DB2 Universal Database Programming Workshop for Linux, UNIX, and Windows
(Course Code CF10)

Student Notebook

ERC 8.1

IBM Certified Course Material
Trademarks

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

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

<table>
<thead>
<tr>
<th>AIX</th>
<th>Approach</th>
<th>CICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>DB2 Connect</td>
<td>DB2 Universal Database</td>
</tr>
<tr>
<td>Distributed Relational Database Architecture</td>
<td>DRDA</td>
<td>IMS</td>
</tr>
<tr>
<td>iSeries</td>
<td>Lotus</td>
<td>MVS/ESA</td>
</tr>
<tr>
<td>OS/390</td>
<td>OS/400</td>
<td>QMF</td>
</tr>
<tr>
<td>RACF</td>
<td>RISC System/6000</td>
<td>S/390</td>
</tr>
<tr>
<td>SP</td>
<td>VisualAge</td>
<td>WebSphere</td>
</tr>
<tr>
<td>z/OS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Java is a trademark of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries, or both.

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

Intel is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a trademark of Intel Corporation in the United States, other countries, or both.

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

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

June 2005 Edition

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

© Copyright International Business Machines Corporation 1997, 2005. All rights reserved.
This document may not be reproduced in whole or in part without the prior written permission of IBM.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Unit 3. Program Preparation ........................................... 3-1
  Unit Objectives ....................................................... 3-3
  3.1 Program Preparation ........................................... 3-5
    The Big Picture .................................................. 3-6
    Write the Source ................................................ 3-7
    The Big Picture - Precompile .................................. 3-8
    Precompile ........................................................ 3-9
    Bindfile (Linux, UNIX, and Windows) ............................ 3-11
    The Big Picture - Compile and Link .............................. 3-12
    Compile and Link ................................................ 3-13
    The Big Picture - Bind .......................................... 3-14
    Bind ............................................................... 3-16
    The Big Picture - Run .......................................... 3-17
    Run ............................................................... 3-18
  3.2 Program Preparation - Additional Considerations ............ 3-19
    Associating Load Modules and Packages ........................ 3-20
    Precompiling versus Binding (Linux, UNIX, and Windows) ... 3-22
    Adjusting Qualifier and Owner (Linux, UNIX, and Windows) .. 3-23
    Bind using QUALIFIER Option ................................... 3-24
    DB2 Optimizer .................................................... 3-25
    BIND Access Strategy ............................................ 3-26
    Implicit Rebinding .............................................. 3-27
    Explicit Rebinding .............................................. 3-29
    Privileges Required for Programming ........................... 3-30
    Protecting Resources Through Programs ......................... 3-32
  3.3 UNIX and Windows Considerations ............................... 3-33
    Precompile (Linux, UNIX, Windows) .............................. 3-34
    Precompile with Bindfile Option (Linux, UNIX, Windows) ... 3-36
    DB2 BIND REOPT Option (Linux, UNIX, Windows) ............... 3-38
  3.4 Final Summary ................................................... 3-41
    The Big Picture ................................................ 3-42
    Checkpoint ...................................................... 3-43
    Unit Summary .................................................... 3-44

Unit 4. Program Structure II .......................................... 4-1
  Unit Objectives ..................................................... 4-3
  4.1 Using Cursors .................................................. 4-5
    Processing Multiple Rows ....................................... 4-6
    Structured Programming - Sequential File Read ............... 4-7
    SELECT with FETCH - C Program ................................ 4-8
    Answer Set Materialization ..................................... 4-10
    DELETE Via A Cursor - C Program ............................... 4-12
    UPDATE Via a Cursor - C Program .............................. 4-13

© Copyright IBM Corp. 1997, 2005

Course materials may not be reproduced in whole or in part
without the prior written permission of IBM.
# 4.2 Compound SQL - DB2 for Linux, UNIX, and Windows

- **UPDATE/DELETE Via a Cursor** ........................................... 4-15
- **OPTIMIZE FOR n ROWS** ................................................. 4-16
- **FETCH FIRST k ROWS ONLY** ............................................ 4-18
- **Cursor Manipulation** .................................................. 4-19
- **WITH HOLD Cursor** .................................................... 4-20
- **Cursor Repositioning** ................................................ 4-22
- **Cursor Repositioning - C (1 of 2)** ................................. 4-23
- **Cursor Repositioning - C (2 of 2)** ................................. 4-25

## Unit 5. Recovery and Locking Concepts .................................. 5-1

### 5.1 Logical Unit of Work

- **Unit Objectives** .......................................................... 5-3
- **Unit of Recovery** ....................................................... 5-5
- **Commit and Rollback** .................................................. 5-7
- **Commit versus Rollback** .............................................. 5-8
- **Controlling Transactions with Savepoints** ...................... 5-10
- **Savepoint Restrictions** ............................................... 5-12
- **Savepoint Considerations** ............................................ 5-13
- **Data Recovery (1 of 2)** .............................................. 5-15
- **Data Recovery (2 of 2)** .............................................. 5-16

### 5.2 Locking

- **Attributes of Locks** .................................................. 5-19
- **Locking - Size** .......................................................... 5-20
- **Lock Modes** .............................................................. 5-21
- **Table Lock versus Row or Page Locking** ......................... 5-23
- **Isolation Levels - Duration** ........................................ 5-26
- **WITH RR/RS/CS/UR** ................................................... 5-28
- **U-Lock Usage** ............................................................ 5-31

### 5.3 Connection Considerations (Linux, UNIX, and Windows)

- **Prep to Bind (Linux, UNIX, Windows)** ............................ 5-33
- **CONNECT (Type 1) and CONNECT (Type 2) (Linux, UNIX, Windows)** ........................................ 5-35
- **Remote Unit of Work - Type 1 Connect (Linux, UNIX, Windows)** ........................................... 5-36
- **Distributed Unit of Work - Type 2 Connect (Linux, UNIX, Windows)** ..................................... 5-37
- **Releasing a Connection (Linux, UNIX, Windows)** ............ 5-38
- **Multisite Update Wizard (Linux, UNIX, Windows)** ........... 5-39
- **Connection-Oriented Precompile Options (Linux, UNIX, Windows)** ................................... 5-40
- **Syncpoint Considerations (Linux, UNIX, Windows)** .......... 5-41
- **Multisite Bind (Linux, UNIX, Windows)** ........................ 5-42
- **Checkpoint** ............................................................... 5-43
- **Unit Summary** ........................................................... 5-44
## Unit 6. Dynamic SQL Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Objectives</td>
<td>6-1</td>
</tr>
<tr>
<td>6.1 Embedded Dynamic SQL</td>
<td>6-2</td>
</tr>
<tr>
<td>Types of Dynamic SQL Statements</td>
<td>6-3</td>
</tr>
<tr>
<td>Complete Non-SELECT Statement</td>
<td>6-4</td>
</tr>
<tr>
<td>Complete Non-SELECT Statement - C</td>
<td>6-6</td>
</tr>
<tr>
<td>Parameterized Non-SELECT Statement</td>
<td>6-7</td>
</tr>
<tr>
<td>Parameterized Non-SELECT Statement - C</td>
<td>6-9</td>
</tr>
<tr>
<td>SELECT Statement with Fixed List - C</td>
<td>6-10</td>
</tr>
<tr>
<td>SELECT Statement with Varying List</td>
<td>6-11</td>
</tr>
<tr>
<td>Unit Summary</td>
<td>6-12</td>
</tr>
<tr>
<td></td>
<td>6-13</td>
</tr>
</tbody>
</table>

## Unit 7. Managing Test Data.

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Objectives</td>
<td>7-1</td>
</tr>
<tr>
<td>7.1 Managing Test Data</td>
<td>7-2</td>
</tr>
<tr>
<td>Single Row INSERT</td>
<td>7-3</td>
</tr>
<tr>
<td>Multiple Row INSERT</td>
<td>7-4</td>
</tr>
<tr>
<td>Data Movement Utilities</td>
<td>7-5</td>
</tr>
<tr>
<td>Importing Data (Linux, UNIX, Windows)</td>
<td>7-6</td>
</tr>
<tr>
<td>Import Utility Example (Linux, UNIX, Windows)</td>
<td>7-7</td>
</tr>
<tr>
<td>Data Utilities - RUNSTATS</td>
<td>7-9</td>
</tr>
<tr>
<td>REORG</td>
<td>7-11</td>
</tr>
<tr>
<td>Unit Summary</td>
<td>7-12</td>
</tr>
<tr>
<td></td>
<td>7-13</td>
</tr>
</tbody>
</table>

## Unit 8. Performance Considerations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Objectives</td>
<td>8-1</td>
</tr>
<tr>
<td>8.1 Index Concepts</td>
<td>8-3</td>
</tr>
<tr>
<td>DB2 Optimizer</td>
<td>8-5</td>
</tr>
<tr>
<td>Index Concepts</td>
<td>8-6</td>
</tr>
<tr>
<td>Typical Access Paths</td>
<td>8-7</td>
</tr>
<tr>
<td>Relation Scan</td>
<td>8-8</td>
</tr>
<tr>
<td>Matching Index Scan / Range Delimiting</td>
<td>8-9</td>
</tr>
<tr>
<td>Index-Only Access</td>
<td>8-10</td>
</tr>
<tr>
<td>Non-Matching Index Scan</td>
<td>8-11</td>
</tr>
<tr>
<td>Access Path Exercise</td>
<td>8-12</td>
</tr>
<tr>
<td>Clustered and Non-Clustered Indexes</td>
<td>8-13</td>
</tr>
<tr>
<td>8.2 General Guidelines</td>
<td>8-15</td>
</tr>
<tr>
<td>General Guidelines - Avoid Unnecessary Conversions</td>
<td>8-17</td>
</tr>
<tr>
<td>General Guidelines - Select What You Need</td>
<td>8-18</td>
</tr>
<tr>
<td>General Guidelines - Minimize DB2 Sorts</td>
<td>8-19</td>
</tr>
<tr>
<td>General Guidelines - Use Views</td>
<td>8-20</td>
</tr>
<tr>
<td>General Guidelines - Examine Lock Strategy</td>
<td>8-21</td>
</tr>
<tr>
<td>General Guidelines - Dynamic versus Static SQL</td>
<td>8-22</td>
</tr>
<tr>
<td>General Guidelines - Scalar Functions</td>
<td>8-23</td>
</tr>
<tr>
<td>General Guidelines - Statements</td>
<td>8-24</td>
</tr>
<tr>
<td>General Guidelines - Exploit Blocking (1 of 2)</td>
<td>8-25</td>
</tr>
<tr>
<td>General Guidelines - Exploit Blocking (2 of 2)</td>
<td>8-26</td>
</tr>
</tbody>
</table>
Trademarks

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

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

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

AIX®, AT®, CICS®,
CICS/6000®, DB2®, DB2 Connect™
DB2 Extenders™, DB2 Universal Database™
DRDA®, IMS™, Distributed Relational Database Architecture™
Lotus®, MVS™, iSeries™
OS/390®, OS/400®, MVS/ESA™
RACF®, RETAIN®, QMF™
S/390®, S/390®, RISC System/6000®
WebSphere®, z/OS®, VisualAge®
1-2-3®

Java is a trademark of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries, or both.

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

Intel is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a trademark of Intel Corporation in the United States, other countries, or both.

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

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

DB2 Universal Database Programming Workshop for Linux, UNIX, and Windows

Duration: 3 days

Purpose

This course enables you to acquire the skills necessary to produce application programs that manipulate DB2 databases. Emphasis is on embedding Structured Query Language (SQL) statements and preparing programs for execution.

Audience

Application programmers who need to write embedded SQL programs in COBOL or PL/I (on z/OS) or in C language (on Linux, UNIX, and Windows).

Prerequisites

Prior to attending this course, the student should already have experience with one of the supported programming languages, COBOL or PL/I for z/OS, or C language for Linux, UNIX, and Windows. Students are also expected to already be able to construct and use SQL statements.

Objectives

After completing this course, you should be able to:

- Incorporate static SQL statements in an application program
- Prepare the program for execution
- Validate execution results are correct
- Produce code to support multiple rows being returned from the database manager using cursors
- Identify considerations regarding units of work, concurrency, and restart of programs
- Identify differences between static and dynamic SQL
- Provide test data for applications
• Discuss program and DB2 options relative to performance of static SQL
Agenda

Day 1
Welcome
Unit 1 - DB2 Concepts
Lab - DDL
Unit 2 - Program Structure - Part 1
Lab - Program Structure I

Day 2
Unit 3 - Program Preparation
Lab - Program Preparation
Unit 4 - Program Structure - Part 2
Lab - Program Structure II
Unit 5 - Recovery and Locking Concepts
Lab - Cursor Restart

Day 3
Unit 6 - Dynamic SQL Introduction
Lab - Dynamic SQL
Unit 7 - Managing Test Data
Lab - Loading Data
Unit 8 - Performance Considerations
Lab - Performance
Unit 1. DB2 Concepts

What This Unit Is About

The DB2 family includes products on multiple platforms. The DB2 Universal Database products for client/server application development require certain product components.

While many application programs will be written using static SQL, there are other alternatives, including stored procedures and Call Level Interface.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Identify DB2 family products
• Explain DB2 workstation component functions
• Identify DB2 objects
• Identify the key differences between static SQL and other application alternatives for accessing DB2 data

How You Will Check Your Progress

Accountability:

• Machine lab

References

DB2 UDB for Linux, UNIX, and Windows

SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849  Call Level Interface Guide and Reference, Volume 1
SC09-4850  Call Level Interface Guide and Reference, Volume 2
SC09-4828  Command Reference
SC09-4844  SQL Reference - Volume 1
SC09-4845  SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

- Identify DB2 family products
- Explain DB2 component functions
- Identify DB2 objects
- Identify the key differences between static SQL and other application alternatives for accessing DB2 data

Notes:
1.1 DB2 Family
Notes:

DB2 UDB is a relational database management system (RDBMS) that enables users to create, update, and control relational databases using Structured Query Language (SQL). Designed to meet the information needs of small and large businesses alike, it is available on a variety of platforms, including large systems such as z/OS, OS/390, VM, and VSE; mid-sized systems such as OS/400, AIX, Linux, HP-UX, and Solaris; and single or LAN-based systems such as Windows 2000, Windows NT, Windows XP, and Windows .NET. Clients running V6 or V7 can access DB2 UDB V8 servers.

All members of the DB2 UDB family have similar external architecture and use many of the same key algorithms, thereby ensuring portability for customers across platforms.
Notes:

- DB2 provides a full-function, robust, relational database management system (RDBMS) along with a set of related products designed for the universal database platforms. DB2 brings to these platforms the support of the IBM SQL technology that is available on all members of the DB2 family.

- A Communication Support product must be installed on the server to enable remote client access.

- Distributed Relational Database Architecture (DRDA). Application Requesters represent DRDA requests coming from, for example, z/OS, VM, and OS/400. No special product or enablement, other than support of remote clients, is required on the DB2 Server to support DRDA Application Requestors.

- The DB2 Run-Time Client is a lightweight client that provides the functionality required for an application to access DB2 Universal Database servers and DB2 Connect servers. Functionality includes communications support protocol support and support for application interfaces, such as JDBC, SQLJ, ODBC, CLI, and OLE DB.
- The DB2 Application Development Client is a collection of graphical and non-graphical tools and components for developing character-based, multimedia, and object-oriented applications. Special features include the Development Center and sample applications for all supported programming languages. The Application Development Client also includes the tools and components provided as part of the DB2 Administration Client product. The Application Development Client comes with the Developer’s Editions.

The Personal Developer’s Edition contains several CD-ROMs with all the code that you need to develop and test your applications on Windows and Linux operating environments.

DB2 Universal Developer’s Edition provides the application developer with the tools needed to develop DB2 client/server, mobile, and Web-ready applications across all of the supported environments for DB2 Universal Database and DB2 Connect products.

DB2 Personal Developer’s Edition is available to you at no-charge. This product provides the DB2 Universal Database and DB2 Connect Personal Editions media for Windows XP, NT/2000, and Linux. These are licensed for your use for development, evaluation, testing, and demonstrations. It can be downloaded from the Web, at no charge, from:

http://www.software.ibm.com/db2/

Alternatively, the DB2 Personal Developer’s Edition can be ordered as a program package, complete with CD-ROMs, with a shipping and handling charge.

- The DB2 Administration Client provides the ability for workstations from a variety of platforms to access and administer DB2 databases. The Administration Client has all the features of DB2 Run-Time Client and also includes all the DB2 administration tools and support for Thin Clients.

- DB2 Connect provides DRDA application requestor (AR) function. With Communications Support, DB2 Connect enables applications running on DB2 UDB for Linux, UNIX, and Windows platforms to access and update data on DB2 for z/OS, DB2 for OS/390, DB2 for VM and VSE, DB2 for iSeries, and other database management systems that are DRDA compliant. The DB2 Connect product incorporates the following utilities and functions of DB2:
  - Command Line Processor (CLP) that allows you to issue SQL statements to access host platforms.
  - Database system monitor utility which is an aid in problem determination area.
  - The bind command, which allows you to bind packages to the host platforms.
  - The IMPORT and EXPORT utilities, which allow you to move data between host platforms and the DB2 Connect workstations.
  - The ability to catalog databases for the host platform.
Notes:

- The DB2 Universal Database can run applications locally and can be accessed by applications running on remote clients that have the DB2 Run-Time client installed. The DB2 Run-Time client provides a run-time environment that enables client applications to access one or more remote databases. Software on the DB2 Client Pack CD-ROM provides support for clients on many platforms including: Windows, AIX, HP-UX, Linux, and Solaris. If you want to access data across the LAN from a database server, install Workgroup Server Edition or Enterprise Server Edition on the database server. DB2 Run-Time client must be installed on every client workstation that wants to access the database server. The supported LAN communication protocols are: Named Pipes, NetBIOS, or TCP/IP.

- DB2 Personal Developer’s Edition provides the DB2 Universal Database Personal Edition and DB2 Connect Personal Edition media for Windows and Linux. These CD-ROMs are provided to you for testing your applications only. If you need to install and use a database, you have to get a valid license by purchasing the DB2 Universal Database product.
DB2 Personal Developer’s Edition can be downloaded from the Web, at no charge, from:

http://www.ibm.com/db2

Alternatively, the DB2 Personal Developer’s Edition can be ordered as a program package, complete with CD-ROMs.

- The DB2 Universal Developer’s Edition contains CD-ROMs for all the operating systems supported by DB2, and include the following:
  - DB2 Connect Personal Edition and DB2 Connect Enterprise Edition
  - Administration clients for all platforms. These clients contain tools for administering databases, such as the Control Center and the Event Analyzer. These clients also allow you to run applications on any system.
  - Application development clients for all platforms. These clients have application development tools, sample programs, and header files. Each DB2 AD client includes everything you need to develop your applications.
  - Run-time clients for all platforms. An application can be run from a run-time client on any system. The run-time client does not have some of the features of the administration client, such as the DB2 Control Center and Event Analyzer, and so takes up less space.
  - DB2 Extenders
  - DB2 XML Extender
  - VisualAge for Java, Professional Edition (Windows)
  - Websphere Studio
  - Websphere Application Server, Standard Edition
  - Query Management Facility (try and buy)

In addition, for both Developer’s Editions you get copies of other software that you may find useful for developing applications. This software may vary from time to time, and is accompanied by license agreements for use.
1.2 Access to Data
Accessing DB2

Notes:

The main DB2 software components are:

- **Database Engine** - Provides the base functions of DB2. It manages and controls all access to data. It generates packages (stored access paths to the data), provides transaction management, ensures data integrity and data protection, and provides concurrency control.

