supported.
- A maximum of 6 concurrent database connections can be open at the same time.
- The ability to update multiple databases as part of the same transaction (two-phase commit) is not supported.

IBM eNetwork Personal Communication customers that need these additional functions should upgrade to DB2 Connect Personal Edition or DB2 Connect Enterprise Edition products.

Additional Operating System Support

Version 5.2

SCO UnixWare 7

The DB2 Universal Database products listed below support the SCO UnixWare 7 operating system:

- DB2 Universal Database Workgroup Edition
- DB2 Universal Database Enterprise Edition
- DB2 Universal Database Developer's Edition
- DB2 Connect Enterprise Edition

See the *Quick Beginnings for UNIX* for information on installing DB2 Universal Database on SCO. For information on installing and configuring clients, see the *Installing and Configuring DB2 Clients* book. Information on the compilers and pre-compilers supported by the DB2 Universal Database Developer's Edition can be found in the *Building Applications for UNIX Environments* book.

Windows 98

The DB2 Universal Database products listed below support the Windows 98 operating system:

- DB2 Universal Database Personal Edition
- DB2 Connect Personal Edition

For information on installing and configuring clients, see the *Installing and Configuring DB2 Clients* book.

Windows NT

DB2 Universal Database Extended Enterprise Edition supports the Windows NT operating system. See the *DB2 Extended Enterprise Edition for Windows NT Quick Beginnings* for information on installing DB2 Universal Database on Windows NT. For information on installing and configuring clients, see the *Installing and Configuring DB2 Clients* book. Information on the compilers and pre-compilers supported by the DB2 Universal Database Developer's Edition can be found in Appendix D, "Building Applications for Windows and OS/2 Environments Updates" on page 149.

Solaris

DB2 Universal Database Extended Enterprise Edition supports the Solaris operating system. See the *DB2 Extended Enterprise Edition for UNIX Quick Beginnings* for information on installing DB2 Universal Database on Solaris. For information on installing and configuring clients, see the *Installing and Configuring DB2 Clients* book.

Client Application Enabler

The DB2 Universal Database Client Application Enabler Pack now includes Version 5.2 clients for SGI, SCO UnixWare 7, and Windows 98. See the *Installing and Configuring DB2 Clients* book for information on installing and configuring clients.

Note: DB2 Universal Database clients are also available in the Developer's Edition products.

Migration

Version 5

Migrating to Version 5

Support is provided to allow back-level DB2 databases and directories to be converted to a format usable by DB2 Universal Database Version 5. The following is a list of DB2 database releases that are supported in the DB2 Universal Database Version 5 database migration process:

- DB2 for OS/2 Version 1.x and Version 2.x
- DB2 for AIX Version 1.x and Version 2.x
- DB2 for HP-UX Version 2.x
- DB2 for Solaris Operating Environment Version 2.x
- DB2 for Windows NT Version 2.x
- DB2 for Windows 95 Version 2.x
- DB2 Parallel Edition for AIX Version 1.x to Version 5

See the Quick Beginnings manuals for details.

With the addition of all the new features in this release, some incompatibilities have been introduced. They are described in the *Quick Beginnings* manuals.

Version 5.2

Migrating to Version 5.2

Support is provided to allow back-level DB2 databases and directories to be converted to a format usable by DB2 Universal Database Version 5.2. The following is a list of DB2 database releases that are supported in the DB2 Universal Database Version 5.2 database migration process:

- DB2 for OS/2 Version 1.x and Version 2.x
- DB2 for AIX Version 1.x and Version 2.x
- DB2 for HP-UX Version 2.x
- DB2 for Solaris Operating Environment Version 2.x
- DB2 for Windows NT Version 2.x
- DB2 for Windows 95 Version 2.x
- DB2 Parallel Edition for AIX Version 1.x

See the Quick Beginnings manuals for details.

With the addition of all the new features in this release, some incompatibilities have been introduced, including:

- SQLCODE -311 will be returned when the length value of a varying length string is found to be greater than the maximum length allowed (the SQLLEN value). Prior to Version 5.2, SQLCODE -804 was returned.
- SQLCODE -804 will be returned if the SQLLEN value associated with a particular variable is negative. Prior to Version 5.2, SQLCODE -311 was returned.

For more information on incompatibilities see the *Quick Beginnings* manuals.

Note: No migration is required for DB2 Universal Database Version 5 customers moving to Version 5.2. However you should run the **db2upd52** command to enable Version 5.2's new functionality for your catalogs and existing databases. See the *Quick Beginnings* manuals for details.

Security

Version 5

Authentication

The *Trusted Clients* option is now enabled on all platforms when authentication type is CLIENT. It allows the administrator to choose whether to Trust All Clients (the default) or trust only those clients that come from systems where there is security inherent in the operating system. See the *Quick Beginnings* book for your operating system for details.

Communications

Version 5

Named Pipe Support in Windows

Named pipe is now a supported communication protocol for Windows 95 and Windows NT clients. It is also supported for DB2 Universal Database servers and DB2 Connect gateways on Windows NT.

The new command CATALOG NPIPE NODE lets you add a named pipe node entry to the node directory. The equivalent REXX API is provided. See the *Quick Beginnings for Windows NT*, *Command Reference*, and *API Reference* for details.

Version 5.2

Named Pipe Support in Windows 98

Named pipe support has been extended to Windows 98 clients (see “Named Pipe Support in Windows”).

Chapter 3. Extensibility

With each new release, DB2 extends its capabilities, which in turn expands the abilities of the end-user, administrator, and application programmer. DB2 Common Server included strong transaction processing. DB2 Parallel Edition could support complex queries and very large databases. DB2 Universal Database not only incorporates the best of DB2 Common Server and DB2 Parallel Edition, but also offers new capabilities that enable the processing and analysis required for today's business intelligence applications.

SQL Enhancements

Version 5

All enhancements comply with the SQL92 Entry Level standards. Features from higher levels of SQL92 and the future SQL3 have also been added.

See the *SQL Reference* for details about the enhancements described in this section.

Outer Join Support

A left, right, and full outer join operation is now supported using SQL92 syntax; that is, a join operation whose result includes unmatched rows in addition to matching rows.

Additional Authorization Support

The following functionality has been added for authorization support:

- Column level UPDATE privilege.
- Column level REFERENCES privilege.
- WITH GRANT OPTION on GRANT (for tables/views/columns).
- PUBLIC privileges for static SQL and views.

REAL Data Type Support

A single-precision floating-point data type using the keyword REAL is now supported. See "BIGINT Data Type" on page 18 for information on 64-bit integer support available in Version 5.2.

New CREATE SCHEMA and DROP SCHEMA Statements

A CREATE SCHEMA statement and a DROP SCHEMA statement are now supported. This allows privileges to be associated with the schema to control which users can create, alter, or drop objects in the schema.

See "CURRENT SCHEMA" on page 19 for information on the Version 5.2 special register that identifies the schema name used to qualify unqualified object references.

User-Defined Table Functions Supported

SQL users can now access data that is not stored in the relational format and can use of the query capabilities of the relational database.

It is often difficult if not impossible to subject data from non-relational data stores to relational operations. User-defined table functions are an extension to SQL that address this issue. A table function is an external user-defined function that constructs a derived table. The program for the function can access data from the various sources and format it into a tabular form that is returned from the table function. Once the table function is written, it can be used in the FROM clause of queries. Table functions can be used not only to subject this external data to the power of SQL, but also to capture external data permanently into relational tables.

See the *Embedded SQL Programming Guide* for details.

Unique Constraints Supported

Unique constraint support has been added as follows:

- Support for one or more UNIQUE constraints on tables in addition to PRIMARY KEYs.
- Foreign keys can reference unique constraints
- Unique constraint checking is deferred to the end of statement
- Specified constraint names used as index names (applies to primary keys also).

CUBE and ROLLUP Aggregations

The GROUP BY clause has been extended to support "super groups". One type of super group is a "ROLLUP group"; a result set that contains "sub-total" and "overall total" rows in addition to the regular grouped rows. Another type of super group is a "CUBE group"; a result set that contains "cross-tabulation" rows in addition to all the rows that would be in a ROLLUP group for the same columns.

New RENAME TABLE Statement

Support is now provided for renaming an existing table while maintaining current authorizations and indexes from the source table on the renamed table.

See “APPEND Parameter for ALTER TABLE” on page 36, “Increasing VARCHAR Column Length” on page 18, and “Table-Level Locks” on page 29 for Version 5.2 enhancements to the ALTER TABLE statement.

Friendly Arithmetic and Conversion

Friendly arithmetic and conversion allow a query to proceed and provide some returned results even though some data items could not be evaluated. This function enhances compatibility with DB2 for OS/390.

Built-in Functions

The following built-in functions are now available:

- COUNT_BIG

  Returns the number of rows or values in a set of rows or values for tables with a large number of rows. Use COUNT_BIG instead of COUNT when the value returned may exceed 2 billion.

- GENERATE_UNIQUE

  Returns a unique value that can be used to provide unique values in a table.

500 Table Columns

Up to 500 columns in a table are now supported on 4 KB pages. For information on Version 5.2 enhancements that support tables with up to 1012 columns see “Multiple Page Size Support” on page 19.

Version 5.2

DATALINK Data Type

DB2 is being enhanced with a new predefined data type, DATALINK. A DATALINK value in a database represents an object stored in a storage system outside of the database system. DB2 will treat the DATALINK value as if it were stored in the database, even though it is not. These means that the value is robust in terms of integrity, access control, and recovery.

The DB2 extension, called the DB2 File Manager, is important because it enables the asset management of files stored on file servers outside of the database management system. Using DATALINKs and the DB2 File Manager means that external files can be backed up with the database and SQL Data Control Language statements can be used to control permission to those files (for example, GRANT and REVOKE).

Users can create indexes on text, images, and videos, and store those attributes in relational tables along with the DATALINK value. The DATALINK value is a pointer or a uniform resource locator (URL) to an external file. The DB2 File Manager allows DB2 to treat this external data as if it was stored in the database.

DB2 File Manager is not available for Extended Enterprise Edition systems. The DATALINK data type can not be used on Extended Enterprise Edition systems to reference files on a DB2 File Manager.

DATALINKs does not support Windows 3.1 clients. Clients on other operating systems must be at the Version 5.2 level of DB2 Universal Database, except for AIX clients who can be on Version 5 at FixPak U453782 or higher.

Detailed information on the DB2 File Manager technology and the DATALINK data type can be found in the following:

- *DB2 File Manager Quick Beginnings*
- *Administration Guide*
- *API Reference*
- *Command Reference*
- *Messages Reference*
- *SQL Reference*
- Call Level Interface see “DATALINK” on page 76

BIGINT Data Type

An SQL data type of BIGINT is available for supporting 64-bit integers. As platforms introduce native support for 64-bit integers, the processing of large numbers with BIGINT is more efficient than processing with DECIMAL, and more precise than DOUBLE or REAL. This new data type allows:

- Tables and views to include BIGINT columns
- User-defined functions and procedures to pass and return BIGINT types
- Applications to define BIGINT host variables and retrieve data into 64-bit integer types (when supported by the programming language).
- The range for BIGINT is -9223372036854775808 to +9223372036854775807.

See “BIGINT” on page 75, “BIGINT Data Type” on page 99, the *SQL Reference* and the *Administration Guide* for details.

Increasing VARCHAR Column Length

The length of an existing VARCHAR column in a table can be increased to up to 4000 bytes. The ALTER *column-alteration* parameter for the ALTER TABLE statement allows for increasing the size of a VARCHAR column length. See the *SQL Reference* for details.

Free Space on Pages

A PCTFREE parameter has been added to the ALTER TABLE and CREATE INDEX statements. It is used to indicate the amount of free space left on each page. The free space is necessary to allow data to be inserted on a target page, instead of being appended to the end of a table. Free space is an important consideration when using clustering indexes (see “Clustering Indexes” on page 35). See the *SQL Reference* and the *Administration Guide* for details.

An INDEXFREESPACE parameter has been added to the LOAD command to specify the percentage of free space to leave on each index page when loading an index. See the *Command Reference* for details.

Replication of Long Fields

The Long Field Manager log records have been updated so that data capture capability can be extended to LONG VARCHAR/LONG VARGRAPHIC columns. See the *API Reference* for details.

An additional clause has been added to the DATA CAPTURE parameter of the ALTER TABLE statement to indicate when a LONG VARCHAR or LONG VARGRAPHIC column is included. See the *SQL Reference* for details.

Multiple Page Size Support

Information can be stored on page sizes of 4 KB and 8 KB. A page size of 4 KB supports table spaces of up to 64 GB and tables of up to 500 columns (see "500 Table Columns" on page 17). A page size of 8 KB allows table spaces of 128 GB, tables of 1012 columns, and row lengths of 8101 bytes. Page size is set during buffer pool creation. Once the buffer pool page size is fixed, then table spaces and tables can be created. See the *SQL Reference* and the *Administration Guide* for details.

CURRENT SCHEMA

The CURRENT SCHEMA special register contains the default qualifier to be used for unqualified object references for dynamic SQL statements issued within a specific DB2 connection. The qualifier can be changed by the SET SCHEMA statement. See the *SQL Reference* for details.

The QUALIFIER option of the BIND command controls the schema name used to qualify unqualified database object references for static SQL statements. See the *Command Reference* for details.

Note: For compatibility with DB2 for OS/390, CURRENT SCHEMA is synonymous with the CURRENT SQLID special register.

Typed Tables and Views

A new CREATE TYPE statement supports the definition of user-defined structured types that enhance DB2's object management capabilities. Subtyping is supported, so a structured type can either be created on its own or as a subtype of another structured type (thereby inheriting the attributes of that type). For example, an employee table might have some employees who are part of the subtype part-time. In this way, DB2 users can now create structured type hierarchies that are similar to class hierarchies in Java or C++. In addition, the creation of a structured type T also makes a corresponding reference type, REF(T), available for use as an attribute or column type when defining structured types, tables, and views.

The CREATE TABLE statement has been extended so that a structured type can be used as the basis for defining a table of objects of that type. Rows in such a table have an object ID column plus columns that correspond to each attribute of the type specified in the type-based version of the CREATE TABLE statement. To manage a table (or more properly, table hierarchy) that contains instances of a type plus one or more of its subtypes, the CREATE TABLE statement allows a typed table to be created UNDER another typed table; as a subtable of that table.

The CREATE VIEW statement has been similarly extended so that a structured type can be used as the basis for defining an object view or an object view hierarchy.

The SQL language has been extended with new functionality in its SELECT, INSERT, UPDATE, and DELETE statements to support queries and updates to table hierarchies (and subsets thereof), and the expression portion of SQL has been extended with support for a dereference operator (->) that enables users to traverse references using a C++ like path notation. More details on all of these new features can be found in the *SQL Reference*.

See “IMPORT and EXPORT” on page 21 and “RUNSTATS” on page 22 for commands that can be used on table hierarchies. See the *Administration Guide* and the *SQL Reference* for more information on structured types and table and view hierarchies. See “Defined Types” on page 76 for information about the Call Level Interface (CLI) impact of these extensions.

Command and API Enhancements

Version 5

LOAD and IMPORT

A new option has been added for handling decimal data. MODIFIED BY IMPLIEDDECIMALPOINT allows LOAD and IMPORT to insert an implied decimal point based on the column definition. Another new option, MODIFIED BY BINARYNUMERIC, allows all numeric data to be imported in binary or packed decimal format. See the *Command Reference* for details.

Table Space Recovery to a Point in Time

A selected subset of table spaces can now be rolled forward to a specified point in time. You can also restore a subset of the table spaces from a table space backup. See the *Administration Guide* and *Command Reference* for details.

Restore Subset of Table Spaces from Backup

You can now selectively choose the table spaces to be restored from the full database backup image. See the *Administration Guide* and the *Command Reference* for details.

Query Table Spaces

Various APIs and commands for querying table spaces and table space containers have been enhanced.

The LIST TABLESPACES command is enhanced to display the additional information when SHOW DETAIL is specified.

See the *Command Reference* and the *API Reference* for details.

FORCE

The FORCE command has been modified so that on a DB2 Universal Database Extended Enterprise Edition system, any application can be forced from any database partition using the agent ID (an identifier that uniquely identifies an application). See the *Command Reference* for details.

Version 5.2

Audit Function

The DB2 audit function generates, and allows you to maintain an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. These records can be used to detect and deter penetration of the DB2 system. With analysis, the records can reveal usage patterns that identify system misuse. For details on the audit function and the **db2audit** command see the *Administration Guide*.

Client Information

The sqleseti - Set Client Information API allows an application to set client information associated with a specific connection. A successful API call means that the client information has accepted and will be propagated on subsequent connections. See the *API Reference* for details.

The sqleqryi - Query Client Information API allows you to query current client information. If a specific connection is requested, then the latest values for that connection are returned. If all connections are specified, then the values associated with all connections specified by the last sqleseti API are returned. See the *API Reference* for details.

History File Information

Administration information is now available in the history file. The administrative events recorded will contain ROLLFORWARD, ALTER TABLESPACE, REORG, RUNSTATS, and DROP TABLE information. See the *API Reference* for details on obtaining this information with the sqluhops - Open Recovery History File Scan API. For information on extracting information with the LIST HISTORY command see the *Command Reference*.

IMPORT and EXPORT

New parameters have been added to provide support for transferring objects between table hierarchies (see “Typed Tables and Views” on page 19). Data residing in typed tables can be exported and imported as an entire table hierarchy, a subhierarchy, or a single typed table or subtable. See the *Administration Guide* for more details on table hierarchies.

See the *Command Reference* for information on the changes to the IMPORT and EXPORT commands. See the *API Reference* for information on the changes to sqluexpr - EXPORT and sqluimpr - IMPORT APIs.

LOAD and RESTORE

New parameters have been added to the LOAD and RESTORE commands extending support to the new DATALINK data type. See the *Command Reference* for details.

RECONCILE

The RECONCILE command validates the references to files for the DATALINK data in a table. See the *Command Reference* for details.

RUNSTATS

Statistics support has been extended to hierarchical data (see "Typed Tables and Views" on page 19). Statistics are still collected using the RUNSTATS command and updated through regular UPDATE commands on the system catalog tables. See the RUNSTATS command in the *Command Reference* and the sqlustat - Runstats API in the *API Reference* for details.

Log Sequence Number

The db2flsn - Find Log Sequence Number command allows users to determine which log file contains the log record identified by a given log sequence number (LSN). Locating an LSN is of particular interest to users of IBM DataPropagator Relational (DPROPR) and the sqlurlog - Asynchronous Read Log API. See the *Command Reference* for details.

Application Development Enhancements

Version 5

Open Database Connectivity 3.0 Support

The DB2 Call Level Interface (DB2 CLI) has been updated to the latest Microsoft ODBC 3.0 specifications. This enables ODBC 3.0 applications to run with DB2. Scrollable Cursor support has also been added. See the *CLI Guide and Reference* for details.

Additional Support in the DB2 Precompiler and APIs

The following support has been added:

- DB2 API for REXX called SQLDB2. It provides REXX support for new and existing DB2 APIs that do not return any data except for the SQLCA.

- The TARGET option of the DB2 PREP command and the precompiler API now support BORLAND_C and BORLAND_CPLUSPLUS to create DB2 applications with the Borland C++ compiler on OS/2.
- Support for the 32–bit version of Micro Focus COBOL on the OS/2 operating system.

See the *API Reference* for a full description of DB2 APIs. See the *Embedded SQL Programming Guide* for a description of the precompiler options.

User-Defined Functions

Enhancements have been made for table functions and scalar functions:

- A new CREATE FUNCTION specification of DBINFO | NO DBINFO enables the function creator to specify whether the UDF receives additional information as an argument.
- A new AS LOCATOR clause has been added to the parameter/result data type specified in the CREATE FUNCTION statement. It is valid for large object (LOB) types and for distinct types based on the LOB types. Instead of passing the entire LOB value across the DB2 / UDF interface, a locator is passed. Five new APIs are available to UDFs in NOT FENCED mode only, enabling the UDFs to directly manipulate the locators and (subsets of) the LOB values which they represent.

  Depending on how a UDF needs to process a LOB value, this can result in a great improvement in both performance and in memory utilization.
- UDF run-time support is extended with OLE automation (controller part) to invoke external UDFs that are methods of OLE automation servers. OLE automation is the standard protocol for interoperability on Windows operating systems.

See the *Embedded SQL Programming Guide* and the *SQL Reference* for details.

Precompile and Bind Enhancements

The precompile and bind enhancements are categorized as follows:

Long Host Variable Names

Long host variable names that correspond to the variable name length of the programming language are now supported. Host variable names can now be up to 255 characters in length for all DB2-supported languages.

SQL Statement Flagging

The grammar and all syntax rules (that do not require catalog access) of SQL statements are now checked and those not compliant with the SQL92 Entry Level standard are flagged.

Support for Java Programming

DB2 enables you to develop applications and applets that access and manipulate DB2 databases. DB2 makes this possible by providing support for the Sun Microsystem's Java Database Connectivity (JDBC) API. DB2 provides this support through a DB2 JDBC driver that comes with DB2. The JDBC API provides a standard way to access databases from Java code. Your Java code passes SQL statements as function argu-

ments to the DB2 JDBC driver. The driver handles the JDBC API calls from your Java code.

You can also use the Java programming language to develop user-defined functions and stored procedures which run on the server.

See the *Embedded SQL Programming Guide* and *DB2 Connect* documentation for details.

See “Extended Support for Java Applets” and “Support for JDBC and SQLJ” for information on Java support in DB2 Universal Database Version 5.2.

Version 5.2

Microsoft Transaction Server

Applications running under Microsoft Transaction Server (MTS) on Windows NT, Windows 95, and Windows 98 operating systems can use MTS to coordinate two-phase commit with multiple DB2 Universal Database databases and other MTS-compliant resource managers. See the *Administration Guide* and “Microsoft Transaction Server” on page 93 for details.

See the *DB2 Connect Enterprise Edition Quick Beginnings* manual for information on how host and AS/400 databases can participate in distributed transactions managed by MTS.

MTS support is enabled by setting the *tp_mon_name* configuration parameter (see the *Administration Guide*).

Extended Support for Java Applets

The DB2 Java Database Connectivity (JDBC) Applet Server can now be started as a Windows NT service. It must be registered as a service before it is made available in the services section of the control panel. See “JDBC Applet and Application Support” on page 102 and the *Quick Beginnings* documentation for details.

Support for JDBC and SQLJ

DB2 Universal Database provides support for Java Database Connectivity (JDBC) and for Java Embedded SQL (SQLJ).

JDBC support consists of:

- Support for Java UDFs and stored procedures on the server.
- Support for client applications and applets written in Java that use JDBC to access DB2.

See “Programming in JDBC” on page 101 for details.

SQLJ support allows you to run Java embedded SQL applications and applets on the client against DB2 databases on the server. It also provides support for Java embedded SQL UDFs and stored procedures on the server. See “Embedded SQL for Java (SQLJ) Programming” on page 117 for details.

Perl Interface

The Database Independent Interface for Perl (Perl DBI) is an Application Programming Interface (API) that provides database access for the Perl language. The DBI defines a set of functions, variables, and conventions that provide a consistent database interface independent of the actual database being used. A specific database driver is required to work in conjunction with the DBI in order to access a particular database.

In Version 5.2 of DB2 Universal Database, support for the Perl DBI is provided on the AIX, HP-UX, and Solaris platforms. This support is available with the DBD::DB2 database driver. The driver and documentation can be obtained through FTP and the Web. For more information, see `http://www.software.ibm.com/data/db2/perl`.

Configuration Parameters

Version 5.2

Number of Commits to Group (mincommit)

This parameter has been modified so that changes to this value take effect immediately. You no longer have to wait for all applications to disconnect from the database. See the *Administration Guide* for details.

Default Database System Monitor Switches (dft_monswitches)

This parameter has been modified so that changes to its values take effect immediately. You no longer have to manually stop and restart the database manager. See the *Administration Guide* and “System Monitor Switches” on page 72 for details.

National Language Support

Version 5.2

Bidirectional CCSID Support

Bidirectional (BiDi) layout transformations are implemented in DB2 Universal Database Version 5.2 using the new Coded Character Set Identifiers (CCSID) definitions. For the new BiDi-specific CCSIDs, DB2 Universal Database performs layout transformations instead of or in addition to code page conversions. The following cases exist:

- When using RA protocol (workstation-to-workstation connections), conversions are done by the DB2 Universal Database server or DB2 Connect gateway. Layout transformations will be performed by the same machine that does code page conversion.
- For DRDA connections code page conversions are performed by the receiver. Since host platforms may not support BiDi-specific CCSIDs at the same time as DB2 Universal Database, DB2 will provide layout transformations in both directions (DB2 Universal Database as the client or the server).

See the *Administration Guide* for details.

Euro Support

DB2 Universal Database is a EuroReady product. Code page 850 and the Microsoft Windows ANSI code pages have been modified to include the euro. DB2 Universal Database Version 5.2 uses these definitions by default. If you would like to continue to use the previous definition of these code pages, you should contact DB2 Universal Database Service. See the *Administration Guide* for more details.

DB2 Extenders

DB2 Extenders give you the ability to store, access, and manipulate text, images, audio, and video in a DB2 database.

Version 5.2

Partitioned Databases

DB2 Extenders Version 5.2 adds support for DB2 Universal Database Extended Enterprise Edition on Windows NT, Solaris, and AIX operating systems. When DB2 runs a query in parallel, any DB2 Extender user-defined function (UDF) in the query is run in

parallel on the individual partitions. The Query by Image Content (QBIC) feature of the DB2 Image Extender will manage the parallel operation of the QBIC UDFs. See the *DB2 Universal Database Extender Administration and Programming* guides for details.

Commands

DB2 Extender's support for DB2 Extended Enterprise Edition adds the following commands:

- REDISTRIBUTE NODEGROUP completes the redistribution process so that redistributed data can be used with DB2 Extenders.
- RECREATE QBIC CATALOG recovers a QBIC catalog on a specified node.
- DMBNCRT adds a node to a DB2 Extender instance.
- DMBNDROP drops a node from a DB2 Extender instance.
- DMBNLIST lists all the partitions for a DB2 Extender instance.

The following DB2 Extender commands have changed:

- START SERVER, STOP SERVER, and GET SERVER STATUS process all nodes for the connected database.
- DMBSTART and DMBSTOP start or stop all nodes or a specified node for the extender instance.
- DMBICRT creates a partitioned DB2 Extender instance.

See the *DB2 Universal Database Extender Administration and Programming* guides for details.

Migration

Users using DB2 Extenders Version 5 can migrate their data for use with DB2 Extenders Version 5.2, but the data will not be partitioned.

Net.Data

Version 5.2

Net.Data is an application that allows Web developers to easily build dynamic Internet applications using *Web Macros*. Net.Data Web Macros have the simplicity of HTML with the power of dynamic SQL. Net.Data Version 2 offers the following new features:

- **FastCGI:** combines the best aspects of Common Gateway Interface (CGI) and vendor API's to optimize Net.Data applications.
- **Cache Manager:** the new cache improves response time for end users.

- **Macro file enhancements:** capabilities have been extended by adding looping constructs and nested IFs.
- **Java:** new servlets help in the development and management of macros in Java environments.

Information on using Net.Data with DB2 can be found on the Web at `http://www.software.ibm.com/data/net.data`.

Legato

Version 5.2

The Legato NetWorker BusinesSuite Module for DB2 provides on-line, non-disruptive backup for DB2 Universal Database on AIX. In addition to providing high-speed on-line backup, customers will have the ability to centrally manage and back up multiple DB2 servers. The following key points apply:

- The BusinesSuite module for DB2 is 1.0.
- NetWorker server software is NW for UNIX 4.2.5 (or higher), and NW for Windows NT 4.4 (or higher).
- For invocation the user must use the LOAD option on BACKUP or RESTORE, specifying the liblgto.a library which is shipped in the `sqllib/lib` directory. For example:

```
db2 backup db sample LOAD /u/dmcinnis/sqllib/lib/liblgto.a
```

Note: This is an AIX only feature and the liblgto.a library is always installed in the db2instance's `sqllib/lib` directory.

Chapter 4. Performance

Applications not only need to be up, they need to be up and running-fast. That's why DB2 Universal Database improves its performance with each new release.

Query

Version 5

Index ANDing

The performance has been improved for queries that use columns which are key columns of different indexes over the same table. DB2 uses dynamic bitmap technology to efficiently combine multiple indexes.

Star Joins

The performance of queries involving *star joins* has been improved. Star queries are characterized as multiway joins between several small dimension tables and a large fact table. DB2's new star join algorithm exploits dynamic bitmaps to join a large fact table with a series of relatively small dimension tables, minimizing data I/O.

Version 5.2

Table-Level Locks

Users with the appropriate authority can now specify the size (granularity) of locks used when a table is accessed. By default row-level locks are used when tables are created. Changes to the ALTER TABLE statement allow locking to be pushed up to the table level. Using table-level locks may improve the performance of queries by reducing the number of locks that need to be obtained and released. See the *Administration Guide* and the *SQL Reference* for details.

Limit Fetch Size

The new Fetch First N Rows Only feature allows users to limit the size of the result set of a query to a specified value. This feature improves the performance of queries that have potentially large results, when only a limited number of rows are of interest. For example, a user might be interested in viewing only information on the 10 highest paid employees in an organization. In this case the user could issue a SELECT statement

with a FETCH of only the first 10 rows. See “Fetch-first-clause” on page 100 and the *SQL Reference* for details.

Retrieval Using Index-Only Access

Index-only access means that queries can be satisfied by accessing only the index, providing that the SELECT matches the included columns. See “Index-Only Access” on page 35 for more information.

Correlated Predicates

The DB2 Universal Database optimizer has been enhanced with regards to choosing an execution plan for queries that contain joins with more than one join predicate joining two tables. The dependence or independence of predicates can affect the performance of optimizer’s chosen plan. The new DB2_CORRELATED_PREDICATES registry variable helps the optimizer detect and compensate for the correlation of join predicates. When this variable is true, the optimizer will use the KEYCARD information of unique index statistics to detect cases of correlation and dynamically adjust the combined selectivities of the correlated predicates. This results in a more accurate estimate of the join size and cost. See the *Administration Guide* for details.

Summary Tables

Summary tables can be used to improve query performance. You can create a summary table that holds a derived result and keeps that result updated. For example, you could monitor the highest salary in your company (`select max(salary) from employee`) and keep that information up-to-date in a summary table. Then, whenever a user requests the highest employee salary, the result from the summary table is returned, rather than recalculating it from the employee table.

Summary tables can be created to hold the results of simple queries, or a collection of joins involving multiple tables. See the *Administration Guide* and the *SQL Reference* for details.

Note: The REFRESH IMMEDIATE option is not available in Version 5.2. It will return SQLCODE SQL0628N if used. You can use the REFRESH TABLE statement to update your summary tables.

Replicated Tables

DB2 Extended Enterprise Edition users can specify that the data stored in a table is physically replicated on each database partition of the nodegroup for the table space where the table is defined. Replicated tables are particularly useful for joins in which you have a large fact table and small dimension tables. A list of all customers or a list of all transactions (orders and sales) could be large fact tables, while a list of countries might be a small dimension table. If the customer table and the transaction table are stored on separate database partitions, and both are involved in joins with the country table, then the country table is a good candidate for replication in your multi-node environment. To create a replicated table, you use the CREATE TABLE statement with the REPLICATED parameter. See the *SQL Reference* and the *Administration Guide* for details.

Hash Joins

A hash join will first compare *hash codes* before comparing predicates for tables involved in a join. In a hash join, one table (selected by the optimizer) is scanned and rows are copied into memory buffers drawn from the sort heap allocation. The memory buffers are divided into partitions based on a hash code computed from the columns of the join predicates. Rows of the other table involved in the join are matched to rows from the first table by comparing the hash code. If the hash codes match, the actual join predicate columns are compared.

Hash join requires one or more predicates in the form `table1.columnX=table2.columnY`, and for which the column types are the SAME. For columns of type CHAR, the length must be the same. For columns of type DECIMAL, the precision and scale must be the same. The column type cannot be a LONG field column, or a large object (LOB) column.

For more details on hash joins see the *Administration Guide*. For information on monitoring hash joins see “Hash Join” on page 68.

Cache

Version 5

Global SQL Cache

The aim of the Global SQL Cache is to minimize the amount of catalog access required for sections of static SQL statements and to maximize the sharing of sections for dynamic DML statements by eliminating many of the previous restrictions. This is done by establishing a global cache shared by all agents connected to the same database or partition of a database (in the case of DB2 Universal Database Extended Enterprise Edition), in which sections for both static and dynamic SQL statements will be placed. This global cache acts as a public repository, or library, for different sections being used on the database at any given time.

Version 5.2

Package Lock Avoidance

The DB2_NO_PKG_LOCK registry variable allows the Global SQL Cache to operate without the use of package locks to protect cached package entries. In Version 5 package entries in the cache (while in use) were protected from being dropped or modi-

fied by cache-level locks. Since acquiring a lock is expensive in terms of performance, users now have the option of working in a no package lock mode. In this mode certain database operations will not be allowed in order to protect cached package entries (for example: DROP TABLE). See the *Administration Guide* for details.

Recovery

Version 5

Faster Restart

Enhancements have been made to speed up restart which means that database crash recovery is now faster. This enhancement also applies to roll-forward recovery.

Buffer Pools

Version 5

Multiple Buffer Pools

You can now create multiple buffer pools of various sizes (number of pages) and assign table spaces to them using the CREATE BUFFERPOOL SQL statement. This provides the database administrator greater control of the data in memory. See the *Administration Guide* for details.

Extended Storage

Extended storage provides a secondary level of storage for buffer pools. This allows a user to access memory beyond the maximum allowed for each process. Support now exists for very large physical memory (64-bit memory support). DB2 exploits 64-bit systems and 32-bit systems capable of supporting greater than 4 GB of real memory. See the *Administration Guide* for details.

Version 5.2

Page Size

DB2 Universal Database supports the storage of information on page sizes of 4 KB and 8 KB (see “Multiple Page Size Support” on page 19). In order to use the new 8K page size a buffer pool and attached table space with a page size of 8K must be created. See the CREATE BUFFERPOOL and CREATE TABLESPACE statements in the *SQL Reference* for details.

Fast Communications Manager Buffers

The DB2_FORCE_FCM_BP environment variable allows DB2 Extended Enterprise Edition for AIX customers to create their FCM buffers in a separate shared memory segment. This allows for faster communications (see “Multiple Logical Nodes” on page 36).

Commands

Version 5

LOAD

In addition to SMP exploitation of load, LOAD performance for DEL and ASC data has been improved for the simple column types CHAR and INT. Other performance improvements have been made for other data formats and column types.

LOAD now has a new MODIFIED BY option called FASTPARSE which provides further performance improvements. See the *Command Reference* for details.

Applications

Version 5

DB2 Governor

The DB2 Governor lets you set rules that indicate how it is to handle application behavior. For example, it monitors the resources used by executing applications and if specified limits are exceeded, it reprioritizes query execution or issues the force command to terminate them.

In addition, the governor can be used to generate accounting records for executing applications that can be used for chargeback accounting. See the *Administration Guide* for details.

Client/Server

Version 5

Deferred Prepare

Deferred prepare provides a performance enhancement when accessing DB2 and DRDA databases by combining the SQL PREPARE statement flow with the associated OPEN, DESCRIBE, or EXECUTE statement flow to minimize the inter-process or network flow.

When deferred prepare is enabled, DB2 defers sending out SQL PREPARE statements until the associated OPEN, DESCRIBE, or EXECUTE statement is issued by the application. See the *CLI Guide and Reference* for details.

A PREPARE that is not eligible for deferral can be sent immediately, and have its cursor opened at the same time. PREPARE statements that normally cannot be deferred or pre-OPEN can also be optimized if the appropriate option is set in the sqlesetc - Set Client API or precompiler option. See the *Command Reference* for details.

Version 5.2

Log File Storage

The *newlogpath* configuration parameter allows users to change the location where log files are stored. Log files can now be stored on raw devices (Windows NT, AIX, and Solaris). This can speed up database operation as there will be less overhead in code path length for each I/O call to read or write the log, and there is no need to initialize the new log path to guarantee the disk space. See the *Administration Guide* for details.

Indexes

Version 5.2

Clustering Indexes

A clustering index allows data records to be clustered on pages based on the sequence of a particular index, and maintains that clustering as much as possible over the course of insert and update activity. Clustering increases the efficiency of data retrieval when it involves accessing sequential value ranges for a particular index (the clustering index). Without a clustering index, data can still be arranged on pages based on a particular index (using REORG), but there is no mechanism to maintain the arrangement as data is added and removed without additional REORGs. See the *Administration Guide* and the *SQL Reference* for details.

Index-Only Access

The CREATE INDEX statement now allows users to specify additional columns to be appended to the set of index key columns. The new INCLUDE parameter identifies columns that are included in the index, but are not part of the unique index key. This means improved performance because queries can be satisfied by accessing only the index, without reading the base table. See the *Administration Guide* and the *SQL Reference* for details.