The basic elements of the database engine are database objects, system catalogs, directories, and configuration files. All access to data takes place through the SQL interface.

- **SQL Interfaces** - Command line or file input interfaces to execute SQL.
  - **SPUFI** - SQL Processor Using File Input. Tool on z/OS for entering SQL via file input.
  - **QMF** - Query Management Facility. z/OS or workstation tool to allow interactive SQL entry and processing.
- **Command Line Processor (CLP)** - CLP is used to dynamically execute SQL requests and/or DB2 commands. CLP may be used to access local workstation databases, remote workstation databases, or remote Distributed Relational Database Architecture Application Server (DRDA AS) databases via DB2 Connect Personal Edition or DB2 Connect Enterprise Edition.

**Administration Interfaces** - A collection of GUI tools that help to manage and administer databases, including:
- Control Center - For configuration, backup and recovery, directory management.
- Command Editor - For issuing commands and creating command scripts.
- Task Center - For issuing SQL and creating SQL scripts.
- Event Analyzer - For analyzing event information.
- Journal - For analyzing the status of submitted jobs.
- Health Center - For identifying problem areas.
- Tools Settings - For setting up replications, setting termination characters, and setting up Alert Center options.

**Application Interfaces** - Access the database by using:
- Embedded SQL
- Call Level Interface
  - ODBC
- Java
  - Java Database Connectivity (JDBC)
  - Embedded SQL for Java (SQLJ)
- Application Programming Interfaces (APIs) Linux/UNIX/Windows only.
- Data Objects

**External tools** which can provide a variety of additional function.
Notes:

- The left side of the above diagram represents the traditional dependence of the end user on data processing (DP) personnel.

- As an alternative, end users could use a query tool to access the database directly.

- Lotus Approach or QMF are examples of products that allow authorized end users to access production data directly to make business decisions. The dependence on DP personnel is reduced dramatically.

- The extent to which end users can use query tools to access data directly is dependent on the complexity and sensitivity of the data and the knowledge of the end user. The programmer or the information center consultant may need to support the use of query tools to some extent. However, writing and maintaining applications using query tools may be less time consuming than developing and testing traditional application programs.
**Physical Environment**

**Notes:**

(Linux, UNIX, and Windows):

- Each instance is a unique database manager environment (in UNIX, a virtual copy of the code using symbolic links to the physical code is created). This allows for separate database manager environments (configuration parameters, authentication). In UNIX, this allows a separate physical copy of a new release of code to be installed and instances to be created from the new code.

- **DB2 configuration files** contain parameter values that define the resources allocated to DB2 and individual databases. There are two types of configuration files: database manager configuration file, for the DB2 instance as a whole and the database configuration file, for each individual database.

- The database manager configuration file is created when an instance of DB2 is created, and affects the system resources allocated to DB2 for an individual instance. Its parameters have global applicability, independent of any one database stored in the system.
• A **database configuration file** for an individual database is created when the database is created. There is one configuration file per database. The parameters in this configuration file specify the amount of resources to be allocated to that database. Many of these parameters can be changed to improve the performance or increase capacity. Different changes may be required depending on the type of activity in that specific database.

• DB2 creates and maintains a set of system catalog tables for each database. These tables contain information such as descriptions of tables, views, and packages. A set of read-only views for the system catalog tables is created in the SYSCAT schema. A set of updatable catalog views is created in the SYSSTAT schema.

• Logging transaction changes is also performed at the database level.
Connect to Database

Notes:

- When running an application in a z/OS environment, the application normally inherits a connection that has been acquired through the mechanism that invokes the application, for example, via the Job Control Language (JCL) or Customer Information Control System (CICS) Resource Control Table (RCT).

- In an application running in the UNIX or Windows environment, the CONNECT TO statement connects an application process to a database.

- If you do not want to use the default userid and password to perform the connection, the userid and password can be explicitly provided on the connect statement by using the format:

  `CONNECT TO dbname USER userid USING password`
Notes:

- Each database will have multiple table spaces and index spaces. Table spaces and index spaces are used to manage placement of data on physical storage. Table spaces contain table data and index spaces contain the index key and RID.

- A relational database presents data as a collection of tables. A table consists of a defined number of columns and any number of rows.

- Each table may have multiple indexes. Indexes may provide a faster way to access table data.

- Each table may have multiple views. Views may be associated with more than one base table.
1.3 SQL Statement Types
**Notes:**

- DB2 tables are defined with the CREATE table statement.
- Additional columns can be added to an existing table with the ALTER statement. Other functions can be completed via the ALTER statement as well.
- DB2 tables are deleted with the DROP statement.
Notes:

- SELECT retrieves information from one or more columns of data in a table.
- UPDATE changes values in one or more columns.
- INSERT adds one or more rows.
- DELETE removes one or more rows.
- All of the statements act on a set of rows, ranging from no rows to all rows in the table.
Notes:

- GRANT authorizes a user for some privilege.
- REVOKE removes a privilege from a user.
- CONNECT connects an application process to an application server.
- COMMIT terminates a unit of work and commits database changes that were made by that unit of work.
- ROLLBACK terminates a unit of work and backs out the database changes that were made by that unit of work.
1.4 Application Alternatives
Application Alternatives

- Static SQL
- Dynamic SQL
- Call Level Interface
  - ODBC
- Java Development
  - JDBC and SQLJ
- Stored Procedures
- User-Defined Functions

Notes:

- This course will concentrate on static SQL, but the next few pages will also present the concepts of other alternatives.
**Static and Dynamic SQL**

**Notes:**

- DB2 UDB supports both dynamic and static SQL.
- Dynamic SQL is where the application program provides the SQL statement to DB2 UDB at execution time. This means that DB2 UDB determines the strategy of how it accesses the data at execution time.
- Static SQL is where the application program provides the SQL statement to DB2 UDB at application development time. This means that DB2 UDB can determine the strategy of how it accesses the data at program development time, and store the strategy for later execution.
- When an SQL statement is prepared, whether for dynamic or static SQL, DB2 access the database catalog definitions and statistics and creates a strategy, or access plan. The access plan contains the information that the database manager needs to process the SQL statements against the database.
• For dynamic SQL, this strategy is kept in memory in the package cache, where it can be used for a later execution of the same statement. However, when the database shuts down, and dynamic SQL strategies in the package cache cease to exist.

• For static SQL, while the program is being developed, it is precompiled and bound to the database. The process of binding prepares the strategy and stores it in the database catalog as a package. The package is persistent and can be used in subsequent executions of the associated program, even executions after the database shuts down and then comes back up.
Call Level Interface

**Notes:**

- DB2 Call Level Interface (CLI) is a callable SQL programming interface for the C/C++ programming language.

- A callable SQL interface is an application program interface (API) for database access, which uses function calls to invoke dynamic SQL. It is an alternative to embedded dynamic SQL. The important difference between embedded dynamic SQL and DB2 CLI lies in the invocation mechanism for SQL.

- An application that uses an embedded SQL interface requires a precompiler to convert the SQL statements into code, which is then compiled, bound to the database, and executed. A DB2 CLI application does not require precompilation or binding, but instead uses a standard set of functions to execute SQL statements and related services at run time.

- This difference is important because, traditionally, precompilers have been specific to a database product, which effectively ties your applications to that product. DB2 CLI enables you to write portable applications that are independent of any particular database product. This independence means a DB2 CLI application does not have to
be re-precompiled or rebound to access different database products, but rather selects
the appropriate database modules from link libraries at link time or dynamically at run
time.

- DB2 CLI eliminates the need for the application controlled global data areas, such as
  SQLCA and SQLDA, used in embedded SQL applications. Instead, DB2 CLI allocates
  and controls the necessary data structures, and provides a handle for the application to
  reference them. This moves the responsibility of managing these data structures from
  the application to DB2 CLI.

Reference:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC09-4849</td>
<td>DB2 Call Level Interface Guide and Reference, Volume 1</td>
</tr>
<tr>
<td>SC09-4850</td>
<td>DB2 Call Level Interface Guide and Reference, Volume 2</td>
</tr>
</tbody>
</table>
**Notes:**

The X/Open Company and the SQL Access Group jointly developed a specification for a callable SQL interface referred to as the *X/Open Call Level Interface*. The goal of this interface is to increase the portability of applications by enabling them to become independent of any one database vendor’s programming interface. Most of the X/Open Call Level Interface specification has been accepted as part of the ISO Call Level Interface International Standard (ISO/IEC 9075-3:1995 SQL/CLI).

Microsoft developed a callable SQL interface called Open Database Connectivity (ODBC) for Microsoft operating systems based on a preliminary draft of X/Open CLI.

The ODBC specification also includes an operating environment where database-specific ODBC drivers are dynamically loaded at run time by a driver manager based on the data source (database name) provided on the connect request. The application is linked directly to a single driver manager library rather than to each DBMS’s library. The driver manager mediates the application’s function calls at run time and ensures that they are directed to the appropriate DBMS-specific ODBC driver. Since the ODBC driver manager knows only about the ODBC-specific functions, DBMS-specific functions cannot be accessed in an
ODBC environment. DBMS-specific dynamic SQL statements are supported via a mechanism called an escape clause.

ODBC is not limited to Microsoft operating systems; other implementations are available on various platforms.

The DB2 CLI load library can be loaded as an ODBC driver by an ODBC driver manager. For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from http://www.microsoft.com/data/. For non-Windows platforms, the ODBC SDK is provided by other vendors. When developing ODBC applications that may connect to DB2 servers, use the IBM DB2 UDB Call Level Interface Guide and Reference (for information on DB2-specific extensions and diagnostic information), in conjunction with the ODBC Programmer’s Reference and SDK Guide available from Microsoft.

Applications written directly to DB2 CLI link directly to the DB2 CLI load library. DB2 CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2-specific functions.

The following DB2 features are available to both ODBC and DB2 CLI applications:

- Double-byte (graphic) data types
- Stored procedures
- Distributed Unit of Work (DUOW), two-phase commit
- Compound SQL
- User-defined types (UDT)
- User-defined functions (UDF)

Reference:

SC09-4849  DB2 Call Level Interface Guide and Reference, Volume 1
SC09-4850  DB2 Call Level Interface Guide and Reference, Volume 2
SC18-7425  DB2 ODBC Guide and Reference
OLE DB (Windows Only)

- OLE DB Consumer Support - ability to access OLE DB Providers
- OLE DB Provider Support - ability to provide data to OLE DB Consumers

Notes:

Another standard interface option that DB2 UDB supports is Microsoft OLE DB. Support for OLE DB is provided only on Windows platforms.

Microsoft OLE DB is a set of OLE/COM interfaces that provide applications with uniform access to data stored in diverse information sources. The OLE DB architecture defines OLE DB consumers and providers. An OLE DB consumer is any system or application that uses OLE DB interfaces; an OLE DB provider is a component that exposes OLE DB interfaces.

On Windows, DB2 UDB can act as an OLE DB consumer. It then directly accesses rowset data, or composes a command text in a language native to the provider and retrieves rowset data. The OLE DB consumer in DB2 UDB retrieves data from any OLE DB provider. The user registers a table function of language type OLE DB, which refers to an OLE DB provider, and the relevant rowset.
The DB2 OLE DB consumer offers the following features:

- Access to OLE DB sources, including MS Access and MS SQL Server
- Read-only support, including the ability to move data from OLE tables into DB2 tables, and the ability to merge OLE tables with DB2 tables
- SQL language support, including retrieving and manipulating data

On Windows, DB2 UDB can act as an OLE DB Provider. This gives OLE DB-based applications the ability to extract or query DB2 using the native OLE interface. The OLE DB provider is named IBMDADB2. It enables OLE DB consumers to access data on a DB2 UDB server. If DB2 Connect is installed, these OLE DB consumers can also access data on host DBMSs such as DB2 for z/OS, DB2 for VM/VSE, or DB2 for OS/400.

The OLE DB Provider function in DB2 offers the following features:

- Level 0 support of the OLE DB provider specification, including some additional Level 1 interfaces
- A free-threaded provider implementation enabling the application to create components in one thread and use those components in any other thread
- An Error Lookup Service that returns DB2 error messages
Microsoft .NET Support

- Support for .NET Framework 1.1
- Support for Visual Studio .NET 2003
- Deeper integration with Visual Studio:
  - DB2 administrator add-in
  - SQL debugger
  - Functional and usability improvements in existing add-ins
- Integrated support of CLR (Common Language Runtime)
  - Create DB2 stored procedures and UDFs in any CLR language (C#, VB, VC++, etc.)
- DB2 .NET Provider enhancements
  - Increased performance / increased functionality

Notes:
Here are some of the new things you are able to do using the features of the DB2 Development Add-In for Visual Studio .NET:

- Discover and work with remote DB2 data connections using server host name/IP, port, and database name
- Create new tables, views, indexes, and triggers using functionally rich scripting wizards
- Extend the SQL IntelliSense support beyond the SQL editor to include multi-line edit controls in your wizards and dialogs
- Retrieve data for tables and views using data grids in read/write mode
- Access table and view details including columns, indexes, and triggers
- Generate create scripts for tables, views, functions, and procedures
- Develop and deploy DB2 CLR procedures using C# and VB.NET.
- Manage the deployment of CLR assemblies to local and remote DB2 for Windows servers
• Create re-usable data adapters directly under your data connections, and share them with other developers, across projects and form designers
• Discover or manually define the shape of your data adapter result sets
• Preview results from your data adapters using data grids in read/write mode
• Automatically generate Web methods and Web services for your data adapters or their commands
• Customize your add-in using an extended set of DB2 tools settings
• Add collapsible regions to your DB2 database script files
Java Applications

There might be cases where you need an application that can access DB2 databases across the Internet. Using the Java programming language, you can develop applications and applets that access and manipulate data in DB2 databases.

DB2 provides support for the Sun Microsystem's Java Database Connectivity (JDBC) API. The support is provided through a DB2 JDBC driver (JDBC 3.0 compliance) that comes with DB2. The JDBC API, which is similar to ODBC APIs, provides a standard way to access databases from Java code. The Java code passes SQL statements as function arguments to the DB2 JDBC driver. The driver handles the JDBC API calls from the client Java code.

DB2's Java enablement has three independent components:
- Support for client applications and applets written in Java using JDBC to access DB2
- Precompile and binding support for client applications and applets written in Java using SQLJ to access DB2
- Support for Java UDFs and stored procedures on the server
The graphic illustrates how a DB2 JDBC application works. You can think of a DB2 JDBC application as a DB2 CLI application, only you write it using the Java language. Calls to JDBC are translated to calls to DB2 CLI through Java native methods. JDBC requests flow from the DB2 client through DB2 CLI to the DB2 server.

SQLJ applications use this JDBC support, and in addition require the SQLJ run-time classes to authenticate and execute any SQL packages that were bound to the database at the precompiling and binding stage.
Notes:

- Stored procedures allow an application running on a client to call a procedure stored on a database server. This stored procedure executes and accesses the database locally and returns information to the client application.

- To use this technique, an application must be written in two separate procedures. The calling procedure is contained in a client application and executes on the client. The stored procedure executes at the location of the database on the database server.

- Applications that use stored procedures have the following advantages:
  - Reduced network traffic
  - Improved performance of server intensive work
  - Access to features that exist only on the database server
  - Reduced maintenance cost for database code changes

Stored Procedures can be written in various languages for example, COBOL, PL/1, C, C++, Java, and Assembler.
Development Center

- Development of DB2 stored procedures made simple
  - Supports development and debugging of Java stored procedures on all DB2 UDB Server platforms
  - Supports static (SQLJ), dynamic (JDBC) SQL and SQL Procedure Language
  - GUI development environment
  - Users can focus on the logic of their SPs
    - Wizards simplify many tasks
    - Tool automates build process
  - Integrated with MS Visual Studio

- Benefits:
  - Ease of development
  - Portability across DB2 family

Notes:

The Development Center assists you in creating a stored procedure that runs on a database server. The client application must be written separately.

The Development Center is a graphical application that supports the rapid development of DB2 stored procedures. Using Development Center, you can perform the following tasks:

- Create new stored procedures
- Build stored procedures on local and remote DB2 servers
- Modify and rebuild existing stored procedures
- Test and debug the execution of installed stored procedures.
Development Center Example

Notes:

The Development Center provides an easy-to-use interface for developing routines such as stored procedures (Java or SQL PL = Procedure language) and user-defined functions (UDFs). A set of wizards makes it easy to perform your development tasks. The Development Center provides a single development environment that supports the entire DB2 family ranging from the workstation to z/OS. No C compiler is needed for SQL procedures.

The Development Center provides the following functions:

- Rapid, iterative development of Java or SQL stored procedures, user-defined functions, and types.
- Multiple concurrent projects can be under development at same time.
- Project view of development objects; server view of live database objects.
User-Defined Functions

SELECT EURO(SALARY) FROM EMP

Notes:

- A user-defined function (UDF) can be written for use in performing operations within an SQL statement that returns a single scalar value. UDFs are useful for such things as transforming data values, performing calculations on one or more data values, or extracting parts of a value (such as extracting parts of a large object). They are also quite flexible as they can be written in a high-level language (C, C++, Java, COBOL, PL/I or Assembler).

References:

- SC09-4825 Application Development Guide: Building and Running Applications
- SC09-4826 Application Development Guide: Programming Client Applications
- SC09-4827 Application Development Guide: Programming Server Applications
1.5 Command Line Processor (CLP) (Linux, UNIX, and Windows)
Notes:

- These options may be entered after the `db2`, and before the command or statement. If the command or statement will not fit on one line, use the \ (backslash) character for continuation.
Starting a CLP Session

\textit{non-interactive mode}

\textbf{db2} connect to eddb
\textbf{db2} "select * from syscat.tables" | more
(double quotes may be required)

\textit{interactive mode}

\textbf{db2} select * from syscat.tables
\textbf{db2=>} connect to eddb
\textbf{db2=>} select * from syscat.tables

\textbf{Notes:}

- Prefix all CLP commands/requests with \textbf{db2} or use CLP in interactive mode by typing \textbf{db2} and then pressing the Enter key. In the interactive mode, the user can type CLP commands without prefixing them by DB2.

- In UNIX, when using CLP without being in interactive mode, remember to put quotes around special characters when issuing SQL statements and commands. A simple habit which will avoid misinterpretation of special characters would be to surround the entire statement or command with double quotes, for example:

\textbf{db2} select * from atable

- Interactive mode does not allow you to do operating system commands directly at the interactive CLP command. To execute operating system commands without quitting interactive mode, issue !\texttt{<operating-system-command>}.