Tables

Version 5.2

APPEND Parameter for ALTER TABLE

An APPEND parameter has been added to the ALTER TABLE statement. It is used to indicate whether data is appended to the end of a table or inserted where free space is available. Specifying APPEND ON can improve performance, as it allows for faster inserts and eliminates the maintenance of free space information. See “Altering Tables” on page 100 and the *SQL Reference* for details.

Communications

Version 5.2

Virtual Interface Architecture Support

The Virtual Interface Architecture (VIA) communication architecture is supported for inter-node communications. VIA enables high-speed communication amongst clusters of Windows NT SMP machines. The DB2VIAENABLE registry variable specifies if the VIA communications protocol is used. See the *Administration Guide* for details.

Multiple Logical Nodes

Direct communication is available between nodes on the same host (multiple logical nodes). In previous releases the DB2 Fast Communication Manager used a dedicated communication daemon (FCM daemon) to service communications requests whether they were local or remote. Now the FCM daemon can be bypassed when the remote nodes reside on the same physical machine. This enables high-speed communication amongst multiple logical nodes for Extended Enterprise Edition users on AIX. The DB2_FORCE_FCM_BP environment variable allows direct communication between multiple logical nodes (MLNs). See the *Administration Guide* for details.

Chapter 5. Scalability

DB2 Universal Database supports you whether you are working on a local database or on a database that is distributed across the largest parallel system. That's important, because as you start using DB2 Universal Database you can be confident that if you need more capacity or processing power, you have a scalable database system that is capable of handling your growing needs.

Partitioned Databases

DB2 has extended the strength of its database manager to the parallel, multinode environment. DB2 Universal Database Extended Enterprise Edition allows a database to be partitioned across multiple, independent computers connected by a LAN. Each database partition or node (node was the term used in DB2 Parallel Edition for AIX) is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. To the end-user and application developer, the database still appears as a single entity on a single processor. There are two main benefits that this provides. First, this enables an application to use a database that is simply too large for a single processor to handle efficiently. Second, SQL operations can operate in parallel on the individual database partitions, thereby speeding up the execution time of a single query or utility.

There are two types of query parallelism:

- *Intra-partition parallelism* refers to the ability to break up a query into many parts. This usually means subdividing what is usually considered a single database operation, for example an SQL query, into multiple parts, many or all of which can be executed in parallel within a single database partition.
- *Inter-partition parallelism* refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one or more machines.

As your needs change you may find that your configuration is no longer appropriate. DB2 Universal Database lets you scale your configuration. Whether you are adding memory, storage, processors, database partitions, or changing environments (moving to SMP or MPP) DB2 can meet your current and future needs.

DB2 Universal Database offers many options on how best to match your hardware and application requirements with a specific DB2 product configuration.

Version 5

Single Partition on a Single Processor

This environment consists of memory and disk, but contains only a single CPU. It has been given many names: standalone database, client/server database, serial database, and single node/non-parallel environment.

Single Partition with Multiple Processors

This environment is typically made up of several equally powerful processors within the same machine (symmetric multiprocessor or SMP). Resources such as memory and disk space are shared. Machines with multiple processors have more memory and disks than those with a single processor.

Multiple Partitions Each with Its Own Processor

In this environment there are many database partitions, but each is on a separate machine and that has its own processor, memory, and disks. Machines are connected by a communication facility. This environment is known as a massively parallel processing (MPP) environment or a shared-nothing configuration.

Version 5.2

Multiple Partitions with Multiple Processors

In a multipartition environment each partition can have multiple processors. This configuration combines the advantages of SMP and MPP parallelism. Queries can be performed in a single partition across multiple processors, or they can be performed in parallel across multiple partitions. This environment is called an SMP cluster.

Windows NT and Solaris

In addition to AIX, DB2 Universal Database Extended Enterprise Edition is now available for the Windows NT and Solaris operating systems.

Migrating from DB2 Parallel Edition

Version 5.2

While enhancements to SQL commands and DB2 application programming interfaces (APIs) have been made and new commands and APIs have been added to support the parallel environment, most data manipulation language (DML) commands are unchanged and you need not change your programs or SQL statements to run in the parallel environment. *DB2 Extended Enterprise Edition Quick Beginnings* for your operating system provides installing, migrating, and configuring information.

The DB2 Parallel Edition Version 1.2 features that are now available in DB2 Version 5.2 are documented in the DB2 Version 5.2 product documentation. Some restrictions apply to the DB2 Parallel Edition Version 1.2 features that are available in DB2 Version 5.2. They are documented in the following manuals:

- *Administration Guide*
- *SQL Reference*
- *API Reference*
- *Command Reference*
- *System Monitor Guide and Reference*

Symmetric Multi-Processor (SMP) Enhancements

The multiple processors available in an SMP environment allow database operations to be completed significantly more quickly than with databases assigned to only a single processor.

Version 5

Intra-Partition Parallelism

The I/O parallelism provided in DB2 Version 2 has been enhanced. It now exploits CPU parallelism in a single SQL query. Technology is introduced which exploits multiple processors of a symmetric multiprocessor (SMP) to speed up the execution of a single SQL query. Prior to this work, SMP exploitation was limited to that between multiple queries and to parallel I/O. A key purpose of this technology is to exploit the fact that disks, memory, and processors can be shared uniformly by multiple processors in an SMP system. This provides two important benefits:

- It allows workload to be divided more evenly among the processors, thereby achieving better scalability. This is possible since all processors have the choice to

work on all data. This is analogous to a single queue serviced by multiple bank tellers.

- It provides flexibility in designing query execution strategies that is not possible with a shared nothing approach. For example, a correlated subquery can be executed by the same processor that executes the parent query.

See the *Administration Guide* for more information.

LOAD Utility

Support now exists for the LOAD utility to exploit multiple processors in SMP machines.

While loading data into a table in a multinode nodegroup, the data files must have been processed by the splitter utility (db2split), which writes a header to each file. Only ASC and DEL files may be used to load in a table that exists on a multinode nodegroup. In addition, you now have the option to pick a set of partitions, which may be the same or different from the partitions being loaded, to participate in the parallel split process. See the *Command Reference* for more information.

As well, you can specify the degree of LOAD parallelism desired for CPU and disk I/O. See the *Command Reference* and the *API Reference* for more information.

BACKUP and RESTORE Utilities

Backup and restore of multiple table spaces on SMP machines are performed in parallel. See the *Command Reference* for more information.

Version 5.2

Index Generation

Multiple processors of an SMP are now used to speed up the process of generating an index. An index can be generated with the CREATE INDEX statement, the addition of a primary key or unique constraint to a table, or the REORG TABLE command. See the *Command Reference* for more information.

Degree of Parallelism

If a query is run with DEGREE = ANY, the database manager chooses the degree of intra-partition parallelism based on a number of factors including the number of processors and the characteristics of the query. The actual degree used at run time may be lower than the number of processors depending on these factors.

The degree of parallelism is determined by the SQL optimizer when the statement is compiled and may be adjusted before query execution depending on the database activity. The degree of parallelism may be lower than that chosen by the SQL optimizer if the system is heavily utilized. This occurs since intra-partition parallelism aggressively

uses system resources to reduce the elapsed time of the query which may adversely affect the performance of other database users. See the *Administration Guide* for details.

Autoloader Utility

Version 5.2

In a partitioned database, large amounts of data are divided across many partitions. Partitioning keys are used to determine the particular database partition where each portion of the data resides. Data must pass through a splitting phase before it can be loaded at the correct database partition. This split and load process is accomplished by the Autoloader utility.

The Autoloader utility has been improved for users of DB2 Extended Enterprise Edition on AIX, and is now available for users of DB2 Extended Enterprise Edition on Windows NT and Solaris. operating systems

The Autoloader utility uses a hashing algorithm to partition the data into as many output sockets as there are database partitions in the nodegroup in which the table was defined. It then loads these output sockets concurrently across the set of database partitions in the nodegroup. A key feature of the Autoloader utility is that it uses sockets for all data transfer required in the split and load process. It also allows the use of multiple database partitions for the splitting phase, thereby improving performance significantly. See the *Administration Guide* for details.

Chapter 6. Usability

The strengths of a database management system are not obvious if that system is hard to use. DB2 Universal Database has significantly enhanced its user interface and administration tools to provide greater ease of use.

Graphical User Interface (GUI)

Version 5

Control Center

The Control Center (called Database Director in previous releases) has been significantly enhanced to provide greater ease of use. An administrator of a DB2 database is provided with the tools necessary to set up, manage, monitor, and tune the database. You can manage a stand-alone database, or multiple remote databases from a single point of control. In addition to management of database instances, databases, and table spaces, Control Center now includes management functions for schemas, tables, views, indexes, users, user groups, user-defined types, user-defined functions, triggers, and stored procedures, as well as the ability to discover databases on the systems being managed. The ability to explain SQL graphically has been enhanced.

Also new in Version 5 is a built-in Scheduler that allows a job to run unattended at a given time, every x hours / days / weeks / months, multiple times a week, or multiple times a month.

The Control Center now includes support for data replication setup, allowing registration of replication sources and definition of replication subscriptions. This function is part of the DB2 base engine.

Monitoring capabilities provide early warning of potential problems, or automated actions to correct problems discovered without human intervention.

The Control Center runs on OS/2, Windows NT, Windows 95, and Windows 98 operating systems, from which you can administer DB2 databases on OS/2, Windows NT, AIX, HP-UX, and the Solaris Operating Environment and other UNIX operating systems.

In addition to the function described above, there is support for the management of DB2 Universal Database Extended Enterprise Edition objects. There are additional performance monitoring views to monitor these objects.

See the online help that accompanies the database administrative tools for instructions for using the tools. See the *Administration Getting Started* for an introduction to DB2 database administration using the database administrative tools.

See "Web Control Center" on page 45 for information on the Control Center support available in Version 5.2.

SmartGuides

The Control Center has been further enhanced through the inclusion of SmartGuides, which step you through a task. SmartGuides are included for database performance configuration, create/restore/backup database, create table space, and create table.

Visual Explain

The new Portable Snapshot function in Visual Explain makes the explain snapshot from any operating system usable by Visual Explain on any other operating system. For example, an HP-UX snapshot can be viewed with a Windows NT client.

See the online help that accompanies Visual Explain for instructions on how to use the tool.

Command Center

The Command Center provides an interactive window that lets you perform tasks such as:

- Execute SQL statements, DB2 commands and operating system commands.
- See the execution result of the one or many SQL statements and DB2 commands in a result window. You can scroll through the results and save the output to a file.
- Save a sequence of SQL statements and DB2 commands to a script file. You can then schedule the script to run as a job. When a saved script is modified, all jobs dependent on the saved script inherit the new modified behavior.
- Recall and execute a script file.
- See the execution plan and statistics associated with an SQL statement before execution. You do this by invoking Visual Explain with a simple click of a button in the interactive window.
- Get quick access to the database administrative tools such as the Control Center and the Scheduler from the main tool bar.
- Display all the command scripts known to the system through the Script Center, with summary information listed for each.

For complete information on all the commands see the *Command Reference* and *SQL Reference*. The DB2 Client Application Enabler contains a subset of the Command Center functions. See the *Quick Beginnings* book for your operating system for details.

See the online help that accompanies the Command Center for instructions on how to use the tool.

The Command Line Processor (CLP) continues to be available on all operating systems.

Version 5.2

Web Control Center

The Web Control Center is the Java port of the DB2 Universal Database Control Center. It eliminates the need to install the Client Application Enabler on your administrative workstations and gives you the ability to administer DB2 Universal Database over an intranet.

The Web Control Center's design is almost identical to that of Version 5, but it provides you with a more flexible network centric administration environment. It is implemented as a Java applet that uses DB2's JDBC support. It runs from any Java-enabled Web browser that supports the Java Development Kit 1.1.5. See "Web Control Center Installation and Configuration" on page 185 for details on installing, configuring, and using the Web Control Center.

NetQuestion Search Service and the Web Control Center

The NetQuestion Search Service, in combination with the Web Control Center, gives easy access to the DB2 Universal Database information library. See "Enabling the Web Control Center Remote Documentation Searches" on page 190 for details on installing and using the search function.

Connectivity and Protocol Configuration and Auto-Discovery

Version 5

The task of connecting a client to databases on servers is now simplified. Many of the steps performed on the server and the client are now automated.

From the server, DB2 *auto-discovers* the protocols it has detected on the workstation. You select the appropriate protocol and tell DB2 to configure the server and client.

From the client, it is now much simpler to make a new database connectable. You can ask DB2 to auto-discover the databases available, and then you can select a database from that list and ask DB2 to configure it for connection.

Server Communications Configuration

From an instance in the Control Center, the Setup Communications option lets you configure a DB2 server instance and update the communications protocol settings to support connections to the instance's databases from remote clients. You can set up communications for a new instance, or maintain the communication configuration of an existing instance. Many of the required steps are automated. You simply need to select the communication protocols you wish to have the server instance support.

Server Communications Configuration is available on OS/2, Windows NT, Windows 95, and Windows 98 operating systems. See the *Quick Beginnings* book for your operating system for details.

Client Configuration Assistant

This assistant leads you through the steps necessary to configure and manage a DB2 client, while at the same time automating some of the steps required. Once a database has been configured for connection using either the automated or manual method, you can perform other actions on it:

- DB2 connection testing
- Connection information maintenance
- DB2 CLI/ODBC administration
- Application binding
- Database removal.

Connections can be configured to both DB2 Common Server databases and DRDA databases.

The Assistant is available on OS/2, Windows NT, Windows 95, and Windows 98 operating systems. The Client Configuration Assistant does not configure remote clients. See the *Quick Beginnings* book for your operating system for details.

Version 5.2

Client Configuration Assistant Enhancements for Version 5.2

The Client Configuration Assistant (CCA) can be used to configure TCP/IP connections to DB2 for VM and DB2 for AS/400 database servers. You can also use it to configure Communications Server for NT (CS/NT) and IBM Personal Communications SNA stacks if you are using an SNA network. See the *DB2 Connect Quick Beginnings* manuals for details.

Instance Profile Manager

Version 5

The DB2 environment variables are consolidated into the DB2 Instance Profile Registry.

This helps you by:

- Supporting multiple environment profiles (one per instance)
- Providing a machine-wide default (global) profile
- Supporting a machine-wide default DB2 instance name
- Eliminating the need to reboot the machine when a variable is modified
- Supporting user/application overrides
- Centralizing control of environment variables
- Providing a command line tool to modify the externalized variables
- Enabling remote administration via the command line and the graphical tools.

A new command, DB2SET, displays, sets, or removes DB2 profile variables. See the *Command Reference* for details.

Licensing

Version 5

The usability surrounding entering a license key has been improved. In addition, a database administrator can now enable or disable counting concurrent users on some operating systems (OS/2, Windows NT, Windows 95, and AIX). See the *Quick Beginnings* book for your operating system for details.

Chapter 7. Serviceability

Technical support for DB2 Universal Database is improved through the addition of new and more precise messages and enhancements to its problem solving tools.

Version 5

Version 5

DB2 Database Repair Tool

Enhancements have been made to the DB2 Database Repair Tool (DB2DART), an analysis and repair tool, which provides improved DB2 serviceability. See the *Trouble-shooting Guide* for additional details.

Messages

The messages have been improved so that the location of the reported problem is defined more precisely, and so that they are clearer and more helpful. See the *Trouble-shooting Guide* for additional details.

Version 5.2

Version 5.2

Service Level Tool

The db2level tool will return information on the level of DB2 Universal Database being run. It will include items such as:

- Internal build level (for example `c980310`)
- PTF number (for example `WR09024`).

This type of information will be helpful when users call DB2 Service.

Appendix A. System Monitor Guide and Reference Updates

The *System Monitor Guide and Reference* has not been updated for DB2 Universal Database Version 5.2. The following sections document any Version 5.2 changes and enhancements to the system monitor and should be used in conjunction with the Version 5 edition of the *System Monitor Guide and Reference* by Version 5.2 users. These include:

- "System Monitoring for DB2 Connect": You can now use the system monitor to collect more information on your DB2 Connect applications. New parameters have been added to the GET SNAPSHOT and LIST DCS APPLICATIONS commands that will return new and enhanced data elements that provide DB2 Connect application details.
- "New System Monitor Data Elements" on page 68: New data elements have been added to help you monitor hash joins, page reorganizations, and lock escalation.
- "Application Identification" on page 72: Application identification can be different depending on your version of DB2.
- "System Monitor Switches" on page 72: Certain DB2 configuration parameters can now be updated dynamically, including the default database system monitor switches.
- "Changes to Number of Log Pages Written" on page 73: The Number of Log Pages Written data element does not measure pages written by DB2.

System Monitoring for DB2 Connect

This section lists the system monitor data elements that can be monitored for DB2 Connect in DB2 Universal Database Version 5.2. These data elements fall into two groupings:

existing Data elements that were available in Version 5, but not for DB2 Connect. Starting with Version 5.2, these elements can be used to monitor DB2 Connect activity.

new Data elements with DB2 Connect implications that have been added to the system monitor in Version 5.2.

These DB2 Connect data elements (both existing and new) will be returned in the following data structures. These data structures also fall into two categories:

changed These data structures existed in Version 5, but have been modified in Version 5.2 to include data elements for DB2 Connect. They are:

- sqlm_db2
- sqlm_appl
- sqlm_dcs_applinfo.

new These data structures that return DB2 Connect data elements have been added to the system monitor in Version 5.2:

- sqlm_dcs_appl
- sqlm_dcs_appl_stats
- sqlm_dcs_appl_xid
- sqlm_dcs_applid_info
- sqlm_dcs_dbase
- sqlm_dcs_stmt
- sqlm_tpmon_info

Note: The data structures are defined in the **sqlmon.h** header file.

Changed Commands

The following commands associated with system monitoring have been enhanced to return more DB2 Connect information:

GET SNAPSHOT
Depending on the specified clause, information can be collected on:

- A specific DCS database
- All DCS databases
- All DCS applications
- All DCS applications currently connected to a specific DCS database
- A specific DCS application running on the DB2 Connect gateway.

LIST DCS APPLICATIONS
The EXTENDED parameter returns additional DCS information:

- DCS application status
- Status change time
- Client platform
- Client protocol
- Client code page
- Process ID of the client application
- Host coded character set ID (CCSID).
- Database alias at the gateway
- DCS database name

RESET MONITOR
The DCS parameter has been added to this command to indicate that monitor data should only be reset for DCS databases.

For command details see the *Command Reference*.

Existing Data Elements

The following table lists the Version 5 system monitor data elements that can collect DB2 Connect information starting in Version 5.2. These data elements will be returned in the data structures listed in the **API Structure** column in Version 5.2, in addition to their original Version 5 data structures.

Note: You should refer to the Version 5 System Monitor Guide and Reference for detailed information on these data elements.

Data Element	API Element Name	Level	API Structure
Database Identification and Status			
Database Name	db_name	DCS Database DCS Application	sqlm_dcs_dbase sqlm_dcs_applid_info
Application Identification and Status			
Application Handle (agent ID)	agent_id	DCS Application	sqlm_dcs_applid_info
ID of Code Page Used by Application	codepage_id	DCS Application	sqlm_dcs_applid_info
Application Status Change Time	status_change_time	DCS Application	sqlm_dcs_applid_info
User Login ID	execution_id	DCS Application	sqlm_dcs_applid_info
Client Process ID	client_pid	DCS Application	sqlm_dcs_applid_info
Client Operating Platform	client_platform	DCS Application	sqlm_dcs_applid_info
Client Communication Protocol	client_protocol	DCS Application	sqlm_dcs_applid_info
Previous Unit of Work Completion Timestamp	prev_uow_stop_time	DCS Application	sqlm_dcs_appl
Unit of Work Start Timestamp	uow_start_time	DCS Application	sqlm_dcs_appl
Unit of Work Stop Timestamp	uow_stop_time	DCS Application	sqlm_dcs_appl
Unit of Work Completion Status	uow_comp_status	DCS Application	sqlm_dcs_appl
Application Idle Time	appl_idle_time	DCS Application	sqlm_dcs_appl
SQL Statement Activity			
Failed Statement Operations	failed_sql_stmts	DCS Database DCS Application	sqlm_dcs_dbase sqlm_dcs_appl_stats
Commit Statements Attempted	commit_sql_stmts	DCS Database DCS Application	sqlm_dcs_dbase sqlm_dcs_appl_stats
Rollback Statements Attempted	rollback_sql_stmts	DCS Database DCS Application	sqlm_dcs_dbase sqlm_dcs_appl_stats
SQL Statement Details			
Statement Operation	stmt_operation	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt
Package Name	package_name	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt
Section Number	section_number	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt

Data Element	API Element Name	Level	API Structure
Application Creator	creator	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt
Statement Operation Start Timestamp	stmt_start	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt
Statement Operation Stop Timestamp	stmt_stop	DCS Application DCS Statement	sqlm_dcs_appl sqlm_dcs_stmt
SQL Dynamic Statement Text	n/a	DCS Statement	sqlm_dcs_stmt
Number of Successful Fetches	fetch_count	DCS Statement	sqlm_dcs_stmt
Query Cost Estimate	query_cost_estimate	DCS Statement	sqlm_dcs_stmt
Query Number of Rows Estimate	query_card_estimate	DCS Statement	sqlm_dcs_stmt
SQL Snapshot Monitor Elements			
Last Reset Timestamp	last_reset	DCS Database DCS Application	sqlm_dcs_dbase sqlm_dcs_appl

New Data Elements

This section lists all the new Version 5.2 data elements that can be monitored for DB2 Connect. You should refer to the Version 5 System Monitor Guide and Reference for instructions on how to collect information with the DB2 system monitor.

DCS Database Name

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
DCS Database	sqlm_dcs_dbase	Basic
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	dcs_db_name	
Element Type	information	
Related Information	• “Host Database Name” on page 55 • “Database Alias at the Gateway” on page 55	

Description: The name of the DCS database as catalogued in the DCS directory.

Usage: Use this element for problem determination on DCS applications.

Host Database Name

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Database	sqlm_dcs_dbase	Basic
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	host_db_name	
Element Type	information	
Related Information	• "DCS Database Name" on page 54 • "Database Alias at the Gateway" on page 55	

Description: The real name of the host database for which information is being collected or to which the application is connected. This is the name that was given to the database when it was created.

Usage: Use this element for problem determination on DCS applications.

Database Alias at the Gateway

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	gw_db_alias	
Element Type	information	
Related Information	• "DCS Database Name" on page 54 • "Host Database Name" on page 55	

Description: The alias used at the DB2 Connect gateway to connect to the host database.

Usage: Use this element for problem determination on DCS applications.

DB2 Connect Gateway First Connect Timestamp

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Database	sqlm_dcs_dbase	Basic
DCS Application	sqlm_dcs_appl	Basic
Resettable	No	
API Element Name	gw_con_time	
Element Type	timestamp	
Related Information	• None	

Description: The date and time of the first connection to the host database from the DB2 Connect gateway.

Usage: Use this element for problem determination on DCS applications.

Maximum Number of Concurrent Connections

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Database	sqlm_dcs_dbase	Basic
Resettable	No	
API Element Name	gw_connections_top	
Element Type	water mark	
Related Information	• "Total Number of Attempted Connections for DB2 Connect" on page 57 • "Current Number of Connections for DB2 Connect" on page 57	

Description: The maximum number of concurrent connections to a host database that have been handled by the DB2 Connect gateway since the first database connection.

Usage: This element will help you understand the level of activity at the DB2 Connect gateway and the associated use of system resources.

Total Number of Attempted Connections for DB2 Connect

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
DCS Database	sqlm_dcs_dbase	Basic
Resettable	No	
API Element Name	gw_total_cons	
Element Type	water mark	
Related Information	• "Maximum Number of Concurrent Connections" on page 56 • "Current Number of Connections for DB2 Connect" on page 57	

Description: The total number of connections for DB2 Connect attempted at the DB2 Connect gateway since the `db2start` command or the last reset.

Usage: This element will help you understand the level of activity at the DB2 Connect gateway and the associated use of system resources.

Current Number of Connections for DB2 Connect

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
DCS Database	sqlm_dcs_dbase	Basic
Resettable	No	
API Element Name	gw_cur_cons	
Element Type	gauge	
Related Information	• "Maximum Number of Concurrent Connections" on page 56 • "Total Number of Attempted Connections for DB2 Connect" on page 57	

Description: The current number of connections to host databases being handled by the DB2 Connect gateway.

Usage: This element will help you understand the level of activity at the DB2 Connect gateway and the associated use of system resources.

Number of Connections Waiting for the Host to Reply

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
DCS Database	sqlm_dcs_dbase	Basic
Resettable	No	
API Element Name	gw_cons_wait_host	
Element Type	gauge	
Related Information	• "Current Number of Connections for DB2 Connect" on page 57 • "Number of Connections Waiting for the Client to Send Request" on page 58	

Description: The current number of connections to host databases being handled by the DB2 Connect gateway that are waiting for a reply from the host.

Usage: This value can change frequently. It should be sampled at regular intervals over an extended period in order to obtain a realistic view of gateway usage.

Number of Connections Waiting for the Client to Send Request

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
DCS Database	sqlm_dcs_dbase	Basic
Resettable	No	
API Element Name	gw_cons_wait_client	
Element Type	gauge	
Related Information	• "Current Number of Connections for DB2 Connect" on page 57 • "Number of Connections Waiting for the Host to Reply" on page 58	

Description: The current number of connections to host databases being handled by the DB2 Connect gateway that are waiting for the client to send a request.

Usage: This value can change frequently. It should be sampled at regular intervals over an extended period in order to obtain a realistic view of gateway usage.

Elapsed Time Spent on DB2 Connect Gateway Processing

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl	Statement
DCS Statement	sqlm_dcs_stmt	Statement
Resettable	Yes (at application) No (at other levels)	
API Element Name	gw_exec_time	
Element Type	time	
Related Information	• None	

Description: The time (in microseconds) at the DB2 Connect gateway to process an application request (since the connection was established), or to process a single statement.

Usage: Use this element to determine what portion of the overall processing time is due to DB2 Connect gateway processing.

Number of SQL Statements Attempted

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Database	sqlm_dcs_dbase	Basic
DCS Application	sqlm_dcs_appl	Basic
Resettable	Yes	
API Element Name	sql_stmts	
Element Type	counter	
Related Information	• Snapshot Time	

Description: The number of SQL statements that have been attempted since the latter of: application start up, database activation, or last reset.

Usage: Use this element to measure database activity at the database or application level. To calculate the SQL statement throughput for a given period, you can divide this element by the elapsed time between two snapshots.

Number of Open Cursors

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl	Basic
Resettable	No	
API Element Name	open_cursors	
Element Type	gauge	
Related Information	• None	

Description: The number of cursors currently open for an application.

Usage: Use this element to assess how much memory is being allocated. The amount of memory allocated by the DB2 client, DB2 Connect, or the database agent on the target database is related to the number of cursors that are currently open. Knowing this information can help with capacity planning. For example, each open cursor that is doing blocking has a buffer size of `RQRIOBLK`. If *deferred_prepare* is enabled, then two buffers will be allocated.

DCS Application Status

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_applinfo	Basic
Resettable	No	
API Element Name	dcs_appl_status	
Element Type	information	
Related Information	• "Host Coded Character Set ID" on page 61 • "Outbound Communication Protocol" on page 61 • "Outbound Communication Address" on page 62 • "Inbound Communication Address" on page 62	

Description: The status of a DCS application at the DB2 Connect gateway.

Usage: Use this element for problem determination on DCS applications. Values are:

SQLM_DCS_CONNECTPEND_OUTBOUND
: The application has initiated a database connection from the DB2 Connect gateway to a host database, but the request has not completed yet.

SQLM_DCS_UOWWAIT_OUTBOUND
: The DB2 Connect gateway is waiting for the host database to reply to the application's request.

SQLM_DCS_UOWWAIT_INBOUND
: (1) The connection from the DB2 Connect gateway to the host database has been established and the gateway is waiting for SQL requests from the application.

(2) The DB2 Connect gateway is waiting on behalf of the unit of work in the application. This usually means that the application's code is being executed.

Host Coded Character Set ID

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	host_ccsid	
Element Type	information	
Related Information	• "DCS Application Status" on page 60 • "Outbound Communication Protocol" on page 61 • "Outbound Communication Address" on page 62 • "Inbound Communication Address" on page 62	

Description: This is the coded character set identifier (CCSID) of the host database.

Usage: Use this element for problem determination on DCS applications.

Outbound Communication Protocol

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo	Basic
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	outbound_comm_protocol	
Element Type	information	
Related Information	• "DCS Application Status" on page 60 • "Host Coded Character Set ID" on page 61 • "Outbound Communication Address" on page 62 • "Inbound Communication Address" on page 62	

Description: The communication protocol used between the DB2 Connect gateway and the host.

Usage: Use this element for problem determination on DCS applications. Valid values are:

- SQLM_PROT_APPC
- SQLM_PROT_TCPIP

Outbound Communication Address

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo	Basic
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	outbound_comm_address	
Element Type	information	
Related Information	• "DCS Application Status" on page 60 • "Host Coded Character Set ID" on page 61 • "Outbound Communication Protocol" on page 61 • "Inbound Communication Address" on page 62	

Description: This is the communication address of the target database. For example, it could be an SNA net ID and LU partner name, or an IP address and port number for TCP/IP.

Usage: Use this element for problem determination on DCS applications.

Inbound Communication Address

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_applinfo	Basic
DCS Application	sqlm_dcs_applid_info	Basic
Resettable	No	
API Element Name	inbound_comm_address	
Element Type	information	
Related Information	• "DCS Application Status" on page 60 • "Host Coded Character Set ID" on page 61 • "Outbound Communication Protocol" on page 61 • "Outbound Communication Address" on page 62	

Description: This is the communication address of the client. For example, it could be an SNA net ID and LU partner name, or an IP address and port number for TCP/IP.

Usage: Use this element for problem determination on DCS applications.

Inbound Number of Bytes Received

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl sqlm_dcs_stmt	Basic Statement
Resettable	Yes	
API Element Name **Element Type**	inbound_bytes_received counter	
Related Information	• "Outbound Number of Bytes Sent" on page 63 • "Outbound Number of Bytes Received" on page 64 • "Inbound Number of Bytes Sent" on page 64	

Description: The number of bytes received by the DB2 Connect gateway from the client, excluding communication protocol overhead (for example, TCP/IP or SNA headers).

Usage: Use this element to measure the throughput from the client to the DB2 Connect gateway.

Outbound Number of Bytes Sent

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl sqlm_dcs_stmt	Basic Statement
Resettable	Yes	
API Element Name **Element Type**	outbound_bytes_sent counter	
Related Information	• "Inbound Number of Bytes Received" on page 63 • "Outbound Number of Bytes Received" on page 64 • "Inbound Number of Bytes Sent" on page 64	

Description: The number of bytes sent by the DB2 Connect gateway to the host, excluding communication protocol overhead (for example, TCP/IP or SNA headers).

Usage: Use this element to measure the throughput from the DB2 Connect gateway to the host database.

Outbound Number of Bytes Received

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl sqlm_dcs_stmt	Basic Statement
Resettable	Yes	
API Element Name **Element Type**	outbound_bytes_received counter	
Related Information	• "Inbound Number of Bytes Received" on page 63 • "Outbound Number of Bytes Sent" on page 63 • "Inbound Number of Bytes Sent" on page 64	

Description: The number of bytes received by the DB2 Connect gateway from the host, excluding communication protocol overhead (for example, TCP/IP or SNA headers).

Usage: Use this element to measure the throughput from the host databases to the DB2 Connect gateway.

Inbound Number of Bytes Sent

Snapshot Information Level	API Structure(s)	Monitor Switch
DCS Application	sqlm_dcs_appl sqlm_dcs_stmt	Basic Statement
Resettable	Yes	
API Element Name **Element Type**	inbound_bytes_sent counter	
Related Information	• "Inbound Number of Bytes Received" on page 63 • "Outbound Number of Bytes Sent" on page 63 • "Outbound Number of Bytes Received" on page 64	

Description: The number of bytes sent by the DB2 Connect gateway to the client, excluding communication protocol overhead (for example, TCP/IP or SNA headers).

Usage: Use this element to measure the throughput from the DB2 Connect gateway to the client.

Transaction ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_appl_xid	Unit of Work
DCS Application	sqlm_appl_xid	Unit of Work
Resettable	No	
API Element Name	xid	
Element Type	information	
Related Information	• None	

Description: A unique transaction identifier (across all databases) generated by a transaction manager in a two-phase commit transaction.

Usage: This identifier can be used to correlate the transaction generated by the transaction manager with the transactions executed against multiple databases. It can be used to help diagnose transaction manager problems by tying database transactions involving a two-phase commit protocol with the transactions originated by the transaction manager.

TP Monitor Client User ID

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_tpmon_info	Basic
DCS Application	sqlm_tpmon_info	Basic
Resettable	No	
API Element Name	tpmon_client_userid	
Element Type	information	
Related Information	• "TP Monitor Client Workstation Name" on page 66 • "TP Monitor Client Application Name" on page 66 • "TP Monitor Client Accounting String" on page 67	

Description: The client user ID generated by a transaction manager and provided to the server, if the **sqleseti** API is used.

Usage: Use this element in application server or TP monitor environments to identify the end-user for whom the transaction is being executed.

TP Monitor Client Workstation Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_tpmon_info	Basic
DCS Application	sqlm_tpmon_info	Basic
Resettable	No	
API Element Name	tpmon_client_wkstn	
Element Type	information	
Related Information	• "TP Monitor Client User ID" on page 65 • "TP Monitor Client Application Name" on page 66 • "TP Monitor Client Accounting String" on page 67	

Description: Identifies the client's system or workstation (for example CICS EITERMID), if the **sqleseti** API was issued in this connection.

Usage: Use this element to identify the user's machine by node ID, terminal ID, or similar identifiers.

TP Monitor Client Application Name

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_tpmon_info	Basic
DCS Application	sqlm_tpmon_info	Basic
Resettable	No	
API Element Name	tpmon_client_app	
Element Type	information	
Related Information	• "TP Monitor Client User ID" on page 65 • "TP Monitor Client Workstation Name" on page 66 • "TP Monitor Client Accounting String" on page 67	

Description: Identifies the the server transaction program performing the transaction, if the **sqleseti** API was issued in this connection.

Usage: Use this element for problem determination and accounting purposes.

TP Monitor Client Accounting String

Snapshot Information Level	API Structure(s)	Monitor Switch
Application	sqlm_tpmon_info	Basic
DCS Application	sqlm_tpmon_info	Basic
Resettable	No	
API Element Name	acc_str_length acc_str_offset	
Element Type	information	
Related Information	• "TP Monitor Client User ID" on page 65 • "TP Monitor Client Workstation Name" on page 66 • "TP Monitor Client Application Name" on page 66	

Description: The data passed to the target database for logging and diagnostic purposes, if the **sqleseti** API was issued in this connection.

Note: Two variables in the `sqlm_tpmon_info` data structure are used to determine the TP monitor client accounting string:

acc_str_length	length of the accounting string
acc_str_offset	relative offset from the start of the `sqlm_tpmon_info` data structure

Usage: Use this element for problem determination and accounting purposes.

Transaction Processor Monitoring

In a transaction monitor or application server (multi-tier) environment, application users do not issue SQL requests directly. Instead, they request the transaction processor monitor (for example, CICS, TUXEDO, or ENCINA running on a UNIX, OS/2, or Windows NT server) or application server to execute a business transaction. Each business transaction is an application part that issues SQL requests to the database server. Because the SQL requests are issued by an intermediate server, the database server has no information about the original client that caused the execution of the SQL request.

Developers of transaction processor monitor (TP monitor) transactions or application server code can use the sqleseti - Set Client Information API to provide information about the original client to the database server. This information can be found in the following data elements:

- "TP Monitor Client User ID" on page 65
- "TP Monitor Client Workstation Name" on page 66
- "TP Monitor Client Application Name" on page 66
- "TP Monitor Client Accounting String" on page 67.

New System Monitor Data Elements

In addition to the new data elements for DB2 Connect (see “New Data Elements” on page 54), Version 5.2 adds data elements to help you monitor:

- Hash joins
- Page reorganizations
- Lock escalations

Hash Join

Hash join is an additional option for the optimizer (see “Hash Joins” on page 31 for more details).

Total Hash Joins

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	total_hash_joins	
Element Type	counter	
Related Information	• “Hash Join Threshold” on page 69 • “Total Hash Loops” on page 69 • “Hash Join Overflows” on page 70 • “Hash Join Small Overflows” on page 70	

Description: The total number of hash joins executed.

Usage: At the database or application level, use this value in conjunction with Hash Join Overflows and Hash Join Small Overflows to determine if a significant percentage of hash joins would benefit from modest increases in the sort heap size.

Hash Join Threshold

Snapshot Information Level	API Structure(s)	Monitor Switch
Database Manager	sqlm_db2	Basic
Resettable	Yes	
API Element Name	post_threshold_hash_joins	
Element Type	counter	
Related Information	• "Total Hash Joins" on page 68 • "Total Hash Loops" on page 69 • "Hash Join Overflows" on page 70 • "Hash Join Small Overflows" on page 70	

Description: The total number of times that a hash join heap request was limited due to concurrent use of shared or private sort heap space.

Usage: If this value is large (greater than 5% of "Hash Join Overflows" on page 70), the sort heap threshold should be increased.

Total Hash Loops

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	total_hash_loops	
Element Type	counter	
Related Information	• "Total Hash Joins" on page 68 • "Hash Join Threshold" on page 69 • "Hash Join Overflows" on page 70 • "Hash Join Small Overflows" on page 70	

Description: The total number of times that a single partition of a hash join was larger than the available sort heap space.

Usage: Values for this data element indicate inefficient execution of hash joins. This might indicate that the sort heap size is too small or the sort heap threshold is too small. Use this value in conjunction with the other hash join variables to tune the sort heap size (*sortheap*) and sort heap threshold (*sheapthres*) configuration parameters.

Hash Join Overflows

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	hash_join_overflows	
Element Type	counter	
Related Information	• "Total Hash Joins" on page 68 • "Hash Join Threshold" on page 69 • "Total Hash Loops" on page 69 • "Hash Join Small Overflows" on page 70	

Description: The number of times that hash join data exceeded the available sort heap space.

Usage: At the database level, if the percentage of Hash Join Small Overflows is greater than 10% of this value, then you should consider increasing the sort heap size. Values at the application level can be used to evaluate hash join performance for individual applications.

Hash Join Small Overflows

Snapshot Information Level	API Structure(s)	Monitor Switch
Database	sqlm_dbase	Basic
Application	sqlm_appl	Basic
Resettable	Yes	
Event Type	**Event Record(s)**	
Database	sqlm_db_event	
Connection	sqlm_conn_event	
API Element Name	hash_join_small_overflows	
Element Type	counter	
Related Information	• "Total Hash Joins" on page 68 • "Hash Join Threshold" on page 69 • "Total Hash Loops" on page 69 • "Hash Join Overflows" on page 70	

Description: The number of times that hash join data exceeded the available sort heap space by less than 10%.

Usage: If this value and Hash Join Overflows are high, then you should consider increasing the sort heap threshold. If this value is greater than 10% of Hash Join Overflows, then you should consider increasing the sort heap size.

Page Reorganization

When a new row is being inserted or an existing row is being updated, resulting in an increased record size, the page where this record is to be placed may have enough free space, but that space could be fragmented. In these cases the page may require reorganization, which moves all fragmented space to a contiguous area, where the new record can be written. Such a page reorganization (page reorg) is very expensive to perform.

Page Reorganizations

Snapshot Information Level	**API Structure(s)**	**Monitor Switch**
Table	sqlm_table	Table
Resettable	Yes	
Event Type	**Event Record(s)**	
Table	sqlm_table_event	
API Element Name	page_reorgs	
Element Type	counter	
Related Information	• Rows Inserted • Rows Updated	

Description: The number of page reorgs executed for a table.

Usage: Too many page reorgs can result in less than optimal insert performance. You can use the REORG TABLE utility to reorganize a table and eliminate fragmentation. You can also use the APPEND parameter for the ALTER TABLE statement to indicate that all inserts are appended at the end of a table and so avoid page reorgs.

In situations where updates to rows causes the row length to increase, the page may have enough space to accommodate the new row, but a page reorg may be required to defragment that space. Or if the page does not have enough space for the new larger row, an overflow record is created being created causing *Accesses to Overflowed Records* during reads. You can avoid both situations by using fixed length columns instead of varying length columns.

Lock Escalations

Lock escalation occurs when the number of locks used by a given application exceeds the configured limit or the number of locks configured for all applications is exhausted. During lock escalation a few highly restrictive locks are used to replace many less restrictive locks. Although this reduces the total number of locks used, it also increases the number of lock waits and potentially causes deadlocks.

Lock Escalations

Snapshot Information Level	API Structure(s)	Monitor Switch
Lock	sqlm_lock	Lock
Lock	sqlm_lock_wait	Lock
Resettable	No	
Event Type	**Event Record(s)**	
Deadlock	sqlm_dlconn_event	
API Element Name	lock_escalation	
Element Type	information	
Related Information	• other lock data elements	

Description: Indicates whether a lock request was made as part of a lock escalation.

Usage: Use this element to better understand the cause of deadlocks. If you experience a deadlock that involves applications doing lock escalation, you may want to increase the amount of lock memory or change the percentage of locks that any one application can request.

Application Identification

The *Application Handle (agent ID)* data element has different behavior depending on your version of DB2. When taking snapshots from DB2 Version 1 or Version 2 to a DB2 Universal Database (Version 5 or greater) database, the agent_id returned is not usable as an application identifier. In these cases an agent_id is still returned for back-level compitability, but internally the DB2 Universal Database server will not recognize the value as an agent_id.

System Monitor Switches

DB2 Universal Database Version 5.2 provides support for dynamically updating certain database configuration parameters, including *dft_monswitches* (Default Database System Monitor Switches).

The section *Monitor Switches Control Data Collected by the Database Manger* in the *System Monitor Guide and Reference* should now indicate that switches can be set **without stopping** the database management system. These switches can now be dynamically updated, but the application doing the update must be explicitly attached to the instance for the updates to take effect.

Note: Any existing snapshot applications will not be affected. In order to pick up the new default values for *dft_monswitches*, a monitoring application must terminate and reestablish its instance connection.

Changes to Number of Log Pages Written

The following note is being added to the Number of Log Pages Written data element (log_writes).

Note: When log pages are written to disk, the last page might not be full. In such cases, the partial log page remains in the log buffer, and additional log records are written to the page. Therefore log pages might be written to disk by the logger more than once. You should not use this data element to measure the number of pages produced by DB2.

Changes to Query Number of Rows Estimate

The Query Number of Rows Estimate data element (query_card _estimate) now returns information for the following SQL statements when you are monitoring DB2 Connect.

- INSERT, UPDATE, and DELETE

  Indicates the number of rows affected.

- PREPARE

  Estimate of the number of rows that will be returned. Only collected if the DRDA server is DB2 Universal Database, DB2 for VM and VSE, or DB2 for OS/400.

- FETCH

  Set to the number of rows fetched. Only collected if the DRDA server is DB2 for OS/400.

If information is not collected for a DRDA server, then the data element is set to zero.

Changes to Query Cost Estimate

The Query Cost Estimate data element (query_cost _estimate) now returns information for the following SQL statements when you are monitoring DB2 Connect.

- PREPARE

  Represents the relative cost of the prepared SQL statement.

- FETCH

  Contains the length of the row retrieved. Only collected if the DRDA server is DB2 for OS/400.

If information is not collected for a DRDA server, then the data element is set to zero.

Note: If the DRDA server is DB2 for OS/390, this estimate could be higher than 2**32 - 1 (the maximum integer number that can be expressed through an unsigned long variable). In that case, the value returned by the System Monitor for this data element will be 2**32 - 1.

Appendix B. Call Level Interface Guide and Reference Updates

The *CLI Guide and Reference* has not been refreshed for DB2 Universal Database Version 5.2. The following sections document any Version 5.2 changes and enhancements to the call level interface and should be used in conjunction with the Version 5 edition of the *CLI Guide and Reference* by Version 5.2 users. These include:

- "Data Types": You can use the new DB2 Universal Database data types with the DB2 Call Level Interface (CLI).
- "Configuration Keywords" on page 77: Changes and additions have been made to some CLI/ODBC configuration keywords.
- "Functions" on page 83: Enhancements have been made to certain CLI functions.
- "Messages" on page 93: Some DB2 messages affecting CLI have changed.
- "Microsoft Transaction Server" on page 93: Support has been extended to the Microsoft Transaction Server.

Data Types

The following data types are available with DB2 Universal Database Version 5.2 and can be used with the call level interface (CLI):

- "BIGINT"
- "DATALINK" on page 76
- "Defined Types" on page 76

BIGINT

The following tables are updated in the CLI Guide and Reference as a result of the added support for the BIGINT data type. See the *SQL Reference* for more information on this new data type.

Table 3, "SQL Symbolic and Default Data Types":

- SQL Data Type: BIGINT
- Symbolic SQL Data Type: SQL_BIGINT
- Default Symbolic C Data Type: SQL_C_BIGINT

Table 4, "C Data Types":

- C Symbolic Data Type: SQL_C_BIGINT
- C Type: SQLBIGINT
- Base C Type: long int (or __int64)

Table 22, "SQLBindParameter Arguments", under `SQLBindParameter()`**:**

- Add SQL_C_BIGINT to the row for argument "ValueType".
- Add SQL_BIGINT to the row for argument "ParameterType".

Table 116, "SQLGetTypeInfo Arguments, under `SQLGetTypeInfo()`**:**
Add SQL_BIGINT to the list of supported types for argument "DataType".

DATALINK

The following additions apply to Chapter 2. Writing a DB2 CLI Application of the *CLI Guide and Reference* as a result of support for the new DATALINK data type.

Three tables require an extra row each:

Table 3. SQL Symbolic and Default Data Types

- SQL Data Type field: DATALINK
- Symbolic SQL Data Type field: SQL_DATALINK
- Default Symbolic C Data Type: SQL_C_DATALINK

Table 4. C Data Types

- C Symbolic Data Type field: SQL_C_DATALINK
- C Type field: SQLCHAR
- Base C type field: unsigned char

Table 7. Supported Data Conversions

The SQL Data Type SQLDATALINK only converts with SQL_C_DATALINK.

Defined Types

The following section is added to Chapter 3. Using Advanced Features.

Using Reference Types

In addition to the distinct types, user defined structured types can also be defined and used as the type for a table or view. Tables or views that are defined using a structured type are called *typed tables* or *typed views*. Structured types can be defined in a hierarchy with *subtypes* and *supertypes*. These structured types are created using the CREATE TYPE statement. Rows of a typed table or view are identified with an object identifier (OID) that is a reference type. A reference type is defined to have a target type, which must be a structured type. When the root structured type (the structured type without a supertype) is defined, the representation type for the reference type is defined as based on a built-in data type. Similar to user defined types (UDTs), a reference type shares its internal representation with an existing type, but is considered to be a separate and incompatible type for most operations.

Reference types provide a means of referring to rows in typed tables or typed views. Applications continue to work with C data types for application variables, and only need to consider the reference types when constructing C statements.

This means:

- All SQL to C data type conversion rules that apply to the reference type's underlying SQL built-in type apply to the reference type.
- The reference type will have the same default C Type as the underlying SQL built-in type.

- `SQLDescribeCol()` will return the representation (built-in) type information. The target type name can be obtained by calling `SQLColAttribute()` with the input descriptor type set to SQL_DESC_REFERENCE_TYPE.
- SQL predicates that involve parameter markers must be explicitly cast to the reference type. This is required since the application can only deal with the built-in types, so before any operation can be performed using the parameter, it must be cast from the C built-in type to the reference type; otherwise an error will occur when the statement is prepared.

For complete rules and a description of reference types refer to the *SQL Reference*.

Configuration Keywords

The following changes and additions have been made to the CLI/ODBC configuration keywords.

- "CLISCHEMA"
- "CURRENTREFRESHAGE" on page 78
- "CURRENTSCHEMA" on page 79
- "IGNOREWARNLIST" on page 79
- "OPTIMIZESQLCOLUMNS" on page 79
- "PATCH1 and PATCH2 Values" on page 80

For specific details on setting CLI/ODBC configuration keywords, see the *Quick Beginnings* manual for your platform.

CLISCHEMA

Keyword Description: The DB2 ODBC catalog view to use.

db2cli.ini Keyword Syntax: CLISCHEMA=*ODBC catalog view*

Default Setting: None - No ODBC catalog view is used

DB2 CLI/ODBC Settings Tab: This keyword cannot be set using the CLI/ODBC Settings notebook. To set these keywords use either the UPDATE CLI CONFIGURATION command in the Command Line Processor, or modify the db2cli.ini file directly.

Equivalent Connection Attribute: SQL_ATTR_CLISCHEMA

Usage Notes:

The DB2 ODBC catalog is designed to improve the performance of schema calls for lists of tables in ODBC applications that connect to host DBMSs through DB2 Connect.

The DB2 ODBC catalog, created and maintained on the host DBMS, contains rows representing objects defined in the real DB2 catalog, but these rows include only the columns necessary to support ODBC operations. The tables in the DB2 ODBC catalog are pre-joined and specifically indexed to support fast catalog access for ODBC applications.

System administrators can create multiple DB2 ODBC catalog views, each containing only the rows that are needed by a particular user group. Each end user can then select the DB2 ODBC catalog view they wish to use (by setting this keyword).

Use of the CLISCHEMA setting is completely transparent to the ODBC application; you can use this option with any ODBC application.

While this keyword has some similar effects as the SYSSCHEMA keyword, CLISCHEMA should be used instead (where applicable).

CLISCHEMA improves data access efficiency: The user-defined tables used with SYSSCHEMA were mirror images of the DB2 catalog tables, and the ODBC driver still had to join rows from multiple tables to produce the information required by the ODBC user. Using CLISCHEMA also results in less contention on the catalog tables.

CURRENTREFRESHAGE

Keyword Description: Set the value of the CURRENT REFRESH AGE special register.

db2cli.ini Keyword Syntax: CURRENTREFRESHAGE=ANY | a numeric constant

Default Setting: 0 - summary tables defined with REFRESH DEFERRED will not be used to optimize the processing of a query

DB2 CLI/ODBC Settings Tab: This keyword cannot be set using the CLI/ODBC Settings notebook. To set these keywords use either the UPDATE CLI CONFIGURATION command in the Command Line Processor, or modify the db2cli.ini file directly.

Usage Notes:

For information on Summary Tables and the SET CURRENT REFRESH AGE statement, see the *SQL Reference*.

This keyword can be set to one of the following values:

0

Indicates that summary tables defined with REFRESH DEFERRED will not be used to optimize the processing of a query (default).

99999999999999

Indicates that any summary tables defined with REFRESH DEFERRED or REFRESH IMMEDIATE may be used to optimize the processing of a query. This value represents 9999 years, 99 months, 99 days, 99 hours, 99 minutes, and 99 seconds.

ANY

This is a shorthand for 99999999999999.

CURRENTSCHEMA

Keyword Description: Issue a SET CURRENT SCHEMA statement upon a successful connect.

db2cli.ini Keyword Syntax: CURRENTSCHEMA=*schema*

Default Setting: No statement is issued.

DB2 CLI/ODBC Settings Tab: This keyword cannot be set using the CLI/ODBC Settings notebook. To set these keywords use either the UPDATE CLI CONFIGURATION command in the Command Line Processor, or modify the db2cli.ini file directly.

Usage Notes:

Upon a successful connect, if this keyword is set then a SET CURRENT SCHEMA statement is sent to the server. This allows the application to name SQL objects without having to qualify them with a schema name.

See the SET SCHEMA statement in the *SQL Reference* for more information.

IGNOREWARNLIST

Keyword Description: Ignore specified sqlstates.

db2cli.ini Keyword Syntax: IGNOREWARNLIST="'sqlstate1', 'sqlstate2', ..."

Default Setting: Warnings are returned as normal

DB2 CLI/ODBC Settings Tab: This keyword cannot be set using the CLI/ODBC Settings notebook. To set these keywords use either the UPDATE CLI CONFIGURATION command in the Command Line Processor, or modify the db2cli.ini file directly.

Usage Notes:

On rare occasions an application may not correctly handle some warning messages, but does not want to ignore all warning messages. This keyword can be used to indicate which warnings are not to be passed on to the application. The IGNOREWARNINGS keyword should be used if all database manager warnings are to be ignored.

If an sqlstate is included in both IGNOREWARNLIST and WARNINGLIST, it will be ignored altogether.

Each sqlstate must be in uppercase, delimited with single quotes and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
IGNOREWARNLIST="'01000', '01004','01504'"
```

OPTIMIZESQLCOLUMNS

Keyword Description: Optimize SQLColumns() call with explicit Schema and Table Name.

db2cli.ini Keyword Syntax: OPTIMIZESQLCOLUMNS=1 | 0

Default Setting: 1 - Optimization on

DB2 CLI/ODBC Settings Tab: This keyword cannot be set using the CLI/ODBC Settings notebook. To set these keywords use either the UPDATE CLI CONFIGURATION command in the Command Line Processor, or modify the db2cli.ini file directly.

Equivalent Connection Attribute: SQL_ATTR_OPTIMIZESQLCOLUMNS

Usage Notes:

If OPTIMIZESQLCOLUMNS is on (set to 1), then all calls to `SQLColumns()` will be optimized if an explicit (no wildcard specified) Schema Name, explicit Table Name, and % (ALL columns) for Column Name are specified. The DB2 CLI/ODBC Driver will optimize this call so that the system tables will not be scanned.

If the call is optimized then the COLUMN_DEF information (which contains the default string for the columns) is not returned. If the application needs the COLUMN_DEF information then OPTIMIZESQLCOLUMNS should be set to 0.

PATCH1 and PATCH2 Values

The DB2 CLI/ODBC driver default behavior can be modified by specifying values for both the PATCH1 and PATCH2 keyword through either the db2cli.ini file or through the SQLDriverConnect() or SQLBrowseConnect() CLI API.

The PATCH1 keyword is specified by adding together all keywords that the user wants to set. For example, if patch 1, 2, and 8 were specified, then PATCH1 would have a value of 11. Following is a description of each keyword value and its effect on the driver:

1 This makes the driver search for "count(exp)" and replace it with "count(distinct exp)". This is needed because some versions of DB2 support the "count(exp)" syntax, and that syntax is generated by some ODBC applications. Needed by Microsoft applications when the server does not support the "count(exp)" syntax.

2 Some ODBC applications are trapped when SQL_NULL_DATA is returned in the SQLGetTypeInfo() function for either the LITERAL_PREFIX or LITERAL_SUFFIX column. This forces the driver to return an empty string instead. Needed by Impromptu 2.0.

4 This forces the driver to treat the input time stamp data as date data if the time and the fraction part of the time stamp are zero. Needed by Microsoft Access.

8 This forces the driver to treat the input time stamp data as time data if the date part of the time stamp is 1899-12-30. Needed by Microsoft Access.

16 Not used.

32 This forces the driver to not return information about SQL_LONGVARCHAR, SQL_LONGVARBINARY, and SQL_LONGVARGRAPHIC columns. To the application it appears as though long fields are not supported. Needed by Lotus 123.

64 This forces the driver to NULL terminate graphic output strings. This is needed by Microsoft Access in a double byte environment.

128 This forces the driver to let the query "SELECT Config, nValue FROM MSysConf" go to the server. Currently the driver returns an error with associated SQLSTATE value of S0002 (table not found). Needed if the user has created this configuration table in the database and wants the application to access it.

256 This forces the driver to return the primary key columns first in the SQLStatistics() call. Currently, the driver returns the indexes sorted by index name, which is standard ODBC behavior.

512 This forces the driver to return FALSE in SQLGetFunctions() for both SQL_API_SQLTABLEPRIVILEGES and SQL_API_SQLCOLUMNPRIVILEGES.

1024 This forces the driver to return SQL_SUCCESS instead of SQL_NO_DATA_FOUND in SQLExecute() or SQLExecDirect() if the executed UPDATE or DELETE statement affects no rows.

2048 Not used.

4096 This forces the driver to not issue a COMMIT after closing a cursor when in autocommit mode.

8192 This forces the driver to return an extra result set after invoking a stored procedure. This result set is a one row result set consisting of the output values of the stored procedure. Can be accessed by Powerbuild applications.

32768 This forces the driver to make Microsoft Query applications work with DB2 MVS synonyms.

65536 This forces the driver to manually insert a "G" in front of character literals which are in fact graphic literals. This patch should always be supplied when working in a double byte environment.

131072 This forces the driver to return a time stamp column as a CHAR(26) column instead. Needed by Microsoft applications when the time stamp column is part of a unique index.

262144 This forces the driver to use the pseudo-catalog table db2cli.procedures instead of the SYSCAT.PROCEDURES and SYSCAT.PROCPARMS tables.

524288 This forces the driver to use SYSTEM_TABLE_SCHEMA instead of TABLE_SCHEMA when doing a system table query to a DB2/400 Version 3.x system. This results in better performance.

1048576 This forces the driver to treat a zero length string through SQLPutData() as SQL_NULL_DATA.

2097152 This forces the driver to report that SQLParamOptions() is not supported.

The PATCH2 keyword differs from the PATCH1 keyword. In this case, multiple patches are specified using comma separators. For example, if patch 1, 4, and 5 were specified, then PATCH2 would have a value of "1,4,5". Following is a description of each keyword value and its effect on the driver:

1 This forces the driver to convert the name of the stored procedure in a CALL statement to uppercase.

2 Not used.

3 This forces the driver to convert all arguments to schema calls to uppercase.

4 This forces the driver to return the Version 2.1.2 like result set for schema calls (that is, SQLColumns(), SQLProcedureColumns(), and so on), instead of the Version 5 like result set.

5 This forces the driver to not optimize the processing of input VARCHAR columns, where the pointer to the data and the pointer to the length are consecutive in memory.

6 This forces the driver to return a message that scrollable cursors are not supported. This is needed by Visual Basic programs if the DB2 client is Version 5 and the server is DB2 UDB Version 5.

7 This forces the driver to map all GRAPHIC column data types to the CHAR column data type. This is needed in a double byte environment.

8 This forces the driver to ignore catalog search arguments in schema calls.

9 This flag prevents an autocommit from occurring when the cursor is closed at the server when the last block of data is returned. The commit will occur when the cursor is closed by the application.

10 This setting should only be used in an EUC (Extended Unix Code) environment. It ensures that the CLI driver provides data for character variables (CHAR, VARCHAR, etc...) in the proper format for the JDBC driver. The data in these character types will not be usable in JDBC without this setting.

11 Pretend to support SQL_CATALOG_LOCATION, SQL_CATALOG_NAME_SEPARATOR, and SQL_MAX_CATALOG_NAME_LEN. Workaround for MS InterDev.

12 Remove extraneous double quotes (") from calls to schema functions. Workaround for MS InterDev.

13 Do not append keywords from the db2cli.ini file to the output string of SQLDriverConnect().

14 Do not return an error and ignore the schema argument for SQLProcedures() and SQLProcedureColumns().

15 Ignore locale specific decimal separator and force a period for conversions between Character and:

- Decimal
- Float
- Double

Functions

The following CLI functions have been added or enhanced:

SQLBuildDataLink - Build DATALINK Value

Purpose

Specification:	DB2 CLI 5.2		ISO CLI

SQLBuildDataLink() returns a DATALINK value built from input arguments.

Syntax

```
SQLRETURN   SQLBuildDataLink(SQLHSTMT          StatementHandle,
                             SQLCHAR     FAR   *LinkType,
                             SQLINTEGER        LinkTypeLength,
                             SQLCHAR     FAR   *DataLocation,
                             SQLINTEGER        DataLocationLength,
                             SQLCHAR     FAR   *Comment,
                             SQLINTEGER        CommentLength,
                             SQLCHAR     FAR   *DataLinkValue,
                             SQLINTEGER        BufferLength,
                             SQLINTEGER  FAR   *StringLengthPtr);
```

Function Arguments

Table 1. SQLBuildDataLink Arguments

Data Type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Used only for diagnostic reporting.
SQLCHAR *	LinkType	input	Always set to SQL_DATALINK_URL.
SQLINTEGER	LinkTypeLength	input	The length of the *LinkType* value.
SQLCHAR *	DataLocation	input	The complete URL value to be assigned.
SQLINTEGER	DataLocationLength	input	The length of the *DataLocation* value.
SQLCHAR *	Comment	input	The comment, if any, to be assigned.
SQLINTEGER	CommentLength	input	The length of the *Comment* value.
SQLCHAR *	DataLinkValue	output	The DATALINK value that is created by the function.
SQLINTEGER	BufferLength	input	Length of the DataLinkValue buffer.
SQLINTEGER	*StringLengthPtr	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **DataLinkValue*. If *DataLinkValue* is a null pointer, no length is returned. If the number of bytes available to return is greater than BufferLength minus the length of the null-termination character, then SQLSTATE 01004 is returned. In this case, subsequent use of the DATALINK value may fail.

Usage

The function is used to build a DATALINK value. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Return Codes

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 2. SQLGetDatLinkAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
01004	Data truncated.	The data returned in **DataLinkValue* was truncated to be *BufferLength* minus the length of the null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI was unable to allocate memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified one of the arguments (*LinkTypeLength*, *DataLocationLength*, or *CommentLength*) was less than 0 but not equal to SQL_NTS or *BufferLength* is less than 0.

Restrictions

None.

References

- “SQLGetDataLinkAttr - Get Datalink Attribute Value” on page 86

SQLGetDataLinkAttr - Get Datalink Attribute Value

Purpose

Specification:	DB2 CLI 5.2		ISO CLI

SQLGetDataLinkAttr() returns the current value of an attribute of a datalink value.

Syntax

```
SQLRETURN   SQLGetDataLinkAttr(SQLHSTMT          StatementHandle,
                               SQLSMALLINT       Attribute,
                               SQLCHAR     FAR   *DataLink,
                               SQLINTEGER        DataLinkLength,
                               SQLPOINTER        *ValuePtr,
                               SQLINTEGER        BufferLength,
                               SQLINTEGER  FAR   *StringLengthPtr);
```

Function Arguments

Table 3 (Page 1 of 2). SQLGetDataLinkAttr Arguments

Data Type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Used only for diagnostic reporting.
SQLSMALLINT	Attribute	input	Identifies the attribute of the DataLink that is to be extracted. Possible values are: SQL_ATTR_DATALINK_COMMENT SQL_ATTR_DATALINK_LINKTYPE SQL_ATTR_DATALINK_URLCOMPLETE (complete URL to access a file) SQL_ATTR_DATALINK_URLPATH (to access a file within a file server) SQL_ATTR_DATALINK_URLPATHONLY (file path only) SQL_ATTR_DATALINK_URLSCHEME SQL_ATTR_DATALINK_URLSERVER
SQLCHAR *	DataLink	input	The DATALINK value from which the attribute is to be extracted.
SQLINTEGER	DataLinkLength	input	The length of the DATALINK value.
SQLPOINTER *	ValuePtr	output	A pointer to memory in which to return the value of the attribute specified by *Attribute*.
SQLINTEGER	BufferLength	input	Length of the Attribute buffer.

Table 3 (Page 2 of 2). SQLGetDataLinkAttr Arguments

Data Type	Argument	Use	Description
SQLINTEGER	*StringLength	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **Attribute*. If *Attribute* is a null pointer, no length is returned. If the number of bytes available to return is greater than BufferLength minus the length of the null-termination character, then SQLSTATE HY090 is returned.

Usage

The function is used with a DATALINK value that was retrieved from the database or built using SQLBuildDataLink. The *AttrType* value determines the attribute from the DATALINK value that is returned. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Return Codes

- SQL_SUCCESS
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics

Table 4. SQLGetDatLinkAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
01004	Data truncated.	The data returned in **ValuePtr* was truncated to be *BufferLength* minus the length of the null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
HY001	Memory allocation failure.	DB2 CLI was unable to allocate memory required to support execution or completion of the function.
HY009	Invalid argument value.	The value specified for the argument **DataLink* was a null pointer or was not valid.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0 or the values specified for the argument *DataLinkLength* was less than 0 and not equal to SQL_NTS.
HY092	Option type out of range.	The value specified for the argument *AttrType* was not valid.

SQLGetDataLinkAttr

Restrictions

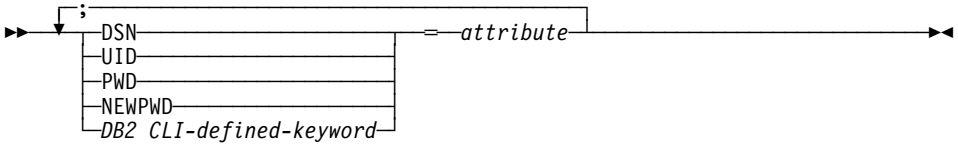
None.

References

- “SQLBuildDataLink - Build DATALINK Value” on page 84

SQLDriverConnect() and NEWPWD Support

A new attribute-keyword, NEWPWD, has been added to the SQLDriverConnect() function. The syntax diagram in the 'Usage' section has been updated to include the new keyword, and to fix the mistake in the Version 5 book (the separator should be a semicolon, not a comma).



This new entry is defined as follows:

NEWPWD New password used as part of a change password request.

The application can either specify the new string to use, for example, NEWPWD=anewpass;; or specify NEWPWD=; and rely on a dialog box generated by the DB2 CLI driver to prompt for the new password (set the *DriverCompletion* argument to anything other than SQL_DRIVER_NOPROMPT).

SQLBrowseConnect() and NEWPWD Support

The attribute-keyword NEWPWD has been added. It has the same description as “SQLDriverConnect() and NEWPWD Support.”

SQLGetInfo()

The following change must be made to the description of the values for SQL_GETDATA_EXTENSIONS in `SQLGetInfo()`. The version 5.0 CLI Guide and Reference incorrectly states that SQL_GD_BLOCK is supported; this is not the case.

The Description and Notes section for SQL_GETDATA_EXTENSIONS should read as follows:

SQL_GETDATA_EXTENSIONS 32-bit mask Indicates whether extensions to the `SQLGetData()` function are supported. The following extensions are currently identified and supported by DB2 CLI:

- SQL_GD_ANY_COLUMN, `SQLGetData()` can be called for unbound columns that precede the last bound column.
- SQL_GD_ANY_ORDER, `SQLGetData()` can be called for columns in any order.

ODBC also defines the following extensions which are not returned by DB2 CLI:

- SQL_GD_BLOCK
- SQL_GD_BOUND

SQLGetDataLinkAttr

SQLGetLength()

The description of the StringLength argument has been updated as follows:

The length of the large object (LOB, CLOB, etc...) referenced by the Locator argument. This value is in bytes, even for DBCLOB data.

SQLSetConnectAttr() - Additional Connection Attributes

The following connection attributes have been added to `SQLSetConnectAttr()`:

- SQL_ATTR_ENLIST_IN_DTC - For Microsoft Transaction Server (MTS) Support
- SQL_ATTR_INFO_USERID
- SQL_ATTR_INFO_WRKSTNNAME
- SQL_ATTR_INFO_APPLNAME
- SQL_ATTR_INFO_ACCTSTR

In Chapter 5 "Functions", in the section `SQLSetConnectAttr()`, Table 147 "When Connection Attributes can be Set" has been updated to include the following new rows:

Table 5. When Connection Attributes can be Set

Attribute	Before connection	After connection	After statements allocated
SQL_ATTR_ENLIST_IN_DTC	No	Yes	Yes
SQL_ATTR_INFO_USERID	No	Yes	Yes
SQL_ATTR_INFO_WRKSTNNAME	No	Yes	Yes
SQL_ATTR_INFO_APPLNAME	No	Yes	Yes
SQL_ATTR_INFO_ACCTSTR	No	Yes	Yes

Also, in the section "Attribute *ValuePtr Contents" for `SQLSetConnectAttr()`, the following new information has been added:

SQL_ATTR_ENLIST_IN_DTC (DB2 CLI v5)

An SQLPOINTER which can be either:

- non-null transaction pointer:

  Change the state of the connection from non-distributed transaction state to distributed transaction state. The connection will be enlisted with the Distributed Transaction Coordinator (DTC).

- null:

  Change the state of the connection from distributed transaction state to a non-distributed transaction state.

Each time this attribute is used with a non-null transaction pointer, the previous transaction is assumed to be ended and a new transaction is initiated. The application must call the ITransaction member function Endtransaction before calling this API with a non-null pointer. Otherwise the

previous transaction will be aborted. The application can enlist mutiple connections with the same transaction pointer.

Note: This connection attribute is specified by MTS automatically for each transaction and is not coded by the user application.

There must not be concurrent SQL statements executing on 2 different connections into the same database within the same distributed unit of work (DUOW).

SQL_ATTR_INFO_USERID (DB2 CLI v5)

A null-terminated character string used to identify the client user ID sent to the host database server when using DB2 Connect.

Usage notes:

- When the value is being set, some servers may not handle the entire length provided and may truncate the value.

  DB2 for OS/390 servers support up to a length of 16 characters.

- This user-id is not to be confused with the authentication user-id. This user-id is for identification purposes only and is not used for any authorization.
- To ensure that the data is converted correctly when transmitted to a DRDA server, use only the characters A to Z, 0 to 9, and the underscore(_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_WRKSTNNAME (DB2 CLI v5)

A null-terminated character string used to identify the client workstation name sent to the host database server when using DB2 Connect.

Usage notes:

- When the value is being set, some servers may not handle the entire length provided and may truncate the value.

  DB2 for OS/390 servers support up to a length of 18 characters.

- To ensure that the data is converted correctly when transmitted to a DRDA server, use only the characters A to Z, 0 to 9, and the underscore(_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_APPLNAME (DB2 CLI v5)

A null-terminated character string used to identify the client application name sent to the host database server when using DB2 Connect.

Usage notes:

- When the value is being set, some servers may not handle the entire length provided and may truncate the value.

SQLGetDataLinkAttr

DB2 for OS/390 servers support up to a length of 32 characters.

- To ensure that the data is converted correctly when transmitted to a DRDA server, use only the characters A to Z, 0 to 9, and the underscore(_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_ACCTSTR (DB2 CLI v5)

A null-terminated character string used to identify the client accounting string sent to the host database server when using DB2 Connect.

Usage notes:

- When the value is being set, some servers may not handle the entire length provided and may truncate the value.

  DB2 for OS/390 servers support up to a length of 200 characters.

- To ensure that the data is converted correctly when transmitted to a DRDA server, use only the characters A to Z, 0 to 9, and the underscore(_) or period (.).

Note: This is an IBM defined extension.

SQLSetStmtAttr()

The original Version 5 documentation for the the `SQLSetStmtAttr()` function concerning the SQL_ATTR_CURSOR_TYPE statement contained one mistake. If SQL_ATTR_CURSOR_TYPE is set to either SQL_CURSOR_KEYSET_DRIVEN or SQL_CURSOR_DYNAMIC (neither supported by DB2 CLI) then the value used will be SQL_CURSOR_STATIC, not SQL_CURSOR_FORWARD_ONLY as originally documented.

The complete documentation is as follows:

SQL_ATTR_CURSOR_TYPE (DB2 CLI v2)

A 32-bit integer value that specifies the cursor type. The supported values are:

- **SQL_CURSOR_FORWARD_ONLY** = The cursor only scrolls forward.
- SQL_CURSOR_STATIC = The data in the result set is static, and can be scrolled through forwards or backwards. Absolute row numbers can also be specified.

The default value is SQL_CURSOR_FORWARD_ONLY.

This option cannot be specified for an open cursor.

For more detailed information, the application should call `SQLGetInfo()` with an InfoType of SQL_<cursor type>_ATTRIBUTES1 and SQL_<cursor type>_ATTRIBUTES2 (SQL_STATIC_CURSOR_ATTRIBUTES2, for instance).

Note: The following values have also been defined by ODBC, but are not supported by DB2 CLI:

- SQL_CURSOR_KEYSET_DRIVEN
- SQL_CURSOR_DYNAMIC

If either of these values is used, DB2 CLI sets the statement attribute to SQL_CURSOR_STATIC and returns SQLSTATE 01S02 (Option value changed). In this case the application should call `SQLGetStmtAttr()` to query the actual value.

Messages

SQLSTATE 22003 - Numeric value out of range

Before DB2 UDB Version 5.2 when converting integer to float data types:

- SQL_C_SLONG, SQL_CLONG or SQL_C_ULONG to SQL_FLOAT
- SQL_INTEGER to SQL_C_FLOAT

a number of functions returned SQLSTATE 22003 - Numeric value out of range, if the integer contained more significant digits than the float could handle.

This is not correct according to the ODBC or SQL standard. Beginning with DB2 UDB v5.2, the error will not be returned in this case.

See 'Appendix E. SQLSTATE Cross Reference' of the *CLI Guide and Reference* for a list of CLI functions that return this SQLSTATE.

Microsoft Transaction Server

A new connection attribute has been created in `SQLSetConnectAttr()` to support the Microsoft Transaction Server (MTS). See the information on SQL_ATTR_ENLIST_IN_DTC in “SQLSetConnectAttr() - Additional Connection Attributes” on page 90 for more information.

Appendix C. Embedded SQL Programming Guide Updates

The *Embedded SQL Programming Guide* has not been refreshed for DB2 Universal Database Version 5.2. The following sections document any Version 5.2 changes and enhancements and should be used in conjunction with the Version 5 edition of the *Embedded SQL Programming Guide* by Version 5.2 users. These include:

- "Running CLI/ODBC/JDBC/SQLJ Programs in a DBCS Environment": Provides information on running programs that access DB2 Universal Database in a double-byte character set (DBCS) environment.
- "Host Structure Support in C/C++" on page 96: The precompiler allows the grouping of host variables.
- "SQL Enhancements" on page 99: SQL changes that can be incorporated into programs.
- "Programming in JDBC" on page 101: Using the Java programming language to access DB2 databases.
- "Embedded SQL for Java (SQLJ) Programming" on page 117: Using SQLJ to access DB2 databases.

Running CLI/ODBC/JDBC/SQLJ Programs in a DBCS Environment

Details on running Java programs that access DB2 Universal Database in a double-byte character set (DBCS) environment can be found at `http://www.software.ibm.com/data/db2/java/dbcsjava.html`. This web page currently contains the following information:

JDBC and SQLJ programs access DB2 using the DB2 CLI/ODBC driver and therefore use the same configuration file (db2cli.ini). The following entries must be added to this configuration file if you run Java programs that access DB2 UDB in a DBCS environment:

PATCH1 = 65536

This forces the driver to manually insert a "G" in front of character literals which are in fact graphic literals. This PATCH1 value should always be set when working in a double byte environment.

PATCH1 = 64

This forces the driver to NULL terminate graphic output strings. This is needed by Microsoft Access in a double byte environment. If you need to use this PATCH1 value as well then you would add the two values together (64+65536 = 65600) and set PATCH1=65600. See Note #2 below for more information about specifying multiple PATCH1 values.

PATCH2 = 7

This forces the driver to map all graphic column data types to char column data type. This is needed in a double byte environment.

PATCH2 = 10

This setting should only be used in an EUC (Extended Unix Code) environment. It ensures that the CLI driver provides data for character variables (CHAR, VARCHAR, etc...) in the proper format for the JDBC driver. The data in these character types will not be usable in JDBC without this setting.

Note:

1. Each of these keywords is set in each database specific stanza of the db2cli.ini file. If you want to set them for multiple databases then you need to repeat them for each database stanza in db2cli.ini.
2. To set multiple PATCH1 values you add the individual values and use the sum. To set PATCH1 to both 64 and 65536 you would set PATCH1=65600 (64+65536). If you already have other PATCH1 values set then replace the existing number with the sum of the existing number and the new PATCH1 values you want to add.
3. To set multiple PATCH2 values you specify them in a comma delimited string (unlike the PATCH1 option). To set PATCH2 values 1 and 7 you would set PATCH2="1,7"

For more information about setting these keywords see the "Running CLI/ODBC Programs" section in the Installing and Configuring DB2 Clients manual.

Host Structure Support in C/C++

With host structure support, the C/C++ precompiler allows host variables to be grouped into a single host structure. This provides a shorthand for referencing that same set of host variables in an SQL statement. For example, the following host structure can be used to access some of the columns in the STAFF table of the SAMPLE database:

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
     {
       short   years;
       double salary;
     } info;
   } staff_record;
```

The fields of a host structure can be any of the valid host variable types. These include all numeric, character, and large object types. Nested host structures are also supported up to 25 levels. In the example above, the field `info` is a sub-structure, whereas the field `name` is not, as it represents a VARCHAR field. The same principle applies to

LONG VARCHAR, VARGRAPHIC and LONG VARGRAPHIC. Pointer to host structure is also supported.