- If a command that you are entering exceeds the limit allowed at the command prompt, use a \texttt{\textbackslash} (backslash) as the line continuation character. If you want a blank space between the last character on the current line and the first character on the next line,
don't forget to code a blank space before the backslash. You may wish to use the CLP Flag -t instead of the \ (backslash). It says that the CLP statement terminates with a ; (semicolon).

- In its output, CLP represents a SQL NULL value as a hyphen (-). If the column is numeric, the hyphen is placed at the right of the column. If the column is not numeric, the hyphen is at the left.
Notes:

Use the Command Editor to execute DB2 commands and SQL statements; to execute OS/390, or z/OS commands; to work with command scripts; and to view a graphical representation of the access plan for explained SQL statements.

On the command page, you can perform the following actions:

- Execute an SQL statement or DB2 CLP command. You do not need to precede the command by DB2.
- Run an existing script, by clicking the Execute icon (at the left of the tool bar).
- Execute the commands in sequence.
- Create and save a script. You can optionally store a saved script in the Task Center, where you can schedule the script to run at a specific time.
- Run and/or modify an existing script.

On the Query Results page, you can see the results of the queries. You can also save the results or edit the contents of the table.
**QUIT versus TERMINATE versus CONNECT RESET**

<table>
<thead>
<tr>
<th>CLP COMMAND</th>
<th>Terminate CLP Back-end Process</th>
<th>Disconnect database Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>quit</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>terminate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>connect reset</td>
<td>No</td>
<td>Yes if CONNECT=1 (RUOW)</td>
</tr>
</tbody>
</table>

**Notes:**

- To connect to a local or remote database, specify `db2 connect to dbname` where `dbname` maps to the alias name specified in the System Database Directory. After the `connect` is issued, all SQL requests are executed against the database to which you are connected. `db2 connect reset` terminates the connection, and a subsequent SQL statement will cause connection to the default database, if it is defined. **The default database is defined in the registry variable DB2DBDFT.**

- `terminate` issues a disconnect and also terminates the CLP back-end process. This would be necessary in order for any registry variables to be reset.

- `quit` ends the input mode and returns the user to the command prompt, but quit does not terminate CLP nor the database connection.

- `connect reset` disconnects the current connection and may connect to the default database. Connecting to the default database depends on other options beyond the scope of this discussion.

- To end the CLP and the database connection, issue the terminate command.
1.6 Optional Topic - DB2 Objects
Tablespace and Database Objects

- Create sequence:
  - Create database
  - Create tablespace
  - Create table

**Notes:**

- Before a table can be created, its database and table space must exist. It will probably be the job of the DB2 system administrator or database administrator to create those objects.
- When setting up your test data, you might have to create tables.
Create Table

CREATE TABLE EMP
  (EMPNO CHAR(6) NOT NULL,
   FIRSTNAME VARCHAR(12) NOT NULL,
   MIDINIT CHAR(1) NOT NULL WITH DEFAULT,
   LASTNAME VARCHAR(15) NOT NULL,
   WORKDEPT CHAR(3),
   PHONENO CHAR(4),
   HIREDATE DATE,
   JOB CHAR(8),
   EDLEVEL SMALLINT,
   SEX CHAR(1),
   BIRTHDATE DATE,
   SALARY DECIMAL(9,2),
   BONUS DECIMAL(9,2),
   COMM DECIMAL(9,2))
IN TSX;

Notes:

- The IN keyword places the table into a table space of a database.
  - On z/OS, this would specify the database and table space name.
  - On Linux, UNIX, and Windows, this would specify the table space name. Prior to
    running the statement, you would need to establish a connection to a database with
    a CONNECT TO statement.

- Each table consists of a number of columns. Each column has two fundamental
  characteristics:
    - Data type
    - Null attribute
Data Types - Character and Numeric

- **CHAR(n)** - Fixed length string between 1 and 254 bytes
- **VARCHAR(n)** - Variable length string
- **LONG VARCHAR** - Variable length string
- **GRAPHIC(n)** - Fixed length graphic string (from 1 to 127)
- **VARGRAPHIC(n)** - Variable length graphic string
- **LONG VARGRAPHIC** - Variable length graphic string
- **SMALLINT** - 2-byte integer, values between + and - 32 KB
- **INTEGER** - 4-byte integer, values between + and - 2 GB
- **DECIMAL(p,s)** - Decimal number, max 31 digits
- **FLOAT/REAL/DOUBLE** - Fullword (4 bytes) or double word (8 bytes)

**Notes:**

- Character data can be stored in fixed length character columns (CHAR(n) or CHARACTER(n)), variable length character columns (VARCHAR(n)), or long variable length character columns (LONG VARCHAR).
- Double Byte Character Set (DBCS) data can be stored in graphic column types.
- Numeric data can be stored in whole number columns (SMALLINT, INTEGER, INT) or column types that allow for fractional components (DECIMAL(p,s), DEC(p,s), FLOAT, REAL, DOUBLE).
## Data Types - DATE/TIME Data

<table>
<thead>
<tr>
<th></th>
<th>Internal Representation</th>
<th>External Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>4 bytes</td>
<td>'mm/dd/yyyy'</td>
</tr>
<tr>
<td>USA</td>
<td>Packed decimal without sign</td>
<td>'dd.mm.yyyy'</td>
</tr>
<tr>
<td>EUR</td>
<td>yyyymmdd</td>
<td>'yyyy-mm-dd'</td>
</tr>
<tr>
<td>ISO</td>
<td></td>
<td>'yyyy-mm-dd'</td>
</tr>
<tr>
<td>JIS</td>
<td></td>
<td>10 to 254 bytes</td>
</tr>
<tr>
<td>LOCAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| TIME   | 3 bytes                 | 'hh:mm xM' where x=A or P |
| USA    | Packed decimal without sign | 'hh.mm.ss'             |
| EUR    | hhmmss                  | 'hh:mm:ss'             |
| ISO    |                          | 'hh:mm:ss'             |
| JIS    |                          | 8 to 254 bytes         |
| LOCAL  |                          |                         |

| TIMESTAMP | 10 bytes | 'yyyy-mm-dd-hh.mm.ss.nnnnnn' | 26 bytes |
| LOCAL     |          |                            |         |

|                  |          |                            |         |

**Notes:**

- DB2 supports DATE, TIME, and TIMESTAMP data. DB2 does the necessary verifications on data that is entered in these types of columns. A number of SQL scalar functions are provided to manipulate the date/time data types, such as DAYS(), MONTH(), and YEAR().
Notes:

- **Views** are logical representations of data that exists in base tables.
- Access to data can reference either the base table or the view.
- Data is physically stored in the base table only. When a view is accessed, data is dynamically retrieved from the base table.
- Views can be used for security and simplicity.

**Note:** A detailed discussion of views is included in the prerequisite course, SQL Workshop.
Declared Temporary Tables

Application temporary tables (Linux, UNIX, Windows)

DECLARE GLOBAL TEMPORARY TABLE T1
LIKE REAL_T1
ON COMMIT DELETE ROWS
NOT LOGGED
IN USR_TEMP_TS;
INSERT INTO SESSION.T1
SELECT * FROM REAL_T1 WHERE DEPTNO=:mydept;
/* do other work on T1 */
/* when connection ends, table is automatically dropped */

Notes:

• DECLARE GLOBAL TEMPORARY TABLE (a declared temporary table) is used to hold data temporarily, such as the intermediate results of SQL transactions. Declared temporary tables persist only as long as the application process. The description of a declared temporary table is neither stored nor shareable. Thus, each application process might refer to the same declared temporary table but have its own unique description of it.

• The DECLARE GLOBAL TEMPORARY TABLE statement defines a declared temporary table for the current application process. The declared temporary table description does not appear in the system catalog. It is not persistent and cannot be shared with other application processes. Each application process that defines a declared temporary table of the same name has its own unique description and instance of the temporary table. When the application process terminates, the temporary table is dropped.
Invocation
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization
None are required, unless the LIKE clause is specified when additional privileges might be required.
Referential Integrity (1 of 3)

Some Basic Terms

- **Unique Key** - A column or set of columns that is constrained so that no two of its values are equal.
- **Unique Constraint** - A column or set of columns that uniquely identifies one row in a table from all other rows. A table may have multiple unique constraints identified.
- **Primary Key** - One particular unique constraint which may be the one most frequently used to access the table.
- **Parent Key** - A unique constraint or primary key that is used to provide valid values for another column.
- **Foreign Key** - A column or set of columns in a table defined to be related to the parent key column or set of columns in another table.
- **Parent Table** - Within a given relationship, the table that contains the parent key.
- **Child Table** - Within a given relationship, the table that contains the foreign key.

**Notes:**

- **DEPARTMENT** table
  - **PRIMARY KEY**:
    - DEPTNO
    - DEPTNAME
  - **Parent Key**:
    - DEPTNO
    - DEPTNAME

- **EMPLOYEE** table
  - **PRIMARY KEY**:
    - EMPNO
  - **FOREIGN KEY**:
    - DEPTNO
  - **UNIQUE CONSTRAINT**:
    - DEPTNO
  - **Parent Key**:
    - EMPNO

- **DEPARTMENT** table
  - C01 INFORMATION CENTER
  - E11 OPERATIONS

- **EMPLOYEE** table
  - E11
    - EMPNO: 000090
    - FIRSTNM: EILEEN
    - LASTNM: HENDERSON
    - DEPT: C01
    - TAX_ID: 101-98-7654
  - E11
    - EMPNO: 000290
    - FIRSTNM: JOHN
    - LASTNM: PARKER
    - DEPT: E11
    - TAX_ID: 214-32-8045
  - E11
    - EMPNO: 000300
    - FIRSTNM: PHILIP
    - LASTNM: SMITH
    - DEPT: E11
    - TAX_ID: 750-18-1912

© Copyright IBM Corporation 2005
Referential Integrity (2 of 3)

- By definition, a FOREIGN KEY in one table matches a PARENT KEY
- A PARENT KEY value must be UNIQUE, and cannot be NULL
- A FOREIGN KEY value must match a PARENT KEY or be NULL
- DB2 enforcement of referential constraints:
  - INSERT/UPDATE/LOAD of FOREIGN KEY
    - Must match PARENT KEY or be NULL
  - UPDATE of PARENT KEY
    - Not allowed if any FOREIGN KEYS match original value

Notes:

- DB2 requires a UNIQUE INDEX on the column or set of columns defined as the parent key.
- Foreign key values only permit NULL if the underlying columns of the foreign key permit NULL. Allowing or restricting NULL values in a column is determined when the column is defined.
Referential Integrity (3 of 3)

- **DELETEing a row**

  - When DELETEing a parent key, what should be done to matching foreign keys?
    - DELETE rows with matching values (CASCADE)
    - Change matching values to null (SET NULL)
    - Disallow the DELETE if matching values exist (RESTRICT / NO ACTION)

**Notes:**

The action that should take place when a parent key value is deleted is called the DELETE RULE.

The correct choice for a DELETE RULE is based on business requirements.

On DELETE, RESTRICT and NO ACTION are the same. On UPDATE, the distinction is:

- RESTRICT is enforced before all other constraints.
- NO ACTION is enforced after all other constraints.

There are very few cases where this can make a difference. NO ACTION is the default.
Check Constraints

```
CREATE TABLE EMP

(EMPNO     CHAR(6)      NOT NULL,
FIRSTNME  VARCHAR(12)  NOT NULL,
MIDINIT   CHAR(1)      NOT NULL WITH DEFAULT,
LASTNAME  VARCHAR(15)  NOT NULL,
WORKDEPT  CHAR(3),
PHONENO   CHAR(4),
HIREDATE  DATE,
JOB       CHAR(8),
EDLEVEL   SMALLINT,
SEX       CHAR(1)      CHECK (SEX IN 'M', 'F')),
BIRTHDATE DATE         CHECK (BIRTHDATE < CURRENT DATE),
SALARY    DECIMAL(9,2) CHECK (SALARY > 0),
BONUS     DECIMAL(9,2) CHECK (BONUS < SALARY),
COMM      DECIMAL(9,2)
CONSTRAINT SALCHECK  CHECK ( SALARY < 15000 OR JOB = 'MANAGER')
)
IN DBX.TSX
```

Notes:

Check Constraints are used to enforce data rules at the column and/or table level.

The column-constraint of a column-definition provides a shorthand method of defining a constraint composed of a single column. Thus, if a column-constraint is specified in the definition of column C, the effect is the same as if that constraint were specified as a unique-constraint, referential-constraint, or check-constraint in which C is the only identified column.

**CONSTRAINT constraint-name**

Names the constraint. If a constraint name is not specified, a unique constraint name is generated and stored in the DB2 catalog. If the name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.
Triggers

create trigger reorder
after update
of qty on stock
referencing new as n
for each row mode db2sql
when (n.qty <=5)
insert into reorder values (n.itemno, current timestamp)

Notes:

- The TRIGGER privilege on the table. The privilege set must include at least one of the following:
  - The TRIGGER privilege on the table on which the trigger is defined
  - The ALTER privilege on the table on which the trigger is defined
  - DBADM authority on the database that contains the table
  - SYSADM or SYSCTRL authority
- The SELECT privilege on the table on which the trigger is defined if any transition variables or transition tables are specified
- The SELECT privilege on any table or view to which the search condition of triggered action refers
- The necessary privileges to invoke the triggered SQL statements in the triggered action
Checkpoint

Exercise — Unit 1 Checkpoint

1. Some of the platforms where DB2 is available are:

___________________________________________________________________
___________________________________________________________________
___________________________________________________________________
___________________________________________________________________
___________________________________________________________________
___________________________________________________________________
___________________________________________________________________
___________________________________________________________________

2. True or False? A trigger may be defined on a table to enforce referential integrity.
___________________________________________________________________

3. True or False? Check constraints are defined only at the column definition of a table.
___________________________________________________________________

4. True or False? A stored procedure can only be defined in a client server environment when called from a Java program.
___________________________________________________________________

5. True or False? DB2 Call Level Interface is a callable SQL programming interface for C/C++ programming language.
___________________________________________________________________

6. True or False? DB2 will determine the access strategy for dynamic SQL within an application program before execution time.
___________________________________________________________________

7. If application development is going to be done on an Intel-based workstation running Windows NT, what product should be installed on that machine?
___________________________________________________________________
Unit Summary

Having completed this unit, you should be able to:

- Identify DB2 family products
- Explain DB2 component functions
- Identify DB2 objects
- Identify the key differences between static SQL and other application alternatives for accessing DB2 data

Notes:
Unit 2. Program Structure I

What This Unit Is About

There are certain basic elements that are necessary in any program using embedded SQL. This topic will show how to embed SQL in a program and how to process single-row selects.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Embed INSERT, UPDATE, DELETE and single-row SELECT statements in application programs
• Effectively communicate with DB2 when processing NULL values and determining the success of statement execution
• Demonstrate use of DB2 coding aids
• Code CONNECT statements within an application program
• Identify connection types and impacts on a unit of work.

How You Will Check Your Progress

Accountability:

• Machine lab

References

DB2 UDB for Linux, UNIX, and Windows
SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849 Call Level Interface Guide and Reference, Volume 1
SC09-4850  Call Level Interface Guide and Reference, Volume 2
SC09-4828  Command Reference
SC09-4844  SQL Reference - Volume 1
SC09-4845  SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

- Embed INSERT, UPDATE, DELETE and single-row SELECT statements in application programs
- Effectively communicate with DB2 when processing NULL values and determining success of statement execution
- Demonstrate use of DB2 coding aids
- Code CONNECT statements within an application program
- Identify connection types and impacts on a unit of work

Notes:
2.1 Supporting Embedded SQL
Supporting Embedded SQL

- Use delimiters to identify SQL statements in the application
- Use host variables to provide values on the WHERE clause or receive values from SELECT statements
- Use appropriate techniques to process null data
- Use communications from DB2 to determine success or failure of SQL statements

Notes:

- The code necessary to support each of the above functions will be addressed on the following visuals.
2.2 Delimiters
Delimiters (1 of 2)

- Used by PRECOMPIILER to identify SQL statements to be translated
- Must indicate beginning and end of each embedded SQL statement

<table>
<thead>
<tr>
<th>COBOL</th>
<th>EXEC SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQL statement END-EXEC.</td>
</tr>
<tr>
<td>C; C++; PL/I</td>
<td>EXEC SQL</td>
</tr>
<tr>
<td></td>
<td>SQL statement ;</td>
</tr>
<tr>
<td>Java</td>
<td>#sql { SQL statement } ;</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>X EXEC SQL</td>
</tr>
<tr>
<td></td>
<td>SQL statement</td>
</tr>
</tbody>
</table>

**Notes:**

- The DB2 precompiler uses delimiters to locate embedded SQL statements. When the precompiler locates delimiters, all of the enclosed code is considered SQL to be translated.
- The start and end of each SQL statement must be indicated by delimiters, even if multiple SQL statements appear sequentially in the program's logic.
- The keywords EXEC and SQL must appear on the same line. The remainder of the statement can appear on subsequent lines.
- **COBOL note:**
  - All COBOL delimiters and SQL statements must begin at or after column 12.
Delimiters (2 of 2)

- SQL STATEMENT
  ```
  UPDATE TEMPL
  SET WORKDEPT = 'C02'
  WHERE WORKDEPT = 'C01'
  ```

- SQL STATEMENT in a COBOL PROGRAM
  ```
  EXEC SQL
  UPDATE TEMPL
  SET WORKDEPT = 'C02'
  WHERE WORKDEPT = 'C01'
  END-EXEC.
  ```

- SQL STATEMENT in a C or PL/I PROGRAM
  ```
  EXEC SQL
  UPDATE TEMPL
  SET WORKDEPT='C02'
  WHERE WORKDEPT = 'C01' ;
  ```

- SQL STATEMENT in a JAVA PROGRAM
  ```
  #sql { UPDATE TEMPL
  SET WORKDEPT = 'C02'
  WHERE WORKDEPT = 'C01' } ;
  ```

Notes:
- The above examples illustrate embedding an SQL statement in an application program written in COBOL, Java, or C language.
- Note that the difference in the programming statements is limited to the delimiters.
2.3 Host Variables
Host Variables

Notes:

- The term **HOST VARIABLE** is used to identify the symbolic name of storage that has been declared using the application language storage definition statement.
- The colon is used to separate components of SQL syntax from the symbolic names. Without the colon, the host variable may be misinterpreted as a component of the SQL statement. For example,

  ```sql
  WHERE WORKDEPT = DEPT
  ```

  would be interpreted as matching the value of column WORKDEPT with the value of column DEPT.

**Note:** You can use any valid COBOL name for a host variable. Do not use host variable names that begin with ‘SQL’.

**Note:** Case sensitivity: The C language precompiler will recognize `EXEC SQL` or `exec sql` or any combination of upper and lower case characters. Table and column names specified in your embedded SQL are mapped to upper case as the DB2 catalog stores
them in upper case. Case is preserved for host variable names or string literals included within the SQL statement.
Host Variables - Insert - C Program

- **SQL statement**

  ```sql
  INSERT INTO TEMPL
  ( EMPNO, LASTNAME )
  VALUES ( '000190', 'JONES' )
  ```

- **SQL statement in a program**

  ```sql
  EXEC SQL INSERT INTO TEMPL
  ( EMPNO, LASTNAME )
  VALUES ( :empno, :name );
  ```

  ![Diagram](image)

  000190
  
  empno

  JONES
  
  name

**Notes:**

- The first SQL statement could also be embedded in an application simply by enclosing it in delimiters. However, the statement could only insert the hard-coded values into the table.

- The second statement shows the use of host variables in an embedded SQL statement. This statement could be included in a processing loop with the program's logic assigning various values to the host variables.

  The values assigned to the host variables could be obtained by any application programming technique, such as mapping a screen of input or reading a file.
Host Variables - Update - C Program

- **SQL statement**
  
  ```sql
  UPDATE TEMPL
  SET SALARY = SALARY * 1.05
  WHERE JOBCODE = 54
  ```

- **SQL statement in a program**
  
  ```sql
  EXEC SQL
  UPDATE TEMPL
  SET SALARY = SALARY * :percent
  WHERE JOBCODE = :code;
  ```

```
\[
\text{code} \quad 54
\]
\[
\text{percent} \quad 1.05
\]
```

---

**Notes:**

- Host variables can be used anywhere a value is required in an SQL statement.
- The above example illustrates the use of host variables on the WHERE clause and SET clause of an UPDATE statement.
Host Variables - To Receive a Value

TO RECEIVE A VALUE

SELECT . . . INTO :a . . .

- REQUIRED with SELECT statement
- One HOST VARIABLE for each COLUMN SELECTed
- Host language label in SQL statement is preceded by a colon
- Host variable declaration must match column data type
- Additional variable may be used for indicating NULL characteristic

Notes:

- The SELECT statement requests information to be returned by DB2 to the application program. DB2 requires the name of a host variable that can be used to store the results of a SELECT statement.
- Again, the colon is used to delineate host variables from other components of the SQL language.
- The host variable must match the column data type, and should be identical to avoid conversions and truncation.

Note: The SELECT ... INTO ... syntax can only be used to return one or zero rows. Also, the GROUP BY, HAVING, and ORDER BY clauses are not permitted in this form of SELECT.
Receiving a Value - C Program

EXEC SQL
SELECT LASTNAME, WORKDEPT
    INTO :name, :deptno
FROM TEMPL
WHERE EMPNO = :empid;

Notes:
- Each column selected must have a corresponding host variable.
- The host variables are separated with commas.
- Assignment is positional - the list of columns is assigned to the list of host variables.
Notes:

- The boxes above represent some of the data types available in DB2.
- The circles above represent some of the languages in which SQL statements can be embedded.
- The C precompiler recognizes only a subset of valid C declarations as valid host variable declarations.
  - User-defined data types are not allowed.
  - The C data type `int` is not allowed because its internal representation is machine dependent.
  - Numeric host variables may be of type double, short, or long.
  - Character host variables may be defined as a fixed character array or variable length strings may be defined as a structure containing their length and data.
- On Linux, UNIX, and Windows, the Application Development Guide: Programming Client Applications provides the Java equivalent of each SQL data type, based on the
JDBC specification for data type mappings. Some mappings depend on whether you use the JDBC Version 1.22 or Version 2.0 driver.

- On Linux, UNIX, and Windows, consult the Application Development Guide: Programming Client Applications for complete reference concerning how specific data types should be defined in any each supported language.

You can declare C++ class data members as host variables. Refer to the Application Development Guide: Programming Client Applications or Application Programming and SQL Guide for examples of how this can be done.

**Note:** DB2 externalizes DATE, TIME and TIMESTAMP data types as character strings. The format of the external DATE and TIME is determined by the database territory, precompiler option, or the CHAR scalar function. The USA format for DATE and TIME, as well as the TIMESTAMP format are shown below:

- **DATE:** mm/dd/yyyy
- **TIME:** hh:mm AM (or PM)
- **TIMESTAMP:** yyyy-mm-dd-hh.mm.ss.nnnn

### SQL Data Types Mapped to Typical COBOL Declarations

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>COBOL data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>S9(4) COMP-4,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S9(4) COMP-5,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S9(4) COMP,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or S9(4) BINARY</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>S9(4) COMP-4,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S9(4) COMP-5,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S9(4) COMP,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or S9(4) BINARY</td>
<td></td>
</tr>
<tr>
<td>DECIMAL(p,s) or NUMERIC(p,s)</td>
<td>S9(p-s)V9(s) COMP-3 or S9(p-s)V9(s) PACKED-DECIMAL DISPLAY SIGN LEADING SEPARATE NATIONAL SIGN LEADING SEPARATE</td>
<td>p is precision; s is scale. 0&lt;=s=p&lt;=31. If s=0, use S9(p)V or SP(p). If s=p, use SV9(s). If the COBOL compiler does not support 31-digit decimal numbers, no exact equivalent exists. Use COMP-2.</td>
</tr>
<tr>
<td>REAL or FLOAT (n)</td>
<td>COMP-1</td>
<td>1&lt;=n=21</td>
</tr>
<tr>
<td>DOUBLE PRECISION, DOUBLE or FLOAT (n)</td>
<td>COMP-2</td>
<td>22&lt;=n=53</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>Fixed-length character string. For example, 01 VAR-NAME PIC X (n).</td>
<td>1&lt;=n=255</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>Varying-length character string. For example, 01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n)</td>
<td>The inner variables must have a level of 49.</td>
</tr>
<tr>
<td>SQL data type</td>
<td>COBOL data type</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>GRAPHIC(n)</td>
<td>Fixed-length graphic string. For example,</td>
<td>$n$ refers to the number of double-byte characters, not to the number of bytes.</td>
</tr>
<tr>
<td></td>
<td>01 VAR-NAME PIC G(n)</td>
<td>1&lt;=$n$&lt;=$127</td>
</tr>
<tr>
<td></td>
<td>USAGE IS DISPLAY-1.</td>
<td></td>
</tr>
<tr>
<td>VARGRAPHIC(n)</td>
<td>Varying-length graphic string. For example,</td>
<td>$n$ refers to the number of double-byte characters, not to the number of bytes.</td>
</tr>
<tr>
<td></td>
<td>01 VAR-NAME.</td>
<td>The inner variables must have a level of 49.</td>
</tr>
<tr>
<td></td>
<td>49 VAR-LEN PIC S9(4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USAGE BINARY.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49 VAR-TEXT PIC G(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USAGE IS DISPLAY-1.</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>Fixed-length character string of length n.</td>
<td>If you are using a date exit routine, $n$ is determined by that routine. Otherwise, $n$ must be at least 10.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 VAR-NAME PIC X (n).</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>Fixed-length character string of length n.</td>
<td>If you are using a time exit routine, $n$ is determined by that routine. Otherwise, $n$ must be at least 6; to include seconds, $n$ must be at least 8.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 VAR-NAME PIC X (n).</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Fixed-length character string of length n.</td>
<td>$n$ must be at least 19. To include microseconds, $n$ must be 26; if $n$ is less than 26, truncation occurs on the microseconds part.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01 VAR-NAME PIC X (n).</td>
<td></td>
</tr>
<tr>
<td>Result set locator</td>
<td>SQL TYPE IS RESULT-SET-LOCATOR</td>
<td>Use this data type only for receiving result sets. Do not use this data type as a column type.</td>
</tr>
<tr>
<td>Table locator</td>
<td>SQL TYPE IS TABLE LIKE table-name AS LOCATOR</td>
<td>Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. Do not use this data type as a column type.</td>
</tr>
<tr>
<td>BLOB locator</td>
<td>SQL TYPE IS BLOB-LOCATOR</td>
<td>Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type.</td>
</tr>
<tr>
<td>CLOB locator</td>
<td>SQL TYPE IS CLOB-LOCATOR</td>
<td>Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type.</td>
</tr>
<tr>
<td>DBCLOB locator</td>
<td>SQL TYPE IS DBCLOB-LOCATOR</td>
<td>Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type.</td>
</tr>
<tr>
<td>BLOB(n)</td>
<td>SQL TYPE IS BLOB(n)</td>
<td>1&lt;=$n$&lt;=$2147483647</td>
</tr>
<tr>
<td>CLOB(n)</td>
<td>SQL TYPE IS CLOB(n)</td>
<td>1&lt;=$n$&lt;=$2147483647</td>
</tr>
<tr>
<td>DBCLOB(n)</td>
<td>SQL TYPE IS DBCLOB(n)</td>
<td>$n$ is the number of double-byte characters.</td>
</tr>
<tr>
<td></td>
<td>1&lt;=$n$&lt;=$1073741823</td>
<td></td>
</tr>
<tr>
<td>ROWID</td>
<td>SQL TYPE IS ROWID</td>
<td></td>
</tr>
</tbody>
</table>

This table was copied from the *Application Programming and SQL Guide*.

### SQL Data Types Mapped to Typical PL/I Declarations

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>PL/I equivalent</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>BIN FIXED(n)</td>
<td>1&lt;=$n$&lt;=$15</td>
</tr>
<tr>
<td>INTEGER</td>
<td>BIN FIXED(n)</td>
<td>16&lt;=$n$&lt;=$31</td>
</tr>
<tr>
<td>SQL data type</td>
<td>PL/I equivalent</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| DECIMAL(p,s) or NUMERIC(p,s)      | If p=16: DEC FIXED(p) or DEC FIXED(p,s) | p is precision; s is scale. 1<=p<=31 and 0<=s<=p.  
If p>15, the PL/I compiler must support 31-digit decimal variables. |
| REAL or FLOAT (n)                  | BIN FLOAT(p) or DEC FLOAT(m) | 1<=n<=21; 1<=p<=21; and 1<=m<=6                                      |
| DOUBLE PRECISION, DOUBLE or FLOAT (n) | BIN FLOAT(p) or DEC FLOAT(m) | 22<=n<=53; 22<=p<=53; and 7<=m<=16                                  |
| CHAR(n)                           | CHAR(n)         |                                                                      |
| VARCHAR(n)                        | CHAR(n) VAR     | 1<=n<=255                                                             |
| GRAPHIC(n)                        | GRAPHIC(n)      |                                                                      |
| VARGRAPHIC(n)                     | GRAPHIC(n) VAR  |                                                                      |
| DATE                              | CHAR(n)         | If you are using a date exit routine, that routine determines n; otherwise, n must be at least 10. |
| TIME                              | CHAR(n)         | If you are using a time exit routine, that routine determines n; otherwise, n must be at least 6; to include seconds, n must be at least 8. |
| TIMESTAMP                         | CHAR(n)         | n must be at least 19. To include microseconds, n must be 26; if n is less than 26, microseconds part is truncated. |
| Result set locator               | SQL TYPE IS RESULT_SET_LOCATOR | Use this data type only for receiving result sets. Do not use this data type as a column type. |
| Table locator                     | SQL TYPE IS TABLE LIKE table-name AS LOCATOR | Use this data type only in a user-defined function or stored procedure to receive rows of a transition table. Do not use this data type as a column type. |
| BLOB locator                      | SQL TYPE IS BLOB_LOCATOR | Use this data type only to manipulate data in BLOB columns. Do not use this data type as a column type. |
| CLOB locator                      | SQL TYPE IS CLOB_LOCATOR | Use this data type only to manipulate data in CLOB columns. Do not use this data type as a column type. |
| DBCLOB locator                    | SQL TYPE IS DBCLOB_LOCATOR | Use this data type only to manipulate data in DBCLOB columns. Do not use this data type as a column type. |
| BLOB(n)                           | SQL TYPE IS BLOB(n) | 1<=n<=2147483647                                                      |
| CLOB(n)                           | SQL TYPE IS CLOB(n) | 1<=n<=2147483647                                                      |
| DBCLOB(n)                         | SQL TYPE IS DBCLOB(n) | n is the number of double-byte characters. 1<=n<=1073741823            |
| ROWID                             | SQL TYPE IS ROWID |                                                                      |

This table was copied from the *Application Programming and SQL Guide*. 

© Copyright IBM Corp. 1997, 2005
### SQL Data Types Mapped to C/C++ Declarations

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>C/C++ Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>16-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>short int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlint16</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>long</td>
<td>32-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>long int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlint32</td>
<td></td>
</tr>
<tr>
<td>BEGININT</td>
<td>long long</td>
<td>64-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__int64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlin64</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
<td>Single-precision floating point.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
<td>Double-precision floating point.</td>
</tr>
<tr>
<td>DECIMAL (p, s)</td>
<td>No exact equivalent; use</td>
<td>Packed decimal.</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>(Consider using the CHAR and DECIMAL functions to manipulate packed decimal fields as character data.)</td>
</tr>
<tr>
<td>CHAR(1)</td>
<td>char</td>
<td>Single character</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>No exact equivalent; use</td>
<td>Fixed-length character string.</td>
</tr>
<tr>
<td></td>
<td>char[n+1] where n is large enough to hold the data 1 &lt;= n &lt;= 254</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>struct tag {</td>
<td>Non null-terminated varying character string with 2-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>short int;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char[n]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 32672</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alternatively, use char[n+1] where n is large enough to hold the data 1 &lt;= n &lt;= 32762</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>struct tag {</td>
<td>Non null-terminated varying character string with 2-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>short int;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char[n]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32673 &lt;= n &lt;= 32700</td>
<td></td>
</tr>
<tr>
<td>CLOB(n) character</td>
<td>SQL TYPE IS</td>
<td>Non null-terminated varying string with 4-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>clob(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 1=2147483647</td>
<td></td>
</tr>
<tr>
<td>CLOB locator variable</td>
<td>SQL TYPE IS</td>
<td>Identifies CLOB entries residing on the server.</td>
</tr>
<tr>
<td></td>
<td>clob_locator</td>
<td></td>
</tr>
<tr>
<td>CLOB file reference variable</td>
<td>SQL TYPE IS</td>
<td>Descriptor for file containing CLOB data.</td>
</tr>
<tr>
<td></td>
<td>clob_file</td>
<td></td>
</tr>
<tr>
<td>BLOB(n)</td>
<td>SQL TYPE IS</td>
<td>Non null-terminated varying string with 4-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>blob(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 2147483647</td>
<td></td>
</tr>
<tr>
<td>BLOB locator variable</td>
<td>SQL TYPE IS</td>
<td>Identifies BLOB entries residing on the server.</td>
</tr>
<tr>
<td></td>
<td>blob_locator</td>
<td></td>
</tr>
<tr>
<td>BLOB file reference variable</td>
<td>SQL TYPE IS</td>
<td>Descriptor for file containing BLOB data.</td>
</tr>
<tr>
<td></td>
<td>blob_file</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>Null-terminated character form</td>
<td>Allow at least 11 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>Null-terminated character form</td>
<td>Allow at least 9 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Null-terminated character form</td>
<td>Allow at least 27 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
</tbody>
</table>

---

Student Notebook

SQL Data Types Mapped to C/C++ Declarations

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>C/C++ Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>16-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>short int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlint16</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>long</td>
<td>32-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>long int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlint32</td>
<td></td>
</tr>
<tr>
<td>BEGININT</td>
<td>long long</td>
<td>64-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__int64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sqlin64</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
<td>Single-precision floating point.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
<td>Double-precision floating point.</td>
</tr>
<tr>
<td>DECIMAL (p, s)</td>
<td>No exact equivalent; use</td>
<td>Packed decimal.</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>(Consider using the CHAR and DECIMAL functions to manipulate packed decimal fields as character data.)</td>
</tr>
<tr>
<td>CHAR(1)</td>
<td>char</td>
<td>Single character</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>No exact equivalent; use</td>
<td>Fixed-length character string.</td>
</tr>
<tr>
<td></td>
<td>char[n+1] where n is large enough to hold the data 1 &lt;= n &lt;= 254</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>struct tag {</td>
<td>Non null-terminated varying character string with 2-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>short int;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char[n]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 32672</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alternatively, use char[n+1] where n is large enough to hold the data 1 &lt;= n &lt;= 32762</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>struct tag {</td>
<td>Non null-terminated varying character string with 2-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>short int;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>char[n]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32673 &lt;= n &lt;= 32700</td>
<td></td>
</tr>
<tr>
<td>CLOB(n) character</td>
<td>SQL TYPE IS</td>
<td>Non null-terminated varying string with 4-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>clob(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 1=2147483647</td>
<td></td>
</tr>
<tr>
<td>CLOB locator variable</td>
<td>SQL TYPE IS</td>
<td>Identifies CLOB entries residing on the server.</td>
</tr>
<tr>
<td></td>
<td>clob_locator</td>
<td></td>
</tr>
<tr>
<td>CLOB file reference variable</td>
<td>SQL TYPE IS</td>
<td>Descriptor for file containing CLOB data.</td>
</tr>
<tr>
<td></td>
<td>clob_file</td>
<td></td>
</tr>
<tr>
<td>BLOB(n)</td>
<td>SQL TYPE IS</td>
<td>Non null-terminated varying string with 4-byte string length indicator.</td>
</tr>
<tr>
<td></td>
<td>blob(n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 &lt;= n &lt;= 2147483647</td>
<td></td>
</tr>
<tr>
<td>BLOB locator variable</td>
<td>SQL TYPE IS</td>
<td>Identifies BLOB entries residing on the server.</td>
</tr>
<tr>
<td></td>
<td>blob_locator</td>
<td></td>
</tr>
<tr>
<td>BLOB file reference variable</td>
<td>SQL TYPE IS</td>
<td>Descriptor for file containing BLOB data.</td>
</tr>
<tr>
<td></td>
<td>blob_file</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>Null-terminated character form</td>
<td>Allow at least 11 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>Null-terminated character form</td>
<td>Allow at least 9 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Null-terminated character form</td>
<td>Allow at least 27 characters to accommodate the null-terminator.</td>
</tr>
<tr>
<td></td>
<td>VARCHAR structured form</td>
<td></td>
</tr>
</tbody>
</table>
Reference:
SC09-4826  Application Development Guide: Programming Client Applications
Declaring Host Variables - C Program

EXEC SQL BEGIN DECLARE SECTION;
    struct {
        char empno[7];
        char lastname[21];
    } employee;
    double percent;
    long code;
EXEC SQL END DECLARE SECTION;

Notes:

- All host variables referenced in embedded SQL are required (on Linux, UNIX, and Windows) to be declared within the section of code delimited by the BEGIN and END DECLARE statements.
- Host variable names cannot begin with EXEC, SQL, sql, DB2, or db2.
- Host variable names should be considered global to a module rather than local to the function in which they are defined.
- Variables declared outside a declare section must not have the same name as variables declared within a Declare Section.
- Multiple declare sections may be used in one source file.
- The BEGIN DECLARE SECTION statement may be coded in the application program wherever variable declarations can appear in accordance with the rules of the host language. A host variable section ends with an END DECLARE SECTION statement.
- The BEGIN DECLARE SECTION and the END DECLARE SECTION statements must be paired and may not be nested.
- Host variable declarations can be specified by using the SQL INCLUDE statement. Otherwise, a host variable declaration section must not contain any statements other than host variable declarations.
Declaration Considerations For C

CREATE TABLE SIMPLE
(TITLE CHAR (20) NOT NULL
 NUMBER SMALLINT NOT NULL)

EXEC SQL BEGIN DECLARE SECTION;
char title [21];
short number;
EXEC SQL END DECLARE SECTION;

main( )
{
    EXEC SQL
        SELECT TITLE INTO :title
        FROM SIMPLE
        WHERE NUMBER = :number;
}

Notes:

- To support character string data, provide storage for the database data and the null terminator.
- Since int is environment-dependent, it is not permitted for host variable declarations.
- Consult the Application Development Guide: Programming Client Applications for other details, including variations for declaring storage to support character data.