There are two ways to reference the host variables grouped in a host structure in an SQL statement:

1. The host structure name can be referenced in an SQL statement.

   ```
   EXEC SQL SELECT id, name, years, salary
        INTO :staff_record
        FROM staff
        WHERE id = 10;
   ```

   The precompiler converts the reference to `staff_record` into a list, separated by commas, of all the fields declared within the host structure. Each field is qualified with the host structure names of all levels to prevent naming conflicts with other host variables or fields. This is equivalent to the following method.

2. Fully qualified host variable names can be referenced in an SQL statement.

   ```
   EXEC SQL SELECT id, name, years, salary
        INTO :staff_record.id, :staff_record.name,
             :staff_record.info.years, :staff_record.info.salary
        FROM staff
        WHERE id = 10;
   ```

   References to field names must be fully qualified even if there are no other host variables with the same name. Qualified sub-structures can also be referenced. In the example above, `:staff_record.info` can be used to replace `:staff_record.info.years, :staff_record.info.salary`.

Since a reference to a host structure (first example) is equivalent to a comma-separated list of its fields, there are instances where this type of reference may lead to an error. For example:

```
EXEC SQL CONNECT TO :staff_record;
```

Here, the CONNECT statement expects a single character-based host variable. By giving a host structure instead, the statement results in a precompile-time error:

```
SQL0087N  Host variable "staff_record" is a structure used where structure
references are not permitted.
```

Other uses of host structures, which may cause an SQL0087N error to occur, include PREPARE, EXECUTE IMMEDIATE, CALL, indicator variables and SQLDA references. Host structures with exactly one field are permitted in such situations, as are references to individual fields (second example).

Indicator Tables

An indicator table is a collection of indicator variables to be used with a host structure. It must be declared as an array of short integers. For example:

```
short ind_tab[10];
```

The example above declares an indicator table with 10 elements. The following shows the way it can be used in an SQL statement:

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record INDICATOR :ind_tab
     FROM staff
     WHERE id = 10;
```

The following lists each host structure field with its corresponding indicator variable in the table:

staff_record.id ind_tab[0]

staff_record.name ind_tab[1]

staff_record.info.years ind_tab[2]

staff_record.info.salary ind_tab[3]

Note: An indicator table element, for example `ind_tab[1]`, cannot be referenced individually in an SQL statement. The keyword INDICATOR is optional. The number of structure fields and indicators do not have to match; any extra indicators are unused, and any extra fields do not have indicators assigned to them.

A scalar indicator variable can also be used in the place of an indicator table to provide an indicator for the first field of the host structure. This is equivalent to having an indicator table with only 1 element. For example:

```
short scalar_ind;

EXEC SQL SELECT id, name, years, salary
          INTO :staff_record INDICATOR :scalar_ind
          FROM staff
          WHERE id = 10;
```

If an indicator table is specified along with a host variable instead of a host structure, only the first element of the indicator table, for example `ind_tab[0]`, will be used:

```
EXEC SQL SELECT id
          INTO :staff_record.id INDICATOR :ind_tab
          FROM staff
          WHERE id = 10;
```

If an array of short integers is declared within a host structure:

```
struct tag
{
  short i[2];
} test_record;
```

The array will be expanded into its elements when `test_record` is referenced in an SQL statement making `:test_record` equivalent to `:test_record.i[0], :test_record.i[1]`.

SQL Enhancements

BIGINT Data Type

The BIGINT data type is a newly supported SQL data type that allows applications to define BIGINT host variables and retrieve data into 64-bit integer types when this is supported by the programming language.

The sections below show the mapping between the BIGINT data type and C and C++, COBOL, and Java.

BIGINT Data Type in C and C++

The following table shows the C/C++ equivalent column type for the new BIGINT data type. When the precompiler finds a host variable declaration, it determines the appropriate SQL type value. The database manager uses this value to convert the data exchanged between the application and itself.

Table 6. BIGINT Data Type Mapped to C/C++ Declarations

SQL Column Type[1]	C/C++ Data Type[2]	SQL Column Type Description
BIGINT (492 or 493)	long long long long int __int64	64-bit signed integer

Notes:

1. The first number under **Column Type** indicates that an indicator variable is not provided, and the second number indicates that an indicator variable is provided. An indicator variable is needed to indicate NULL values, or to hold the length of a truncated string. These are the values that would appear in the SQLTYPE field of the SQLDA for these data types.
2. Windows operating systems use __int64, where __ represents 2 underscores.

BIGINT Data Type in COBOL

The table below shows the COBOL equivalent of each column type. When the precompiler finds a host variable declaration, it determines the appropriate SQL type value. The database manager uses this value to convert the data exchanged between the application and itself.

Table 7 (Page 1 of 2). BIGINT Data Type Mapped to COBOL Declarations

SQL Column Type[1]	COBOL Data Type	SQL Column Type Description
BIGINT (492 or 493)	01 name PIC S9(18) COMP-5.	64-bit signed integer

Table 7 (Page 2 of 2). BIGINT Data Type Mapped to COBOL Declarations

SQL Column Type[1]	COBOL Data Type	SQL Column Type Description

Notes:

1. The first number under **Column Type** indicates that an indicator variable is not provided, and the second number indicates that an indicator variable is provided. An indicator variable is needed to indicate NULL values, or to hold the length of a truncated string. These are the values that would appear in the SQLTYPE field of the SQLDA for these data types.

BIGINT Data Type in Java

When you call UDFs and stored procedures that are implemented as Java methods, DB2 converts SQL types to and from Java types for you. The table below shows how BIGINT is converted.

Table 8. BIGINT Data Type and Java Objects

SQL Type	Java Type (UDF)	Java Type (Stored Procedure)
BIGINT (492 or 493)	long	long

Fetch-first-clause

Use the FETCH FIRST clause with the SELECT statement to set a maximum number of rows that can be retrieved. It lets the database manager know that the application does not want to retrieve more than a specific number of rows, regardless of how many rows there might be in the result table when this clause is specified.

For more detailed information on how to specify the FETCH FIRST clause and the SELECT statement, refer to the *SQL Reference*.

Altering Tables

When a table is created, the default for append mode is OFF, meaning when the table has data inserted, the data is placed where free space is available in data pages.

If you want new table data appended to the end of the last table page, change the append option on the ALTER TABLE statement to ON.

Recognizing Equivalence of Repeated Host Variables

Users are strongly advised not to change the SQLDA after binding a DB2 embedded SQL program. If there are two or more occurences of a host variable in an SQL statement, the SQLDA will have duplicate entries for the variable, but only the first instance of the variable will be used by DB2. If a variable reference is change to a different variable, it will have no affect on the program. In general, users are advised not to change the output from the precompiler.

Programming in JDBC

This section deals with using JDBC and Java to access data from your DB2 databases. For information on SQLJ, refer to “Embedded SQL for Java (SQLJ) Programming” on page 117.

DB2 together with JDBC provides support for:

- Developing Java applications and applets that access DB2 databases through the Java Development Kit (JDK) Version 1.1. The JDK includes the Java Database Connectivity (JDBC) API from Sun Microsystems. See the Web Page at `http://www.software.ibm.com/data/db2/java` for information on this specification.
- Creating user-defined functions (UDFs) and stored procedures in Java.

This section describes how to access DB2 databases using JDBC. The specific topics discussed are:

- Getting Started
- How Does It Work?
- Creating and Running JDBC Applets and Applications
- Creating Java UDFs and Stored Procedures

Getting Started

You can use DB2's JDBC support to build both:

- Java applications, which rely on the DB2 Client Application Enabler (CAE) to connect to DB2.
- Java applets, that do not require any DB2 component code on the client.

If your application or applet uses JDBC, you need to familiarize yourself with the JDBC specification, *JDBC: A Java SQL API,* available from Sun Microsystems. This specification describes how to call JDBC APIs to access a database and manipulate data in that database.

You should also read through this section to learn about DB2's extensions to JDBC and its few limitations (refer to “Extensions” on page 105). If you plan to create UDFs or stored procedures in Java, refer to “Creating and Using Java User-Defined Functions” on page 108 and “Creating and Using Java Stored Procedures” on page 110, as there are considerations that are different for Java than for other languages.

To help you begin coding your program, sample applications and applets are provided in the `sqllib/samples/java` directory. (All instances of `sqllib/samples/java` in this book also refer to the `%DB2PATH%\samples\java` directory on Windows and OS/2, where %DB2PATH% is the path where DB2 is installed.) If you have created the SAMPLE database, you can also run the samples. See the *SQL Reference* for information about the SAMPLE database.

How Does It Work?

There are two independent components to DB2's Java enablement:

- Support for client applications and applets written in Java using JDBC to access DB2 (see "JDBC Applet and Application Support").
- Support for Java UDFs and stored procedures on the server (see "Java UDFs and Stored Procedures" on page 103)

JDBC Applet and Application Support

Figure 1 illustrates how the JDBC applet driver works. The driver consists of a JDBC client and a JDBC server. The JDBC client driver is loaded on the Web browser along with the applet. When a connection to a DB2 database is requested by the applet, the client opens a TCP/IP socket to the JDBC server on the machine where the Web server is running. After a connection is set up, the client sends each of the subsequent database access requests from the applet to the JDBC server though the TCP/IP connection. The JDBC server then makes corresponding CLI (ODBC) calls to perform the task. Upon completion, the JDBC server sends the results back to the client through the connection.

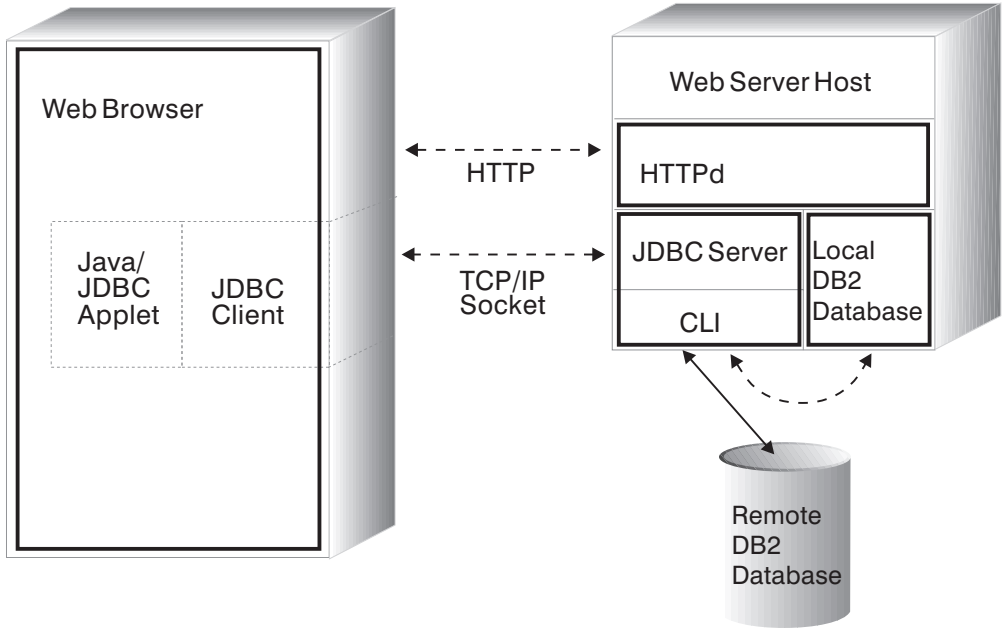


Figure 1. DB2's JDBC Applet Implementation

Figure 2 on page 103 illustrates how a DB2 JDBC application works. You can think of a DB2 JDBC application as a DB2 CLI application, only you write it using the Java language. Calls to JDBC are translated to calls to DB2 CLI, through Java native methods. This dependency requires that the DB2 CAE component be installed at the

client. A JDBC request flows through DB2 CLI to the DB2 server through the normal CAE communication flow.

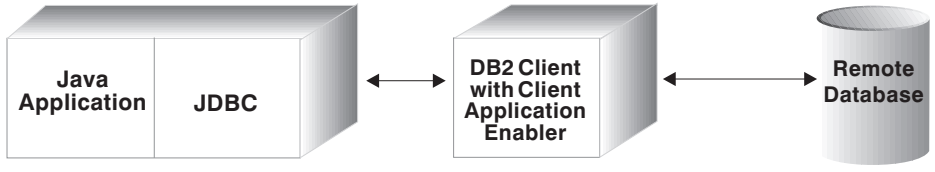


Figure 2. DB2's JDBC Application Implementation

Writing a Java JDBC application or applet is very similar to writing a C application using DB2 CLI or ODBC to access the database. The primary difference between applications and applets is that an application requires the DB2 CAE to be installed on the client, and uses the CAE to communicate with DB2; the applet depends on a Java-enabled Web browser, and does not require any DB2 code installed on the client.

You need to treat applets differently than applications, as applets are delivered over the network (intranet or Internet). You must install the DB2 server or client on the same machine as your Web server. The JDBC applet and application support is installed as part of DB2.

Java UDFs and Stored Procedures

Creating Java UDFs and stored procedures is very similar to creating UDFs and stored procedures in other supported programming languages. Once you have created and registered them, you can call them from programs in any language. Typically, you may call JDBC APIs from your stored procedures, but you can not call them from UDFs.

Note: DB2 does not support Java stored procedures and Java user-defined functions (UDFs) accessing DB2 databases on HP-UX and SCO UnixWare servers.

To run your UDFs and stored procedures, DB2 calls the Java interpreter on the server. DB2 does not include a Java interpreter; your database administrator must install and configure the appropriate Java Development Kit (JDK) on your DB2 server before starting up the database.

The runtime libraries for the Java interpreter must be available in the system search paths (PATH or LIBPATH or LD_LIBRARY_PATH, and CLASSPATH). For more information on setting up the Java environment on OS/2 and Windows, see Appendix D, "Building Applications for Windows and OS/2 Environments Updates" on page 149. For more information on setting up the Java environment on UNIX platforms see *Building Applications for UNIX Environments*.

DB2 loads or starts the Java interpreter on the first call to a Java UDF or stored procedure (see "Creating and Using Java User-Defined Functions" on page 108 or "Creating and Using Java Stored Procedures" on page 110). For unfenced UDFs and stored procedures, DB2 loads one Java interpreter per database instance, and runs it inside the

database engine's address space for best performance. For fenced UDFs, DB2 uses a distinct interpreter inside the db2udf process; similarly, fenced stored procedures use a distinct interpreter inside the db2dari process. In all cases, the Java interpreter stays loaded until the embedding process ends.

Creating and Running JDBC Applets and Applications

Whether you're writing an application or applet, you would typically call JDBC APIs to:

1. Import the appropriate Java packages and classes (`java.sql.*`).
2. Load the appropriate JDBC driver (`COM.ibm.db2.jdbc.app.DB2Driver` for applications; `COM.ibm.db2.jdbc.net.DB2Driver` for applets).
3. Connect to the database, specifying the location with a URL (as defined in Sun's JDBC specification) and using the `db2` subprotocol. For applets, you must also provide the userid, password, host name, and the port number for the applet server; for applications, the Client Application Enabler provides the required values.
4. Pass SQL statements to the database.
5. Receive the results.
6. Close the connection.

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

Distributing and Running a JDBC Applet

Like other Java applets, you distribute your JDBC applet over the network (intranet or Internet). Typically you would imbed the applet in a hypertext markup language (HTML) page. For example, to call the sample applet `DB2Applt.java`, (provided in `sqllib/samples/java`) you might use the tag:

```
<applet code="DB2Applt.class" width=325 height=275 archive="db2java.zip">
```

To run your applet, you need only a Java-enabled Web browser on the client machine. When you load your HTML page, the applet tag downloads the Java applet to your machine which then downloads the Java class files, including the `COM.ibm.db2.jdbc.net` class which is DB2's JDBC driver. When your applet calls the JDBC API to connect to DB2, the JDBC driver establishes separate communications with the DB2 database through the JDBC applet server residing on the Web server.

For your applets to run, you need to ensure that the correct files are installed in the proper places, as follows:

1. Install DB2 (server or client) on the same machine as your Web server. The Java applet and JDBC support is installed as part of DB2.
2. Create your own HTML file, or use a modified version of the supplied `DB2Applt.html` file and move it to the same directory as your class file. Copy your `.class` file, or for the sample, `sqllib/samples/java/samples.zip` and the `sqllib/java/db2java.zip` file to this directory.
3. Start DB2's JDBC applet server on your Web server by typing:

```
db2jstrt portno
```

where *portno* is an unused TCP/IP port number that applets can use.

4. On your client machine, start your Web browser and load your HTML file to run your applet.

Distributing and Running a JDBC Application

Distribute your JDBC application as you would any other Java application. As the application uses DB2's CAE to communicate with the DB2 server, you have no special security concerns; authority verification is performed by the CAE.

To run your application on a client machine, you must also have installed on that machine:

- The Java interpreter, which you need to run any Java code.
- DB2's Client Application Enabler, which also includes the DB2 JDBC driver.

Start your application from the GUI or command line, like any other application.

A sample application, DB2Appl.java, is provided in the `sqllib/samples/java` directory. If you have created the SAMPLE database, you can also run the sample by adding `samples/java/samples.zip` to your CLASSPATH environment variable, changing to the `sqllib/samples/java` directory, and entering the following command:

```
java DB2Appl
```

See the *SQL Reference* for information on the SAMPLE database.

Extensions

DB2 Universal Database supports the JDK Version 1.1 JDBC specification. You may have to modify applications and applets that are written to the JDBC Version 1.0 to work properly with DB2 Version 5.

There are also special considerations for graphical and large objects (LOBs).

Using Graphical and Large Objects: The JDBC specification does not explicitly mention large objects (LOBs) or graphic types.

Treat LOBs as the corresponding LONGVAR type. Because LOB types are declared in SQL with a maximum length, ensure that you do not return arrays or strings longer than the declared limit. This consideration applies to SQL string types as well.

For GRAPHIC and DBCLOB types, treat them as the corresponding CHAR types. The following JDBC APIs behave as described below:

setString	Converts from Unicode to database format.[1]
setAsciiStream	Converts from local code page to database format.[2]
setUnicodeStream	Converts from Unicode to database format.[1]
getString	Converts from database format to Unicode.[1]

getAsciiStream Converts from database format to local code page format.[2]

getUnicodeStream Converts from database format to Unicode.[1]

Notes:

1. The DB2 client first converts the data from the database format to the client format; DB2 then converts the data to Unicode
2. The DB2 client performs this type conversion. See your DB2 client information for supported code pages and conversions.

Creating Java UDFs and Stored Procedures

Along with supporting client-side Java code, DB2 also supports creating user-defined functions (UDF) and stored procedures in Java that reside on the server. This Java support does not alter the support for UDFs and stored procedures in other programming languages.

UDFs and stored procedures written in Java provide the same capability as existing UDFs and stored procedures; they are simply methods in Java classes. Once you create and register these UDFs and stored procedures, place the Java classes in the correct file location, described in “Where to Put Java Classes” on page 108. You can then call them from a program in any language. DB2 calls the Java interpreter to run them; they run as if they are a part of a Java application, and therefore are not subject to applet security restrictions.

DB2 handles type conversion (see “Mapping Between SQL Types and Java Objects”) between SQL types and Java objects for you, as it does for other programming languages. Because SQL string and LOB types are declared in SQL with a maximum length, ensure that your Java methods do not return arrays or strings that are longer than the declared limit. DB2 detects many possible errors in data conversion and signals them by throwing an exception.

Because you can overload Java methods, two methods with the same name but different argument lists can coexist in the same Java class. Make sure that your Java methods that implement UDFs and stored procedures have the exact *signature* expected, that is, the list of formal arguments and the method name.

Mapping Between SQL Types and Java Objects

When you call UDFs and stored procedures that are implemented as Java methods, DB2 converts SQL types to and from Java types for you as described in Table 9 on page 107. Several of these classes are provided in the Java package `COM.ibm.db2.app`.

Table 9. DB2 SQL Types and Java Objects

SQL Type	Java Type (UDF)	Java Type (Stored Procedure)
SMALLINT (500/501)	short	short
INTEGER (496/497)	Int	Int
BIGINT (492/493)	long	long
FLOAT (480/481)	double	double
REAL (480/481)[1]	float	float
DECIMAL(p,s) (484/485)	BigDecimal	BigDecimal
NUMERIC(p,s) (504/505)	BigDecimal	BigDecimal
CHAR(*n*) (452/453)	String	String
CHAR(*n*) FOR BIT DATA (452/453)	Blob	Blob
C null-terminated string (400/401)[2]	n/a	String
VARCHAR(*n*)(448/449)	String	String
VARCHAR(*n*) FOR BIT DATA (448/449)	Blob	Blob
LONG VARCHAR (456/457)	Clob	Clob
LONG VARCHAR FOR BIT DATA (456/457)	Blob	Blob
GRAPHIC(*n*) (468/469)	String	String
C null-terminated graphic string (460/461)[2]	n/a	String
VARGRAPHIC(*n*) (464/465)	String	String
LONG VARGRAPHIC (472/473)[3]	Clob	Clob
BLOB(*n*)(404/405)[3]	Blob	Blob
CLOB(*n*) (408/409)[3]	Clob	Clob
DBCLOB(*n*) (412/413)[3]	Clob	Clob
DATE (384/385)[4]	String	String
TIME (388/389)[4]	String	String
TIMESTAMP (392/393)[4]	String	String

Notes:

1. The difference between REAL and DOUBLE in the SQLDA is the length value (4 or 8).
2. Parenthesized types, such as the C null-terminated graphic string, occur in stored procedures when the calling application uses embedded SQL with some host variable types.
3. The Blob and Clob classes are provided in the `COM.ibm.db2.app` package. Their interfaces include routines to generate an InputStream and OutputStream for reading from and writing to a Blob, and a Reader and Writer for a Clob. Refer to "Classes for Java Stored Procedures and UDFs" on page 112 for descriptions of the classes.
4. SQL DATE, TIME, and TIMESTAMP values use the ISO string encoding in Java, as they do for UDFs coded in C.

Instances of classes COM.ibm.db2.app.Blob and COM.ibm.db2.app.Clob represent the LOB data types (BLOB, CLOB, and DBCLOB). These classes provide a limited interface to read LOBs passed as inputs, and write LOBs returned as outputs. Reading and writing of LOBs occur through standard Java I/O stream objects. For the Blob class, the routines getInputStream() and getOutputStream() return an InputStream or OutputStream object through which the BLOB content may be processed bytes-at-a-time. For a Clob, the routines getReader() and getWriter() will return a Reader or Writer object through which the CLOB or DBCLOB content may be processed characters-at-a-time.

If such an object is returned as an output using the set() method, code page conversions may be applied in order to represent the Java Unicode characters in the database code page.

Where to Put Java Classes

Store all Java class files that implement UDFs or stored procedures in the sqllib/function directory. If you declare a class to be part of a Java package, create the corresponding subdirectories under sqllib/function and place the files in the correct subdirectory. For example, if you create a class ibm.tests.test1, store the corresponding Java byte-code file (named test1.class) in sqllib/function/ibm/tests.

The Java interpreter that DB2 invokes uses the CLASSPATH environment variable to locate Java files. DB2 adds the entries sqllib/function and sqllib/java/db2java.zip to the front of your CLASSPATH setting.

To set your environment so that the Java interpreter can find where you have stored the Java class files you may need to set the jdk11_path configuration parameter, or else use the default value. Also, you may need to set the java_heap_sz configuration parameter to increase the heap size for your application. See the *Administration Guide* for more information on these configuration parameters.

Creating and Using Java User-Defined Functions

You can create and use UDFs in Java just as you would in other languages, with only a few minor differences. After you code the UDF, you register it with the database using the CREATE FUNCTION statement. See the *SQL Reference* for information on registering a Java UDF using this statement. You can then call it from any DB2 application in any language, including Java. The UDF can be fenced or unfenced, and you can also use options to modify how the UDF is run. See “Changing How a Java UDF Runs” on page 110.

Some sample Java UDFs are provided in DB2Udf.java in the sqllib/samples/java directory. To register and invoke the sample UDFs, follow the instructions in the DB2Udf.java file.

Coding a Java UDF: In general, if you declare a UDF taking arguments of SQL types *t1*, *t2*, and *t3*, returning type *t4*, it will be called as a Java method with the expected Java signature:

```
public void name (T1 a, T2 b, T3 c, T4 d)  { .....}
```

Where:

- *name* is the method name
- *T1* through *T4* are the Java types that correspond to SQL types *t1* through *t4*.
- *a*, *b*, and *c* are arbitrary variable names for the input arguments.
- *d* is an arbitrary variable name that represents the UDF result being computed.

For example, given a UDF called `sample!test3` that returns INTEGER and takes arguments of type CHAR(5), BLOB(10K), and DATE, DB2 expects the Java implementation of the UDF to have the following signature:

```
import COM.ibm.db2.app.*;
public class sample implements UDF {
   public void test3(String arg1, Blob arg2, String arg3,
                     int result) { ... }
}
```

Java UDFs that implement table functions have more arguments. Beside the variables representing the input, an additional variable appears for each column in the resulting row. For example, a table function may be declared as:

```
public void test4(String arg1,
             int result1, Blob result2, String result3);
```

SQL NULL values are represented by Java variables that are not initialized. These variables have a value of zero if they are primitive types, and Java null if they are object types, in accordance with Java rules. To tell an SQL NULL apart from an ordinary zero, you can call the function `isNull` for any input argument:

```
{ ....
  if (isNull(1)) { /* argument #1 was a SQL NULL */ }
  else           { /* not NULL */ }
}
```

In the above example, the argument numbers start at one. The `isNull()` function, like the other functions that follow, are inherited from the `COM.ibm.db2.app.UDF` interface. This must be implemented by Java classes containing UDFs.

To return a result from a scalar or table UDF, use the `set()` method in the UDF, as follows:

```
{ ....
  set(2, value);
}
```

Where '2' is the index of an output argument, and value is a literal or variable of a compatible type. The argument number is the index in the argument list of the selected output. In the first example in this section, the `int result` variable has an index of 4; in the second, `result1` through `result3` have indices of 2 through 4. An output argument that is not set before the UDF returns will have a NULL value.

Like C modules used in UDFs and stored procedures, you cannot use the Java standard I/O streams (`System.in`, `System.out`, and `System.err`) in Java UDFs. For an example of a Java UDF, see the file `DB2Udf.java` in the `sqllib/samples/java` directory.

Remember that all Java class files that you use to implement a UDF must reside in the `sqllib/function directory` or an appropriate subdirectory. See "Where to Put Java Classes" on page 108.

Changing How a Java UDF Runs: Typically, DB2 calls a UDF many times, once for each row of a result set in a query. The implementing Java class is instantiated once per row, and the selected method of each new instance is called once.

You can change this model by declaring the UDF with the SCRATCHPAD option. When you use this option, the Java class is instantiated only once, and the same instance is reused for the entire query. While C-language UDFs can maintain state between calls in a scratchpad area provided by the database engine, Java UDFs can simply use instance variables. Note that there is still a separate Java UDF instance per query reference to that UDF, just as there is for C UDFs. If a UDF is called in several places in a query, each call will have its own Java object.

At the end of a query, if you specify the FINAL CALL option on the CREATE FUNCTION statement, the object's `public void close()` method is called. If you do not define this method, a stub function takes over and the event is ignored.

If you specify the ALLOW PARALLEL clause for a Java UDF in the CREATE FUNCTION statement, DB2 may elect to evaluate the UDF in parallel. If this occurs, several distinct Java objects may be created on different partitions. Each object receives a subset of the rows. Note that are no such object instances are created for C or C++ UDFs.

As with other UDFs, Java UDFs can be fenced or unfenced. Unfenced UDFs are run inside the address space of the database engine; fenced UDFs are run in a separate process. Although Java UDFs cannot inadvertently corrupt the address space of their embedding process, they can terminate or slow down the process. Therefore, when you are debugging UDFs written in Java, you should run them as fenced UDFs.

Refer to "COM.ibm.db2.app.UDF" on page 113 for a description of the `COM.ibm.db2.app.UDF` interface. This interface describes other useful calls that you can make within a UDF, such as `setSQLstate` and `getDBinfo`.

Creating and Using Java Stored Procedures

As with UDFs, you can create and use stored procedures in Java just like you can for other programming languages. There are some programming considerations (as discussed in "Coding Java Stored Procedures" on page 111) that you need to know when you write your Java code. You also need to *register* your Java stored procedure. Refer to the CREATE PROCEDURE statement in the *SQL Reference* for information on how to register your stored procedure.

Note: If you are running a *database server with local clients* node type, you must set the `maxdari` database manager configuration parameter to a non-zero value before you invoke a Java stored procedure.

A sample Java stored procedure, DB2Stp.java, is provided in `sqllib/samples/java`.

Remember that all Java class files that you use to implement a stored procedure must reside in the `sqllib/function` directory or appropriate subdirectory (as discussed in "Where to Put Java Classes" on page 108).

Coding Java Stored Procedures: Java stored procedures are public instance methods. Within the classes, the stored procedures are identified by their method name and signature. When you call a stored procedure, its signature is generated dynamically based on the variable types that you pass to it.

Java stored procedures are very similar to the Java UDFs described in "Creating and Using Java User-Defined Functions" on page 108. Like table functions, they can have multiple outputs. They also use the same conventions for NULL values, and the same `set` routine for output. The main difference is that a Java class that contains stored procedures must implement the `COM.ibm.db2.app.StoredProc` interface instead of the `COM.ibm.db2.app.UDF` interface. Refer to "COM.ibm.db2.app.StoredProc" on page 112 for a description of the `COM.ibm.db2.app.StoredProc` interface.

This interface provides the following routine to fetch a JDBC connection to the embedding application context:

```
public java.sql.Connection getConnection()
```

You can use this handle to run SQL statements. Other methods of the `StoredProc` interface are listed in the file `sqllib/samples/java/StoredProc.java`.

The following is a small stored procedure with one input and two outputs. It executes the given SQL query, and returns the number of rows in the result, and the SQLSTATE:

```
import COM.ibm.db2.app.*;
import java.sql.*;
public class sample2 implements StoredProc {
   public void donut(String query, int rowCount,
                     String sqlstate) throws Exception {
      try {
         Statement s = getConnection().createStatement();
         ResultSet r = s.executeQuery(query);
         int counter = 0;
         while(r.next()) {
             counter ++;
         }
         r.close();  s.close();
         set(2, counter);
      } catch(SQLException x) {
         set(3, x.getSQLState());
      }
   }
}
```

Classes for Java Stored Procedures and UDFs

There are five classes/interfaces that you can use with Java Stored Procedures or UDFs:

- COM.ibm.db2.app.StoredProc
- COM.ibm.db2.app.UDF
- COM.ibm.db2.app.Lob
- COM.ibm.db2.app.Blob
- COM.ibm.db2.app.Clob

The following sections describe the public aspects of these classes' behavior:

COM.ibm.db2.app.StoredProc: A Java class that contains methods intended to be called as stored procedures must be public and must implement this Java interface. You must declare such a class as follows:

```
public class <user-STP-class> implements COM.ibm.db2.app.StoredProc{ ... }
```

You can only call inherited methods of the `COM.ibm.db2.app.StoredProc` interface in the context of the currently executing stored procedure. For example, you cannot use operations on LOB arguments, result- or status-setting calls, etc., after a stored procedure returns. A Java exception will be thrown if you violate this rule.

Argument-related calls use a column index to identify the column being referenced. These start at 1 for the first argument. At this time, all arguments of a stored procedure are considered INOUT and thus are both inputs and outputs.

Any exception returned from the stored procedure is caught by the database and returned to the caller with SQLCODE -4302, SQLSTATE 38501. A JDBC SQLException

or SQLWarning is handled specially and passes its own SQLCODE, SQLSTATE etc. to the calling application verbatim.

The following methods are associated with the `COM.ibm.db2.app.StoredProc` class:

```
public StoredProc() [default constructor]
```

This constructor is called by the database before the stored procedure call.

```
public boolean isNull(int) throws Exception
```

This function tests whether an input argument with the given index is an SQL NULL.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

This function sets the output argument with the given index to the given value. The index has to refer to a valid output argument, the data type must match, and the value must have an acceptable length and contents. Strings with Unicode characters must be representable in the database code page. Errors result in an exception being thrown.

```
public java.sql.Connection getConnection() throws Exception
```

This function returns a JDBC object that represents the calling application's connection to the database. It is analogous to the result of a null `SQLConnect()` call in a C stored procedure.

COM.ibm.db2.app.UDF: A Java class that contains methods intended to be called as UDFs must be public and must implement this Java interface. You must declare such a class as follows:

```
public class <user-UDF-class> implements COM.ibm.db2.app.UDF{ ... }
```

You can only call methods of the `COM.ibm.db2.app.UDF` interface in the context of the currently executing UDF. For example, you cannot use operations on LOB arguments, result- or status-setting calls, etc., after a UDF returns. A Java exception will be thrown if this rule is violated.

Argument-related calls use a column index to identify the column being set. These start at 1 for the first argument. Output arguments are numbered higher than the input arguments. For example, a scalar UDF with three inputs uses index 4 for the output.

Any exception returned from the UDF is caught by the database and returned to the caller with SQLCODE -4302, SQLSTATE 38501.

The following methods are associated with the `COM.ibm.db2.app.UDF` class:

```
public UDF() [default constructor]
```

This constructor is called by the database at the beginning of a series of UDF calls. It precedes the first call to the UDF.

```
public void close()
```

This function is called by the database at the end of a UDF evaluation, if the UDF was created with the FINAL CALL option. It is analogous to the final call for a C UDF. If a Java UDF class does not implement this function, a no-op stub will handle and ignore this event.

```
public boolean isNull(int) throws Exception
```

This function tests whether an input argument with the given index is an SQL NULL.

```
public boolean needToSet(int) throws Exception
```

This function tests whether an output argument with the given index needs to be set. This may be false for a table UDF declared with DBINFO, if that column is not used by the UDF caller.

```
public void set(int, short) throws Exception
public void set(int, int) throws Exception
public void set(int, double) throws Exception
public void set(int, float) throws Exception
public void set(int, java.math.BigDecimal) throws Exception
public void set(int, String) throws Exception
public void set(int, COM.ibm.db2.app.Blob) throws Exception
public void set(int, COM.ibm.db2.app.Clob) throws Exception
```

This function sets the output argument with the given index to the given value. The index has to refer to a valid output argument, the data type must match, and the value must have an acceptable length and contents. Strings with Unicode characters must be representable in the database code page. Errors result in an exception being thrown.

```
public void setSQLstate(String) throws Exception
```

This function may be called from a UDF to set the SQLSTATE to be returned from this call. A table UDF should call this function with "02000" to signal the end-of-table condition. If the string is not acceptable as an SQLSTATE, an exception will be thrown.