Reference:

SC09-4826 Application Development Guide: Programming Client Applications
### DCLGEN

- Generate table declaration -
  - `db2dclgn` (DCLGEN)
  - Creates structures for host variables
  - Languages supported
    - C, Java, COBOL, FORTRAN
  - `db2dclgn -D sample -T employee -L Java`

---

**Notes:**

DCLGEN is a tool that generates declarations for a given table in a database. It eliminates the need for the developers to look up those declarations in the documentation themselves, allowing faster development. The generated declarations should be used as a basis if further tailoring is desired.

- `db2dclgn -D <database name> -T <tablename> [<options>]`
### Option Description Parameter Default Setting

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Parameter</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Action</td>
<td>ADD, REPLACE</td>
<td>ADD</td>
</tr>
<tr>
<td>-b</td>
<td>LOB variable type</td>
<td>LOB, LOCATOR, FILE</td>
<td>LOB</td>
</tr>
<tr>
<td>-c</td>
<td>Column as suffix</td>
<td>none</td>
<td>FALSE</td>
</tr>
<tr>
<td>-i</td>
<td>Indicator variables</td>
<td>none</td>
<td>FALSE</td>
</tr>
<tr>
<td>-l</td>
<td>Host Language</td>
<td>C, COBOL, FORTRAN, JAVA</td>
<td>C</td>
</tr>
<tr>
<td>-n</td>
<td>Field name prefix</td>
<td>prefix</td>
<td>none</td>
</tr>
<tr>
<td>-o</td>
<td>Output file</td>
<td>filename</td>
<td>tablename.x</td>
</tr>
<tr>
<td>-p</td>
<td>Password</td>
<td>password</td>
<td>none</td>
</tr>
<tr>
<td>-r</td>
<td>Remarks</td>
<td>none</td>
<td>FALSE</td>
</tr>
<tr>
<td>-s</td>
<td>Structure name</td>
<td>structure name</td>
<td>table name</td>
</tr>
<tr>
<td>-u</td>
<td>Userid</td>
<td>userid</td>
<td>none</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose</td>
<td>none</td>
<td>FALSE</td>
</tr>
<tr>
<td>-w</td>
<td>DBCS Variable type</td>
<td>SQLDBCHAR, WCHAR_T</td>
<td>SQLDBCHAR</td>
</tr>
<tr>
<td>-y</td>
<td>DBCS symbol</td>
<td>G, N</td>
<td>G</td>
</tr>
</tbody>
</table>

- Where .x is:
  - .h for C
  - .cbl for COBOL
  - .java for Java
  - .f for FORTRAN on UNIX
  - .for for FORTRAN on Intel

The -i option specifies whether indicator variables are to be generated. Since host structures are supported in C and COBOL, an indicator table of size equal to the number of columns is generated, whereas for JAVA and FORTRAN, individual indicator variables are generated for each column. The names of the indicator table and the variable are the same as the table name and the column name, respectively, prefixed by “IND-” (for COBOL) or “ind_” (for other languages). The default behavior is to not generate indicator variables.

The -c option specifies whether the column name is to be used as suffix in the file name when a prefix (-n) is specified. If no prefix is specified, this option is ignored. The default behavior is to not use the column name as a suffix, but instead to use the column number, which starts at 1.

The -s option specifies the structure name that is to be generated to group all the fields in the declarations. The default behavior is to use the unqualified table name.
For example, to generate the declarations for the STAFF table in the SAMPLE database in C in the output file staff.h, issue the following command:

```
db2dclgn -d sample -t staff -l C
```

The resulting staff.h file contains:

```c
struct {
    short id;
    { short length;
      char data[9];
    } name;
    short dept;
    char job[5];
    short years;
    double salary;
    double comm;
} staff;
```
Host Structure Support in C

EXEC SQL BEGIN DECLARE SECTION;
  struct
   {
     short id;
     struct
       {
         short length;
         char data[10];
       } name;
     struct
       {
         short years;
         double salary;
       } info;
  } staff_record;
EXEC SQL END DECLARE SECTION;

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

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;

Notes:
2.4 Processing Null Data
Indicator Variables

SELECT COLA INTO :a:aind

- value
  a

- aind

- INDICATOR VARIABLE (SMALLINT) is required if SELECTed column allows NULL
- If column is NULL, INDICATOR VARIABLE is set to negative value, VALUE VARIABLE is untouched
- INDICATOR VARIABLE can be set to negative value by program to indicate NULL on UPDATE or INSERT

Notes:
- DB2 will set the indicator to a positive value (usually 0) if the column does not contain NULL.
- DB2 will set the indicator to a negative value (usually -1) if the column contains NULL.
- If DB2 attempts to indicate the presence of a NULL and the program does not provide an indicator variable, an error results.
- Sample C declare for an indicator:

```c
EXEC SQL BEGIN DECLARE SECTION;
short aind;
char a[9];
EXEC SQL END DECLARE SECTION;
```

- Optionally, the specification of the keyword INDICATOR can be included:

```sql
EXEC SQL
SELECT COLA INTO a INDICATOR :aind;
```
Indicator Variables Example - C Program

```
CREATE TABLE TEMPL
(EMPNO      CHAR(6)       NOT NULL,
LASTNAME   VARCHAR(20)   NOT NULL,
JOBCODE    CHAR(2),
WORKDEPT   CHAR(3)       NOT NULL,
PHONENO    CHAR(10))

EXEC SQL

SELECT JOBCODE, WORKDEPT, PHONENO
INTO :JC :JCI, :DPT, :PHO:PHOI
FROM TEMPL
WHERE EMPNO = :ID;

EMPNO (6) | LASTNAME (20) | JOBCODE 0-99 | WORKDEPT (3) | PHONENO (10)
----------|--------------|--------------|--------------|-------------
000070    | JOHNSON      | 54           | C01          | 5137853210  
000120    | SCOTT        | ?            | C01          | 8592743091  
000320    | MILLIGAN     | ?            | C01          | ?            
```

Notes:

- Each column that allows NULL should have a corresponding indicator variable.
- The indicator is associated with a host variable in each statement by position. Notice that the host variable and indicator variable are not separated by a comma.

Question - If the WORKDEPT column in the above table permitted NULLs, is the above code subject to an execution time error?
Testing for NULL on Select - C Program

EXEC SQL
SELECT PHONENO, SEX
INTO :phoneno , :phoneind , :sex
FROM TEMPL
WHERE EMPNO = :eno;

if (phoneind < 0)
    null_phone ();
else
    good_phone ();

Notes:

- In the above example, DB2 manipulates the indicator variable.
- The application program interrogates the indicator after execution.
Passing NULL Value - C Program

```c
if (some condition)
    phoneind = -1;
else
    phoneind = 0;
```

**EXEC SQL**

```sql
UPDATE TEMPL
SET PHONENO = :newphoneno :phoneind
WHERE EMPNO = :eno;
```

**Notes:**

- In the above example, the application program sets the indicator variable to indicate whether a null is desired before execution of the SQL statement.
- If DB2 finds a negative value in the indicator, the column is set to null for the given set of rows. The value in the host variable is ignored. If the indicator contains a positive value or zero, the value in the corresponding host variable will be used.
- The keyword NULL is permitted in embedded SQL statements. The following syntax could be used to avoid the use of indicator variables for an UPDATE statement:

```c
if (some condition)
    EXEC SQL
    UPDATE TEMPL
    SET PHONENO = NULL
    WHERE EMPNO = :eno;
else
    EXEC SQL
    UPDATE TEMPL
```
SET PHONENO = :newphone
WHERE EMPNO = :eno;

This technique is not favored since maintenance of this code would require duplicate changes should any portion of the UPDATE statement require modification.
# Indicator Variable Usage - NULL Values

<table>
<thead>
<tr>
<th></th>
<th>HOST VARIABLE :CD</th>
<th>INDICATOR VARIABLE :CDI</th>
<th>COLUMN JOBCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT/FETCH</strong></td>
<td>60 UNCHANGED</td>
<td>0 &lt;0</td>
<td>60 NULL</td>
</tr>
<tr>
<td><strong>UPDATE/INSERT</strong></td>
<td>50 N/A</td>
<td>0 &lt;0</td>
<td>50 NULL</td>
</tr>
</tbody>
</table>

**DB2 MANIPULATES INDICATOR**

**PROGRAM MANIPULATES INDICATOR**

---

**Notes:**

- Indicator variables are set by DB2 during execution of SELECT and FETCH statements, and should be checked by the application program after execution.

  **Note:** The FETCH statement is used to retrieve a single row from a result set.

- Indicator variables are set by the application program before execution of UPDATE and INSERT statements to indicate that a null should be placed in the database table by DB2.
Indicator Variable Usage - Numeric Conversion

- Numeric conversion is handled by the database manager
  - Completed when host variable is of different type from column
  - Normally transparent to the application

- If column value cannot be stored in host variable
  - For example, DECIMAL(15) with 12 digits assigned to INTEGER host variable
  - Results in -2 value in the indicator variable

Notes:

- Applications can be written to check indicator variables for values less than zero (0) in order to determine if the associated host variable should be ignored and that a null value was returned.

- An indicator variable value of negative two (-2) means the associated host variable should be ignored and a null value was returned. However, the reason for the returned null is that a numeric conversion could not be performed. An application may or may not need to handle this particular reason specifically.
2.5 SQLCA
Notes:

- An application program containing executable SQL statements must either provide a structure named SQLCA or a stand-alone integer variable named SQLCODE.
- The success or failure of the last SQL statement executed is described in the SQLCA.
- The programmer should check the SQLCA after every executable SQL statement.
SQLCODE and SQLSTATE

**Notes:**

- SQLSTATE is set by the database manager after execution of each SQL statement. Thus, application programs can check the success of the execution of SQL statements by testing SQLSTATE instead of SQLCODE.
- SQLSTATE is a character string variable in the SQLCA.
- SQLSTATE provides application programs with common codes for common error conditions. Furthermore, SQLSTATE is designed so that application programs can test for specific errors (by looking at the whole field) or classes of errors (by looking at the first two bytes). The coding scheme is the same for all database managers and is based on the ISO/ANSI SQL92 standard.

**Note:** If the application uses the platform-defined error message generation routine, it will determine the correct error message to use.
### SQLCA Codes

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>INTEGER SQLCODE</th>
<th>CHAR(5) SQLSTATE</th>
<th>REQUEST STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>&lt;0</td>
<td>&quot;02nnn&quot;</td>
<td>FAILED</td>
</tr>
<tr>
<td>WARNING</td>
<td>&gt;0 &amp; &lt;&gt; 100</td>
<td>&quot;01nnn&quot;</td>
<td>SATISFIED WITH SPECIAL CONDITION (SQLWARN0 = 'W')</td>
</tr>
<tr>
<td>NOTFOUND</td>
<td>+100</td>
<td>&quot;02nnn&quot;</td>
<td>(MORE) DATA NOT FOUND</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>&quot;00000&quot;</td>
<td>SUCCESS</td>
</tr>
</tbody>
</table>

**Notes:**

- All negative return codes are considered **errors**.
- All positive return codes not equal to 100 are considered **warnings**. Also, a 'W' in SQLWARN0 indicates a **warning**, even if SQLCODE = 0.
- The meaning of SQLCODE values other than 0 and 100 is product specific.
- Descriptions of error messages can be found in the *Messages Reference*.
- Some common SQLCODE values:
  - -551 - `<authorization-ID>` does not have the privilege to perform operation `<operation>` on object `<name>`. (SQLSTATE ‘42501’)
  - -818 - A timestamp conflict occurred. (SQLSTATE ‘51013’)
  - -805 - Package `<package-name>` was not found. (SQLSTATE ‘51002’)
  - -904 - Unsuccessful execution caused by an unavailable resource. Reason code: `<reason-code>`, type of resource: `<resource-type>`, and resource name: `<resource-name>`. (SQLSTATE ‘57011’)

---

© Copyright IBM Corporation 2005
- -911 - The current transaction has been rolled back because of a deadlock or timeout. Reason code <reason-code>. (SQLSTATE ‘40001’)
- -305 - The NULL value cannot be assigned to a host variable in the SELECT or FETCH statement because no indicator variable is specified. (SQLSTATE ‘22002’)
- -811 - The result of a scalar fullselect, SELECT INTO statement, or VALUES INTO statement is more than one row. (SQLSTATE ‘21000’)
- +100 - No row was found for FETCH, UPDATE, or DELETE; or the result of a query is an empty table. (SQLSTATE ‘02000’)

Reference:

GC09-4840  Messages Reference, Volume 1
GC09-4841  Messages Reference, Volume 2
# SQLCA Warnings

**Figure 2-24. SQLCA Warnings**

### Notes:

- SQLWARN0 can be interrogated for the presence of a 'W'. If such a value is found, the application program can check SQLWARN1 - SQLWARNA to discover the causes of the WARNING.

- SQLWARN5 and SQLWARN7 are reserved for future use.

In the C or C++ languages, `sqlwarn` is defined as an array of size 11. There are preprocessor `#defines` to map these to SQLWARN0 through SQLWARNA for a `struct sqlca` defined with a name of `sqlca`. Generally, you should use the array names (`sqlwarn[0]` through `sqlwarn[10]`) that correspond to each warning when coding in C or C++.

### IF SQLWARN0 contains:

<table>
<thead>
<tr>
<th>SQLWARN0</th>
<th>SQLWARN1</th>
<th>SQLWARN2</th>
<th>SQLWARN3</th>
<th>SQLWARN4</th>
<th>SQLWARN5</th>
<th>SQLWARN6</th>
<th>SQLWARN7</th>
<th>SQLWARN8</th>
<th>SQLWARN9</th>
<th>SQLWARNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>`' '</td>
<td>String value truncated on assignment to host variable</td>
<td>NULL value(s) eliminated from evaluation of a column function</td>
<td>Number of columns greater that number of host variables</td>
<td>Prepared update or delete has no &quot;WHERE&quot; clause</td>
<td>DATE or TIMESTAMP value adjusted to correct an invalid date resulting from arithmetic operation</td>
<td>Character that could not be converted was replaced with a substitution character</td>
<td>Arithmetic expressions with errors ignored during column function processing</td>
<td>A conversion error was detected when converting a character data value one of the fields in the SQLCA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notes:

- The third element of the array SQLERRD (SQLERRD[2] in C language) will contain the number of rows changed by the last UPDATE, INSERT, or DELETE. It does not contain information about rows SELECTed.

- SQLERRD[3] (or SQLERRD[2] in C language) can be checked for reasonability
  - If the number is reasonable, COMMIT WORK.
  - If the number is unreasonable, ROLLBACK WORK.

Consult the SQL Reference for additional details if desired.
2.6 Coding Aids
Coding Aids

- WHenever Statement
  - For exception handling

- Error message formatting routine
  - (z/OS platform) CALL 'DSNTIAR'
  - (UNIX/Windows platform) sqlaintp/sqlgintp

- INCLUDE statement and INCLUDE files
  - To add SQLCA/SQLDA structures to program and to provide function prototypes

Notes:

- These coding aids can be used in dealing with errors and including structures in applications.

- Error message formatting routine:
  - sqlaintp() for C on UNIX and Windows
  - sqlgintp() for COBOL or FORTRAN on UNIX and Windows
WHenever Statement

EXEC SQL WHENEVER Condition Action;

- Condition:
  - SQLERROR
    - Negative SQLCODE
  - SQLWARNING
    - Positive SQLCODE (not +100)
    - Or SQLWARN0 = 'W'
  - NOT FOUND
    - SQLCODE = +100

- Action:
  - GO TO :X
    - Control transferred to statement labelled X
  - CONTINUE
    - Program continues with next statement
    - Used to cancel effect of prior WHENEVER

Notes:
- The DB2 precompiler will resolve WHENEVER statements to if......go to...... checks after every EXEC SQL statement that follows the WHENEVER statement (until the WHENEVER statement is changed or cancelled).
- The precompiler does not evaluate logical branches to determine placement of checks generated by WHENEVER statements. Rather, checks are inserted solely on the basis of a top to bottom scan of the source code.
- GO TO and GOTO are equivalent.
- The use of a colon before the LABEL is optional.
- Examples of WHENEVER statements for C:
  
  EXEC SQL WHENEVER SQLWARNING GOTO :warning;
  EXEC SQL WHENEVER SQLWARNING GO TO :warning;
  EXEC SQL WHENEVER NOT FOUND GOTO :missing;
  EXEC SQL WHENEVER SQLERROR GOTO oh_no;
  EXEC SQL WHENEVER SQLERROR CONTINUE;
Get Error Message

- Invoke `sqlaintp()` and pass it:
  - Output buffer
  - Size of buffer
  - Line lengths
  - Address of SQLCA

- **FORTRAN and COBOL** use `sqlgintp()`

Notes:

- One message is returned per call.
- An LS/NULL sequence is placed at the end of each message.
- If a positive line width is specified, LF is inserted between words so that the lines do not exceed the line length.
- If a word is longer than a line width, the line is filled with as many characters as will fit; an LF is inserted; and the remaining characters are placed on the next line.
Alternative to WHENEVER - C Program

SQL statement

```c
if (SQLCODE < 0)
    process_error(&sqlca);
if (SQLCODE == 100)
    printf("A record for key %d was not found \n", host_var);
```

```c
void process_error (struct sqlca * sqlca)
{
    char buffer [512];
    short buff_size = sizeof(buffer);
    short line_len = 50;

    sqlaintp (buffer, buff_size, line_len, sqlca);
    printf(buffer);
}
```

Notes:

- Since the WHENEVER statement resolves into a `goto`, it is not favored in structured programming shops. For that reason, you may consider using function calls to process errors that may occur following executable SQL statements.

- The alternative to WHENEVER can also use a case expression. An example using C:

```c
CASE SQLCODE
    WHEN 0 ....
    WHEN 100 ...
    WHEN OTHER - handle error
END;
```
SQL INCLUDE Statement - C Program

To copy SQLCA, SQLDA, or textfile:

file.sqc

```sql
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL INCLUDE 'myfile.h';
EXEC SQL END DECLARE SECTION;
```

file.c

```c
struct sqlca sqlca;
/* myfile.h contains my host variables */
```

Notes:

- The INCLUDE statement inserts source code into a source program at precompile time.
- Alternatives:
  ```sql
  EXEC SQL INCLUDE SQLCA;  #include <sqlca.h>
  struct sqlca sqlca;
  
  EXEC SQL INCLUDE 'myfile.h';  #include "myfile.h";
  ```
- Note that the EXEC SQL INCLUDE is processed during precompile so the code included is available to the precompiler. If `myfile.h` contains host variable declarations, it must be included with the EXEC SQL INCLUDE.
- Include files (header files) provide the necessary definitions of APIs, constants, data structures, macros, and constraints required by applications.
- Some others in addition to those above are:
  - `sql.h` - Defines function prototypes for certain APIs including error message retrieval.
- sqlda.h - Defines the SQL Descriptor Area (SQLDA) structure.
- sqlenv.h - Defines language-specific calls for database environment APIs, and the structure, constants, and return codes for those interfaces.
- sglutil.h - Defines the language-specific calls for the utility APIs, and the structures, constants, and codes required for those interfaces.
Program Flow Pseudocode

INCLUDE SQLCA

BEGIN DECLARE SECTION (optional in COBOL and PL/I)
    define host variables
END DECLARE SECTION (optional in COBOL and PL/I)

Establish connection

Do database processing

COMMIT

Notes:
- Optionally, EXEC SQL WHENEVER statements could be added.
Checkpoint

Exercise — Unit 2 Checkpoint

1. What are the delimiters to identify an SQL statement in a C program?

___________________________________________________________________

2. How do you indicate a host variable in the program?

___________________________________________________________________
___________________________________________________________________

3. How can you find out if the value of a column is NULL?

___________________________________________________________________

4. What should you check after every executable SQL statement?

___________________________________________________________________

5. What API (for the C language) can you call to get the error message for an SQLCODE?

___________________________________________________________________
Unit Summary

Having completed this unit, you should be able to:

- Embed INSERT, UPDATE, DELETE and single-row SELECT statements in application programs
- Effectively communicate with DB2 when processing NULL values and determining success of statement execution
- Demonstrate use of DB2 coding aids
- Code CONNECT statements within an application program
- Identify connection types and impacts on a unit of work

Notes:
Unit 3. Program Preparation

What This Unit Is About

Once an application program is coded with embedded SQL, it must be processed by the precompiler, compiler, and binder in order to create the executable application module as well as the package to identify how the database should be accessed.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Identify the additional steps necessary to prepare a program that contains embedded SQL for execution
• Describe the functions of the DB2 PRECOMPILE and BIND processes
• Describe factors relevant to the BIND process, including RUNSTATS positioning, package status, parameters, and authorization requirements

How You Will Check Your Progress

Accountability:

• Machine lab

References

DB2 UDB for UNIX and Windows
SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849  Call Level Interface Guide and Reference, Volume 1
SC09-4850  Call Level Interface Guide and Reference, Volume 2
SC09-4828  Command Reference
SC09-4844  SQL Reference - Volume 1
SC09-4845  SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

- Identify the additional steps necessary to prepare a program that contains embedded SQL for execution
- Describe the functions of the DB2 PRECOMPILE and BIND processes
- Describe factors relevant to the BIND process, including RUNSTATS positioning, package status, parameters, and authorization requirements

Notes:
3.1 Program Preparation
Notes:

- When an embedded SQL program is written, it contains application code and SQL statements. Before the program can be compiled by the appropriate source code language compiler, the EXEC SQL statements embedded in the program must be removed and replaced with something that will be valid for the compiler to process.

- The process of replacing the SQL statements with program library calls is done by a DB2 process called the precompiler.

- For static embedded SQL programs, the SQL statements will be additionally processed by the DB2 bind process to create a package.

- The DBRM (z/OS) or bindfile (UDB) is input to DB2 bind process to create a package.
Write the Source

```sql
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE EMP END-EXEC.
...
EXEC SQL
  SELECT FIRSTNME, LASTNAME, PHONENO
  INTO :FIRSTNME,:LASTNAME,:PHONENO
  FROM EMP
  WHERE EMPNO   = :EMPNO
END-EXEC.
```

COBOL

```cobol
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE EMP;
...
EXEC SQL
  SELECT FIRSTNME, LASTNAME, PHONENO
  INTO :firstnme,:lastname,:phoneno
  FROM EMP
  WHERE EMPNO   = :empno;
```

Notes:

The source code file will contain SQL statements and host language statements.

Linux, UNIX, and Windows:

- The source program must have one of the following extensions:
  a. C language - sqc
  b. C++ language (AIX) - sqC
     C++ language (Windows) - sqx
  c. COBOL language - sqb
  d. FORTRAN language - sqf
  e. Java language - sqlj

© Copyright IBM Corporation 2005
The Big Picture - Precompile

Notes:
The precompiler prepares the source program for compilation by replacing the EXEC SQL statements with a CALL to a language specific library, and by commenting out the EXEC SQL statement.

For future DB2 processing:
- In z/OS, it generates a DBRM (Database Request Module)
- In UNIX and Windows, it optionally creates a Bindfile.

Either a DBRM or a bindfile contains all the SQL statements that were extracted from the source module.

On UNIX, and Windows, the precompiler requires a database connection so the precompile process can verify the validity of the usage of the database objects.
Precompile

Notes:

The precompiler prepares the source program for compilation by replacing EXEC SQL by a CALL and commenting out the EXEC SQL statement. It verifies the SQL syntax. The precompiler also produces a consistency token and statement number for each SQL statement in the source program.

The precompile process will include any files included with an EXEC SQL INCLUDE statement, including files produced by DCLGEN.

Linux, UNIX, and Windows:

- By default, when a program is precompiled, it is also bound to the database, creating a package directly in the database. It does not create a bind file that would require a subsequent bind step. This is called an inline bind.

- If you use the BINDFILE option, the precompiler creates a bind file (with extension .bind) that contains the data required to create a package. This file can be used later with the BIND command to bind the application to one or more databases. If you specify BINDFILE and do not specify the PACKAGE option, binding is deferred until you invoke
the BIND command. Note that for the Command Line Processor (CLP), the default for
PREP does not specify the BINDFILE option. Thus, if you are using the CLP and want
the binding to be deferred, you need to specify the BINDFILE option.
Bindfile

A Bindfile contains the program's source SQL statements

- A Bindfile
  - Contains extracted, parsed SQL source
  - Is stored as a filesystem file
  - One member or file created per precompile
  - Will become input to BIND

Notes:

Linux, UNIX, and Windows

- One bind file corresponds to exactly one source module.
- The bind file consists of:
  - A package name consisting of a two-part name.
  - A consistency token.
  - Data manipulation statements identified by section number.

In the z/OS environment, this object is referred to as a Database Request Module (DBRM).
The Big Picture - Compile and Link

Notes:
The precompile creates a modified source file that then needs to be processed by the compiler.
Notes:

The modified program is now ready to be compiled and link-edited. The link-edit will have to include the necessary modules for the call to work properly. The modified source file is compiled and linked normally, producing an executable file or load module.

Linux, UNIX, and Windows

- The prep command also allows an option `PREPROCESSOR`. `preprocessor-command` to pass information to the language preprocessor. This specifies the preprocessor command that can be executed by the precompiler before it processes the embedded SQL statements. This enables the use of macros within the declare section.
The Big Picture - Bind

Notes:

- When the bind file or DBRM is bound to the database, a package is created in the database. This package contains instructions about how to execute the SQL statements that were included in the source code.

- The bind file or DBRM consists of:
  - A package name.
  - A consistency token.
  - Data manipulation statements identified by section number.

- Static binding involves:
  - Checking the status of the referenced objects to determine if an SQL statement is executable.
  - Creating an optimized form of the SQL statements for improved performance and compactness.
  - Identifying the most efficient path for each SQL statement.
- Checking authorization.
- Producing the output from the bind process, a PACKAGE.
Notes:

- Validation checks on the existence of necessary DB2 objects, for example, tables, views, and so forth.
- With the resolution of table names, DB2 implicitly appends an unqualified table name with its owner's ID.
- Authority checking is performed to make sure that the BINDer or owner has the authority to create a program to access the data as required.
- Access path selection consists of evaluating a number of different access paths and calculating their costs. The cheapest one will be retained. Cost estimates are based primarily on statistics which are stored in the catalog. The RUNSTATS utility must be run to keep those statistics up-to-date.
- The executable access code is stored in the directory, and the source SQL statements are stored in the DB2 catalog. They may be needed if the access path is evaluated at some later time.
The Big Picture - Run

Figure 3-11. The Big Picture - Run

Notes:
Notes:

- When a database application is executed, SQL calls identify which package to use by passing the program identification string to the DBMS.

- Each statement is identified by a section number that is passed as part of the SQL statement API. This statement is located by the DBMS in its package definition and executed. The results of the API calls are sent back to the application program through the SQLCA.
3.2 Program Preparation - Additional Considerations
Notes:

- During PRECOMPILE, a consistency token is placed in both the modified source and the DBRM (z/OS) or bindfile (UDB). (The term timestamp is also used as a synonym for consistency token.) This token is necessary to ensure that the resultant load module and package were derived from the same original source. (If a bindfile is not created, the token is placed directly in the resultant package).

- Also at PRECOMPILE time, the name of the package that the executable load module will request is stored in the modified source module. By default, this will be the same as the name of the source module (the name excluding .sqc), truncated to eight characters.

- A consistency token contains a timestamp of when the precompile was done. It is often referred to as a timestamp, but we will use the term consistency token both to avoid ambiguity and for consistency with other platforms.

- If a package that matches the creator and name cannot be found, the application receives a -805 SQLCODE.
• If a package is found, the consistency token passed by the load module is compared to the token in the package. If the values match, the load module and package were created from the same original source and execution proceeds. If the tokens do not match, the application receives a -818 SQLCODE which indicates that the load module and package were created from sources created by different precompiles.

• Any changes to original source code require both SQL and non-SQL program preparation steps to be completed.

Why might you want to separate the precompile and bind activities and have them done by different individuals?

Answer: If you request a bind file at precompile time, certain object existence and authentication SQLCODES are treated as warnings instead of errors. This enables you to precompile a program and create a bind file without requiring that the referenced objects are present, or having to be authenticated. So, Jack would not need to have the authority to access the objects referenced in the SQL, as long as Jill did when she did the bind.
**Precompiling versus Binding**

![Diagram showing precompiling versus binding process in Linux, UNIX, and Windows environments.]

**Notes:**
- Unqualified references are qualified with the ID of the binder in the package definition.

**Linux, UNIX, and Windows**
- The full name of the package is JACK.MYAPP, even though JILL actually created the package by completing the BIND.
Adjusting Qualifier and Owner

Jill issues:

```
db2 bind myapp.bnd  qualifier U1  owner U2
```

- Unqualified SQL uses U1 schema
- U2 owns package
  - Can drop, rebind, grant privileges

Notes:

QUALIFIER provides an implicit qualifier for unqualified objects contained in the package. The default is the owner’s authorization ID, whether or not OWNER is explicitly stated.

OWNER designates an authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements in the package. The default is the primary authorization ID for the precompile/bind process if this option has not been explicitly specified.

Linux, UNIX, and Windows

- QUALIFIER and OWNER are CHAR(128) fields. QUALIFIER is stored in the DEFAULT_SCHEMA column of SYSCAT.PACKAGES. OWNER is stored in the BOUNDBY column of SYSCAT.PACKAGES.
Bind using QUALIFIER Option

Program coded:

```
EXEC SQL
SELECT LASTNAME FROM EMPLOYEE
```

BIND PACKAGE ... QUALIFIER PATTI

Program executes:

```
EXEC SQL
SELECT LASTNAME FROM PATTI.EMPLOYEE
```

Notes:

When unqualified references are used, the userid of the binder is used to fully qualify the reference. The QUALIFIER option of the BIND command may override this feature. In this example, the table EMPLOYEE is unqualified. If the user PAUL executes a bind on the package, the table that will be accessed will be PATTI.EMPLOYEE. The QUALIFIER option may be used to explicitly specify what schema will be used to qualify an unqualified reference.
Notes:

- When compiling an SQL statement, the SQL optimizer estimates the execution cost of different ways of satisfying your request.

- Based on this evaluation, the database manager selects what it believes to be the optimal access plan. An access plan specifies the order of operations required to resolve an SQL statement.

- When an application program is bound, a package is created. This package contains access plans for all of the static SQL statements in that application program.

If the BIND VALIDATE option is used and option RUN is specified, validation is attempted at bind time. If all objects do not exist, warning message is produced and the package is created.
**BIND Access Strategy**

```sql
SELECT * FROM TEMPL
WHERE SEX = 'F'
  AND ( JOBCODE = 54 OR SALARY > 1000 )
  AND WORKDEPT IN ('E11', 'D14')
```

- Which search columns have indexes?
- What is the I/O cost if an index is used? What if an index is not used?
- In which order should the tests be performed?

*** RUNSTATS utility produces statistics ***
*** used in developing access strategy ***

**Notes:**

- The DB2 BIND process produces a *package* based on the available statistics which describe the physical characteristics of the databases being accessed.
- RUNSTATS is a utility that updates statistics about tables and indexes. It must be run periodically in order to keep those statistics accurate.
- **GOOD STATISTICS ==> GOOD ACCESS STRATEGIES**
Implicit Rebinding

**Implicit rebinding** is automatic at next execution and triggered if:

- Drop object referenced in package
- Drop primary key on table referenced in package
- Drop/add referential constraint on parent or child table referenced in package
- Revoke **privilege required by owner** to execute static SQL statement embedded in package

### Packages catalog information:

<table>
<thead>
<tr>
<th>NAME or PKGNAME</th>
<th>OWNER or BOUNDBY</th>
<th>VALID</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

- **authorization id of package binder**

**Notes:**

- **Implicit rebinding** is available on DB2. It is totally automatic at execution time and is triggered under the above circumstances.

- When **VALID** contains an **N**, then an implicit rebind is triggered at execution time. When an implicit rebind is performed, the access paths for all the SQL statements are rebound. The owner authorization ID, which represents the ID of the original package binder or the ID specified as the **OWNER** on the bind, is checked for the appropriate authority and privileges to execute the embedded static SQL statements.

- During an implicit bind, if an object does not exist or a privilege has been revoked but not regranted, the execution of the package will fail.

- **VALID** is set to **N** when an index is dropped. The package will be implicitly rebound and **VALID** will equal **Y** even if the index has not been re-created if the package can be recreated using another access strategy that does not use the index.
- VALID is set to N when a table or view is dropped. The package will be implicitly rebound and VALID will equal Y only if the table or view has been re-created before the implicit rebind.

**Linux, UNIX, and Windows**

- VALID is set to X if the package is inoperative because some function instance that it depends on has been dropped. This would occur if the package were dependent on a user-defined function and that user-defined function were dropped.
Explicit Rebinding

- NEW INDEXES AVAILABLE
- STATISTICS IN CATALOG HAVE CHANGED
- CONTROL WHEN INVALID PACKAGES ARE BOUND

Notes:
- Rebinding is the process of re-creating a package for an application program that was previously bound.
- Invalid packages are automatically (or implicitly) rebound by the database manager when they are executed.
- Inoperative packages must be explicitly bound again (using the BIND command).
- You may choose to rebind a package to take advantage of a newly created index or make use of updated statistics after executing the RUNSTATS command.
- When an implicit or explicit rebind is done, the authorization for static SQL statements is based on the previous binder’s privileges.
## Privileges Required for Programming

<table>
<thead>
<tr>
<th>Action</th>
<th>Privileges Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a new package</td>
<td>BINDADD privilege&lt;br&gt;Privileges need to execute each static SQL statement&lt;br&gt;CREATEIN (z/OS) or PACKADM (z/OS)</td>
</tr>
<tr>
<td>Modifying an existing package</td>
<td>BIND privilege on package&lt;br&gt;Privileges need to execute each static SQL statement</td>
</tr>
<tr>
<td>Re-create an existing package</td>
<td>BIND privilege on package</td>
</tr>
<tr>
<td>Execute a package</td>
<td>EXECUTE privilege on package</td>
</tr>
</tbody>
</table>

### Notes:
- This chart summarizes the authorities and privileges needed to:
  - Create a new package (BINDADD).
  - Modify an existing package (BIND).
  - Re-create an existing package (REBIND).
  - Execute (run) a package.
- Simple references are qualified with the ID of the binder.
- For **dynamic applications**, the same privileges are required for preparing and executing programs. However, dynamically prepared statements are based on the authorization ID of the executing process. Therefore, to execute a dynamic statement, connect on the database, execute on the package, **AND** underlying SQL authority is required.
- **z/OS**
  - BINDADD is a privilege granted at the system level.
Linux, UNIX, and Windows

- BINDADD is a privilege granted at the database level.
- You will require CONNECT authority on a database to do any programming.
- **Note:** Binder **must** have authorization to perform all underlying SQL statements at bind time. The preparer is not required to have the underlying SQL authorizations to create a bindfile; warnings will be generated if the user ID doing the prepare does not have the authorizations to perform all underlying SQL statements. Also, all objects must exist in the database against which the prepare and against which the bind is being executed.
Protecting Resources Through Programs

Notes:

- An individual can EXECUTE a static PACKAGE without having the authority for the underlying SQL statements.
- Programs can be coded, tested, and installed in order to perform database tasks on behalf of the program executor.
- The authority to EXECUTE a program which performs some SQL statement does not imply authority to perform the same SQL in an ad hoc (dynamic) environment.
- PLAN is z/OS only. On z/OS only, bind the DBRM to a package and the package to a PLAN. On z/OS, the PLAN executes the package. In Linux/UNIX/Windows, execute the package.
3.3 UNIX and Windows Considerations
Precompile

Notes:

- PRECOMPILE or PREP are interchangeable. DB2 provides a one-pass precompiler.

- By default, when the program is precompiled, it is also bound to the database, creating a package directly in the database. It does not create a bind file that would require a subsequent bind step. This is called an inline bind.

- By default the precompile option produces two outputs:

  1. The precompiler comments the SQL statements from the source code, replacing them with host language calls. This is called the modified source file. The precompiler will determine but not attempt to clean up any syntax errors in the SQL statements. EXEC SQL WHENEVER and EXEC SQL INCLUDE statements are also resolved.

  2. The SQL statements are placed into a temporary bind file that is processed against the DB2 optimizer to create an access strategy which is stored in a package. This preparation step determines:

     - If the objects referenced in the embedded SQL exist.
- If the person doing the bind has the authority to access the objects.
- The access strategy itself.

- The package and the modified source file contain timestamps (also known as consistency tokens) indicating when they were produced by the precompiler. These timestamps will be inherited by all derivative files and objects, and must match at run time. If there is a change in any part of the code, you must start back at the precompile step, so that the resulting modified source file and bind file will have been produced by the same invocation of the precompiler.
**Notes:**

- When the BINDFILE option is used, it is referred to as a **deferred bind**. The BINDFILE precompile option produces two outputs:
  1. The precompiler removes the SQL statements from the source code, replacing them with host language calls. This is called the **modified source file**.
  2. The SQL statements are placed into a saved bind file. A package is not automatically created. A separate bind step must be run to produce a package.

- The bind file and the modified source file contain timestamps indicating when they were produced by the precompiler. These timestamps will be inherited by all derivative files and objects, and must match at run time. If there is a change in any part of the code, you must start back at the precompile step, so that the resulting modified source file and bind file will have been produced by the same invocation of the precompiler.

- If the embedded SQL accesses multiple DB2 databases, the bindfile must be bound against all databases. The program will have to be precompiled, compiled, and linked only one time. But it must be bound multiple times to produce several packages in
different databases. The timestamps on the packages must match the timestamp on the executable file. This is accomplished with the BINDFILE option.