```
public void setSQLmessage(String) throws Exception
```

This function is similar to the `setSQLstate` function. It sets the SQL message result. If the string is not acceptable (for example, longer than 70 characters), an exception will be thrown.

```
public String getFunctionName() throws Exception
```

This function returns the name of the executing UDF.

```
public String getSpecificName() throws Exception
```

This function returns the specific name of the executing UDF.

```
public byte[] getDBinfo() throws Exception
```

This function returns a raw, unprocessed DBINFO structure for the executing UDF, as a byte array. You must first declare it with the DBINFO option.

```
public String getDBname() throws Exception
public String getDBauthid() throws Exception
public String getDBtbschema() throws Exception
public String getDBtbname() throws Exception
public String getDBcolname() throws Exception
public String getDBver_rel() throws Exception
public String getDBplatform() throws Exception
```

These functions return the value of the appropriate field from the DBINFO structure of the executing UDF.

```
public int[] getDBcodepg() throws Exception
```

This function returns the SBCS, DBCS, and composite code page numbers for the database, from the DBINFO structure. The returned integer array has the respective numbers as its first three elements.

```
public byte[] getScratchpad() throws Exception
```

This function returns a copy of the scratchpad of the currently executing UDF. You must first declare the UDF with the SCRATCHPAD option.

```
public void setScratchpad(byte[]) throws Exception
```

This function overwrites the scratchpad of the currently executing UDF with the contents of the given byte array. You must first declare the UDF with the SCRATCHPAD option. The byte array must have the same size as `getScratchpad()` returns.

COM.ibm.db2.app.Lob: This class provides utility routines that create temporary Blob or Clob objects for computation inside user-defined functions or stored procedures.

The following methods are associated with the COM.ibm.db2.app.Lob class:

```
public static COM.ibm.db2.app.Blob newBlob() throws Exception
```

This function creates a temporary Blob. It will be implemented using a LOCATOR if possible.

```
public static COM.ibm.db2.app.Clob newClob() throws Exception
```

This function creates a temporary Clob. It will be implemented using a LOCATOR if possible.

COM.ibm.db2.app.Blob: An instance of this class is passed by the database to represent a BLOB as UDF or stored procedure input, and may be passed back as output. The application may create instances, but only in the context of an executing UDF or stored procedure. Uses of these objects outside such a context will throw an exception.

The following methods are associated with the `COM.ibm.db2.app.Blob` class:

```
public static COM.ibm.db2.app.Blob new() throws Exception
```

This function creates a temporary Blob. It will be implemented using a LOCATOR if possible.

```
public long size() throws Exception
```

This function returns the length (in bytes) of the BLOB.

```
public java.io.InputStream getInputStream() throws Exception
```

This function returns a new InputStream to read the contents of the BLOB. Efficient seek/mark operations are available on that object.

```
public java.io.OutputStream getOutputStream() throws Exception
```

This function returns a new OutputStream to append bytes to the BLOB. Appended bytes become immediately visible on all existing InputStream instances produced by this object's `getInputStream()` call.

COM.ibm.db2.app.Clob: An instance of this class is passed by the database to represent a CLOB or DBCLOB as UDF or stored procedure input, and may be passed back as output. The application may create instances, but only in the context of an executing UDF or stored procedure. Uses of these objects outside such a context will throw an exception.

Clob instances store characters in the database code page. Some Unicode characters may not be representable in this code page, and may cause an exception to be thrown during conversion. This may happen during an append operation, or during a UDF or StoredProc `set()` call. This is necessary to hide the distinction between a CLOB and a DBCLOB from the Java programmer.

The following methods are associated with the `COM.ibm.db2.app.Clob` class:

```
public static COM.ibm.db2.app.Clob new() throws Exception
```

This function creates a temporary Clob. It will be implemented using a LOCATOR if possible.

```
public long size() throws Exception
```

This function returns the length (in characters) of the CLOB.

```
public java.io.Reader getReader() throws Exception
```

This function returns a new Reader to read the contents of the CLOB or DBCLOB. Efficient seek/mark operations are available on that object.

```
public java.io.Writer getWriter() throws Exception
```

This function returns a new Writer to append characters to this CLOB or DBCLOB. Appended characters become immediately visible on all existing Reader instances produced by this object's `GetReader()` call.

Embedded SQL for Java (SQLJ) Programming

This section describes Embedded SQL for Java (SQLJ) programming. The specific topics discussed are:

- SQLJ and DB2 SQLJ Support
- Basic SQLJ Concepts
- Advanced Features
- Comparison with ANSI/ISO Embedded
- SQLJ Translator Reference

SQLJ and DB2 SQLJ Support

SQLJ is embedded SQL for Java, and DB2 SQLJ support facilitates the creation, building and running of SQLJ programs against DB2 databases.

What is SQLJ?

SQLJ consists of a set of programming extensions that define interaction between SQL and Java. It comprises a set of clauses that extend Java programs to include static SQL constructs. An SQLJ `translator` is a utility that transforms those `SQLJ clauses` into standard Java code that accesses the database through a call interface. The output of an SQLJ translator is a generated Java source program that can then be compiled by any Java compiler. Java programs containing embedded SQL can be subjected to static analysis of SQL statments for the purposes of syntax checking, type checking and schema validation.

SQLJ supports only static SQL constructs. The counterpart to static SQL is `dynamic` SQL, a call interface for passing strings to a database as SQL commands. No analysis or checking of those strings is done until the database receives them at execution time. A dynamic SQL API for Java has been specified by JavaSoft, called JDBC. For detailed information about DB2 JDBC support, please see “Programming in JDBC” on page 101.

SQLJ relies upon JDBC for support of dynamic SQL, and does not attempt to replicate the features of JDBC. Rather, SQLJ contains mechanisms that enable a Java programmer to easily move between the two environments and share state information (for example, connection contexts).

What is DB2 SQLJ Support?

DB2 SQLJ support is provided by the DB2 Software Developer's Kit (DB2 SDK). Along with DB2 JDBC support provided by the DB2 Client Application Enabler (DB2 CAE), DB2 SQLJ support allows you to create, build, and run embedded SQL for Java applications, applets, stored procedures and user-defined functions (UDFs). These contain static SQL and use embedded SQL statements that are bound to a DB2 database.

The SQLJ support provided by the DB2 SDK includes:

- The SQLJ translator, `sqlj`, which replaces embedded SQL statements in the SQLJ program with Java source statements, and generates a serialized profile which contains information about the SQL operations found in the SQLJ program. The SQLJ translator uses the `sqllib/java/sqlj.zip` file.
- The SQLJ profile printer, `profp`, which prints the contents of a profile generated by the SQLJ translator in plain text.
- The DB2 SQLJ profile customizer, `db2profc`, which precompiles the SQL statements stored in the generated profile, customizing them into calls to the SQLJ runtime function, and generates a package in the DB2 database.
- The DB2 SQLJ profile printer, `db2profp`, which prints the contents of a DB2 customized profile in plain text.
- The SQLJ profile auditor installer, `profdb`, which installs (or uninstalls) debugging class-auditors into an existing set of binary profiles. Once installed, all `RTStatement` and `RTResultSet` calls made during application runtime will be logged to a file (or standard output), which can then be inspected to verify expected behavior and trace errors. Note that only those calls made to the underlying `RTStatement` and `RTResultSet` call interface at runtime are audited.
- The SQLJ runtime classes, available in `sqllib/java/runtime.zip`, consisting of the following packages:

  ```
  sqlj.runtime
  sqlj.runtime.ref
  sqlj.runtime.profile
  sqlj.runtime.profile.ref
  sqlj.runtime.profile.util
  sqlj.runtime.error
  ```

- The DB2 SQLJ runtime function, which provides a runtime interface to the DB2 database manager.

For more information on the SQLJ translator, see “SQLJ Translator Reference” on page 143. For more information on the `db2profc` and `db2profp` commands, see the *Command Reference*. For more information on the SQLJ runtime classes, visit the DB2 Java web page at:

```
http://www.software.ibm.com/data/db2/java
```

DB2 Trace Facilities

SQLJ programs access DB2 using the DB2 JDBC driver, which in turn uses the DB2 CLI/ODBC driver. Therefore, both the CLI/ODBC/JDBC trace facility and the DB2 trace facility, `db2trc`, can be used to diagnose problems. Details on how to take the above traces are explained in the *Troubleshooting Guide*.

The SQLJ runtime function includes a utility to install runtime call tracing capability into SQLJ programs. The utility operates on the profiles associated with a program. Suppose a program uses a profile called `App_SJProfile0`. Then, debugging would be installed into the program with the command:

```
profdb App_SJProfile0.ser
```

The profdb script uses the Java Virtual Machine to run the main() method of class sqlj.runtime.profile.util.AuditorInstaller. For more details on usage and options for the AuditorInstaller class, visit the DB2 Java web page at:

http://www.software.ibm.com/data/db2/java

DB2 SQLJ Usage Notes

1. If you do not specify an sqlj.properties file, the following default values will be used:

   ```
   sqlj.url=jdbc:db2:sample
   sqlj.driver=COM.ibm.db2.jdbc.app.DB2Driver
   sqlj.online=sqlj.semantics.JdbcChecker
   sqlj.offline=sqlj.semantics.OfflineChecker
   ```

   If you do specify an sqlj.properties file, make sure the following options are set:

   ```
   sqlj.url=jdbc:db2:dbname
   sqlj.driver=COM.ibm.db2.jdbc.app.DB2Driver
   sqlj.online=sqlj.semantics.JdbcChecker
   sqlj.offline=sqlj.semantics.OfflineChecker
   ```

   where dbname is the name of the database.

   These options can also be set on the command line.

2. To run an SQLJ program with program name pgmname, do the following:

   - Translate the Java source code with Embedded SQL to generate the Java source code pgmname.java and profiles pgmname_SJProfile0.ser, pgmname_SJProfile1.ser, ... (one profile for each connection context):

     ```
     sqlj pgmname.sqlj
     ```

   - Compile the generated Java source code to generate the Java byte-codes pgmname.class:

     ```
     javac pgmname.java
     ```

   - Install DB2 SQLJ Customizers on generated profiles and create the DB2 packages in the DB2 database dbname:

     ```
     db2profc -user=user-name -password=user-password -url=jdbc:db2:dbname
              -prepoptions="bindfile using pgmname0.bnd package using pgmname0"
              pgmname_SJProfile0.ser
     db2profc -user=user-name -password=user-password -url=jdbc:db2:dbname
              -prepoptions="bindfile using pgmname1.bnd package using pgmname1"
              pgmname_SJProfile1.ser
      ...
     ```

   - Execute the SQLJ program:

     ```
     java pgmname
     ```

3. To print the content of the profiles generated by the SQLJ translator in plain text:

   ```
   profp pgmname_SJProfile0.ser
   profp pgmname_SJProfile1.ser
     ...
   ```

4. To print the content of the DB2 customized version of the profile in plain text:

```
db2profp -user=user-name -password=user-password -url=jdbc:db2:dbname
         pgmname_SJProfile0.ser
db2profp -user=user-name -password=user-password -url=jdbc:db2:dbname
         pgmname_SJProfile1.ser
    ...
```

   where `dbname` is the name of the database.

5. The following pre-compile options are not applicable:

```
NOLINEMACRO
OPTLEVEL
OUTPUT
SQLCA
TARGET
WCHARTYPE
CONNECT
DISCONNECT
SYNCPOINT
SQLRULES
SQLFLAG
```

6. All positioned `UPDATE/DELETE` SQL statements will be dynamically prepared and executed during the runtime. The authorization identifier used for the execution of positioned `UPDATE/DELETE` SQL statements is the `authid` of the person executing the cursor package (the DB2 package that contained the corresponding `OPEN CURSOR` operation).

   The DRDA precompile/bind option `DYNAMICRULES BIND` can be specified to indicate that the authorization identifier used for the execution of positioned `UPDATE/DELETE` SQL statement is the cursor package owner. This DRDA precompile/bind option is not supported by DB2 Universal Database. Also, the positioned `UPDATE/DELETE` SQL statement is not a valid sub-statement in a Compound SQL statement.

7. All host variables specified in compound SQL are input host variables by default. You have to specify the parameter mode identifier `OUT` before the host variable in order to mark it as an output host variable. For example:

```
#sql {begin compound atomic static
        select count(*) into :OUT count1 from employee;
      end compound}
```

8. The following SQLJ syntax for the `VALUES` clause is used to invoke the function `F` with host variable `x` and assign the result to host variable `i`:

```
i = { VALUES (  F(:x) ) };
```

   and will be translated by the SQLJ translator and stored as

```
? = VALUES (F (?))
```

   in the generated profile.

   DB2 will customize the `VALUE` statement into:

```
VALUES(F(?)) INTO ?
```

when connecting to a DB2 Universal Database database but into:

```
SELECT F(?) INTO ? FROM SYSIBM.SYSDUMMY1
```

when connecting to a DB2 for OS/390 database.

If we run the DB2 SQLJ profile customizer, `db2profc`, against a DB2 Universal Database database and generate a bindfile, we cannot use that bindfile to bind up to a DB2 for OS/390 database when there is a `VALUES` clause in the bindfile. This also applies to generating a bindfile against a DB2 for OS/390 database and trying to bind with it to a DB2 Universal Database database.

For detailed information on building and running DB2 SQLJ programs on Windows and OS/2, see Appendix D, “Building Applications for Windows and OS/2 Environments Updates” on page 149. For detailed information on building and running SQLJ programs on UNIX platforms, see *Building Applications for UNIX Environments*.

Advantages of SQLJ over JDBC for Static SQL

Dynamic SQL provides greater flexibility than static SQL since a calling program has the ability to construct and process SQL strings that are dynamically created at runtime. This capability comes at a greater programming cost due to the increased complexity of code necessary to support flexible, dynamic operations. However, many applications do not require this level of complexity because the SQL commands they use are predetermined. Embedded SQL is better suited for these applications as it enables early error checking, allows for precompilation of SQL for faster execution at runtime, and significantly reduces program size and complexity.

Here are some major differences between the two:

- SQLJ source programs are smaller than equivalent JDBC programs since the translator can implicitly handle many of the tedious programming chores that dynamic interfaces must make explicit.
- SQLJ programs can use translation time database connections to type-check static SQL code. JDBC, being a completely dynamic API, can not.
- SQLJ provides simplified rules for calling SQL stored procedures and functions. The JDBC specification requires a generic call to a stored procedure (or function), `fun`, to have the following syntax:

  ```
  prepStmt.prepareCall("{call fun(...)}"); //for stored procedures

  prepStmt.prepareCall("{? = call fun(...)}"); //for stored functions
  ```

  whereas SQLJ provides simplified notations:

  ```
  #sql {call fun(...) };  //Stored procedure

  // Declare x
  ...
  #sql x = {VALUES(fun(...)) };  // Stored function
  // where VALUES is the SQL construct
  ```

Consistency with other Embedded SQL Languages

Programming languages containing Embedded SQL are called host languages. Java differs from the traditional host languages C, COBOL, and FORTRAN, in ways that significantly affect its embedding of SQL:

- Java has automatic storage management (also known as "garbage collection") that simplifies the management of storage for data retrieved from databases.
- All Java types representing composite data, and data of varying sizes, have a distinguished value, `null`, which can be used to represent the SQL NULL state, giving Java programs an alternative to NULL indicators that are a fixture of other host languages.
- Java is designed to support programs that are automatically heterogeneously portable (also called "super portable" or simply "downloadable"). That, along with Java's type system of classes and interfaces, enables component software. In particular, an SQLJ translator, written in Java, can call components that are specialized by database vendors, in order to leverage the existing authorization, schema checking, type checking, transactional, and recovery capabilities that are traditional of databases, and to generate code optimized for particular databases.
- Java is designed for binary portability in heterogeneous networks, which promises to enable binary portability for database applications that use static SQL.

Basic SQLJ Concepts

The following kinds of SQL constructs may appear in SQLJ programs:

- Queries: SELECT statements and expressions.
- SQL Data Change Statements (DML): INSERT, UPDATE, DELETE.
- Data Statements: FETCH, SELECT..INTO.
- Transaction Control: COMMIT, ROLLBACK, etc.
- Data Definition Language (DDL, also known as Schema Manipulation Language): CREATE, DROP, ALTER.
- Calls to stored procedures: e.g., CALL MYPROC(:x, :y, :z)
- Invocations of stored functions: e.g., VALUES(MYFUN(:x))

Host Variables

Arguments to embedded SQL statements are passed through *host variables*, which are variables of the host language that appear in the SQL statement. Host variables are prefixed by a colon, `:`. A host variable contains an optional parameter mode identifier: `IN`, `OUT`, or `INOUT`, followed by a Java host variable that is a Java identifier for a parameter, variable, or field. The evaluation of a Java identifier does not have side effects in a Java program, so it may appear multiple times in the Java code generated to replace an SQLJ clause.

The following query contains host variable, `:x`, (which is the Java variable, field, or parameter `x` visible in the scope containing the query):

```
SELECT  COL1,  COL2  FROM  TABLE1  WHERE  :x  > COL3
```

SQLJ Clauses

Static SQL statements in SQLJ appear in *SQLJ clauses*. SQLJ clauses represent the mechanism by which SQL statements in Java programs are communicated to the database.

Each SQLJ clause begins with the token `#sql`, which is not a legal Java identifier, and is terminated by a semicolon, and as such makes the clause and its SQL contents recognizable to an SQLJ translator.

The simplest SQLJ clauses are *executable clauses* and consist of the token `#sql` followed by an SQL statement enclosed in braces. For example, the following SQLJ clause may appear wherever a Java statement may legally appear and its purpose is to delete all of the rows in the table named `TAB`:

```
#sql { DELETE FROM TAB };
```

In an SQLJ executable clause, the tokens that appear inside the braces are SQL tokens, except for the host variables. All host variables are distinguished by the colon character so the translator can identify them. SQL tokens never occur outside the braces of an SQLJ executable clause. For example, the following Java method inserts its arguments into an SQL table. The method body consists of an SQLJ executable clause containing the host variables `x`, `y`, and `z`:

```
void m (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

In general, SQL tokens are case insensitive (except for identifiers delimited by double quotes), and can be written in upper, lower, or mixed case. Java tokens, however, are case sensitive. For clarity in examples, case insensitive SQL tokens are written uppercase, and Java tokens are lowercase or mixed case. Throughout this document, the lowercase `null` is used to represent the Java "null" value, and the uppercase `NULL` to represent the SQL null value.

The following example SQLJ application uses static SQL to retrieve and update data from the EMPLOYEE table of the DB2 `sample` database. The program declares two cursors to retrieve data. After connecting to the database, `cursor1` selects data from the EMPLOYEE table, which is printed out. After this, the table is updated, and then `cursor2` extracts data from the updated table, which is also printed out. Finally, the changes are rolled back before the program ends.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ;
#sql iterator App_Cursor2 (String) ;
```

```
class App
{

   /**********************
    **   Register Driver   **
   **********************/

   static
   {
      try
      {
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }

   /********************
    **       Main          **
   ********************/

   public static void main(String argv[])
   {
      try
      {
         App_Cursor1 cursor1;
         App_Cursor2 cursor2;

         String str1 = null;
         String str2 = null;
         long   count1;

         // URL is jdbc:db2:dbname
         String url = "jdbc:db2:sample";

         DefaultContext ctx = DefaultContext.getDefaultContext();
         if (ctx == null)
         {
           try
           {
             // connect with default id/password
             Connection con = DriverManager.getConnection(url);
             con.setAutoCommit(false);
             ctx = new DefaultContext(con);
           }
           catch (SQLException e)
           {
             System.out.println("Error: could not get a default context");
             System.err.println(e) ;
             System.exit(1);
```

```
  }
  DefaultContext.setDefaultContext(ctx);
}

// retrieve data from the database
System.out.println("Retrieve some data from the database...");
#sql cursor1 = { SELECT empno, firstnme from employee };

// display the result set
// cursor1.next() returns false when there are no more rows
System.out.println("Received results:");
while (cursor1.next())
{
   str1 = cursor1.empno();
   str2 = cursor1.firstnme();

   System.out.print (" empno= " + str1);
   System.out.print (" firstname= " + str2);
   System.out.print ("\n");
}
cursor1.close();

// retrieve number of employee from the database
System.out.println("\nRetrieve the number of rows in employee table...");
#sql { SELECT count(*) into :count1 from employee };
if (1 == count1)
   System.out.println ("There is " + count1 + " row in employee table.");
else
   System.out.println ("There are " + count1 + " rows in employee table.");

// update the database
System.out.println("\n\nUpdate the database... ");
#sql { UPDATE employee set firstnme = 'SHILI' where empno = '000010' };

// retrieve the updated data from the database
System.out.println("\nRetrieve the updated data from the database...");
str1 = "000010";
#sql cursor2 = { SELECT firstnme from employee where empno = :str1 };

// display the result set
// cursor2.next() returns false when there are no more rows
System.out.println("Received results:");
while (true)
{
   #sql { FETCH :cursor2 INTO :str2 };
   if (cursor2.endFetch()) break;

   System.out.print (" empno= " + str1);
   System.out.print (" firstname= " + str2);
   System.out.print ("\n");
}
cursor2.close();
```

```
            // rollback the update
            System.out.println("\n\nRollback the update...");
            #sql { ROLLBACK work };
            System.out.println("Rollback done.");
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
    }
}
```

Connection Context

Each SQLJ executable clause requires, either explicitly or implicitly, a *connection context object* that designates the database connection at which the SQL operation specified in that clause will be executed. The connection context object is an optional expression, delimited by brackets, that immediately follows the token `#sql`. For example, in the following SQLJ clause, the connection context is the value of the Java variable `myconn`:

```
#sql [ myconn ] { UPDATE TAB2 SET COL1 = :w
                      WHERE :v < COL2 };
```

The connection context object designates a database at which the SQL statements will be executed, and the session and transaction in which they are executed. A connection context is an object of a *connection context class*, which is defined by means of an SQLJ connection clause. A connection context class has methods for opening a connection to a database, given a URL or other connection string, a user name, and password. At run time, an SQLJ program must call those methods to establish a database connection before any SQLJ clauses are executed. The following illustrates an SQLJ connection clause that defines a connection context class named `Inventory`:

```
#sql context Inventory;
```

Note: The connection context object implicitly specifies a database and associated schemas as per a JDBC Connection instance. If an SQLJ executable clause specifies an SQL Session management statement (e.g., `SET SCHEMA`), then that clause will not affect any other SQLJ executable clause. Session management directives are only in conjunction with a connection context object and these objects are initialized only at translation, installation, and customization times in a vendor specific manner.

When an SQLJ clause contains an expression designating the connection context object on which it will be executed, then that clause is said to use an *explicit connection*. When the connection context object is omitted from a clause, then that clause is said to use the *default connection*. Portable applications should always use explicit connection contexts.

As an example, if an invocation of an SQLJ translator indicates that the default connection context class is class `Green`, then all SQLJ clauses that use the default con-

nection will be translated as if they used the explicit connection context object `Green.getDefaultContext()`.

Programs may install a connection context object as the default connection by calling `setDefaultContext`. For example:

```
Green.setDefaultContext(new Green(argv[0], autoCommit));
```

Note: argv[0] is assumed to contain a url naming a database, user ID, and password. `autoCommit` is a boolean flag that is true if auto commit mode is on, and false otherwise.

The default connection context object for a program is stored in a static variable of the default connection context class. To avoid using static variables with some SQLJ programs, such as applets, reentrant libraries, and some multi-threaded programs, you can use SQLJ clauses with explicit connection contexts objects.

When an SQLJ program is executing inside a database as a stored procedure, or is otherwise executing in an environment that automatically provides a connection context, calls to method `ConnectionContext.getDefaultContext` always returns an object representing the schema in which the program is executing. An SQLJ program can detect whether it is executing in an environment that implicitly supplies a connection context by calling `ConnectionContext.getDefaultContext` before it calls `ConnectionContext.setDefaultContext` to install a connection context object. An execution environment that automatically supplies a connection context will return a non-null connection context object.

Schema checking using exemplar schemas. At translation time, a connection context class plays a different role. It symbolizes the "type" of database schema to which the SQLJ program will connect at run time. The notion of the "type of a database schema" is informal. It includes the names, and privileges associated with tables and views, the "shapes" of their rows, stored programs, and so forth. The type of a schema is symbolized by an *exemplar schema*, which is simply a database schema that contains the tables, views, programs, and privileges that would be required in order for the SQL operations in SQLJ clauses to execute successfully. An exemplar schema may be the actual runtime schema, or may be another schema that is a "typical" schema, in ways relevant to the SQLJ program being translated.

The invoker of an SQLJ translator must provide a mapping of connection context classes to exemplar schemas. An SQLJ translator connects to the exemplar schema in order to provide syntax checking, type checking and schema checking for all SQLJ clauses that will be executed in the connection context of the class "exemplified" by that schema. In that way, the exemplar schema represents the database schema to which the application will connect at runtime. It is the responsibility of the application developer to pick an exemplar schema that represents the run time schemas in relevant ways, e.g., having tables, views, stored functions, and stored procedures with the same names and types, and having privileges set appropriately.

If no appropriate exemplar schema is available, or if it is inconvenient to connect to a database during SQLJ program development, then the programmer may omit the

exemplar schema for a connection type. SQLJ clauses to be executed on connections of that type will not then be schema checked at translation time, and will instead be checked later at installation or customization time.

The mapping of connection context classes to exemplar schemas is provided to an SQLJ translator in an implementation-dependent way, typically by pairing connection context class names with connect strings and passwords. For example, a client side SQLJ translator may require such mapping on the command line in an invocation of the translator. Those connect strings and passwords are then used as arguments of the connection context class constructors that establish a database connection to the exemplar schema.

Since the connection context is optional in an SQLJ clause, when the connection context is absent from an SQLJ clause there is a default connection context class specified. The clause is then checked against the exemplar schema corresponding to the class of the default connection context object for the program.

Here is a sample program demonstrating connection contexts:

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import COM.ibm.db2.jdbc.app.*;
import COM.ibm.db2.app.*;

public class connect
{

  /*********************
  **   Register Driver   **
  *********************/

  static
  {
    try
    {
      Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();
    }
    catch (Exception e)
    {
      System.out.println ("\n  Error loading DB2 Driver...\n");
      e.printStackTrace ();
    }
  }

  /*******************
  **       Main          **
  *******************/

  public static void main (String args[]) throws SQLException
  {
```

```
// URLs used in Connect to DB

String url = "jdbc:db2:sample";
String userid = "userid";          // update with your user ID
String password = "password";      // update with your password

// Get default connection context if it exists

DefaultContext ctx = DefaultContext.getDefaultContext();

// If no default connection context, connect to database

if (ctx == null)
{
  try
  {
    Connection con = DriverManager.getConnection(url,userid,password);
    // AutoCommit can be set to "true" or "false"
    con.setAutoCommit(true);
    ctx = new DefaultContext(con);
  }
  catch (SQLException e)
  {
    System.out.println("*** Error connecting to database.\n");
    System.err.println(e) ;
    System.exit(1);
  }

  // Set default context which will be used whenever connection context
  // is not specified

  DefaultContext.setDefaultContext(ctx);
  System.out.println("*** Connected as ADM ID successfully.\n");
}

int SqlCode=0;   // Variable to hold SQLCODE
int SqlState=0;  // Variable to hold SQLSTATE

try
{
  #sql {..... your SQL statement here ......};
}

/* Here's how you can check for SQLCODEs and SQLSTATE */

catch (SQLException e)
{
  SqlCode = e.getErrorCode() // Get SQLCODE
  SqlState = e.getSQLState() // Get SQLSTATE

  if (SqlCode = -1234)
  {
```

```
        // Your code here to handle -1234 SQLCODE
      }
      else
      {
        // Your code here to handle other errors
      }
      System.err.println(e) ; // Print the exception
      System.exit(1);         // Exit
    }

    // More SQL statements ....

     /*************************************************************
     * You can drop the connection context when it is no longer
     * needed.  This is similar to disconnecting from the database
     * in a C program.  If you do not drop the connection context,
     * it will be dropped automatically when the program ends
     *************************************************************/

    if (ctx != null) ctx.close();

  }  // End main

}  // End connect
```

Result Set Iterators

A capability central to SQL is the ability to execute queries that retrieve a "result set" of rows from the database. An SQLJ clause may evaluate a query and return a *result set iterator* object containing the result set selected by that query. Depending on the type of the iterator object, it may be used with the `FETCH..INTO` idiom of Embedded SQL to extract data into host variables, or it may retrieve, through accessor methods, column data consisting of the names and types of columns returned by the query.

An SQLJ *result set iterator* is a Java object from which the data returned by an SQL query can be retrieved. In that role, it corresponds to the *cursor* of Embedded SQL, from which data are fetched. Unlike the cursor, however, an iterator is a first class object. An iterator can be passed as a parameter to a method, and can be used outside the SQLJ translation unit that creates it, without losing its static type for the purposes of type-checking of component interfaces.

An iterator has one or more columns with associated Java types. Names that are Java identifiers can optionally be provided for the iterator columns. The columns of an iterator (which have Java types) are conceptually distinct from the columns of a query (which have SQL types). SQLJ supports two mechanisms for matching iterator columns to query columns. They are *bind by position* and *bind by name*.

Bind by position means that the left to right order of declaration of the iterator columns places them in correspondence with the expressions selected in an SQL query. Traditional `FETCH..INTO` syntax is used to retrieve data from the iterator object into Java variables. *Bind by name* means that the name of each iterator column is matched to the

name of a column returned by the SQL query, independent of the order in which that column appeared in the query. Named accessor methods are generated by the SQLJ translator for each column of the iterator. The name of an accessor method matches the name of a column returned by a query and its return type is the Java type of the iterator column. The *FETCH..INTO* syntax may not be used with an iterator of this type, as the accessor methods provide the mechanism for transferring the data.

An iterator declaration clause designates whether objects of that iterator type use bind by position or bind by name. The two styles of accessing result set data are mutually exclusive: an iterator class supports either bind by position or bind by name, but not both. Program development tools may prefer to generate SQLJ programs using bind by position, since these tools can generate SQLJ code that is "correct by construction". People writing SQLJ programs "by hand" may prefer to use bind by name, to make their applications resilient against changes to the program or database schema.

Positional binding to columns. The following is an example of an iterator class declaration that binds by position. It declares an iterator class called `ByPos`, with two columns of types `String` and `int`:

```
#sql public iterator ByPos (String, int);
```

Assume a table `PEOPLE` with columns `FULLNAME` and `BIRTHYEAR`:

```
TABLE PEOPLE ( FULLNAME VARCHAR(50),
               BIRTHYEAR NUMERIC(4,0) )
```

An *iterator object* of type `ByPos` is used in conjunction with a `FETCH..INTO` statement to retrieve data from table `PEOPLE` as illustrated in the following example:

```
{
    ByPos positer; // declare iterator object
    String name = null;
    int year = 0;
    // populate it
    #sql positer = { SELECT FULLNAME,
                      BIRTHYEAR FROM PEOPLE };
    #sql { FETCH :positer INTO :name, :year };
    while ( !positer.endFetch() )
    {
               System.out.println(name + " was born in " +
                           year);
       #sql { FETCH :positer INTO :name, :year };
    }
}
```

The predicate method `endFetch()` of the iterator object returns true when no more rows are available from the iterator (specifically, it becomes true following the first FETCH that returns no data).

The first SQLJ clause in the block above effectively executes its query and constructs an iterator object containing the result set returned by the query, and assigns it to variable `positer`. The type of the iterator object is derived from the assignment target, which is of type `ByPos`.

The second SQLJ clause in that block contains a *FETCH..INTO* statement. The SQLJ translator checks that the types of host variables in the *INTO* clause match the types of the iterator columns that correspond by position. The types of the SQL columns in the query are convertible to the types of the positionally corresponding iterator columns, according to the SQL to Java type mapping of SQLJ. Those conversions are statically checked at translation time if a database connection to an exemplar schema is provided to the translator.

Named binding to columns. The following is an example of an iterator class declaration that binds by name. It declares an iterator class called `ByName`, with columns named `FULLNAME` and `BIRTHYEAR`:

```
#sql public iterator ByName
              (String fullNAME, int birthYEAR);
```

That iterator class can then be used as follows:

```
{
   ByName namiter; // define iterator object

   #sql namiter = { SELECT FULLNAME, BIRTHYEAR
                               FROM PEOPLE };
   String s;  int i;

   // advances to next row
   while ( namiter.next() )
   {
      i = namiter.birthYEAR();
                // returns column named BIRTHYEAR
      s = namiter.fullNAME();
                // returns column named FULLNAME
      System.out.println(s + " was born in " + i);

   }
}
```

In this example, the first SQLJ clause constructs an iterator object of type `ByName`, as that is the type of the assignment target in that clause. That iterator has generated accessor methods `birthYEAR()` and `fullNAME()` that return the data from the result set columns with those names. The names of the generated accessor methods *are an exact case sensitive match* with their definitions on the iterator declaration clause. Matching a specific accessor method to a specific column name in the `SELECT` list expressions is performed using a case insensitive match. Two column names that differ only in case sensitivity must use the SQL `AS` clause to alias one of the column names in order to avoid ambiguity.

Method `next()` advances the iterator object to successive rows of the result set. It returns `true` when a next row is available, and `false` after it fails to retrieve a next row because the iterator contains no more rows.

A Java compiler will detect type mismatch errors in the uses of column accessor methods. Additionally, if a connection to an exemplar schema is provided at translation

time, then the SQLJ translator will statically check the validity of the types and names of the iterator columns against the SQL queries associated with it.

Providing names for columns of queries. When the expressions selected by a query are unnamed, or have SQL names that are not legal Java identifiers, then SQL column aliases may be used to name them. Consider a table named `"Trouble!"` with a column called `"Not a legal Java identifier"`:

```
CREATE TABLE "Trouble!"
(
   "Not a legal Java identifier"  VARCHAR(10),
   col2 FLOAT
)
```

The following line generates an iterator class called `xY`. The iterator declaration clause may appear wherever a Java class definition may appear:

```
#sql iterator xY (String x, double Y);
```

The SQLJ clause in the following block uses column aliases to associate that column's name with an expression in the query:

```
{
    xY it;
    #sql it = { SELECT "Not a legal Java identifier" AS "x",
                       COL2 * COL2 AS Y
                       FROM "Trouble!" };
    while (it.next()) { System.out.println(it.x() + it.Y()); }
}
```

The first line declares a local variable of that iterator class. The second line initializes that variable to contain a result set obtained from the specified query. The `while()` loop calls the column accessor methods of the iterator to obtain and print data from its rows.

Calls to Stored Procedures and Functions

Databases may contain *stored procedures* and *stored functions*. User-defined procedures and functions are named schema objects that execute in the database. An SQLJ executable clause appearing as a Java statement may call a stored procedure by means of the `CALL` statement. For example:

```
#sql { CALL SOME_PROC(:INOUT myarg) };
```

Stored procedures may have `IN`, `OUT`, or `IN OUT` parameters. In the above case, the value of host variable `myarg` is changed by the execution of that clause. An SQLJ executable clause may call a stored function by means of the SQL `VALUES` construct. For example, assume a stored function `F` that returns an integer. The following example illustrates a call to that function that then assigns its result to Java local variable `x`:

```
{
    int x;
    #sql x = { VALUES( F(34) ) };
}
```

Note: DB2 does not support Java stored procedures and Java user-defined functions (UDFs) accessing DB2 databases on HP-UX and SCO UnixWare servers.

Advanced Features

The sections that follow discuss more advanced programming techniques including multiple connections, dynamic SQL and multi-threading.

Using Multiple SQLJ Contexts and Connections

SQLJ supports connecting to multiple schemas at the same time. The various schemas used at runtime are modeled as distinct connection context classes in SQLJ programs, which allows type checking using the same schemas at translation time. The following program demonstrates the use of multiple contexts by connecting to two DB2 databases, `sample` and `sample2`.