- If the program will be bound to multiple databases, or if you don't want object existence or authentication errors to prevent the creation of the bindfile, the BINDFILE option should be used. This is not the default.
DB2 BIND REOPT Option

New choices for reoptimizing static and dynamic SQL statements using REOPT parameter to bind:

- **NONE** - Default estimates are used, based on estimates for host variables, special registers and parameters (Pre-V8.2 approach, and still the default)
- **ONCE** - Reoptimized with values when first used
- **ALWAYS** - Reoptimized every time that the statement is used

**Notes:**

The bind option REOPT enables reoptimization of static and dynamic SQL statements at run-time.

Specifically, the access path for an SQL statement containing host variables, special registers, or parameter markers is optimized using the values of these variables, rather than the default estimates determined by the compiler. This optimization takes place at query execution time when these values are available.

You can set the bind option REOPT to one of the following three values:

- **NONE** The values of any host variables, parameter markers, or special registers in an SQL statement are not used to optimize this statement's access path. The default estimates for these variables are used instead. This is the default behavior.
- **ONCE** When the statement is first executed, the access path for a given SQL statement is optimized using the actual values for any host variables,
parameter markers, or special registers. This access path will be used in all subsequent executions of the statement.

**ALWAYS** Each time that a statement containing host variables, parameter markers, or special registers is executed, the values of these variables are used to optimize the access path for the statement.
3.4 Final Summary
The Big Picture

![Diagram showing the process flow of Source, Modified Source, Libraries, Object file, Executable, DBRM/Bindfile, and Package.]

**Notes:**
Checkpoint

Exercise — Unit 3 Checkpoint

1. What command do you need to run against your source code before you compile it?

___________________________________________________________________

2. What option should you use on the precompile command so you can separately bind the program?

___________________________________________________________________

3. Should RUNSTATS be run before or after you bind a program on which you want optimum performance?

___________________________________________________________________
Unit Summary

Having completed this unit, you should be able to:

- Identify the additional steps necessary to prepare a program that contains embedded SQL for execution
- Describe the functions of the DB2 PRECOMPILE and BIND processes
- Describe factors relevant to the BIND process, including RUNSTATS positioning, package status, parameters, and authorization requirements

Notes:
Unit 4. Program Structure II

What This Unit Is About

An application program can only process the results of a select statement one row at a time, so it uses a mechanism called a cursor to move through the result set created by DB2 row by row.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Use DECLARE, OPEN, FETCH, and CLOSE CURSOR statements to handle select criteria that may return multiple rows in application programs
- Issue positioned UPDATE and DELETE statements
- (Linux, UNIX, and Windows) Use compound SQL

How You Will Check Your Progress

Accountability:

- Machine lab

References

**DB2 UDB for UNIX and Windows**

- SC09-4820 Administration Guide: Implementation
- SC09-4821 Administration Guide: Performance
- SC09-4822 Administration Guide: Planning
- SC09-4824 Administrative API Reference
- SC09-4825 Application Development Guide: Building and Running Applications
- SC09-4826 Application Development Guide: Programming Client Applications
- SC09-4827 Application Development Guide: Programming Server Applications
- SC09-4849 Call Level Interface Guide and Reference, Volume 1
<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC09-4850</td>
<td>Call Level Interface Guide and Reference, Volume 2</td>
</tr>
<tr>
<td>SC09-4828</td>
<td>Command Reference</td>
</tr>
<tr>
<td>SC09-4844</td>
<td>SQL Reference - Volume 1</td>
</tr>
<tr>
<td>SC09-4845</td>
<td>SQL Reference - Volume 2</td>
</tr>
</tbody>
</table>
Unit Objectives

After completing this unit, you should be able to:

- Use DECLARE, OPEN, FETCH, and CLOSE CURSOR statements to handle select criteria that may return multiple rows in application programs
- Issue positioned UPDATE and DELETE statements

Notes:
4.1 Using Cursors
Notes:

- There is no method to determine the number of rows that will satisfy the WHERE clause of a SELECT statement before actually receiving data from DB2. Therefore, it is not possible to allocate storage in the application program to receive an entire SET of data.

- DB2 provides the use of cursors to process SETS of data. The cursor is used to retrieve one row of data at a time. It is possible to allocate storage in the application program to receive one row of data at a time.

- Each FETCH of the cursor retrieves the next row in the SET of data that met the WHERE clause.
Structured Programming - Sequential File Read

1. ? →
2. ? →
3. ? →
   while (not EOF)
   {
   < process file >
   }
4. ? →
5. ? →

Notes:
Fill in the statements (or pseudocode) to read a file using structured programming techniques.

1.
2.
3.
4.
5.

(1) Define file access
(2) Open the file for reading
(3) Read the first record
(4) Read the next record
(5) Close the file
SELECT with FETCH - C Program

- **DEFINE A CURSOR**
  ```sql
  EXEC SQL
  DECLARE K9 CURSOR FOR
  SELECT EMPNO, LASTNAME
  FROM TEMPL
  WHERE DEPTNO = :dpt;
  ```

- **OPEN the CURSOR**
  ```sql
  EXEC SQL OPEN K9;
  ```

- **FETCH RESULT ROWS ONE AT A TIME**
  ```sql
  EXEC SQL FETCH K9 INTO :empno, :name;
  ```

- **CLOSE CURSOR when finished**
  ```sql
  EXEC SQL CLOSE K9;
  ```

**Notes:**

- The **DECLARE CURSOR** statement relates a CURSOR to a SELECT statement. The **DECLARE CURSOR** is not executable code. At DB2 PRECOMPILE time, the definition of the CURSOR is stored in the bindfile, and the statement is commented.

- The **OPEN CURSOR** statement generates executable code. This code will allow subsequent FETCH statements to access the SET of data that meets the definition of the DECLARE CURSOR's underlying SELECT statement. Values assigned to host variables are relevant at OPEN. Changing the values assigned to host variables after OPEN will not change the result set.

  **Note:** DB2 will not implicitly OPEN a cursor on the first FETCH.

- The FETCH statement retrieves a row of data from the SET made accessible by the OPEN. The FETCH statement is usually coded in a program loop until SQLCODE 100 is returned. This indicates that the entire SET of rows has been processed. Processing all the rows in a SET is not a requirement.
• The CLOSE CURSOR releases the CURSOR from the SET of data. The addressability to the SET of data is lost. The control blocks required by DB2 for the OPEN cursor are released.

Note: DB2 will close any OPEN cursors at program termination. However, it is recommended that the programmer explicitly CLOSE cursors that are no longer required unless the program will soon terminate.

• Comparing cursor logic to sequential file processing:
  - The SET of rows defined by the DECLARE CURSOR is analogous to a sequential file.
  - The OPEN CURSOR statement is analogous to OPENing a sequential file.
  - FETCHing a cursor is analogous to READing a sequential file. Therefore, the first FETCH is usually coded outside any loop processing, and a check for no rows returned is coded before execution of the loop. The next FETCH is coded at the end of the loop.
  - The CLOSE CURSOR statement is analogous to CLOSING a sequential file.
Answer Set Materialization

Notes:

- The rows of a results table may be derived at OPEN or during execution of subsequent FETCH statements.
  - If DB2 can avoid materialization and still provide the correct results set, it will usually derive the rows of the results set as they are FETCHED.
  - If a sort is required to satisfy the SQL statement, DB2 will retrieve the results set at OPEN and use a workfile for sorting purposes.
  - In the above example, assume there is no INDEX on NAME. DB2 must use a workfile to produce the results set in order to return the rows in the requested order. Subsequent FETCH statements will retrieve the rows from the workfile.
  - If an index did exist on the NAME column, DB2 MAY use the index to avoid the sort and the use of a workfile. In this case, the rows would be derived at FETCH.

EXPLAIN can be used to determine if sorts are required.

- If the temporary table is created, the S row locks for Cursor Stability applications are obtained and released during the OPEN of the cursor. FETCH from the temporary table...
may therefore present a row to the application that has since been changed in the underlying base table.
DELETE Via a Cursor - C Program

EXEC SQL DECLARE CE CURSOR FOR
SELECT . . . FROM TEMPL . . . ;

EXEC SQL OPEN CE ;

EXEC SQL FETCH CE INTO . . . ;

EXEC SQL DELETE FROM TEMPL
WHERE CURRENT OF CE ;

EXEC SQL CLOSE CE ;

Notes:

• This skeleton code demonstrates using a cursor to identify rows to be DELETED.
• The DECLARE, OPEN, FETCH, and CLOSE CURSOR logic is not changed.
• DELETE WHERE CURRENT OF CURSOR deletes the row presently identified by the cursor.
• This form of DELETE can be used if the data contained in a row is needed by the application before the row is to be DELETED. One requirement for such logic would be to generate a report of rows DELETED by the application.
• Reminder - simple DELETE statements can be embedded in application programs.
• Although not required by syntax, it is many times appropriate to use the FOR UPDATE OF clause on the declaration of a cursor that may be used for DELETE purposes. See the next visual for a discussion of this clause.

Note: The DELETE WHERE CURRENT OF CURSOR does not need to be executed against every row in the results set. Also, note that the FETCH statement must be executed after the DELETE in order to position the cursor on the next row.
UPDATE Via a Cursor - C Program

```
EXEC SQL
   DECLARE CX CURSOR FOR
      SELECT EMPNO, LASTNAME
      FROM TEMPL
      WHERE DEPTNO = :dpt
      FOR UPDATE OF LASTNAME;

EXEC SQL OPEN CX;

EXEC SQL FETCH CX INTO :empno, :name;

EXEC SQL UPDATE TEMPL
   SET LASTNAME = :newname
   WHERE CURRENT OF CX;

EXEC SQL CLOSE CX;
```

Notes:

- This skeleton code demonstrates using a cursor to identify rows to be UPDATED.

- The DECLARE CURSOR has an additional FOR UPDATE OF clause. The additional clause identifies columns that may be UPDATED via the cursor. This clause will also cause DB2 to use a different row locking strategy when rows are retrieved. This row locking strategy supports the possibility that a row or set of rows MAY be UPDATED via the CURSOR. The strategy, which employs U-locks, can minimize the possibility for deadlocks on rows.

  Note: The FOR UPDATE OF clause can also be specified for a CURSOR that is intended for DELETE WHERE CURRENT OF CURSOR. Specifying the clause will cause DB2 to employ the U-lock strategy which will minimize row deadlocking.

It is possible to use the Multivendor Integration Compliance Architecture (MIA) or SQL92E (ISO/ANS SQL92) SQL standard with DB2. If the MIA or SQL92E standard is used, the FOR UPDATE OF clause is not required but is allowed. However, if FOR UPDATE OF is not used, DB2 will not exploit U-locks.
• More information on U-locks is presented on a subsequent page.
• The OPEN, FETCH, and CLOSE CURSOR logic is not changed.
• UPDATE WHERE CURRENT OF CURSOR updates the row presently identified by the cursor.
• This form of UPDATE can be used if the results set is needed by the application and some or all of the rows in the result set need to be updated. One requirement for such logic is the need to UPDATE the first row in a result set that meets given criteria. The row UPDATED is arbitrary.

For example, assume a retail store sells television sets. A customer would like to purchase a given model. The underlying application may use a CURSOR to SELECT the available serial numbers for the model, but only the first available is assigned to the customer.

• Reminder - Simple UPDATE statements can be embedded in application programs.
UPDATE/DELETE Via a Cursor

-RESTRICTIONS-

DECLARE CURSOR statement may not contain "FOR UPDATE OF", nor may the cursor be used for "DELETE WHERE CURRENT OF", if the SELECT statement contains:

- ORDER BY
- GROUP BY
- DISTINCT
- Set operators (UNION)
- Function
- Join
- FOR FETCH ONLY/FOR READ ONLY

Notes:

- If the SELECT of a DECLARE CURSOR statement contains any of the components listed, it is a READ-ONLY CURSOR.
- The use of the syntax FOR UPDATE OF is not permitted in combination with syntax that defines a READ-ONLY CURSOR.
OPTIMIZE FOR n ROWS

Notes:
In some cases, the actual rows required from a query is only a subset of the whole result set of the query. In that case, the query can be optimized by specifying the expected (smaller) number of rows.

The OPTIMIZE FOR clause requests special processing of the SELECT statement. If the clause is omitted, it is assumed that all rows of the result table will be retrieved. If it is specified, it is assumed that the number of rows retrieved will probably not exceed n.

The value of n must be a positive integer, and it will determine a suitable communication buffer size to improve the performance of blocked cursors.

This clause does not limit the number of rows that can be fetched or affect the result in any other way than performance. Using OPTIMIZE FOR n ROWS can improve the performance if no more than n rows are retrieved, but may degrade performance if more than n rows are retrieved.

For example, assume an application issues the following statement:

```
  SELECT NAME, SALARY FROM EMPLOYEE ORDER BY NAME
```
Assume there is an index on NAME but the data is clustered according to a different index. The optimizer may choose a table scan followed by a sort as the best strategy to return all the data. However, if the application will only fetch the first 10 rows and display them to a terminal, it would be more efficient to use the index to avoid the scan + sort regardless of clustering. The application can be coded to indicate this intent to the optimizer by including the clause OPTIMIZE FOR 10 ROWS.

The OPTIMIZE FOR n ROWS clause should never be used in batch applications that will access the entire answer set.

The OPTIMIZE FOR n ROWS will also help in minimizing the amount of data transmitted across the network.
FETCH FIRST k ROWS ONLY

Notes:

When retrieving rows using a SELECT statement, you might also want to consider using the FETCH FIRST k ROWS ONLY clause. This clause sets the maximum number of rows that can be retrieved from within a SELECT statement. Limiting the result table to the first several rows can improve performance. Unlike OPTIMIZE FOR n ROWS, the database manager ceases processing the query once the specified number of rows have been retrieved when FETCH FIRST k ROWS ONLY is specified.

When the number of rows to be retrieved is small, there is no need to specify the OPTIMIZE FOR n ROWS clause in addition to the FETCH FIRST k ROWS ONLY clause. However, if k is large and you want to optimize by getting the first n rows quickly with a possible delay for the subsequent rows, specify both. The communication buffers are sized based on the lesser of n and k.
Cursor Manipulation

• POSITIONING
  OPEN →
  FETCH FETCH FETCH
  CLOSE

• To REUSE a CURSOR
  OPEN →

◆ COMMIT MAY CLOSE OPEN CURSORS
◆ CLOSE DOES *NOT* COMMIT

Notes:
• CLOSE CURSOR releases the cursor, and addressability to the buffer is lost.
• If a cursor is to be reused, it must be CLOSED before it is OPENED.
• COMMIT will CLOSE any CURSOR that has not been defined WITH HOLD. (See the next visual.)
• Connection makes an immediate connection to exactly one remote system.
• Connection (Type 2) does not release any current connections. More than one connection statement can be executed within the same unit of work.
• Connection (Type 1) only one connection statement can be executed within the same unit of work.
WITH HOLD Cursor

DECLARE E1 CURSOR WITH HOLD FOR
SELECT EMPNO, LASTNAME, PHONE, SALARY
FROM TEMPL WHERE WORKDEPT < :dpt
ORDER BY EMPNO

• Cursor held until:
  Close cursor statement
  ROLLBACK
  CONNECT  (if connect type 1)

• When commit is issued, open cursors declared WITH HOLD will remain open and associated table locks will be held. Row locks are released.

Notes:
• Applications that need to maintain position of a CURSOR across COMMIT points can take advantage of the WITH HOLD option. Cursors declared with this option remain OPEN through COMMIT points. Therefore, the next row in an answer set can be retrieved with FETCH following a COMMIT.
• Remember that cursors defined WITH HOLD will also cause a connection to be maintained across a COMMIT.
• BATCH applications and conversational BROWSE applications may be able to take advantage of this option.
• In pseudo-conversational CICS transactions, the task terminates when screens are conversed since control is returned to CICS. Therefore, the position of the cursor cannot be held.
• CONNECT will close all cursors associated with CONNECT TYPE 1 applications. However, distributed unit of work (DUOW) applications require explicit CLOSE of such cursors before an application can DISCONNECT from the database.
• If a change occurs that invalidates the package (for example, an object is dropped that the package has a dependency upon), when a commit occurs in the program, the next access of the package will cause the automatic rebind of the package. If following the commit, the program's logic was dependent on having the position of the cursor from the previous unit of work, code must be included to possibly reposition the cursor following the commit. If the package had to be rebound, the program will receive a SQLCODE of -501, indicating that the cursor is not open.
Cursor Repositioning

• REQUIREMENT:
  – UPDATE of a large number of rows, using a CURSOR. To avoid accumulating an excessive number of ROW LOCKS, issue a COMMIT after every 500 UPDATEs.

• Consider:
  – If restart capabilities must be included in the program, position of the cursor must be reestablished.

• SOLUTION:
  – After ABEND, reestablish position based upon a saved column value.

Notes:

• The WITH HOLD option does not eliminate the need to code programs with restart capabilities, particularly large BATCH applications.

• Cursor positioning may need to be reestablished after a screen map is conversed in CICS pseudo-conversational transactions.
Cursor Repositioning - C (1 of 2)

EXEC SQL
DECLARE C1 CURSOR
WITH HOLD FOR
SELECT X,Y,Z FROM T1
WHERE X> :storx
ORDER BY X;

EXEC SQL
SELECT VALUE INTO :storx
FROM RESTART;

CTR=0;

EXEC SQL
OPEN C1;

EXEC SQL
FETCH C1
INTO :storx,:story,:storz;

Notes:

• The WITH HOLD option eliminates the need to OPEN the CURSOR every time a COMMIT is issued. The WITH HOLD option may actually save rematerialization of answer sets in some cases and offer significant performance gains. (However, ensure that the duration of the BATCH job is not an issue since subsequent FETCHes are from a workfile and not the actual data table. If this is an issue, the technique of materializing a subset window described later could be used.)

• In this example, the UNIQUE INDEX on X would probably eliminate the need for materialization to satisfy the ORDER BY clause.

• The RESTART table is a one-row table that will contain an initial value or a restart value for X.

• CTR is a simple counter.

• The OPEN statement is based on the value of storx. If this is a normal start, the value obtained from the RESTART table would be the minimum possible value. If this is a restart, the value obtained would be the last value successfully processed.
• The FETCH obtains the first row in the ordered results set.
Cursor Repositioning - C (2 of 2)

while (SQLCODE == 0)
{
    if (some_condition)
    {
        EXEC SQL UPDATE T1
        {
            SET Y=:newy, Z=:newz
            WHERE X=:storx;
            ctr = ctr+1;
            if (ctr == 500)
            {
                EXEC SQL UPDATE RESTART
                {
                    SET VALUE=:storx;
                    EXEC SQL COMMIT;
                    ctr = 0;
                }
            }
        }
    }
    EXEC SQL FETCH C1 INTO :storx, :story, :storz;
}
EXEC SQL
    UPDATE RESTART SET VALUE = :MinValue;
EXEC SQL COMMIT;

Notes:

- If a condition for UPDATE is met, a simple UPDATE is executed and CTR is incremented by 1. Notice that the UPDATE is not based on the CURSOR C1. It is based on the current value of storx, which identifies the current row.