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import COM.ibm.db2.jdbc.app.*;
import COM.ibm.db2.app.*;

public class multicon
{

  /*********************
  **   Register Driver   **
  *********************/

  static
  {
    try
    {
      Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();
    }
    catch (Exception e)
    {
      System.out.println ("\n  Error loading DB2 Driver...\n");
      e.printStackTrace ();
    }
  }

  /*******************
  **       Main          **
  *******************/

  public static void main (String args[]) throws SQLException
  {

    /*********************************
    **  URLs used to connect to DBs **
    *********************************/

    String url1 = "jdbc:db2:sample";   // database SAMPLE
    String url2 = "jdbc:db2:sample2";  // database SAMPLE2
    String userid = "userid";          // change the userid to yours
```

```
String password = "password";      // change the password to yours


System.out.println("*** Begin multicon ***\n");

DefaultContext ctx1 = null;   // connection context 1
DefaultContext ctx2 = null;   // connection context 2


/*********************************
**  Connect to SAMPLE database  **
*********************************/

try
{
  Connection con = DriverManager.getConnection(url1,userid,password);
  con.setAutoCommit(true);
  ctx1 = new DefaultContext(con);
}
catch (SQLException e)
{
  System.out.println("*** Error: could not connect to SAMPLE db.\n");
  System.err.println(e) ;
  // System.exit(1);
}

/* If you want to set a default connection, uncomment the next line of code
   out.  If you have a default connection, it will be used whenever a connection
   context is not specified */

// DefaultContext.setDefaultContext(ctx1);

System.out.println("*** Got a connection to SAMPLE db successfully.\n");

/*********************************
**  Connect to SAMPLE2 database  **
*********************************/

try
{
  Connection con = DriverManager.getConnection(url2,userid,password);
  con.setAutoCommit(true);
  ctx2 = new DefaultContext(con);
}
catch (SQLException e)
{
  System.out.println("*** Error: could not connect to SAMPLE2 db.\n");
  System.err.println(e) ;
  // System.exit(1);
}

// You could also set ctx2 as default connect INSTEAD of ctx1 if you
```

```
    // want to use default connection context
    // DefaultContext.setDefaultContext(ctx2);

    System.out.println("*** Got a connection to SAMPLE2 db successfully.\n");

    /*****************************************************************************
    **  Counting number of tables/views in SAMPLE database using connection ctx1 **
    *****************************************************************************/

    short count=-1;

    try
    {
      #sql [ctx1] {SELECT COUNT(*) INTO :count FROM SYSCAT.TABLES };
    }
    catch (SQLException e)
    {
      System.out.println("*** Error Selecting from SAMPLE's SYSCAT.TABLES.\n");
      System.err.println(e) ;
      // System.exit(1);
    }

    System.out.println("*** Database SAMPLE has " + count + " tables and views.\n");

    /*****************************************************************************
    **  Counting number of tables/views in SAMPLE database using connection ctx2 **
    *****************************************************************************/

    count=-1;  // reset count

    try
    {
      #sql [ctx2] {SELECT COUNT(*) INTO :count FROM SYSCAT.TABLES };
    }
    catch (SQLException e)
    {
      System.out.println("*** Error Selecting from SAMPLE2's SYSCAT.TABLES.\n");
      System.err.println(e) ;
      // System.exit(1);
    }

    System.out.println("*** Database SAMPLE2 has " + count + " tables and views.\n");

    if (ctx1 != null) ctx1.close();   // "disconnect" from SAMPLE
    if (ctx2 != null) ctx2.close();   // "disconnect" from SAMPLE2

  }  // End main

}  // End multicon
```

SQL Execution Control and Status

The execution semantics of SQL operations can be queried and modified via the execution context associated with the operation. An execution context exists as an instance of class `sqlj.runtime.ExecutionContext`.

The following `ExecutionContext` attributes control the execution environment of SQL operations. The `getXXX` and `setXXX` methods read and change the `XXX` value. Once set, they affect all SQL operations subsequently executed on that execution context.

- `MaxRows` specifies the maximum number of rows to be returned by any query.
- `MaxFieldSize` specifies the maximum number of bytes to be returned as data for any column or output variable.
- `QueryTimeout` specifies the number of seconds to wait for an SQL operation to complete.

The following `ExecutionContext` attributes describe the results of the last SQL operation executed:

- `UpdateCount` specifies the number of rows updated, inserted or deleted during the last operation.
- `SQLWarnings` describes any warnings that occurred during the last operation.

An execution context is associated either explicitly or implicitly with each executable SQL operation appearing in an SQLJ program. An execution context may be supplied explicitly as an argument to each SQL operation:

```
ExecutionContext execCtx = new ExecutionContext();
#sql [execCtx] { DELETE FROM emp WHERE sal > 10000 };
```

When explicit execution contexts are used, each SQL operation may be executed using a different execution context instance. If an explicit connection context is also being used, both may be passed as arguments to the SQL operation:

```
#sql [connCtx, execCtx] { DELETE FROM emp WHERE sal > 10000 };
```

If an execution context is not supplied explicitly as an argument to an SQL operation, a default execution context is used implicitly. The default execution context for a particular SQL operation is obtained via the getExecutionContext() method of the connection context used in the operation. For example:

```
#sql [connCtx] { DELETE FROM emp WHERE sal > 10000 };
```

uses the execution context associated with the connection context given by connCtx. When neither a connection context nor an execution context is explicitly supplied, the execution context associated with the default connection context is used.

The following code demonstrates the use of some `ExecutionContext` methods:

```
{
  ExecutionContext execCtx = new ExecutionContext();

  // Wait only 3 seconds for operations to complete
  execCtx.setQueryTimeout(3);

  try {
     // delete using explicit execution context
     // if operation takes longer than 3 seconds,
     // SQLException is raised
     #sql [execCtx] { DELETE FROM emp WHERE sal > 10000 };

     System.out.println
       ("removed " + execCtx.getUpdateCount() + " employees");
  }
  catch(SQLException e) {
      // Assume a timeout occurred
      System.out.println("SQLException has occurred with" +
              " exception " + e );
  }
}
```

Multi-Threading Considerations

SQLJ can be used to write multi-threaded applications. The SQLJ runtime supports multiple threads sharing the same connection context. However, SQLJ programs are subject to synchronization limitations imposed by the underlying DBMS implementation. If a DBMS implementation mandates explicit synchronization of statements executed in a specific connection, then an SQLJ program using that implementation would require a similar synchronization of SQL operations.

Whereas connection contexts may be safely shared between threads, execution contexts should not be shared. If an execution context is shared, the results of an SQL operation performed by one thread will be visible in the other thread. If both threads are executing SQL operations, a race condition may occur in which the results of an execution in one thread are overwritten by the results of an execution in the next thread before the first thread has processed the original results. Furthermore, if a thread attempts to execute an SQL operation using an execution context that is currently being used to execute an operation in another thread, a runtime exception is raised. To avoid such problems, each thread should use a distinct execution context whenever an SQL operation is executed on a shared connection context.

Dynamic SQL and JDBC SQLJ Interoperability

The SQLJ language provides direct support for static SQL operations that are known at the time the program is written. If some or all of a particular SQL statement cannot be determined until runtime, it is a dynamic operation. To perform dynamic SQL operations from an SQLJ program, use JDBC. A ConnectionContext object contains a JDBC Connection object which can be used to create JDBC Statement objects needed for dynamic SQL operations.

Every SQLJ ConnectionContext class includes a constructor that takes as an argument a JDBC Connection. This constructor is used to create an SQLJ connection context instance that shares its underlying database connection with that of the JDBC connection.

Every SQLJ ConnectionContext instance has a `getConnection` method that returns a JDBC Connection instance. The JDBC Connection returned shares the underlying database connection with the SQLJ connection context. It may be used to peform dynamic SQL operations as described in the JDBC API.

Session Sharing. The interoperablity methods described above provide a conversion between the connection abstractions used in SQLJ and those used in JDBC. Both abstractions share the same database session (i.e., the underlying database connection). Accordingly, calls to methods that affect session state on one object will also be reflected in the other object, as it is actually the underlying shared session that is being affected.

JDBC defines the default values for session state of newly created connections. In most cases, SQLJ adopts these default values. However, whereas a newly created JDBC connection has auto commit mode on by default, an SQLJ connection context requires the auto commit mode to be specified explicitly upon construction.

Connection Resource Management. Calling the `close` method of a connection context instance causes the associated JDBC connection instance and the underlying database connection to be closed. Since connection contexts may share the underlying database connection with other connection contexts and/or JDBC connections, it may not be desirable to close the underlying database connection when a connection context is closed. A programmer may wish to release the resources maintained by the connection context (for example, statement handles) without actually closing the underlying database connection. To this end, connection context classes also support a `close` method which takes a boolean argument indicating whether or not to close the underlying database connection: the constant `CLOSE_CONNECTION` if the database connection should be closed, and `KEEP_CONNECTION` if it should be retained. The variant of `close` that takes no arguments is a shorthand for calling `close(CLOSE_CONNECTION)`.

If a connection context instance is not explicitly closed before it is garbage collected, then `close(KEEP_CONNECTION)` is called by the finalize method of the connection context. This allows connection related resources to to be reclaimed by the normal garbage collection process while maintaining the underlying database connection for other JDBC and SQLJ objects that may be using it. Note that if no other JDBC or SQLJ objects are using the connection, then the database connection will also be closed and reclaimed by the garbage collection process.

Both SQLJ connection context objects and JDBC connection objects respond to the `close` method. When writing an SQLJ program, it is sufficient to call the `close` method on only the connection context object. This is because closing the connection context will also close the JDBC connection associated with it. However, it is not sufficient to close only the JDBC connection returned by the `getConnection` method of a connection context. This is because the `close` method of a JDBC connection will not cause the

containing connection context to be closed, and therefore resources maintained by the connection context will not be released until it is garbage collected.

The `isClosed` method of a connection context returns `true` if any variant of the `close` method has been called on the connection context instance. If `isClosed` is `true`, then calling `close` has no effect, and calling any other method is undefined.

Comparison with ANSI/ISO Embedded

ANSI/ISO specifies a "standard embedded language", for FORTRAN, PL/1, COBOL, ADA, MUMPS, and C. An *embedded program* is a mixture of embedded SQL statements and host language statements.

An embedded SQL statement always has an SQL prefix, usually `EXEC SQL`, and a terminator appropriate to its host language, for example, a semi-colon for C, or a new line character for Fortran.

Elements of Embedded SQL fall into four groups, which are treated differently in SQLJ than they are in other embedded languages:

- *Executable SQL statements*: SQLJ directly adopts most of the SQL schema, SQL data and SQL transaction statements, which manipulate SQL data, definitions, and transactions, substantially as they are specified in standard Embedded SQL.
- *Dynamic SQL*: SQLJ does not specify dynamic SQL for Java. Dynamic SQL is handled separately by JDBC.
- *Declarations*: The *declare cursor* and *host variable definition* declarations of Embedded SQL define names for individual data items, which are cursors or host variables, and may be annotated by SQL attributes such as character sets (SQL92). SQLJ replaces those by declarations of Java types for declaring iterator classes and other data items with SQL attributes.
- *Program control*: The *embedded exception declaration*, *SQL session statements*, *SQL connection statements*, and *SQL diagnostics statements* serve to knit together the SQL and host language environments by managing exceptions, database connections, and diagnostics. SQLJ omits all of those statements, because object-oriented languages can directly express the types of exceptions, database connections, and diagnostics, and can manipulate those objects using standard programming techniques.

The following elaborates the differences summarized above between the elements of SQLJ and other embedded languages:

SQL prefix

SQLJ clauses are analogous to the embedded statements described above. An SQLJ clause is introduced by the SQL prefix token `#sql`, chosen for Java since it is not a legal Java identifier, and so cannot conflict with other Java syntax.

cursor name

is a simple identifier in Embedded SQL. The equivalent SQLJ construct is *iterator host variable*, which is a Java variable which must be an instance of a generated iterator class, or a subclass of such a class.

SQL Schema, Data, and Transaction Statements

are treated in SQLJ as *SQLJ clauses* and are substantially as specified by the existing rules for embedded language.

SQL dynamic statements

including `PREPARE`, `DESCRIBE`, and `EXECUTE`, as well as the *dynamic declare cursor* statement, are not used by SQLJ since dynamic operations are subsumed by JDBC.

SQL connection statement

is replaced in SQLJ by direct Java construction and manipulation of connection objects. That enables the capability for SQLJ programs to open multiple connections simultaneously to the same or different databases.

Explicit manipulation of connection objects is supported for Java programs that need to avoid hidden global state (e.g., Java "static variables") that would be used to implement the SQL connection statement. In particular, Java applets and other multi-threaded programs are usually coded to avoid contention of global state. Such programs will store connection objects in local variables and mention them explicitly in SQLJ clauses.

SQLJ allows the possibility that a Java program can manipulate multiple connection objects, connected to different databases. When a program manipulates multiple connections, they are mentioned explicitly in the SQLJ clauses, so they are regular Java objects.

Host variable definition

is specified in Embedded SQL to be preceded by an `EXEC SQL BEGIN DECLARE SECTION` and terminated by an `EXEC SQL END DECLARE SECTION`, so that pre-compilers (translators) can detect the host variable definitions and determine their types by a rudimentary parse of the host program.

SQLJ does not define a host variable definition section. SQLJ translators can take advantage of the portability and component software available for Java in order to have greater parsing ability than traditional pre-compilers, so that the `DECLARE SECTION` is not required for the purpose of confining definitions of host variables to a small portion of the host program. Instead, any Java variable, parameter, or field (of an object) may be used as a host variable.

SQL92 character sets

Java supports a UNICODE character set (ISO 10646) for String data and for identifiers. This allows Java to represent most character data in a uniform way. The SQLJ specification does not address the issue of character sets, since the SQLJ specification is limited to the SQL92 Entry Level specification that does not require them. As per SQL92, no characters may appear in SQLJ clauses that are not defined as an SQL language character with the exception of Java identifiers and Java host variables.

Embedded exception declaration

is not defined by SQLJ. In the ANSI/ISO standard it has these forms:

```
EXEC SQL WHENEVER exception_condition
                    GOTO program_label;
EXEC SQL WHENEVER exception_condition
                    CONTINUE;
```

The Java language does not support the `goto` statement, therefore the direct transliteration of the above construct into Java is not possible. Instead, Java provides a `try..catch` statement that associates a handler for certain exceptions in the Java block in which those exceptions might be raised. For example, assume an exception called `e`:

```
try { block_that_may_throw_exception_e }
  catch (Exception e) { block_that_handles_exception_e }
```

In addition, Java has well developed rules for declaring and handling exceptions, thus the `EXEC SQL WHENEVER` statement does not add value. Other object-oriented languages have facilities for declaring and handling exceptions, similar to those in Java.

JDBC has defined an exception, globally named `java.sql.SQLException`, as the superclass of exceptions that are returned from SQL. SQLJ follows that precedent in order to facilitate interoperability between static SQL and dynamic SQL.

SQL diagnostics statement

SQLJ follows the Java methodology for handling return information traditionally found in the descriptor areas of Embedded SQL. Abnormal termination and certain runtime errors (e.g.., `NULL` retrieval to non-nullable datatypes) are processed using exception handling. Other status information, for example `update count`, are processed by using methods on the connection context and execution context objects.

Declare cursor

declares a single name for both a query and its associated result set in the host program. SQLJ instead distinguishes between a query and the result set that it returns. When an SQLJ clause containing a query is evaluated, it returns an iterator object containing the result set of rows selected by that query. The type of the iterator is a Java class that encodes the number and types (and names) of columns in the result set, allowing type checking of operations on the iterator. The WITH HOLD cursor attribute has the same effect in SQLJ.

Input parameters to SQL statements

SQLJ allows host variables for input parameters to SQL statements, as does standard Embedded SQL.

Extracting column values from result sets

SQLJ supports two approaches to accessing column values from result sets: by position and by name. The familiar `FETCH` of Embedded SQL accesses

columns by position. In the following example, the first column in the row is assigned to var1, the second to var2, and the third to var3:

```
EXEC SQL FETCH cursor1 INTO :var1,:var2,:var3;
```

SQLJ supports a modified version of the FETCH statement. It also supports access to columns by name, through generated methods with the names and types of the columns.

OPEN cursor

Embedded SQL has an OPEN operation to open and re-open its named cursors that represent both a query and its result set:

```
EXEC SQL OPEN cursor1;
```

SQLJ does not have an OPEN operation to open or re-open iterator objects. SQLJ does not name a static query, nor treat it as data. Instead, a query returns an iterator object that is manipulated as data. Of course, a programmer may, in effect, name a query by writing it in an SQLJ clause in the body of a method. Methods are called by their names, and can return result set objects as their values.

SQLJ Translator Reference

Use the script sqlj to run the translator as follows:

```
sqlj [ options ] filelist
```

Alternatively, you can run the translator by using its class name:

```
java sqlj.translator.Main [ options ] filelist
```

The filelist is a list of file names separated by spaces:

```
file1.sqlj [file2.sqlj] ... [foo1.java] [foo2.java]...
```

The files with the .java extension are included to resolve type references, but no output files are created for them. The files with the .sqlj extension include SQLJ clauses. The translator creates .java files, as well as .ser (serialized object) files for them.

The following applies to all options:

- The names of command line options are case sensitive. They are all completely lowercase. The option values are usually case sensitive as well.
- If the same option appears more than once on the command line, or in a property file, SQLJ uses the final option value and ignores the others. For example, if the command line options are:

  ```
  -user=scott  -user=myaccount
  ```

  SQLJ uses the second value, myaccount.

Property Files

Property files can be used to supply options to the translator. Options in a property file appear one per line. Options have the same syntactic form as those appearing on the command line, except that the token sqlj. replaces the initial hyphen. SQLJ ignores

properties without the `sqlj.` prefix in the property file. This allows several programs to share one properties file. Empty lines are ignored. Lines that start with `#` are comments. Here is an example of an `sqlj.properties` file:

```
sqlj.user=scott
sqlj.driver=db2.jdbc.driver.DB2Driver
```

SQLJ processes the options in a property file in order from first to last. A later entry overrides an earlier entry.

SQLJ looks for files called `sqlj.properties` to use as property files when it starts to run. It looks for them in three places, in the following order:

- The Java home directory, if it exists.
- The user's home directory, if it exists.
- The current directory.

It processes each such file it finds, overriding previously set options as it encounters new ones. Thus, options set in the `sqlj.properties` file in the current directory override those set in the `sqlj.properties` file in the user's home or Java home directories.

SQLJ starts by setting all options to their default values, if any. It then reads any default property files it finds, using settings it finds in them to override the original defaults. Finally it looks for options on the command line and uses them to override the settings it has so far. It processes options on the command line from left to right, treating property files specified with the `-props` option on the command line as if their contents were specified inline.

General Options

In the following, all options are described as if they were given on the command line. However, all options except for `-props`, `-help` and `-version` may also appear in a properties file.

-help

The help option causes SQLJ to list all translator options in effect at that time.

Note: The output display contains these lines for each option:

name:	Name of the option.
type:	Datatype or a list of datatypes. Can also be a choice of allowed values.
value:	The current setting in effect.
description:	Description of option.
set from:	Where the option was set (default, property file, inline).

-version

Displays the build version.

-dir

The `dir` option specifies the directory for generated files. For example:

```
-dir=java/files
```

If you do not specify `dir`, SQLJ uses the current directory. The behavior of the `dir` option is similar to the behavior of the `-d` option of `javac`. Suppose we have the files `File1.sqlj` and `File2.sqlj`. `File1.sqlj` has no package declaration. `File2.sqlj` is in the `sqlj.demo` package. If `sqlj` is invoked with the option `-dir=/src`, then `File1.java` is created in `/src`, but `File2.java` is created in `/src/sqlj/demo`. If no `dir` option is specified, then the output file directory is the same as that of the input file.

-warn

You can specify a list of flags for turning warnings on or off with this option. Several values for the warn option must be combined into a single, comma-separated string. SQLJ applies the specified flags in the order in which they appear on the command line. Permitted flags are `all`, `none`, `verbose`, `noverbose`, `null`, `nonull`, `precision`, `noprecision`, `portable`, `noportable`, `strict`, `nostrict`. Default value is `verbose`. For instance:

```
-warn=none,null,precision
```

first turns off all warnings, then turns nullability and precision on. The possible values of the warn option are:

- `all` turns all warnings and informational messages on.
- `none` turns all warnings and informational messages off.
- `precision`, `noprecision` specifies whether or not the SQLJ translator checks for possible loss of precision when moving values from database columns to Java host variables. The default is `precision`. Note that precision checks are part of the semantic analysis and require SQLJ to connect to the database.
- `null`, `nonull` specifies whether or not SQLJ checks nullable columns and nullable Java types for conversion loss when moving values from database columns to Java host variables. If you do not specify one of these options, SQLJ checks nullability by default. Note that nullability checks are part of the semantic analysis and require SQLJ to connect to the database.
- `verbose`, `noverbose` turns on or off informational messages about the semantic analysis process. If you do not specify one of these options, SQLJ is `verbose` by default.
- `portable`, `noportable` turns on or off warning messages about the portability of SQLJ clauses. By default, SQLJ warns about non-portable constructs.
- `strict`, `nostrict` specifies whether SQLJ matches named iterators strictly against the columns returned by the database, or not. If this option is on, a warning will be issued for any column in a result set, that is not matched by a column in the cursor to which the result set is assigned. Default is `strict`.

-props

The props option specifies the name of a property file from which to read options. For example:

```
-props=myapplic.properties
```

Connection Options

These options specify the database connection for online checking. All of these options (except for driver) may be tagged with a ConnectionContext type:

```
-option@ConnectionContextType=value
```

This permits the use of separate exemplar schemas for each of the connection contexts. If you omit the connection context type, when specifying one of these options, the value will be used for any SQL statements that use the default connection context. If no option value is given at a specific ConnectionContextType, then the option value for the default connection context is used.

-user

The user option specifies the username for connecting to a database in order to perform semantic analysis of the SQL expressions embedded in a SQLJ program. It contains the username, for example:

```
-user=scott
```

The user command line option may include a connection context type. For example:

```
-user@Ctx1=scott
```

Whenever a username is required for the connection to a database context `Ctx1`, SQLJ uses the user option that was tagged with `Ctx1`. If it can't find one, SQLJ issues a message and looks for an untagged user option to use instead.

Specifying a user value indicates to SQLJ that online checking is to be performed. If you do not specify the user option, SQLJ does not connect to the database for semantic analysis. There is no default value for the user option.

If you have turned on online checking by default (for example, by specifying `-user=scott`), then in order to disable online checking for a particular connection context type `Ctx2`, you have to explicitly give an empty user name:

```
-user@Ctx2=
```

-password

The password option specifies a password for the user. The password will be requested interactively if it is not supplied. This option can be tagged with a connection context type. The two forms are:

```
-password=tiger
-password@Ctx1=tiger
```

-url

This sub-option specifies a JDBC URL for establishing a database connection:

```
-url=jdbc:db2:sample
```

Semantic Checking Options

These options specify the characteristics of offline and online SQL checking.

-offline

Offline checking assumes that there is no connection to the database so that only SQL syntax and usage of Java types are checked. The offline option specifies the Java class that implements the SQL checking component of SQLJ for offline checking. This option permits customized checking for diverse databases by means of the checker class, `sqlj.semantics.OfflineChecker`.

The offline option can be tagged with a connection context:

```
-offline@myconnect=sqlj.semantics.OfflineChecker
```

-online

Online checking assumes that the database connection exists. You must have specified a user ID by means of the *user* option for checking to actually occur. The online checker passes DML statements to the database for syntactic and semantic analysis, in addition to the features of the offline checker. The online checker also checks stored functions and procedures for overloading by means of the checker class, `sqlj.semantics.JdbcChecker`.

The online option specifies the Java class that implements the SQL checking component of SQLJ via database connections to exemplar schemas. In a similar way to JDBC driver registration, the checker is queried as to whether it is able to perform semantic analysis for the given connection:

```
-online=sqlj.semantics.JdbcChecker
```

The JDBC online checker checks the signature of stored function or procedure calls and matches it with the JDBC types. It also determines the `ResultSetMetaData` for `SELECT` statements. `SELECT` statements are executed to determine the `ResultSetMetaData`. In order to reduce the size of the returned result set to 0, the `WHERE` clause is added or modified to read `WHERE 1=2`. This option can also be tagged with a connection context type:

```
-online@myconnect=sqlj.semantics.JdbcChecker
```

-cache

This option can be used to turn on caching of the results of the online checking in order to avoid database connections during subsequent precompilation runs. The analysis results are cached in the file, `SQLChecker.cache`, in the current directory. The cache may be emptied simply by removing this file.

The cache holds a serialized representation of all SQL statements that have been translated without error or warning messages, with the statements' parameters, return type, the translator settings and modes inferred about the parameters.

The cache is cumulative, adding new statements. If you are just fixing bugs in the Java source, you will not have to re-connect to the database. The boolean value for the cache option can be specified as yes, no, true, false, on, off, 1, or 0. Caching of semantic analysis results is turned off by default. Example:

```
-cache=true
```

-default-block-mode

Specifies the default mode of host variables occuring in an SQL block. It is used during offline checking when the mode cannot be determined. During online checking, the actual mode must correspond to the setting implied by this flag, unless the setting `unknown` is used. Possible settings are `unknown`, `inout`, `in`, and `out`. The default value is `in`. Example:

```
-default-block-mode=unknown
```

-default-function-mode

Specifies the default mode of host variables occuring in arguments of a stored function invocation. It is used during offline checking when the mode cannot be determined. During online checking, the actual mode must correspond to the setting implied by this flag, unless the setting `unknown` is used. Possible settings are `unknown`, `inout`, `in`, and `out`. The default value is `in`. Example:

```
-default-function-mode=unknown
```

-default-procedure-mode

Specifies the default mode of host variables occuring in arguments of a stored function invocation. It is used during offline checking when the mode cannot be determined. During online checking, the actual mode must correspond to the setting implied by this flag, unless the setting `unknown` is used. Possible settings are `unknown`, `inout`, `in`, and `out`. The default value is `in`. Example:

```
-default-procedure-mode=out
```

Sample Programs and Extra Examples

The sample program information in the Version 5 book have been revised for Version 5.2. See “Sample Programs” on page 150 for details.

Appendix D. Building Applications for Windows and OS/2 Environments Updates

The *Building Applications for Windows and OS/2 Environments* has not been refreshed for DB2 Universal Database Version 5.2. The following sections document any Version 5.2 changes and enhancements to building Windows and OS/2 applications and should be used in conjunction with the Version 5 edition of the *Building Applications for Windows and OS/2 Environments* by Version 5.2 users. The updates are arranged by chapter.

Changes to the Preface: About This Book

The following needs to be updated in the *Building Applications for Windows and OS/2 Environments* as a result of DB2 Version 5.2 support for additional operating systems. The paragraph on page vii beginning "This book explains" should be changed to the following:

This book explains how to build applications using the DB2 Software Developer's Kits (DB2 SDKs) for the following operating systems:

- Windows NT
- Windows 98
- Windows 95
- Windows 3.1
- OS/2

Note: Whenever this book mentions Windows NT, Windows 98, or Windows 95, all three operating systems: Windows NT, Windows 98, and Windows 95 are implied, except in the case of Systems Network Architecture (SNA) support, REXX suport, or DB2 Connect, formerly known as Distributed Database Connection Services (DDCS). These are supported on Windows NT only.

On page vii, add the following to the paragraph beginning "Different programming interfaces":

Java Database Connectivity (JDBC) Is a dynamic SQL API for Java. The JDBC API is included in the Java Development Kits available for supported platforms.

Changes to Chapter 1. About the DB2 Software Developer's Kit

The following addition applies to page 1, in the paragraph beginning: "The DB2 SDKs for the Windows and OS/2 platforms". Add the following list item after the list item beginning: "DB2 Java Database Connectivity":

- DB2 embedded SQL for Java (SQLJ) support to develop Java embedded SQL applications and applets.

The following changes apply to page 1, in the paragraph beginning: "The DB2 SDKs for the Windows and OS/2 platforms":

Change the list item beginning "On OS/2, Windows NT" to the following:

- On OS/2 and Windows NT, support to develop database applications that use the REXX language.

Change the list item beginning "A documented API" to the following:

- A documented API to enable other application development tools to implement pre-compiler support for DB2 directly within their products. For example, on OS/2 the IBM PL/I compiler uses this interface. Information on the set of precompiler service APIs, and how to use them, is available from the anonymous FTP site, `ftp://ftp.software.ibm.com`. The PostScript file, called `prepapi.psbin`, is located in the directory `/ps/products/db2/info`. This file is in binary format. If you do not have access to this electronic forum and would like to get a copy of this document, you can call IBM Service as described in the Service Information Flyer.

The "Sample Programs" section on page 4 should be replaced with the following:

Sample Programs

The DB2 SDK comes with sample programs. The file extensions for each supported language, and the directories where the programs can be found on the supported paltforms, are given in Table 10 on page 151. In addition, the locations and extensions for other sample programs can be found in Table 11 on page 151.

The sample programs providing examples of embedded SQL (except for Java), and DB2 API calls are shown in Table 12 on page 154. Log Management User Exit programs are shown in Table 13 on page 159. Command Line Processor (CLP) programs provided by DB2 are shown in Table 14 on page 160.

Java JDBC sample programs are shown in Table 15 on page 160. Java SQLJ sample programs are shown in Table 16 on page 161.

Object Linking and Embedding (OLE) sample programs are shown in Table 17 on page 161. The sample programs demonstrating DB2 CLI calls are shown in Table 18 on page 162.

You can use the sample programs to learn how to code your applications.

Note: Not all sample programs have been ported to all the supported programming languages.

Table 10. Sample Program File Extensions and Locations

Language		Embedded SQL Programs	Non-embedded SQL Programs
C	File Ext.	`.sqc`	`.c`
	Directory	`samples/c`	`samples/c` `samples/cli` (CLI programs)
C++	File Ext.	`.sqC` (UNIX) `.sqx` (Windows & OS/2)	`.C` (UNIX) `.cxx` (Windows & OS/2)
	Directory	`samples/cpp`	`samples/cpp`
COBOL	File Ext.	`.sqb`	`.cbl`
	Directory	`samples/cobol` `samples/cobol_mf`	`samples/cobol` `samples/cobol_mf`
Fortran	File Ext.	`.sqf`	`.f` (UNIX) `.for` (OS/2)
	Directory	`samples/fortran`	`samples/fortran`
JAVA	File Ext.	`.sqlj`	`.java`
	Directory	`samples/java`	`samples/java`
REXX	File Ext.	`.cmd`	`.cmd`
	Directory	`samples/rexx`	`samples/rexx`

Table 11. Other Samples, their Extensions and Locations

Sample Group		
CLP	File Ext.	`.db2`
	Directory	`samples/clp`
OLE	File Ext.	`.bas` (Microsoft Visual Basic) `.CPP` (Microsoft Visual C++)
	Directory	`samples\ole\msvb` (Microsoft Visual Basic) `samples\ole\msvc` (Microsoft Visual C++)
User Exit	File Ext.	`.cad` (Windows & OS/2) `.cadsm` (UNIX) `.cdisk` (UNIX) `.ctape` (UNIX)
	Directory	`samples/c`

Note:

Embedded SQL Programs require precompilation, except for REXX embedded SQL programs where the embedded SQL statements are interpreted when the program is run.

Directory Delimiters	On UNIX are /. On OS/2 and Windows platforms, are \. In the tables, the UNIX delimiters are used unless the directory is only available on Windows and/or OS/2.
IBM COBOL samples	Are only supplied on the OS/2, AIX, Windows NT and Windows 95 platforms in the `cobol` subdirectory.
Micro Focus Cobol Samples	Are supplied on all platforms except the Macintosh and Silicon Graphics IRIX. The 16-bit Micro Focus COBOL examples are supplied in the `cobol_16` subdirectory on OS/2, and the `cobol` subdirectory on Windows 3.1. For all other platforms, the Micro Focus COBOL samples are in the `cobol_mf` subdirectory.
Fortran Samples	Are only supplied on the AIX, HP-UX, Silicon Graphics IRIX, Solaris, and OS/2 platforms.
Java Samples	Are Java Database Connectivity (JDBC) applications, applets, stored procedures and UDFs, and embedded SQL (SQLJ) applications, applets, stored procedures and UDFs. Java samples are available on the AIX, HP-UX, SCO UnixWare 7, Silicon Graphics IRIX, Solaris, OS/2, Windows NT, Windows 98, and Windows 95 platforms.
REXX Samples	Are only supplied on the AIX, OS/2, and Windows NT platforms.
CLP Samples	Are Command Line Processor scripts that execute SQL statements.
OLE Samples	Are for Object Linking and Embedding (OLE) in Microsoft Visual Basic and Microsoft Visual C++, supplied on the Windows NT and Windows 95 platforms only.
User Exit samples	Are Log Management User Exit programs used to archive and retrieve database log files. The files must be renamed with a `.c` extension and compiled as C language programs.

You can find the sample programs in the `samples` subdirectory of the directory where DB2 has been installed. There is a subdirectory for each supported language. The following examples show you how to locate the samples written in C or C++ on each supported platform.

- On UNIX platforms.

  You can find the C source code for embedded SQL and DB2 API programs in `sqllib/samples/c` under your database instance directory; the C source code for

DB2 CLI programs is in sqllib/samples/cli. For additional information about the sample programs in Table 12 on page 154 and Table 18 on page 162, refer to the README file in the appropriate samples subdirectory under your database manager instance. The README file will contain any additional samples that are not listed in this book.

- On OS/2, Windows NT, Windows 98 and Windows 95 platforms.

  You can find the C source code for embedded SQL and DB2 API programs in %DB2PATH%\samples\c under the DB2 install directory; the C source code for DB2 CLI programs is in %DB2PATH%\samples\cli. The variable %DB2PATH% determines where DB2 is installed. Depending on which drive DB2 is installed, %DB2PATH% will point to *drive*:\sqllib. For additional information about the sample programs in Table 12 on page 154 and Table 18 on page 162, refer to the README file in the appropriate %DB2PATH%\samples subdirectory. The README file will contain any additional samples that are not listed in this book.

- On Windows 3.1.

  You can find the C source code for embedded SQL and DB2 API programs in %DB2PATH%\samples\c; the C source code for DB2 CLI programs is in %DB2PATH%\samples\cli. The db2.ini file, which stores the DB2 settings, defines the value for %DB2PATH%, which by default points to *drive*:\sqllib\win. The value of %DB2PATH%, as referenced in the db2.ini file, is only recognized within the DB2 environment. For additional information about the sample programs in Table 12 on page 154 and Table 18 on page 162, refer to the README files in these subdirectories. The README files will contain any additional samples that are not listed in this book.

- On Macintosh.

  You can find the sample programs in the DB2:samples: folder. There are subfolders for sample programs written in C and CLI. For additional information about the sample programs in Table 12 on page 154 and Table 18 on page 162, refer to the README file in the DB2:samples: folder. The README file will contain any additional samples that are not listed in this book.

The sample programs directory is typically read-only on most platforms. Before you alter or build the sample programs, copy them to your working directory. On the Macintosh, copy them to your working folder.

Note: The sample programs that are shipped with DB2 Universal Database have dependencies on the English version of the sample database and the associated table and column names. If the sample database has been translated into another national language on your version of DB2 Universal Database, you need to update the name of the sample database, and the names of the tables and the columns coded in the supplied sample programs, to the names used in the translated sample database. Otherwise, you will experience problems running the sample programs as shipped.

Currently, the sample database is translated into the following languages:

- Brazilian Portuguese

- French
- Korean
- Norwegian
- Simplified Chinese

In Table 12, 'Yes', in the *Embedded SQL* column, indicates that the program contains embedded SQL. A blank indicates that the program does not contain embedded SQL, and thus no precompiling is required.

Table 12 (Page 1 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
adhoc	Yes	Demonstrates dynamic SQL and the SQLDA structure to process SQL commands interactively. SQL commands are input by the user, and output corresponding to the SQL command is returned.
advsql	Yes	Demonstrates the use of advanced SQL expressions like CASE, CAST, and scalar full selects.
asynrlog	Yes	Demonstrates the use of the following API: `ASYNCHRONOUS LOG READ`

backrest		Demonstrates the use of the following APIs:
		`BACKUP DATABASE`
		`RESTORE DATABASE`
		`ROLL FORWARD DATABASE`

blobfile	Yes	Demonstrates the manipulation of a Binary Large Object (BLOB), by reading a BLOB value from the sample database and placing it in a file, the contents of which can be displayed using an external viewer.
bindfile	Yes	Demonstrates the use of the BIND API to bind an embedded SQL application to a database.
calludf	Yes	Demonstrates the use of the library of User-Defined Functions (UDFs) created by `udf` for the `sample` database tables.

client		Demonstrates the use of the following APIs:
		`SET CLIENT`
		`QUERY CLIENT`

columns	Yes	Demonstrates the use of a cursor that is processed using dynamic SQL. This program lists all the entries in the system table, SYSIBM.SYSTABLES, under a desired schema name.
cursor	Yes	Demonstrates the use of a cursor using static SQL.
d_dbconf		Demonstrates the use of the following API: `GET DATABASE CONFIGURATION DEFAULTS`
d_dbmcon		Demonstrates the use of the following API: `GET DATABASE MANAGER CONFIGURATION DEFAULTS`
db2mon		Demonstrates how to use the Database System Monitor APIs, and how to process the output data buffer returned from the Snapshot API.

Table 12 (Page 2 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
dbauth	Yes	Demonstrates the use of the following API: GET AUTHORIZATIONS
dbcat		Demonstrates the use of the following APIs: CATALOG DATABASE CLOSE DATABASE DIRECTORY SCAN GET NEXT DATABASE DIRECTORY ENTRY OPEN DATABASE DIRECTORY SCAN UNCATALOG DATABASE
dbcmt		Demonstrates the use of the following APIs: CHANGE DATABASE COMMENT
dbconf		Demonstrates the use of the following APIs: CREATE DATABASE DROP DATABASE GET DATABASE CONFIGURATION RESET DATABASE CONFIGURATION UPDATE DATABASE CONFIGURATION
dbinst		Demonstrates the use of the following APIs: ATTACH TO INSTANCE DETACH FROM INSTANCE GET INSTANCE
dbmconf		Demonstrates the use of the following APIs: GET DATABASE MANAGER CONFIGURATION RESET DATABASE MANAGER CONFIGURATION UPDATE DATABASE MANAGER CONFIGURATION
dbsnap		Demonstrates the use of the following API: DATABASE SYSTEM MONITOR SNAPSHOT
dbstart		Demonstrates the use of the following API: START DATABASE MANAGER
dbstat	Yes	Demonstrates the use of the following APIs: REORGANIZE TABLE RUN STATISTICS
dbstop		Demonstrates the use of the following APIs: FORCE USERS STOP DATABASE MANAGER
db_udcs		Demonstrates the use of the following APIs in order to simulate the collating behavior of a DB2 for OS/390 CCSID 500 (EBCDIC International) collating sequence: CREATE DATABASE DROP DATABASE

Table 12 (Page 3 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
dcscat		Demonstrates the use of the following APIs: ADD DCS DIRECTORY ENTRY CLOSE DCS DIRECTORY SCAN GET DCS DIRECTORY ENTRY FOR DATABASE GET DCS DIRECTORY ENTRIES OPEN DCS DIRECTORY SCAN UNCATALOG DCS DIRECTORY ENTRY
delet	Yes	Demonstrates static SQL to delete items from a database.
dmscont		Demonstrates the use of the following APIs in order to create a database with more than one database managed storage (DMS) container: CREATE DATABASE DROP DATABASE
dynamic	Yes	Demonstrates the use of a cursor using dynamic SQL.
ebcdicdb		Demonstrates the use of the following APIs in order to simulate the collating behavior of a DB2 for OS/390 CCSID 037 (EBCDIC US English) collating sequence: CREATE DATABASE DROP DATABASE
expsamp	Yes	Demonstrates the use of the following APIs: EXPORT IMPORT in conjunction with a DRDA database.
fillcli	Yes	Demonstrates the client-side of a stored procedure that uses the SQLDA to pass information specifying which table the stored procedure populates with random data.
fillsrv	Yes	Demonstrates the server-side of a stored procedure example that uses the SQLDA to receive information from the client specifying the table that the stored procedure populates with random data.
impexp	Yes	Demonstrates the use of the following APIs: EXPORT IMPORT
inpcli	Yes	Demonstrates stored procedures using either the SQLDA structure or host variables. This is the client program of a client/server example. (The server program is called inpsrv.) The program fills the SQLDA with information, and passes it to the server program for further processing. The SQLCA status is returned to the client program. This program shows the invocation of stored procedures using an embedded SQL CALL statement.
inpsrv	Yes	Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called inpcli.) The program creates a table (PRESIDENTS) in the sample database with the information received in the SQLDA. The server program does all the database processing and returns the SQLCA status to the client program.

Table 12 (Page 4 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
joinsql	Yes	An example using advanced SQL join expressions.
largevol	Yes	Demonstrates parallel query processing in a partitioned environment, and the use of an NFS file system to automate the merging of the result sets.
lobeval	Yes	Demonstrates the use of LOB locators and deferring the evaluation of the actual LOB data.
lobfile	Yes	Demonstrates the use of LOB file handles.
lobloc	Yes	Demonstrates the use of LOB locators.
lobval	Yes	Demonstrates the use of LOBs.
makeapi	Yes	Demonstrates the use of the following APIs: BIND PRECOMPILE PROGRAM START DATABASE MANAGER STOP DATABASE MANAGER
migrate		Demonstrates the use of the following API: MIGRATE DATABASE
monreset		Demonstrates the use of the following API: RESET DATABASE SYSTEM MONITOR DATA AREAS
monsz		Demonstrates the use of the following APIs: ESTIMATE DATABASE SYSTEM MONITOR BUFFER SIZE DATABASE SYSTEM MONITOR SNAPSHOT
nodecat		Demonstrates the use of the following APIs: CATALOG NODE CLOSE NODE DIRECTORY SCAN GET NEXT NODE DIRECTORY ENTRY OPEN NODE DIRECTORY SCAN UNCATALOG NODE
openftch	Yes	Demonstrates fetching, updating, and deleting of rows using static SQL.
outcli	Yes	Demonstrates stored procedures using the SQLDA structure. This is the client program of a client/server example. (The server program is called outsrv.) This program allocates and initializes a one variable SQLDA, and passes it to the server program for further processing. The filled SQLDA is returned to the client program along with the SQLCA status. This program shows the invocation of stored procedures using an embedded SQL CALL statement.
outsrv	Yes	Demonstrates stored procedures using the SQLDA structure. This is the server program of a client/server example. (The client program is called outcli.) The program fills the SQLDA with the median SALARY of the employees in the STAFF table of the sample database. The server program does all the database processing (finding the median). The server program returns the filled SQLDA and the SQLCA status to the client program.

Table 12 (Page 5 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
qload	Yes	Demonstrates the use of the following API: LOAD QUERY
rebind	Yes	Demonstrates the use of the following API: REBIND PACKAGE
rechist		Demonstrates the use of the following APIs: CLOSE RECOVERY HISTORY FILE SCAN GET NEXT RECOVERY HISTORY FILE ENTRY OPEN RECOVERY HISTORY FILE SCAN PRUNE RECOVERY HISTORY FILE ENTRY UPDATE RECOVERY HISTORY FILE ENTRY
recursql	Yes	Demonstrates the use of advanced SQL recursive queries.
regder		Demonstrates the use of the following APIs: REGISTER DEREGISTER
restart		Demonstrates the use of the following API: RESTART DATABASE
sampudf	Yes	Demonstrates the use of User-Defined Types (UDTs) and User-Defined Functions (UDFs). The UDFs declared in this program are all sourced UDFs.
setact		Demonstrates the use of the following API: SET ACCOUNTING STRING
setrundg		Demonstrates the use of the following API: SET RUNTIME DEGREE
static	Yes	Uses static SQL to retrieve information.
sws		Demonstrates the use of the following API: DATABASE MONITOR SWITCH
tabscont		Demonstrates the use of the following APIs: TABLESPACE CONTAINER QUERY OPEN TABLESPACE CONTAINER QUERY FETCH TABLESPACE CONTAINER QUERY CLOSE TABLESPACE CONTAINER QUERY SET TABLESPACE CONTAINER QUERY
tabspace		Demonstrates the use of the following APIs: TABLESPACE QUERY SINGLE TABLESPACE QUERY OPEN TABLESPACE QUERY FETCH TABLESPACE QUERY GET TABLESPACE STATISTICS CLOSE TABLESPACE QUERY
tabsql	Yes	Demonstrates the use of advanced SQL table expressions.

Table 12 (Page 6 of 6). Sample Programs Showing Embedded SQL and APIs

Sample Program Name	Embedded SQL	Program Description
tblcli		Demonstrates a call to a table function (client-side) to display weather information for a number of cities.
tblsrv		Demonstrates a table function (server-side) that processes weather information for a number of cities.
thdsrver	Yes	Demonstrates the use of posix threads APIs for thread creation and management. The program maintains a pool of contexts. A generate_work function is executed from main, and creates dynamic SQL statements that are executed by worker threads. When a context becomes available, a thread is created and dispatched to do the specified work. The work generated consists of statements to delete entries from either the STAFF or EMPLOYEE tables of the sample database. This program is only available on UNIX platforms.
tload	Yes	Demonstrates the use of the following APIs: EXPORT QUIESCE TABLESPACE FOR TABLES LOAD
trigsql	Yes	An example using advanced SQL triggers and constraints.
udf	Yes	Creates a library of User-Defined Functions (UDFs) made specifically for the sample database tables, but can be used with tables of compatible column types.
updat	Yes	Uses static SQL to update a database.
util		Demonstrates the use of the following APIs: GET ERROR MESSAGE GET SQLSTATE MESSAGE INSTALL SIGNAL HANDLER INTERRUPT This program also contains code to output information from an SQLDA.
varinp	Yes	An example of variable input to Embedded Dynamic SQL statement calls using parameter markers.

Table 13 (Page 1 of 2). Log Management User Exit Sample Programs.

Sample File Name	File Description
db2uext2.cadsm	This is a sample User Exit utilizing ADSTAR DSM (ADSM) APIs to archive and retrieve database log files. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only. The Windows NT version is db2uext2.cad. The OS/2 version is db2uexit.cad.
db2uext2.cad	This is the Windows NT version of db2uext2.cadsm. The file must be renamed db2uext2.c and compiled as a C program.

Table 13 (Page 2 of 2). Log Management User Exit Sample Programs.

Sample File Name	File Description
db2uexit.cad	This is the OS/2 version of db2uext2.cadsm. The file must be renamed db2uexit.c and compiled as a C program.
db2uext2.cdisk	This is a sample User Exit utilizing the AIX system copy command to archive and retrieve database log files. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only.
db2uext2.ctape	This is a sample User Exit utilizing the SUN system tape commands to archive and retrieve database log files. All limitations of the SUN system tape commands are limitations of this user exit. The sample provides an audit trail of calls (stored in a separate file for each option) including a timestamp and parameters received. It also provides an error trail of calls in error including a timestamp and an error isolation string for problem determination. These options can be disabled. The file must be renamed db2uext2.c and compiled as a C program. Available on supported UNIX platforms only.

Table 14. Command Line Processor (CLP) Sample Programs.

Sample Program Name	Program Description
const	Creates a table with a CHECK CONSTRAINT clause.
cte	Demonstrates a common table expression. The equivalent sample program demonstrating this advanced SQL statement is tabsql.
flt	Demonstrates a recursive query. The equivalent sample program demonstrating this advanced SQL statement is recursql.
join	Demonstrates an outer join of tables. The equivalent sample program demonstrating this advanced SQL statement is joinsql.
stock	Demonstrates the use of triggers. The equivalent sample program demonstrating this advanced SQL statement is trigsql.
testdata	Uses DB2 built-in functions such as RAND() and TRANSLATE() to populate a table with randomly generated test data.
thaisort	This script is particularly for Thai users. Thai sorting is by phonetic order requiring pre-sorting/swapping of the leading vowel and its consonant, as well as post-sorting in order to view the data in the correct sort order. The file implements Thai sorting by creating UDF functions presort and postsort, and creating a table; then it calls the functions against the table to sort the table data. To run this program, you first have to build the user-defined function program, udf, from the C source file, udf.c.

Table 15 (Page 1 of 2). Java Database Connectivity (JDBC) Sample Programs

Sample Program Name	Program Description
DB2Appl.java	A JDBC application that queries the sample database using the invoking user's privileges.

Table 15 (Page 2 of 2). Java Database Connectivity (JDBC) Sample Programs

Sample Program Name	Program Description
DB2Applt.java	A JDBC applet that queries the sample database using a user and server specified as applet parameters.
DB2Applt.html	An HTML file that embeds the applet sample program, DB2Applt. It needs to be customized with server and user information.
DB2Stp.java	A Java stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client.
DB2Udf.java	A Java UDF that demonstrates several tasks, including integer division, manipulation of Character Large OBjects (CLOBs), and the use of Java instance variables.

Table 16. Embedded SQL for Java (SQLJ) Sample Programs

Sample Program Name	Program Description
App.sqlj	An SQLJ application that uses static SQL to retrieve and update data from the EMPLOYEE table of the sample database.
Applt.sqlj	An SQLJ applet that queries the sample database using a user and server specified as applet parameters.
Applt.html	An HTML file that embeds the applet sample program, Applt. It needs to be customized with server and user information.
Stp.sqlj	An embedded SQL (SQLJ) stored procedure that updates the EMPLOYEE table on the server, and returns new salary and payroll information to the client.
CatUdf.sqlj	An SQLJ program that demonstrates cataloging Java UDFs and creating a sample table, udftest, for testing them.
Udf.sqlj	An SQLJ program that demonstrates calling Java UDFs against the sample table, udftest.
DropUdf.sqlj	An SQLJ program that demonstrates dropping Java UDFs and the sample table, udftest.

Table 17 (Page 1 of 2). Object Linking and Embedding (OLE) Sample Programs

Sample Program Name	Program Description
sales	Demonstrates rollup queries on a Microsoft Excel sales spreadsheet (implemented in Visual Basic).
names	Queries a Lotus Notes address book (implemented in Visual Basic).
inbox	Queries Microsoft Exchange inbox e-mail messages through OLE/Messaging (implemented in Visual Basic).
invoice	An OLE automation user-defined function that sends Microsoft Word invoice documents as e-mail attachments (implemented in Visual Basic).
ccounter	A counter OLE automation user-defined function (implemented in Visual C++).
salarysrv	An OLE automation stored procedure that calculates the median salary of the STAFF table of the sample database (implemented in Visual Basic).

Table 17 (Page 2 of 2). Object Linking and Embedding (OLE) Sample Programs

Sample Program Name	Program Description
salaryclt	A client program that invokes the median salary OLE automation stored procedure `salarysrv` (implemented in Visual Basic and in Visual C++).

Table 18 (Page 1 of 3). Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
Utility files used by most CLI samples	
samputil.c	Utility functions used by most samples
samputil.h	Header file for samputil.c, included by most samples
General CLI Samples	
adhoc.c	Interactive SQL with formatted output (was typical.c)
async.c **	Run a function asynchronously (based on fetch.c)
basiccon.c	Basic connection
browser.c	List columns, foreign keys, index columns or stats for a table
colpriv.c	List column Privileges
columns.c	List all columns for table search string
compnd.c	Compound SQL example
datasour.c	List all available data sources
descrptr.c **	Example of descriptor usage
drivrcon.c	Rewrite of basiccon.c using SQLDriverConnect
duowcon.c	Multiple DUOW Connect type 2, syncpoint 1 (one phase commit)
embedded.c	Show equivalent DB2 CLI calls, for embedded SQL (in comments)
fetch.c	Simple example of a fetch sequence
getattrs.c	List some common environment, connection and statement options/attributes
getcurs.c	Show use of SQLGetCursor, and positioned update
getdata.c	Rewrite of fetch.c using SQLGetData instead of SQLBindCol
getfuncs.c	List all supported functions
getfuncs.h	Header file for getfuncs.c
getinfo.c	Use SQLGetInfo to get driver version and other information
getsqlca.c	Rewrite of adhoc.c to use prepare/execute and show cost estimate
lookres.c	Extract string from resume clob using locators
mixed.sqc	CLI sample with functions written using embedded SQL (Note: This file must be precompiled)
multicon.c	Multiple connections
native.c	Simple example of calling SQLNativeSql, and SQLNumParams

Table 18 (Page 2 of 3). Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
prepare.c	Rewrite of fetch.c, using prepare/execute instead of execdirect
proccols.c	List procedure parameters using SQLProcedureColumns
procs.c	List procedures using SQLProcedures
sfetch.c **	Scrollable cursor example (based on xfetch.c)
setcolat.c	Set column attributes (using SQLSetColAttributes)
setcurs.c	Rewrite of getcurs.c using SQLSetCurs for positioned update
seteattr.c	Set environment attribute (SQL_ATTR_OUTPUT_NTS)
tables.c	List all tables
typeinfo.c	Display type information for all types for current data source
xfetch.c	Extended Fetch, multiple rows per fetch
BLOB Samples	
picin.c	Loads graphic BLOBS into the emp_photo table directly from a file using SQLBindParamToFile
picin2.c	Loads graphic BLOBS into the emp_photo table using SQLPutData
showpic.c	Extracts BLOB picture to file (using SQLBindColToFile), then displays the graphic.
showpic2.c	Extracts BLOB picture to file using piecewise output, then displays the graphic.
Stored Procedure Samples	
clicall.c	Defines a CLI function which is used in the embedded SQL sample mrspcli3.sqc
inpcli.c	Call embedded input stored procedure samples/c/inpsrv
inpcli2.c	Call CLI input stored procedure inpsrv2
inpsrv2.c	CLI input stored procedure (rewrite of embedded sample inpsrv.sqc)
mrspcli.c	CLI program that calls mrspsrv.c
mrspcli2.c	CLI program that calls mrspsrv2.sqc
mrspcli3.sqc	An embedded SQL program that calls mrspsrv2.sqc using clicall.c
mrspsrv.c	Stored procedure that returns a multi-row result set
mrspsrv2.sqc	An embedded SQL stored procedure that returns a multi-row result set
outcli.c	Call embedded output stored procedure samples/c/inpsrv
outcli2.c	Call CLI output stored procedure inpsrv2
outsrv2.c	CLI output stored procedure (rewrite of embedded sample inpsrv.sqc)
Samples using ORDER tables created by create.c (Run in the following order)	
create.c	Creates all tables for the order scenario
custin.c	Inserts customers into the customer table (array insert)
prodin.c	Inserts products into the products table (array insert)
prodpart.c	Inserts parts into the prod_parts table (array insert)
ordin.c	Inserts orders into the ord_line, ord_cust tables (array insert)

Table 18 (Page 3 of 3). Sample CLI Programs in DB2 Universal Database

Sample Program Name	Program Description
ordrep.c	Generates order report using multiple result sets
partrep.c	Generates exploding parts report (recursive SQL Query)
order.c	UDF library code (declares a 'price' UDF)
order.exp	Used to build order library
Version 2 Samples unchanged	
v2sutil.c	samputil.c using old v2 functions
v2sutil.h	samputil.h using old v2 functions
v2fetch.c	fetch.c using old v2 functions
v2xfetch.c	xfetch.c using old v2 functions

Note: Samples marked with a ** are new for this release.

Other files in the `samples/cli` directory include:

- README - Lists all example files.
- makefile - Makefile for all files

Changes to Chapter 2. Setup

The following will be updated. On page 19, change the title, first three paragraphs, and first sentence of the fourth paragraph of "Setting the Windows NT and Windows 95 Environment" to the following:

Setting the Windows NT, Windows 98, and Windows 95 Environment

When you install the DB2 SDK for Windows NT, the install program updates the Windows NT configuration registry with the environment variables `INCLUDE`, `LIB`, `PATH`, `DB2PATH`, and `DB2INSTANCE`. The default instance is DB2.

When you install the DB2 SDK for Windows 98 or the DB2 SDK for Windows 95, the install program updates the `autoexec.bat` file.

You can override these environment variables to set the values for the machine or the currently logged-on user. To override these values, use any of the following:

- The Windows 98, Windows 95 or Windows 3.1 command window
- The Windows NT control panel
- The Windows 98 or Windows 95 `autoexec.bat` file

Note: Exercise caution when changing these environment variables. **Do not** change the `DB2PATH` environment variable.

These environment variables can be updated for running most Windows NT, Windows 98 and Windows 95 programs.

Changes to Chapter 3. Introduction to Embedded SQL Applications

The following change applies to page 30. Replace the note beginning "Of the samples given in Table 7" with the following:

Note: Of the samples given in Table 7 on page 29, the C++ directory, `%DB2PATH%\samples\cpp`, contains C++ versions of `updat`, `outsrv` and `outcli`. It also has several other embedded SQL sample programs including stored procedures. See the README file in the `%DB2PATH%\samples\cpp` directory for more information.

Changes to Chapter 7. Building DB2 Call Level Interface (CLI) Applications

On page 128, replace the first part of the section "Windows NT and Windows 95" with the following:

Windows NT, Windows 98 and Windows 95

Note: All applications on Windows NT, Windows 98 and Windows 95, both embedded SQL and non-embedded SQL, must be built in a DB2 command window, and not from an operating system command prompt.

Microsoft Visual C++ is used in the following batch file, `clibld.bat`.

```
rem  clibld batch file - Windows NT, Windows 98 and Windows 95 - Microsoft Visual C++
rem  Build a CLI sample C program.
rem  Compile the program.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 -I:%DB2PATH%\include clisampl.c
rem Compile common utility functions used by most CLI sample programs.
cl -Z7 -Od -c -W1 -D_X86=1 -DWIN32 -I:%DB2PATH%\include samputil.c
rem  Link the program.
link -debug:full -debugtype:cv -OUT:clisampl.exe clisampl.obj samputil.obj db2cli.lib
```

Compile and Link Options for clibld	
The batch file contains the following compile options:	
`cl`	The Microsoft Visual C++ compiler.
`-Z7`	C7 style CodeView information generated.
`-Od`	Disable optimizations. It is easier to use a debugger with optimization off.
`-c`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-W1`	Set warning level.

Compile and Link Options for clibld	
The batch file contains the following link options:	
`link`	Use the 32-bit linker to link edit.
`-debug:full`	Include debugging information.
`-debugtype:cv`	Indicate the debugger type.
`-OUT:clisampl.exe`	Specify the executable.
`clisampl.obj`	Include the object file.
`samputil.obj`	Include the utility object file for error checking.
`db2cli.lib`	Link with the DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

On page 131, replace the first part of the section "OS/2" with the following:

OS/2

IBM VisualAge C++ is used in the following command file, `clibld.cmd`.

```
rem clibld command file - OS/2 - IBM VisualAge C++ compiler
rem Build a CLI sample C program.
rem Compile the program.
icc -C+ -O- -Ti+ clisampl.c
rem Compile common utility functions used by most CLI sample programs.
icc -C+ -O- -Ti+ samputil.c
rem Link the program.
ilink /NOFREE /NOI /DEBUG /ST:32000 /PM:VIO clisampl.obj
    samputil.obj,clisampl.exe,NUL,db2cli.lib;
```

Compile and Link Options for clibld	
The command file contains the following compile options:	
`icc`	The IBM VisualAge C++ compiler.
`-C+`	Perform compile only; no link. This book assumes that compile and link are separate steps.
`-O-`	No optimization. It is easier to use a debugger with optimization off.
`-Ti+`	Generate debugger information

Compile and Link Options for clibld	
The command file contains the following link options:	
`ilink`	Use the ilink linker to link edit.
`/NOFREE`	No free format.
`/NOI`	No Ignore Case. Force case sensitive identifiers.
`/DEBUG`	Include debugging information.
`/ST:32000`	Specify a stack size of at least 32 000.
`/PM:VIO`	Enable the program to run in an OS/2 window.
`clisampl.obj`	Include the object file.
`samputil.obj`	Include the utility object file for error checking.
`clisampl.exe`	Specify the executable.
`NUL`	Specify a NUL option.
`db2cli.lib`	Link with the DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

Changes to Chapter 8. Building Java Applications and Applets

The following replaces the entire chapter.

You can develop Java programs to access DB2 databases with the appropriate Java Development Kit (JDK) on Windows NT, Windows 98, Windows 95, and OS/2. The JDK includes Java Database Connectivity (JDBC), a dynamic SQL API for Java.

DB2 JDBC support is provided by the DB2 Client Application Enabler (DB2 CAE). With this support you can build and run JDBC applications and applets. These contain dynamic SQL only, and use a Java call interface to pass SQL statements to DB2.

The DB2 Software Developer's Kit (DB2 SDK) provides support for Java embedded SQL (SQLJ). With DB2 SQLJ support and DB2 JDBC support you can build and run SQLJ applications and applets. These contain static SQL and use embedded SQL statements that are bound to a DB2 database.

The SQLJ support provided by the DB2 SDK includes:

- The SQLJ translator, `sqlj`, which replaces embedded SQL statements in the SQLJ program with Java source statements, and generates a serialized profile which contains information about the SQL operations found in the SQLJ program. The SQLJ translator uses the `%DB2PATH%\java\sqlj.zip` file.
- The DB2 SQLJ profile customizer, `db2profc`, which precompiles the SQL statements stored in the generated profile, customizing them into calls to the SQLJ runtime function, and generates a package in the DB2 database. For more information on the `db2profc` command, see the *Command Reference*.
- The DB2 SQLJ runtime function, which provides a runtime interface to the DB2 database manager.

Building and running different types of Java programs requires support from different components of DB2:

- To build JDBC applications requires the DB2 Client Application Enabler (DB2 CAE). To run JDBC applications requires the DB2 CAE in order to connect to DB2.
- To build SQLJ applications requires the DB2 CAE and the DB2 SDK. To run SQLJ applications requires the DB2 CAE in order to connect to DB2.
- To build JDBC applets requires the DB2 CAE. No DB2 component is required to run JDBC applets on a client machine.
- To build SQLJ applets requires the DB2 CAE and the DB2 SDK. No DB2 component is required to run SQLJ applets on a client machine.

For more information on DB2 programming in Java, see “Programming in JDBC” on page 101. This covers creating and running JDBC applications, applets, stored procedures and UDFs. For information on SQLJ applications, applets, stored procedures and UDFs, see “Embedded SQL for Java (SQLJ) Programming” on page 117.

For the latest, updated DB2 Java information, visit the Web Page at `http://www.software.ibm.com/data/db2/java`.

Setting the Environment

Windows NT, Windows 98 and Windows 95

To build Java applications on a supported Windows platform with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1 for Win32 from Sun Microsystems (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for Windows NT and Windows 95 from the DB2 Client Pack. It must be Version 2.1.2 or later.

To run DB2 Java stored procedures or UDFs, you also need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH c:\jdk11
```

where `c:\jdk11` is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the `JDK11_PATH` field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The `JDK11_PATH` field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on a supported Windows platform with DB2 JDBC support, the following environment variables are automatically updated when DB2 is installed, to ensure that:

- CLASSPATH includes "." and the file `%DB2PATH%\java\db2java.zip`

- PATH includes the directory %DB2PATH%\bin

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\runtime.zip
```

OS/2

To build Java applications on OS/2 with DB2 JDBC support, you need to install and configure the following on your development machine:

1. The Java Development Kit (JDK) Version 1.1 for OS/2 from IBM (refer to http://www.software.ibm.com/data/db2/java).
2. The DB2 Client Application Enabler for OS/2 from the DB2 Client Pack. It must be Version 2.1.2 or later.

The JDK must be installed in an HPFS drive to allow long filenames with extensions greater than three characters, such as .java. Your Java working directory must also be on an HPFS drive. If you will be running Java stored procedures or UDFs on an OS/2 server, DB2 must be installed on an HPFS drive on the server in order to allow the stored procedure or UDF .class files to be placed in the %DB2PATH%\function directory.

To run DB2 Java stored procedures or UDFs, you need to update the DB2 database manager configuration on the server to include the path where the JDK is installed on that machine. You can do this by entering the following on the server command line:

```
db2 update dbm cfg using JDK11_PATH c:\jdk11
```

where c:\jdk11 is the path where the JDK is installed.

You can check the DB2 database manager configuration to verify the correct value for the JDK11_PATH field by entering the following command on the server:

```
db2 get dbm cfg
```

You may want to pipe the output to a file for easier viewing. The JDK11_PATH field appears near the beginning of the output. For more information on these commands, see the *Command Reference*.

To run Java programs on OS/2 with DB2 JDBC support, the following environment variables are automatically updated when DB2 is installed, to ensure that:

- CLASSPATH includes "." and the file %DB2PATH%\java\db2java.zip
- PATH includes the directory %DB2PATH%\bin
- LIBPATH includes the directory %DB2PATH%\dll

To build SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\sqlj.zip
```

To run SQLJ programs, CLASSPATH is also updated to include the file:

```
%DB2PATH%\java\runtime.zip
```

Java Sample Programs

DB2 provides sample programs, used in the following sections, to demonstrate building and running JDBC programs that exclusively use dynamic SQL, and SQLJ programs that use static SQL. The Java samples are located in the `%DB2PATH%\samples\java` directory. The directory also contains a README and a makefile. Please see the section “The Java Makefile” on page 170.

Before modifying or building the sample programs, it is recommended that you copy them from the `%DB2PATH%\samples\java` directory to a separate working directory.

On OS/2, your working directory must be on an HPFS drive. Since DB2 sample programs are provided on OS/2 to be compatible with FAT drives, filenames have at most a three character extension. To comply with this restriction, Java sample files have been truncated. After copying the Java files to your working directory, you can rename the truncated files by the following commands:

```
move *.jav *.java
move *.htm *.html
move *.sql *.sqlj
```

To run these sample programs, you must first create and populate the `sample` database by entering:

```
db2sampl
```

General Points for Building and Running DB2 Java Programs

1. You must build and run DB2 Java applications and applets from a window where your environment variables are set. You can do this by running `db2profile`. Refer to Chapter 2, "Setup", on page 19 if you need more information.
2. To build DB2 SQLJ programs, or to run any DB2 Java programs, the database manager on the server must be started. Start the database manager, if it is not already running, by entering the following command on the server:

   ```
   db2start
   ```

The Java Makefile

The `makefile` provided for the Java sample programs is presented below. The `makefile` will only work if a compatible make executable program is resident on your system in a directory included in your PATH variable. A suitable make utility may be provided by another language compiler. Please read the comment at the beginning of the text of the `makefile` for more information.

The make commands used to build specific Java sample programs are cited in the sections that follow. There is one change from makefiles normally used for other languages. The `make clean` command removes all files produced by the java compiler:

`.java` files and `core` dumps; the `make cleanall` command removes these files as well as any files produced by the SQLJ translator.

The makefiles for the supported Windows platforms and for OS/2 are identical except for the header information,

for Windows platforms:

```
# IBM DB2 Universal Database Version 5
# for Windows NT, Windows 98 and Windows 95
# Makefile for DB2 Java samples
```

and for OS/2:

```
# IBM DB2 Universal Database Version 5 for OS/2
# Makefile for DB2 java samples
```

The rest of the makefile is as follows:

```
# The makefile will only work if a compatible make executable program is
# resident on your system in a directory included in your PATH variable.
# Such a make utility may be provided by another language compiler. If you
# do not have a compatible make utility you cannot use this makefile, but
# you can still compile and run these programs by entering the commands
# at the command line, as explained in the README.

# To build your applications using this makefile, you can use one of
# the following commands:

#   make all         - builds all the programs in this directory
#   make <prog_name> - builds a program designated by <prog_name>
#   make clean       - removes all files produced by the java compiler
#                      from your working directory (such as .class files)
#   make cleanall    - removes all files from your working directory produced
#                            by both the sqlj translator and java compiler.

# This file assumes the DB2 instance path is defined by the variable HOME.
# It also assumes that DB2 is installed under the DB2 instance.
# If these conditions are not true, redefine DB2INSTANCEPATH
DB2INSTANCEPATH = $(HOME)

# Use the java compiler
CC= javac

# To connect to another database update the DATASOURCE variable.
# User ID and password are optional. If you want to use them,
# update TESTUID with your user ID, and TESTPWD with your password.

DATASOURCE=sample
TESTUID=
TESTPWD=

COPY = copy
ERASE = del
```

```
# Note: 'all' contains RunCatUdf, which executes the 'java CatUdf' command,
# as this must be run before the 'make Udf' command.
all : DB2Appl DB2Applt DB2Stp DB2Udf App Applt Stp CatUdf RunCatUdf Udf DropUdf

RunCatUdf :
      java CatUdf

clean :
      $(ERASE) *.class
      $(ERASE) core

cleanall : clean
      $(ERASE) App.java
      $(ERASE) Applt.java
      $(ERASE) Stp.java
      $(ERASE) CatUdf.java
      $(ERASE) Udf.java
      $(ERASE) DropUdf.java
      $(ERASE) *.ser

# Build and run the following JDBC application with these commands:
#
#     make DB2Appl
#     java DB2Appl
#
DB2Appl.class : DB2Appl.java
DB2Appl : DB2Appl.class
      $(CC) DB2Appl.java

# After following the setup instructions in the README, you can
# build and run the following JDBC applet with these commands:
#
#     make DB2Applt
#     appletviewer DB2Applt.html
#
DB2Applt.class : DB2Applt.java
DB2Applt : DB2Applt.class
      $(CC) DB2Applt.java

# Build and run the following JDBC stored procedure with these commands:
#
#     make DB2Stp
#     java DB2Stp
#
DB2Stp.class : DB2Stp.java
DB2Stp : DB2Stp.class
      $(CC) DB2Stp.java
      $(COPY) DB2StpSample.class $(DB2PATH)\function

# Build and run the following JDBC UDF with these commands:
#
```

```
#     make DB2Udf
#     java DB2Udf
#
DB2Udf.class : DB2Udf.java
DB2Udf : DB2Udf.class
      $(CC) DB2Udf.java
      $(COPY) DB2UdfSample.class $(DB2PATH)\function

# Build and run the following SQLJ application with these commands:
#
#     make  App
#     java  App
#
App.java : App.sqlj
      sqlj App.sqlj
App.class : App.java App_SJProfile0.ser
App : App.class
      $(CC) App.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using App" App_SJProfile0


# After following the setup instructions in the README, you can
# build and run the following SQLJ applet with these commands:
#
#     make Applt
#     appletviewer Applt.html
#
Applt.java : Applt.sqlj
      sqlj Applt.sqlj
Applt.class : Applt.java Applt_SJProfile0.ser
Applt : Applt.class
      $(CC) Applt.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using Applt" Applt_SJProfile0


# Build and run the following SQLJ stored procedure with these commands:
#
#     make  Stp
#     java  Stp
#
Stp.java : Stp.sqlj
      sqlj Stp.sqlj
Stp.class : Stp.java Stp_SJProfile0.ser
Stp : Stp.class
      $(CC) Stp.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using Stp" Stp_SJProfile0
      $(COPY) Stpsrv.class $(DB2PATH)\function
      $(COPY) Stp_Cursor1.class $(DB2PATH)\function
      $(COPY) Stp_Cursor2.class $(DB2PATH)\function
      $(COPY) Stp_SJProfileKeys.class $(DB2PATH)\function
      $(COPY) Stp_SJProfile0.ser $(DB2PATH)\function
```

```
# The following SQLJ User-Defined Function application requires three programs.
# Build and run the UDF programs with these commands:
#
#     nmake CatUdf
#     java CatUdf
#     nmake Udf
#     java Udf
#     nmake DropUdf
#     java DropUdf
#
# Note: 'java CatUdf' must be executed before 'nmake Udf'. The 'make all'
# command calls RunCatUdf, which executes 'java CatUdf', before calling Udf.
#
CatUdf.java : CatUdf.sqlj
      sqlj CatUdf.sqlj
CatUdf.class : CatUdf.java CatUdf_SJProfile0.ser
CatUdf : CatUdf.class
      $(CC) CatUdf.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using CatUdf" CatUdf_SJProfile0


Udf.java : Udf.sqlj
      sqlj Udf.sqlj
Udf.class : Udf.java Udf_SJProfile0.ser
Udf : Udf.class
      $(CC) Udf.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using Udf" Udf_SJProfile0
      $(COPY) Udfsrv.class $(DB2PATH)\function

DropUdf.java : DropUdf.sqlj
      sqlj DropUdf.sqlj
DropUdf.class : DropUdf.java DropUdf_SJProfile0.ser
DropUdf : DropUdf.class
      $(CC) DropUdf.java
      db2profc -url=jdbc:db2:sample -prepoptions="package using DropUdf" DropUdf_SJProfile0
```

JDBC Programs

Applications

`DB2Appl` demonstrates a dynamic SQL Java application using the JDBC Application driver to access a DB2 database.

Command Line. To build and run this application by commands entered at the command line:

1. Compile `DB2Appl.java` with this command:

   ```
   javac DB2Appl.java
   ```

   to produce the file: `DB2Appl.class`.

2. Run the java interpreter on the application with this command:

```
java DB2Appl
```

makefile. To build this application with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.
2. Build the application with this command:

   ```
   make DB2Appl
   ```
3. Run the java interpreter on the application with this command:

   ```
   java DB2Appl
   ```

Applets

`DB2Applt` demonstrates a dynamic SQL Java applet using the JDBC applet driver to access a DB2 database.

Command Line. To build and run this applet by commands entered at the command line:

1. Ensure that a web server is installed on your DB2 machine (server or client).
2. Modify the `DB2Applt.html` file according to the instructions there.
3. Start the JDBC applet server on the TCP/IP port specified in `DB2Applt.html`; for example, if in `DB2Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```
4. Compile `DB2Applt.java` with this command:

   ```
   javac DB2Applt.java
   ```

   to produce the file `DB2Applt.class`.
5. Ensure that your working directory is accessible by your web browser. If it is not, copy `DB2Applt.class` and `DB2Applt.html` into a directory that is accessible.
6. Copy the file `%DB2PATH%\java\db2java.zip` into the same directory as `DB2Applt.class` and `DB2Applt.html`.
7. On your client machine, start your web browser (which supports JDK 1.1) and load `DB2Applt.html`.

As an alternative to steps (1), (5) and (7), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer DB2Applt.html
```

makefile. To build this applet with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.
2. Ensure that a web server is installed on your DB2 machine (server or client).
3. Modify the `DB2Applt.html` file according to the instructions there.
4. Start the JDBC applet server on the TCP/IP port specified in `DB2Applt.html`; for example, if in `DB2Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```
5. Build the applet with this command:

   ```
   make DB2Applt
   ```
6. Ensure that your working directory is accessible by your web browser. If it is not, copy `DB2Applt.class` and `DB2Applt.html` into a directory that is accessible.
7. Copy the file `%DB2PATH%\java\db2java.zip` into the same directory as `DB2Applt.class` and `DB2Applt.html`.
8. On your client machine, start your web browser (which supports JDK 1.1) and load `DB2Applt.html`.

As an alternative to steps (2), (6) and (8), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer DB2Applt.html
```

Stored Procedures

`DB2Stp` demonstrates how to write a dynamic SQL Java stored procedure using the JDBC Application driver to access a DB2 database.

Command Line. To build and run this stored procedure by commands entered at the command line:

1. Compile `DB2Stp.java` with this command:

   ```
   javac DB2Stp.java
   ```

   This will produce the files `DB2Stp.class` and `DB2StpSample.class`.
2. Copy `DB2StpSample.class` to the `%DB2PATH%\function` directory.
3. Run the java interpreter on the stored procedure with this command:

   ```
   java DB2Stp
   ```

makefile. To build this stored procedure with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.
2. Build the stored procedure with this command:

```
make DB2Stp
```

3. Run the java interpreter on the stored procedure with this command:

   ```
   java DB2Stp
   ```

User-Defined Functions

`DB2Udf` demonstrates implementing dynamic SQL user-defined functions using the JDBC Application driver to access a DB2 database.

Command Line. To build and run this UDF program by commands entered at the command line:

1. Compile `DB2Udf.java` with this command:

   ```
   javac DB2Udf.java
   ```

   This will produce the files `DB2Udf.class` and `DB2UdfSample.class`.

2. Copy `DB2UdfSample.class` to the `%DB2PATH%\function` directory.

3. Run the java interpreter on the UDF program with this command:

   ```
   java DB2Udf
   ```

makefile. To build this UDF program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.

2. Build the UDF program with this command:

   ```
   make DB2Udf
   ```

3. Run the java interpreter on the UDF program with this command:

   ```
   java DB2Udf
   ```

SQLJ Programs

Applications

`App` demonstrates an SQLJ application that accesses a DB2 database.

Command Line. To build and run this application by commands entered at the command line:

1. Translate `App.sqlj` with this command:

   ```
   sqlj App.sqlj
   ```

   This will produce the files `App.java` and `App_SJProfile0.ser`.

2. Compile `App.java` with this command:

   ```
   javac App.java
   ```

   This will produce the files: `App.class`, `App_Cursor1.class`, `App_Cursor2.class` and `App_SJProfileKeys.class`.

3. Customize the generated profile and create the package `App` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample prepoptions="package using App" App_SJProfile0
   ```

4. Run the application with this command:

   ```
   java App
   ```

makefile. To build this application with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java Makefile" on page 170.
2. Build the application with this command:

   ```
   make App
   ```

3. Run the application with this command:

   ```
   java App
   ```

Applets

`Applt` demonstrates an SQLJ applet that accesses a DB2 database.

Command Line. To build and run this applet by commands entered at the command line:

1. Ensure that a web server is installed on your DB2 machine (server or client).
2. Modify the `Applt.html` file according to the instructions there.
3. Start the JDBC applet server on the TCP/IP port specified in `Applt.html`; for example, if in `Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```

4. Translate `Applt.sqlj` with this command:

   ```
   sqlj Applt.sqlj
   ```

   This will produce the files: `Applt.java` and `Applt_SJProfile0.ser`.

5. Compile `Applt.java` with this command:

   ```
   javac Applt.java
   ```

   This will produce the files: `Applt.class`, `Applt_Cursor1.class`, `Applt_Cursor2.class` and `Applt_SJProfileKeys.class`.

6. Customize the generated profile and create the package `Applt` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Applt" Applt_SJProfile0
   ```

7. Ensure that your working directory is accessible by your web browser. If it is not, copy the following files into a directory that is accessible:

```
Applt.html                          Applt.class,
Applt_Cursor1.class,                Applt_Cursor2.class,
Applt_SJProfileKeys.class,          Applt_SJProfile0.ser
```

8. Copy the files `%DB2PATH%\java\db2java.zip` and `%DB2PATH%\java\runtime.zip` into the same directory as your other `Applt` files.
9. On your client machine, start your web browser (which must support JDK 1.1) and load `Applt.html`.

As an Alternative to steps (1), (7) and (9), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer Applt.html
```

makefile. To build this applet with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.
2. Ensure that a web server is installed on your DB2 machine (server or client).
3. Modify the `Applt.html` file according to the instructions there.
4. Start the JDBC applet server on the TCP/IP port specified in `Applt.html`. For example, if in `Applt.html`, you specified:

   ```
   param name=port value='6789'
   ```

   then you would enter:

   ```
   db2jstrt 6789
   ```

5. Build the applet with this command:

   ```
   make Applt
   ```

6. Ensure that your working directory is accessible by your web browser. If it is not, copy the following files into a directory that is accessible:

   ```
   Applt.html,                         Applt.class,
   Applt_Cursor1.class,                Applt_Cursor2.class,
   Applt_SJProfileKeys.class,          Applt_SJProfile0.ser
   ```

7. Copy the files `%DB2PATH%\java\db2java.zip` and `%DB2PATH%\java\runtime.zip` into the same directory as your other `Applt` files.
8. On your client machine, start your web browser (which must support JDK 1.1) and load `Applt.html`.

As an Alternative to steps (2), (6) and (8), you can use the applet viewer that comes with the Java Development Kit by entering the following command in the working directory of your client machine:

```
appletviewer Applt.html
```

Stored Procedures

`Stp` demonstrates an SQLJ stored procedure that accesses a DB2 database.

Command Line. To build and run this stored procedure by commands entered at the command line:

1. Translate `Stp.sqlj` with this command:

   ```
   sqlj Stp.sqlj
   ```

   This will produce the files `Stp.java` and `Stp_SJProfile0.ser`.

2. Compile `Stp.java` with this command:

   ```
   javac Stp.java
   ```

   This will produce the files: `Stp.class`, `Stpsrv.class`, `Stp_Cursor1.class`, `Stp_Cursor2.class` and `Stp_SJProfileKeys.class`.

3. Customize the generated profile and create the package `Stp` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Stp" Stp_SJProfile0
   ```

4. Copy these files to the `%DB2PATH%\function` directory: `Stpsrv.class`, `Stp_Cursor1.class`, `Stp_Cursor2.class`, `Stp_SJProfileKeys.class` and `Stp_SJProfile0.ser`.

5. Run the stored procedure with this command:

   ```
   java Stp
   ```

makefile. To build this stored procedure with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.

2. Build the stored procedure with this command:

   ```
   make Stp
   ```

3. Run the stored procedure with this command:

   ```
   java Stp
   ```

User-Defined Functions

The `%DB2PATH%\samples\java` directory includes a UDF application consisting of three SQLJ programs:

- `CatUdf` demonstrates cataloging Java user-defined functions (UDFs) and creating a sample table, `udftest`, for testing them.
- `Udf` demonstrates calling Java UDFs against the sample table, `udftest`.
- `DropUdf` demonstrates dropping Java UDFs and the sample table, `udftest`.

CatUdf.

Command Line. To build and run this SQLJ program by commands entered at the command line:

1. Translate `CatUdf.sqlj` with this command:

   ```
   sqlj CatUdf.sqlj
   ```

   This will produce the files `CatUdf.java` and `CatUdf_SJProfile0.ser`.

2. Compile `CatUdf.java` with this command:

   ```
   javac CatUdf.java
   ```

   This will produce the files `CatUdf.class` and `CatUdf_SJProfileKeys.class`.

3. Customize the generated profile and create the package `CatUdf` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using CatUdf" CatUdf_SJProfile0
   ```

4. Run `CatUdf` with this command:

   ```
   java CatUdf
   ```

5. Next, run the `Udf` program.

makefile. To build and run this SQLJ program with the `makefile`:

1. Ensure your environment includes a compatible make utility as specified in the section “The Java Makefile” on page 170.
2. Build and run CatUdf with this command:

   ```
   make CatUdf
   ```

3. Next, run the `Udf` program.

Udf.

Command Line. To build and run this SQLJ program by commands entered at the command line:

1. Translate `Udf.sqlj` with this command:

   ```
   sqlj Udf.sqlj
   ```

   This will produce the files `Udf.java` and `Udf_SJProfile0.ser`.

2. Compile `Udf.java` with this command:

   ```
   javac Udf.java
   ```

   This will produce the files: `Udf.class`, `Udf_Cursor1.class`, `Udf_Cursor2.class`, `Udf_Cursor4.class`, `Udf_Cursor5.class`, `Udf_SJProfileKeys.class` and `Udfsrv.class`.

   Note: There is no file `Udf_Cursor3.class`.

3. Customize the generated profile and create the package `Udf` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using Udf" Udf_SJProfile0
   ```

4. Copy the `Udfsrv.class` file into `%DB2PATH%\function`.
5. Run `Udf` with this command:

   ```
   java Udf
   ```

6. Next, run the `DropUdf` program.

makefile. To build this SQLJ program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java Makefile" on page 170.
2. Build `Udf` with this command:

   ```
   make Udf
   ```

3. Run `Udf` with this command:

   ```
   java Udf
   ```

4. Next, run the `DropUdf` program.

DropUdf.

Command Line. To build and run this SQLJ program by commands entered at the command line:

1. Translate `DropUdf.sqlj` with this command:

   ```
   sqlj DropUdf.sqlj
   ```

   This will produce the files `DropUdf.java` and `DropUdf_SJProfile0.ser`.

2. Compile `DropUdf.java` with this command:

   ```
   javac DropUdf.java
   ```

   This will produce the files: `DropUdf.class` and `DropUdf_SJProfileKeys.class`.

3. Customize the generated profile and create the package `DropUdf` in the `sample` database with this command:

   ```
   db2profc -url=jdbc:db2:sample -prepoptions="package using DropUdf"
                  DropUdf_SJProfile0
   ```

4. Run `DropUdf` with this command:

   ```
   java DropUdf
   ```

makefile. To build this SQLJ program with the `makefile`, and then run it:

1. Ensure your environment includes a compatible make utility as specified in the section "The Java Makefile" on page 170.
2. Build `DropUdf` with this command:

   ```
   make DropUdf
   ```

3. Run `DropUdf` with this command:

   ```
   java DropUdf
   ```

General Points for DB2 Java Applets

1. For a larger JDBC or SQLJ applet that consists of several Java classes, you may choose to package all its classes in a single Jar file. For an SQLJ applet, you would also have to package its serialized profiles along with its classes. If you choose to do this, add your Jar file into the `archive` parameter in the "applet" tag. For details, see the JDK Version 1.1 documentation.
2. You may wish to place the file `%DB2PATH%\java\db2java.zip` (and for SQLJ applets, also the file `%DB2PATH%\java\runtime.zip`) into a directory that is shared by several applets that may be loaded from your Web site. In this case, you may need to add a `codebase` parameter into the "applet" tag in the HTML file to identify that directory. For details, see the JDK Version 1.1 documentation.
3. For information on running DB2 Java applets on a webserver, specifically the Domino Go Webserver, see:

   `http://www.software.ibm.com/data/db2/db2lotus/gojava.htm`

Changes to Appendix. Migrating Your Applications

The following new appendix is added for Version 5.2.

When you upgrade to DB2 Universal Database Version 5 from a previous installation of DB2, DB2 Client Application Enabler, or DB2 Software Developer's Kit, your database and node directories are migrated automatically. To migrate your existing databases, use the tools described in the *Administration Guide*.

You do not need to recompile your applications because binary compatibility is provided between DB2 Universal Database Version 5 and previous versions of DB2. If, however, you want to recompile, you may need to recode some of your applications.

In DB2 Universal Database Version 5, all character array items with string semantics have type char, instead of other variations, such as unsigned char. Any applications you code with DB2 Universal Database Version 5 should follow this practice.

If you have DB2 Version 1 applications which use unsigned char, your compiler might produce warnings or errors because of type clashes between unsigned char in Version 1 applications and char in Version 5 function prototypes. If this occurs, use the compiler option `-DSQLOLDCHAR` to eliminate the problem.

Refer to the *SQL Reference* for a list of incompatibilities between DB2 Universal Database Version 5 and previous versions of DB2. Refer to the *API Reference* for a list of API incompatibilities between DB2 Universal Database and previous versions of DB2.

Appendix E. Web Control Center and NetQuestion

The DB2 Universal Database Control Center can now be used from the Web. This can centralize your administration activities by consolidating these activities for all your databases through a single interface. The Web Control Center, in combination with its search service, gives your DB2 administration activities the flexibility available in a Web-based network or intranet. See the following sections for details:

- "Web Control Center Installation and Configuration": Provides information on installing and configuring the Web Control Center.
- "Enabling the Web Control Center Remote Documentation Searches" on page 190: Provides information on enabling remote searching of the DB2 product documentation.

Web Control Center Installation and Configuration

The Web Control Center is the Java version of the DB2 Universal Database Control Center. From a tools perspective, it is modeled on the DB2 Release 5 Control Center, but it provides you with a more flexible network-centric database administration environment. The Web Control Center is implemented as a Java applet that uses DB2's JDBC support.

Information on supported Web browsers and operating systems can be found on the Web at `http://www.software.ibm.com/data/db2/library/browsers.html`

Components to Install

To get started with the Web Control Center, you need to install DB2 Universal Database Server Version 5.2, where the server is any of:

- Workgroup Edition
- Enterprise Edition
- Enterprise - Extended Edition (EEE)

To run the Web Control Center, you need:

- A Web browser; for supported browsers see the Web at `http://www.software.ibm.com/data/db2/library/browsers.html`
- A Web server (optional). See "Machine Configuration" for details on when you need a Web server.

Machine Configuration

You can set up your Web Control Center (Web CC) in a number of different ways. The following table identifies four scenarios, each showing a different way of installing the required components. These scenarios are referenced throughout the Web Control Center Setup section that follows the table.

Scenario	Machine A	Machine B	Machine C
1 - Standalone	Browser Web CC applet JDBC server DB2 server		
2 - Two Tier[1]	Browser Web CC applet JDBC server CAE[3]		DB2 server
3 - Two Tier[2]	Browser	Web CC applet Web server JDBC server DB2 server	
4 - Three Tier	Browser	Web CC applet Web server JDBC server CAE	DB2 server

Note:

1. Browser and JDBC server on one machine.
2. Standalone browser.
3. Client Application Enabler.

In Scenarios 1 and 2, there is no requirement for a Web server because the Web Control Center (Web CC) applet and the browser are on the same machine.

In Scenarios 3 and 4, the Web Control Center applet code is remote, which enables code sharing across several browsers. This means that you can access the Web Control Center from browsers on different machines. Since the applet has not been implemented as a trusted applet, the Web server and the JDBC server must reside on the same machine.

Web Control Center Installation

This section describes how to install the Web Control Center and how you can customize it for your environment.

Web Control Center Setup

1. Start the DB2 servers:
   - On the Windows NT operating system, start the db2 security server. Click on **Start->Control Panel->Services**. Select the db2 security server and click on the **Start** push button.

     Note: This step is not required if your security server is autostarted.
   - On Windows NT or OS/2 operating systems, enter the `db2start` command on the machine with the DB2 Universal Database server.

     On the UNIX operating system, logon to the account where you want to run the JDBC server and issue the `db2start` command.

- On the Windows NT or OS/2 operating system containing the JDBC server, enter:

  ```
  start db2jd 6790
  ```

  On the UNIX operating system containing the JDBC server, enter:

  ```
  db2jd 6790 &
  ```

  The port number 6790 is an example; any 4-digit number that is not already in use can be used. The default port number is 6789.

  Note: The first time you start the JDBC server, it will create several node directory entries, together with various files for administration purposes. In Scenario 1 and 3, all of these administration files and directory entries will be created in the current DB2 instance.

- On the UNIX operating system, logon to the administration server account and issue the `db2admin start` command.

If you are using a UNIX operating system for Scenario 1 or 3, you can do this startup on either the Client Application Enabler or the DB2 Universal Database server.

2. You can start working with the Web Control Center without an existing database by creating a sample database. Enter the `db2sampl` command on the DB2 Universal Database server. On a UNIX operating system, ensure that you are logged on to the DB2 instance before you enter the `db2sampl` command.
3. Load the HTML page to start the Web Control Center:
   - If you are running Scenarios 1 or 2, then you do not need a Web server set up on your machine.
     a. Start the DB2 Web Control Center Launch page. In your Web browser, select **File->Open Page**. The Open Page dialog box appears.

        On a Windows or OS/2 operating system
        Enter `x:/sqllib/java/prime/db2webcc.htm`, where `x:` represents the drive where DB2 is installed.

        On a UNIX operating system
        Enter `INSTHOME/sqllib/java/prime/db2webcc.htm`, where `INSTHOME` represents the home directory of the instance.

        Click on the **Open** push button.

     b. The DB2 Web Control Center Launch page appears. When the control panel appears, click on the **Launch** push button.
   - If you are running Scenarios 3 or 4, then you must have a Web server set up on the machine that contains the Web Control Center applet code and the JDBC server. The Web server must allow access to the `sqllib`directory.
     a. Start the Web Control Center Launch page through your Web server. In your browser, select **File->Open Page**. The Open Page dialog box appears. Enter the URL of your Web server and the main Web Control

Center page and click on the **Open** push button. For example: `http://yourserver/java/prime/db2webcc.htm`.

b. The DB2 Web Control Center Launch page appears. When the control panel appears, select the **Remote** radio button, and enter the address of your Web server in the host entry field. For example: `yourserver.yourdomain.com`.

c. Enter the number of the db2jd port. For example: `6790`.

d. Click on the **Launch** push button.

4. The Control Center Sign On window opens. Enter your user ID and password. This user ID must have an account on the machine that is running the JDBC server. Your initial logon will be used for all database connections. It can be changed from the Control Center pull-down menu. A unique user profile will be assigned to each user ID. Click on **OK**.
5. The DB2 Control Center window opens.

Customizing Your Web Control Center HTML File

If you want the Web Control Center started automatically the next time you open the `db2webcc.htm` file:

- For Scenarios 1 or 2, modify the prompted parameter tag in `db2webcc.htm` from

  ```
  PARAM name="prompted" value="true"
  ```

  to

  ```
  PARAM name="prompted" value="false"
  ```

- For Scenarios 3 or 4, modify the `prompted`, `host`, and `port` parameter tags in `db2webcc.htm` to

  ```
  PARAM name="prompted" value="false"
  PARAM name="host" value="yourserver"
  PARAM name="port" value="6790"
  ```

  where `yourserver` is the host name and `6790` is the port value of the machine you want to connect to.

Configuring Your Web Server to Work with the Web Control Center

Consult the setup documentation that came with your Web server for general Web server configuration information.

To configure your Web server for use with the Web Control Center, use `sqllib` as your home directory. If you choose to use a virtual directory, substitute this directory for the home directory. For example, if you name your virtual directory `temp` then you should use `sqllib/temp`.

If you do not have the DB2 documentation installed and you would like information on configuring your Web server to work with DB2's online documentation, see the Web at `http://www.software.ibm.com/data/pubs/papers/db2html.html`.

Functional Considerations

If you are using the Web Control Center over the Internet, be aware that there is no encryption of the data flow between the JDBC server and the browser.

To use the color options of Visual Explain on Netscape, you must set your operating system to support more than 256 colors.

DB2 does not support the installation of the Web Control Center on a FAT drive for OS/2, because an OS/2 FAT drive does not support long filenames required by Java.

Every activity will be associated with an explicit DB2 connection or attachment. For security purposes, every DB2 activity will be validated.

When you are using the Web Control Center under Scenarios 3 or 4, the local system is Machine B.

Installation Tips for Web Control Center Help on UNIX Operating Systems

When installing the Web Control Center online help on UNIX operating systems you should keep the following in mind:

- You should install the Web Control Center help and the product documentation at the same time. If you install the Web Control Center help and the DB2 online product documentation separately, you can expect the second installation to take some time. This is true regardless of which is installed first.
- You must select the Web Control Center help for any non-English language explicitly. Installing the product messages for a particular language does not mean that the Web Control Center help for that language is automatically installed. However, if you install the Web Control Center help for a particular language, the product messages for that language are installed automatically.
- If you install the product documentation after editing the search form, you must edit your search form again. Installing the product documentation resets the search form to its default configuration.
- If you install the Web Control Center using SMIT rather than db2installer, you must run the `db2insthtml` command to set up document search. For more information on using SMIT to install The Web Control Center on UNIX operating systems, refer to the *DB2 for UNIX Quick Beginnings* or the *DB2 EEE for UNIX Quick Beginnings* manuals.

Differences From DB2 V5.0.0 Control Center

Each user has a profile of their own preferences stored on the machine were the JDBC server exists.

You cannot start a new Control Center from the existing Control Center.

The Event Analyzer and Snapshot Monitor are not included as part of the DB2 Web Control Center for Version 5.2.

Troubleshooting

If you are having problems running the Web Control Center applet:

- Verify that you are using a supported browser and operating system as outlined at `http://www.software.ibm.com/data/db2/library/browsers.html`.
- Check your browser's Java Console window for diagnostic and trace information for the Web Control Center.

If you are having problems connecting to the JDBC or Web server:

- Ensure the JDBC server (db2jd) is running.
- Verify the port number.
- Verify the server name.
- Ensure that you are using the `db2webcc.htm` file from the machine running the JDBC server.
- Ensure that you have an account on the machine running the JDBC server.
- Remember that the Web Control Center works within the Client Application Enabler's locale.

Enabling the Web Control Center Remote Documentation Searches

The Web Control Center allows users to view and search DB2 information (online books and help) that may be stored on a remote system. Viewing information uses a browser and the Web server, while enabling search requires some changes to your local HTML search form and IBM Internet Connection Server (ICS) Lite configuration file. To enable remote searching of the product documentation:

1. Edit the HTML search form.

   On a Windows or OS/2 operating system
   This file is `x:\sqllib\doc\html\db2srch.htm`, where `x:` is the drive where DB2 is installed.

   On an AIX operating system
   This file is `/usr/lpp/db2_05_00/doc/%L/html/db2srch.htm`, where `%L` is your locale.

   On other UNIX operating systems
   This file is `/opt/IBMdb2/V5.0/doc/%L/html/db2srch.htm`, where `%L` is your locale.

   Note: These paths are the defaults and can vary depending on paths set during the installation of DB2.

2. Change the **action=** attribute of the <**form**> tag. Replace `localhost:49213` with `hostname:49214`, where hostname represents the machine where the documentation files are installed.
3. Edit your ICS Lite configuration file.

On a Windows or OS/2 operating system, this file is `httpd.cnf` in the directory `netqos2`, `imnnq_nt`, or `imnnq95`, depending on the operating system. On Windows 98, the directory is `imnnq_95`.

On a UNIX operating system, this file is `httpdlite.conf`. For AIX operating systems, this file is in the `/etc/IMNSearch` directory. For Solaris and HP-UX operating systems, it is in the `/opt/IMNSearch` directory.

- Copy the original ICS Lite configuration file to another file name in the same directory as the original (`db2webcc.cnf` is recommended as the new file name).

  Note: By creating a copy of this configuration file, you can avoid affecting other products installed on your system that rely on the search server.

- Replace the value of the Hostname line in `db2webcc.cnf` (or whatever you named the copy of the ICS Lite configuration file) with the same `hostname:49214` as you did in the HTML search form.

- Add the following line to `db2webcc.cnf`:

  ```
  MaxActiveThreads 40
  ```

  This setting will increase ICS Lite's performance when multiple requests are arriving over the network. You can set the value of MaxActiveThreads to greater than 40, but as this value increases more memory is consumed.

- Edit your Start HTML Search Server icon's properties so that it calls the new `db2webcc.cnf` file. Add the name

  ```
  db2webcc.cnf
  ```

  as the last argument in the icon's target field. If you put the new configuration file in a directory that was different from the original, you must specify the full path name.

  On Windows NT and OS/2 operating systems, if you you have Start HTML Search Server icons in your Startup folder and your DB2 program folder, you should edit the properties for both icons. For information on editing an icon's properties consult your operating system's documentation.

- For each index that is installed on the machine, run the **nqmap** command to set the base location for the documentation files. The exact directory to use depends on your hostname. On a UNIX operating system, it also depends on the locale of the HTML files for that particular language.

  For example, if your server was named *yourserver*, the index you were mapping was Administration (*DB2ADM*), and your locale (*%L*) was set to English (*EN*):

  OS/2 and Windows:

  ```
  nqmap DB2ADMEN "http://yourserver/doc/html/" DB2ADMEN
  ```

  AIX:

  ```
  /usr/IMNSearch/cli/imnmap DB2ADMEN "http://yourserver/doc/%L/html/"
  DB2ADMEN
  ```

Note: For double-byte character (DBCS) indexes, replace `imnmap` with `imqmap`.

Solaris and HP-UX:

/opt/IMNSearch/cli/nqmap DB2ADMEN "http://yourserver/doc/%L/html/" DB2ADMEN

- Repeat these commands for each installed index.
  - For the Application Development index, substitute DB2APD**XX**
  - For the DB2 Connect index, substitute DB2CON**XX**
  - For the Web Control Center Help, substitute DB2HLP**XX**

  In these examples **XX** is the index's locale.
- Stop the search server and start it again.
  - On Windows NT or OS/2 operating systems, select the appropriate icon in your DB2 program folder.

Appendix F. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, and books. This section describes the information that is provided, and how to access it.

To access product information online, you can use the Information Center. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. See “Information Center” on page 202 for details.

SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available through the Control Center. The following table lists the SmartGuides.

Note: Not all SmartGuides are available for the partitioned database environment.

SmartGuide	Helps you to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click on **Add**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**New**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure performance**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to backup and select **Backup**->**Database using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**.

Online Help

Online help is available with all DB2 components. The following table describes the various types of help. You can also access DB2 information through the Information Center. For information see “Information Center” on page 202.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: **?** *command* where *command* is a keyword or the entire command. For example, **? catalog** displays help for all the CATALOG commands, while **? catalog database** displays help for the CATALOG DATABASE command.
Control Center Help	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click on the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message, and any action you should take.	From the command line processor in interactive mode, enter: **?** *XXXnnnnn* where *XXXnnnnn* is a valid message identifier. For example, **? SQL30081** displays help about the SQL30081 message. To view message help one screen at a time, enter: **?** *XXXnnnnn* **| more** To save message help in a file, enter: **?** *XXXnnnnn* > *filename.ext* where *filename.ext* is the file where you want to save the message help.
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: **help** *statement* where *statement* is an SQL statement. For example, **help** *SELECT* displays help about the SELECT statement.

Type of Help	Contents	How to Access...
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five-digit SQL state and the *class-code* is first two digits of the SQL state. For example, **? 08003** displays help for the 08003 SQL state, while **? 08** displays help for the 08 class code.

DB2 Books

The table in this section lists the DB2 books. They are divided into two groups:

Cross-platform books These books contain the common DB2 information for UNIX-based and Intel-based platforms.

Platform-specific books These books are for DB2 on a specific platform. For example, for DB2 on OS/2, on Windows NT, and on the UNIX-based platforms, there are separate *Quick Beginnings* books.

Most books are available in HTML and PostScript format, and in hardcopy that you can order from IBM. The exceptions are noted in the table.

If you want to read the English version of the books, they are always provided in the directory that contains the English documentation.

You can obtain DB2 books and access information in a variety of different ways:

View See “Viewing Online Books” on page 199.
Search See “Searching Online Books” on page 200.
Print See “Printing the PostScript Books” on page 200.
Order See “Ordering the Printed DB2 Books” on page 201.

Book Name	Book Description	Form Number File Name
Cross-Platform Books		
Administration Getting Started	Introduces basic DB2 database administration concepts and tasks, and walks you through the primary administrative tasks.	S10J-8154 db2k0x50
Administration Guide	Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment.	S10J-8157 db2d0x51
API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	S10J-8167 db2b0x51

Book Name	Book Description	Form Number File Name
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	S10J-8159 db2l0x50
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	S10J-8166 db2n0x51
DB2 Connect Enterprise Edition Quick Beginnings	Provides planning, migrating, installing, configuring, and using information for DB2 Connect Enterprise Edition. Also contains installation and setup information for all supported clients.	S10J-7888 db2cyx51
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Personal Edition.	S10J-8162 db2c1x51
DB2 Connect User's Guide	Provides concepts, programming and general using information about the DB2 Connect products.	S10J-8163 db2c0x51
DB2 Connectivity Supplement	Provides setup and reference information for customers who want to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA Application Requesters with DB2 Universal Database servers, and customers who want to use DRDA Application Servers with DB2 Connect (formerly DDCS) application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x51
Embedded SQL Programming Guide	Explains how to develop applications that access DB2 databases using embedded SQL, and includes discussions about programming techniques and performance considerations.	S10J-8158 db2a0x50
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50
Installing and Configuring DB2 Clients	Provides installation and setup information for all DB2 Client Application Enablers and DB2 Software Developer's Kits. **Note:** Available in HTML and PostScript formats only.	No form number db2iyx51
Master Index	Contains a cross reference to the major topics covered in the DB2 library. **Note:** Available in PostScript format and hardcopy only.	S10J-8170 db2w0x50
Messages Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	S10J-8168 db2m0x51

Book Name	Book Description	Form Number File Name
DB2 Replication Guide and Reference	Provides planning, configuring, administering, and using information for the IBM Replication tools supplied with DB2.	S95H-0999 db2e0x52
Road Map to DB2 Programming	Introduces the different ways your applications can access DB2, describes key DB2 features you can use in your applications, and points to detailed sources of information for DB2 programming.	S10J-8155 db2u0x50
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	S10J-8156 db2y0x50
SQL Reference	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views.	S10J-8165 db2s0x51
System Monitor Guide and Reference	Describes how to collect different kinds of information about your database and the database manager. Explains how you can use the information to understand database activity, improve performance, and determine the cause of problems.	S10J-8164 db2f0x50
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169 db2p0x50
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 5.2, including information about Java-based tools.	S04L-6230 db2q0x51
Platform-Specific Books		
Building Applications for UNIX Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a UNIX system.	S10J-8161 db2axx51
Building Applications for Windows and OS/2 Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Windows or OS/2 system.	S10J-8160 db2a1x50
DB2 Personal Edition Quick Beginnings	Provides planning, installing, migrating, configuring, and using information for DB2 Universal Database Personal Edition on OS/2, Windows 95, and the Windows NT operating systems.	S10J-8150 db2i1x50
DB2 SDK for Macintosh Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Macintosh system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0528 sqla7x02
DB2 SDK for SCO OpenServer Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SCO OpenServer system. **Note:** Available for DB2 Version 2.1.2 only.	S89H-3242 sqla9x02

Book Name	Book Description	Form Number File Name
DB2 SDK for SINIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SINIX system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0530 sqla8x00
Quick Beginnings for OS/2	Provides planning, installing, migrating, configuring, and using information for DB2 Universal Database on OS/2. Also contains installing and setup information for all supported clients.	S10J-8147 db2i2x50
Quick Beginnings for UNIX	Provides planning, installing, configuring, migrating, and using information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for all supported clients.	S10J-8148 db2ixx51
Quick Beginnings for Windows NT	Provides planning, installing, configuring, migrating, and using information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for all supported clients.	S10J-8149 db2i6x50
DB2 Extended Enterprise Edition for UNIX Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for UNIX. This book supercedes the *DB2 Extended Enterprise Edition Quick Beginnings for AIX* book, and is suitable for use with all versions of DB2 Extended Enterprise Edition that run on UNIX-based platforms.	S99H-8314 db2v3x51
DB2 Extended Enterprise Edition for Windows NT Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for Windows NT.	S09L-6713 db2v6x51

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e50` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier	Language	Identifier
Brazilian Portuguese	B	Japanese	J
Bulgarian	U	Korean	K
Czech	X	Norwegian	N
Danish	D	Polish	P
English	E	Russian	R
Finnish	Y	Simp. Chinese	C
French	F	Slovenia	L
German	G	Spanish	Z
Greek	A	Swedish	S
Hungarian	H	Trad. Chinese	T

Italian	I	Turkish	M

2. For late breaking information that could not be included in the DB2 books:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/%L directory, where %L is the locale name and DB2DIR is:
     - /usr/lpp/db2_05_00 on AIX
     - /opt/IBMdb2/V5.0 on HP-UX, Solaris, SCO UnixWare 7, and SGI.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.

Viewing Online Books

The manuals included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information, and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can use any HTML Version 3.2-compliant browser to view the online books.

To view online books:

- If you are running DB2 administration tools, use the Information Center. See "Information Center" on page 202 for details.
- Use the open file function of your Web browser. The page you open contains descriptions of and links to DB2 books:
  - On UNIX-based platforms, open the following page:

    `file:/INSTHOME/sqllib/doc/%L/html/index.htm`

    where *%L* is the locale name.
  - On other platforms, open the following page:

    `sqllib\doc\html\index.htm`

    The path is located on the drive where DB2 is installed.

    You can also open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

    Note: The **DB2 Online Books** icon is only available if you do not install the Information Center.

Setting up a Document Server

By default the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, use the following instructions:

1. Copy all files and sub-directories from \sqllib\doc\html on your local system to a web server. Each book has its own sub-directory containing all the necessary

HTML and GIF files that make up the book. Ensure that the directory structure remains the same.

2. Configure the web server to look for the files in the new location. For information, see *Setting up DB2 Online Documentation on a Web Server* at:

   `http://www.software.ibm.com/data/pubs/papers/db2html.html`

3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. Once you are able to view the book files, you should bookmark commonly viewed topics such as:
   - List of books
   - Tables of contents of frequently used books
   - Frequently referenced articles like the *ALTER TABLE* topic
   - Search form.

For information about setting up a search, see the *What's New* book.

Searching Online Books

To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic.
- Click on **Index** at the bottom of any page in an HTML book. Use the Index to find a specific topic in the book.
- Display the Table of Contents or Index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See “Information Center” on page 202 for details.

Printing the PostScript Books

If you prefer to have printed copies of the manuals, you can decompress and print PostScript versions. For the file name of each book in the library, see the table in “DB2 Books” on page 195.

Note: Specify the full path name for the file you intend to print.

On OS/2 and Windows platforms:

1. Copy the compressed PostScript files to a hard drive on your system. The files have a file extension of .exe and are located in the x:\doc*language*\books\ps directory, where x: is the letter representing the CD-ROM drive and *language* is the two-character country code that represents your language (for example, EN for English).
2. Decompress the file that corresponds to the book that you want. The result from this step is a printable PostScript file with a file extension of .psz.

3. Ensure that your default printer is a PostScript printer capable of printing Level 1 (or equivalent) files.
4. Enter the following command from a command line:

   ```
   print filename.psz
   ```

On UNIX-based platforms:

1. Mount the CD-ROM. Refer to your *Quick Beginnings* manual for the procedures to mount the CD-ROM.
2. Change to /cdrom/doc/%L/ps directory on the CD-ROM, where */cdrom* is the mount point of the CD-ROM and *%L* is the name of the desired locale. The manuals will be installed in the previously-mentioned directory with file names ending with .ps.Z.
3. Decompress and print the manual you require using the following command:
   - For AIX:

     ```
     zcat filename | qprt -P PSPrinter_queue
     ```

   - For HP-UX, Solaris, or SCO UnixWare 7:

     ```
     zcat filename | lp -d PSPrinter_queue
     ```

   - For Silicon Graphics IRIX and SINIX:

     ```
     zcat < filename | lp -d PSPrinter_queue
     ```

   where *filename* is the name of the full path name and extension of the compressed PostScript file and *PSprinter_queue* is the name of the PostScript printer queue.

   For example, to print the English version of *Quick Beginnings for UNIX* on AIX, you can use the following command:

   ```
   zcat /cdrom/doc/en/ps/db2ixe50.ps.Z | qprt -P ps1
   ```

Ordering the Printed DB2 Books

You can order the printed DB2 manuals either as a set, or individually. There are three sets of books available. The form number for the entire set of DB2 books is SBOF-8915-00. The form number for the set of books updated for Version 5.2 is SBOF-8921-00. The form number for the books listed under the heading "Cross-Platform Books" is SBOF-8914-00.

Note: These form numbers only apply if you are ordering books that are printed in the English language.

You can also order books individually by the form number listed in “DB2 Books” on page 195. To order printed versions, contact your IBM authorized dealer or marketing representative, or phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Information Center

The Information Center provides quick access to DB2 product information. You must install the DB2 administration tools to obtain the Information Center.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu
- Help menu of the Control Center
- **db2ic** command.

The Information Center provides the following kinds of information. Click on the appropriate tab to look at the information:

Tasks	Lists tasks you can perform using DB2.
Reference	Lists DB2 reference information, such as keywords, commands, and APIs.
Books	Lists DB2 books.
Troubleshooting	Lists categories of error messages and their recovery actions.
Sample Programs	Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.
Web	Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides some search capabilities so you can look for specific topics, and filter capabilities to limit the scope of your searches.

For a full text search, follow the *Search DB2 Books* link in each HTML file, or use the search feature of the help viewer.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server via its icon on the Windows or OS/2 desktop.

Refer to the release notes if you experience any other problems when searching the HTML information.

Appendix G. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

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 Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DatagLANce
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
Lan Distance
MVS/ESA
MVS/XA
NetView
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SAA
SP
SQL/DS
SQL/400
S/370
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc.

HP-UX is a trademark of Hewlett-Packard.

Java, HotJava, Solaris, Solstice, and Sun are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, Visual Basic, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

SCO is a trademark of The Santa Cruz Operation.

SINIX is a trademark of Siemens Nixdorf.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

Telephone

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by accessing the following page:

`http://www.ibm.com/support/`

then performing a search using the keyword “handbook.”

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Part Number: 04L6230

Printed in U.S.A.