- If a condition for UPDATE is not met, the UPDATE and increment of CTR is bypassed.

- As soon as 500 rows have been updated, the value of storx is saved in the restart table, and a COMMIT is issued.

  The value saved will reestablish the position of the cursor in the event of a restart because the OPEN statement will be based on the saved value.

- The next row is FETCHed. The logic is repeated until all rows have been processed.

In Addition:

- Checks of the SQLCODE should be made after each SQL statement. They have been omitted here because of space limitations.

- It is possible to limit the scope of the WHERE clause to a range if materialization of the answer set is a concern. The cursor would not use the WITH HOLD option. Assume
limit = storx + some value. The WHERE clause could be coded as WHERE X BETWEEN :storx AND :limit. This would cause a subset window of rows to be defined instead of all the rows >= :storx.

- The value in the RESTART table would need to be set to the minimum possible value when the job completes, or other logic would need to be provided to distinguish a restart from a normal start.
4.2 Compound SQL - DB2 for Linux, UNIX, and Windows
Compound SQL

- Several SQL substatements in a single executable block
- Reduce DBM overhead
- Reduce network traffic
- No response return before
  - All indicated substatements have completed successfully
  - One sub-statement ends in error (unless NOT ATOMIC is used)

Notes:

- Compound SQL allows the application programmer to identify blocks of SQL that should be completed as a single statement. The programmer can indicate whether the entire set of substatements should be considered as one entity (ATOMIC) or whether certain substatement errors can be ignored (NOT ATOMIC).
- Compound SQL can be exploited in a local or client/server environment.
Compound SQL - C Program

EXEC SQL BEGIN COMPOUND ATOMIC STATIC

UPDATE SAVINGS SET
  BALANCE = BALANCE - :transfer
WHERE ATMCARD = :atmcard;

UPDATE CHECKING SET
  BALANCE = BALANCE + :transfer
WHERE ATMCARD = :atmcard;

INSERT INTO ATMTRANS
  (TTSTMP, CODE, AMOUNT) VALUES
  (CURRENT TIMESTAMP, :code, :transfer);

COMMIT;

END COMPOUND;

© Copyright IBM Corporation 2005

Notes:

• The example illustrates a simple use of compound SQL to transfer money from a savings account to a checking account and record the transaction in an audit table.

• The word ATOMIC designates that all statements must complete successfully, or that all statements within the compound will be rolled back if a single statement fails. The rollback will only apply to statements within the compound. Statements executed before entering the compound are not impacted by the implicit rollback. In the above example, the use of an EXPLICIT COMMIT is shown within the compound. This commit applies to the current unit of work, including any statements executed before entering the block.

• NOT ATOMIC could be indicated instead of ATOMIC. This specifies that, regardless of the failure of any substatements, the compound SQL statement will not undo any changes made to the database by the other substatements. Information in the SQLCA will indicate how many statements processed successfully and other information about errors detected.
• The word STATIC implies that the value of a variable referenced within the compound is established when the compound is invoked, and is not changed by any SQL within the compound. For example:

```sql
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
    UPDATE SAVINGS SET BALANCE = BALANCE - :transfer
    WHERE ATMCARD = :atmcard;
SELECT ACCOUNT INTO :atmcard WHERE CARDHOLDER = 'Cartwright';
UPDATE CHECKING SET BALANCE = BALANCE + :transfer
    WHERE ATMCARD = :atmcard;
INSERT INTO ATMTRANS (ACCOUNT, TTSTMP, CODE, AMOUNT)
    VALUES (:atmcard, CURRENT TIMESTAMP, :code, :transfer);
END COMPOUND;
```

• The value of :atmcard established before the compound was executed will be used for each UPDATE and the INSERT, not the value that corresponds to 'Cartwright'.

**Note:** Non-static behavior is not supported. This means that the substatements should not have interdependences as far as host variables are concerned.

• **STOP AFTER FIRST host_var STATEMENTS** can be coded on the BEGIN COMPOUND to allow the program to specify how many of the substatements should be executed when the compound is executed. For example:

```sql
EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC STOP AFTER FIRST :nbr STATEMENTS
    INSERT INTO TAB1 VALUES (:col1val1,:col2val1);
    INSERT INTO TAB1 VALUES (:col1val2,:col2val2);
    INSERT INTO TAB1 VALUES (:col1val3,:col2val3);
    INSERT INTO TAB1 VALUES (:col1val4,:col2val4);
    INSERT INTO TAB1 VALUES (:col1val5,:col2val5);
    INSERT INTO TAB1 VALUES (:col1val6,:col2val6);
    INSERT INTO TAB1 VALUES (:col1val7,:col2val7);
    INSERT INTO TAB1 VALUES (:col1val8,:col2val8);
    INSERT INTO TAB1 VALUES (:col1val9,:col2val9);
    INSERT INTO TAB1 VALUES (:col1val10,:col2val10);
    INSERT INTO TAB1 VALUES (:col1val11,:col2val11);
END COMPOUND;
```

The host variable :nbr can be set to the number of rows you wish to insert at execution time.

• If COMMIT is included, it must be positioned as the last statement. If a COMMIT is included as the last statement, it will be executed even if the STOP AFTER FIRST host_var STATEMENTS option indicates that less than all of the sub-statements should be executed.

A COMMIT is not permitted if CONNECT TYPE is 2.

• The SQLSTATE and SQLCODE are normally those for the last substatement executed within the compound. Other fields within the SQLCA are used in cumulative fashion for the entire compound, for example, the third element of SQLERRD will contain all rows affected by all statements within the compound. Consult the SQL Reference for other
usage of the SQLCA, such as determining multiple error information for a NOT ATOMIC execution.
Compound SQL Restrictions

- Only NOT ATOMIC supported with DB2 Connect
- Can only include static SQL statements
- Must be Embedded Static SQL

Notes:
The following statements **CANNOT** be included in a compound SQL statement.

- CALL
- CLOSE
- CONNECT
- Compound SQL
- DESCRIBE
- DISCONNECT
- EXECUTE IMMEDIATE
- FETCH
- OPEN
- PREPARE
- RELEASE
- ROLLBACK
- SET CONNECTION

In addition, if COMMIT is coded, it must be the last statement in the compound SQL.
Checkpoint

Exercise — Unit 4 Checkpoint

1. What do you need to declare in order to be able to retrieve multiple rows in a SELECT statement?

2. What is the statement used to get the next row in an answer set?
Unit Summary

Having completed this unit, you should be able to:

- Use DECLARE, OPEN, FETCH, and CLOSE CURSOR statements to handle select criteria that may return multiple rows in application programs
- Issue positioned UPDATE and DELETE statements

Notes:
Unit 5. Recovery and Locking Concepts

What This Unit Is About

Relational databases, including DB2, employ specific terminology for the objects they manage. Recovery units and referential integrity are concepts that application programmers need to understand.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Define a unit of recovery
• Identify the basic locking strategies used by DB2

How You Will Check Your Progress

Accountability:

• Machine lab

References

DB2 UDB for UNIX and Windows
SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849 Call Level Interface Guide and Reference, Volume 1
SC09-4850 Call Level Interface Guide and Reference, Volume 2
SC09-4828 Command Reference
SC09-4844 SQL Reference - Volume 1
Unit Objectives

After completing this unit, you should be able to:

• Define a unit of recovery
• Identify the basic locking strategies used by DB2

Notes:
5.1 Logical Unit of Work
Unit of Recovery

Notes:

- Successful execution of multiple SQL statements may be necessary to complete a logical unit of recovery.
- The application programmer must determine the logical unit of recovery.
- In the example above, assume the checking and savings accounts are stored in two tables. For a transfer of money from checking to savings to complete successfully, two updates must be done.

Both updates are considered the unit of recovery, also referred to as a unit of work.
Commit and Rollback

- Operates on a Unit of Recovery
  - COMMIT
    - Changes will be made permanent
    - Row Locks released
    - Certain Table Locks released
  - ROLLBACK
    - Changes backed out
    - All Locks released

Notes:

- If logical units of work are established by the programmer, it is possible for the database manager to maintain consistency within the database.
- If an application determines that a unit of work is successful, it can indicate such to the database manager via a commit. The database manager can make the changes permanent.
- If an application determines that a unit of work is not successful, it can indicate such to the database manager via a rollback. The database manager can reestablish the state of the database at the last complete unit of work.

COMMIT will not release table locks associated with cursors that have been declared WITH HOLD.
## Commit versus Rollback

<table>
<thead>
<tr>
<th></th>
<th><strong>COMMIT</strong></th>
<th><strong>ROLLBACK</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TSO or UNIX</strong></td>
<td>Normal prog. term. EXEC SQL COMMIT</td>
<td>Prog. abend</td>
</tr>
<tr>
<td></td>
<td>EXEC SQL ROLLBACK</td>
<td>EXEC SQL ROLLBACK</td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td>EXEC SQL COMMIT</td>
<td>Program termination (normal or abend)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXEC SQL ROLLBACK</td>
</tr>
<tr>
<td><strong>CICS</strong></td>
<td>Normal trans. term. SYNC POINT cmd.</td>
<td>Trans. abend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYNCPOINT ROLLBACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cmd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABEND cmd</td>
</tr>
<tr>
<td><strong>IMS</strong></td>
<td>Normal trans. term. CHKP, SYNC, or GU I/O PCB (single-mode)</td>
<td>Trans. abend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROLL or ROLB</td>
</tr>
</tbody>
</table>

### Notes:

- On the UNIX operating system, changes to database information are automatically committed at normal program termination, and rolled back on abnormal program termination. Programs running in TSO will also automatically commit at normal program termination, and roll back on abnormal program termination.
- DB2 on Windows is invoked when a Windows application terminates. However, the Windows operating system does not provide information on whether the termination was normal or abnormal. As a result, DB2 on Windows must roll the changes back. To prevent the roll back of changes when an application terminates normally, the application **must** commit all work before termination.
- Since the CICS or IMS transaction manager controls data synchronization between DB2 and non-DB2 data, the appropriate transaction manager statement must be used to complete or abandon work.
- The COMMIT and ROLLBACK statements will **FAIL** if issued in the CICS or IMS environments.
Note: CICS or IMS are used as an example transaction managers. You should specify the statements required by the transaction manager you are using to complete or abandon a unit of work.
## Controlling Transactions with Savepoints

A savepoint is a mechanism for undoing work done by the DBMS when a database request fails. Savepoints make non-atomic database requests behave atomically. If an error occurs during execution, the savepoint can be used to undo changes made by the transaction between the time the savepoint was started and the time the savepoint rollback is requested.

A savepoint allows you to group several SQL statements into a single executable block. Before the first substatement of the block is executed, a savepoint request to start a savepoint block is required. If any of the sub-statements end in error, only that substatement will be rolled back. At the end of a savepoint block of statements, you can either release the savepoint or rollback to the savepoint.

The following SQL statements enable you to create and control savepoints:

- **SAVEPOINT** - To set a savepoint, issue a SAVEPOINT statement. To improve the clarity of your code, you can choose a meaningful name for the savepoint. Options that can be specified include:

```sql
INSERT INTO MYTAB (C1, C2) VALUES (4,5);
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS;
INSERT INTO MYTAB (C1, C2) VALUES (2,3);
INSERT INTO MYTAB (C1, C2) VALUES (2,1);
INSERT INTO MYTAB (C1, C2) VALUES (245,5);
IF ...
  ROLLBACK TO SAVEPOINT S1;

SAVEPOINT S2 ON ROLLBACK RETAIN CURSORS;
INSERT INTO MYTAB (C1, C2) VALUES (2,300);
INSERT INTO MYTAB (C1, C2) VALUES (200,3);
COMMIT;
```

### Contents of MYTAB:

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
</tr>
</tbody>
</table>
- **UNIQUE** - the application does not intend to reuse this savepoint name while the savepoint is active.

- **ON ROLLBACK RETAIN CURSORS** - (required syntax) indicates that, whenever possible, the cursors are unchanged by a rollback to savepoint.

- **ON ROLLBACK RETAIN LOCKS** - Specifies system behavior upon rollback to this savepoint with respect to locks acquired after the setting of the savepoint. Locks acquired since the savepoint are not tracked and are not rolled back (released) on rollback to the savepoint.

**RELEASE SAVEPOINT** - To release a savepoint, issue a RELEASE SAVEPOINT statement. If you do not explicitly release a savepoint with a RELEASE SAVEPOINT statement, it is released at the end of the transaction.

**ROLLBACK TO SAVEPOINT** - To rollback to a savepoint, issue a ROLLBACK TO SAVEPOINT statement. The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint.

- If the savepoint contains DDL on which a cursor is dependent, the cursor is marked invalid.

- Otherwise, if the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result table.

- Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT.

All locks are retained after a ROLLBACK TO SAVEPOINT statement.

All LOB locators are preserved following a ROLLBACK TO SAVEPOINT.
Savepoint Restrictions

- Cannot be included in triggers
- No limit to number of savepoints in a transaction
- (UNIX/Windows) Cannot be nested

Notes:

DB2 UDB places the following restrictions on your use of savepoints in applications:

- DB2 does not support the use of savepoints in triggers.
- DB2 enables you to set and use as many savepoints as you require within a transaction.
- (Linux, UNIX, and Windows) DB2 does not support the use of a savepoint within another savepoint.
- (Linux, UNIX, and Windows) DB2 does not enable you to use savepoints within atomic compound SQL. You cannot use atomic compound SQL within a savepoint.
Savepoint Considerations

- If savepoint contains DDL and executes ROLLBACK TO SAVEPOINT, cursors dependent on DDL are invalid.

- ROLLBACK TO SAVEPOINT issued:
  - Acts like ROLLBACK WORK on NOT LOGGED INITIALLY table defined during savepoint.
  - Will drop declared temporary table defined during savepoint.

- ROLLBACK TO SAVEPOINT will not impact content of local copy of rows kept for blocked cursor; subsequent fetch may retrieve inserted rows which were rolled back.

Notes:

DB2 enables you to include DDL statements within a savepoint. If the application successfully releases a savepoint that executes DDL statements, the application can continue to use the SQL objects created by the DDL. However, if the application issues a ROLLBACK TO SAVEPOINT statement for a savepoint that executes DDL statements, DB2 marks any cursors that depend on the effects of those DDL statements as invalid.

You can issue a CLOSE statement to close invalid cursors. Other actions against an invalid cursor (such as FETCH, OPEN, UPDATE WHERE CURRENT OF, or DELETE WHERE CURRENT OF) will return negative SQLCODEs.

If your application uses savepoints, consider preventing cursor blocking by precompiling or binding the application with the precompile option BLOCKING NO. While blocking cursors can improve the performance of your application by prefetching multiple rows, the data returned by an application that uses savepoints and blocking cursors may not reflect data that has been committed to the database.
If you precompile the application using blocking, and your application issues a FETCH statement after a ROLLBACK TO SAVEPOINT has occurred, the FETCH statement may retrieve rows which were inserted during the savepoint and subsequently rolled back.

**Linux, UNIX, and Windows**

Within savepoints, DB2 treats tables with the NOT LOGGED INITIALLY property and temporary tables as follows:

- Within a savepoint, you can create a table with the NOT LOGGED INITIALLY property or alter a table to have the NOT LOGGED INITIALLY property. For these savepoints, however, DB2 treats ROLLBACK TO SAVEPOINT statements as ROLLBACK WORK statements and rolls back the entire transaction.

- If a temporary table is declared within a savepoint, a ROLLBACK TO SAVEPOINT statement drops the temporary table. If a temporary table is declared outside a savepoint, a ROLLBACK TO SAVEPOINT does not drop the temporary table.

If your application takes advantage of both buffered inserts and savepoints, DB2 flushes the buffer before executing SAVEPOINT, RELEASE SAVEPOINT, or ROLLBACK TO SAVEPOINT statements.

If there are any active savepoints in an application when an XA compliant transaction manager issues an XA_END request, DB2 issues a RELEASE SAVEPOINT statement.
Data Recovery (1 of 2)

Notes:

- Backups are maintained by database administration for the data in a DB2 system.
- A backup on z/OS may be of the entire DB2 subsystem, data sharing group or table space. In Linux/UNIX/Windows, a backup may be of a table space or database. The key is that a starting point for recovery needs to be established by the database administrator. Further details regarding backup are beyond the scope of this class.
- As applications issue statements that change the database, the DB2 database manager makes entries into the DB2 log noting the changes.
Notes:

- The above diagram illustrates basic DB2 utility recovery.

- In z/OS, for certain types of failure, such as a disk failure on the volume that contains a database, the database manager may use the backup copies and the logs to reestablish the database to the last committed unit of work. In z/OS, the Recovery utility can be used to start the recovery process by copying the backup image for the table space or data set. After the Recovery utility has restored the backup/copy, any committed changes on the log since the last copy will be applied by the Recovery utility.

- For certain types of failure, such as a media failure on the volume that contains data from table spaces, the database manager may use the backup copies and the logs to reestablish the table spaces within a database to the last committed unit of work. The RECOVERY utility can be used to start the recovery process by copying the backup/copy for the table space or data set. After the RECOVERY utility has restored the backup/copy, any committed changes on the log since the last copy will be applied by the RECOVERY utility.
• In Linux/UNIX/Windows, other types of failure will not require the RESTORE / ROLLFORWARD utilities to be executed. However, the concept is the same; the last committed unit of work for z/OS/Linux/UNIX/Windows will be recovered during the restart of DB2.
5.2 Locking
Attributes of Locks

- Size/Object
- Mode
- Duration

Notes:

Database manager locks have the following basic attributes:

**Size** - The scope or level of the lock. The size describes the amount of data controlled. The possible sizes of locks are table space, table, partition (z/OS only), page (z/OS only) and row.

**Mode** - The type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked object. It is sometimes referred to as the state of the lock.

**Object** - The resource being locked. The only types of explicitly lockable objects are tables. The database manager also imposes locks on other types of resources, such as rows, tables and table spaces. The object being locked represents the granularity of the lock.

**Duration** - The length of time a lock is held. Lock durations are affected by isolation levels.
Notes:

- The above visual illustrates the locking strategies that DB2 uses for access to data.
- DB2 always requires a table lock before access to data is permitted. DB2 may also require a table space lock, but this lock is generally not of concern to the application programmer. It will not be discussed in this course any further.
- DB2 may lock only the table (sometimes called strict table locking) or may lock the table and either rows or pages.
  - If the locking strategy does not include row or page locking, the table is locked with a mode that applies to the entire table. The entire table will be locked at the same level of restriction.
  - If the locking strategy includes row or page locking, the table is locked before locking of requested rows or pages. The table is locked at a lesser degree of restriction than the strategy that does not use row or page locking.

DB2 for Linux, UNIX, and Windows, may acquire row locks in addition to table locks. DB2 for z/OS may acquire either page locks or row locks in addition to table locks. A LOCKSIZE
parameter on the table space identifies what size locks will be acquired. LOCKSIZE ROW defined on the table space will cause DB2 for z/OS to acquire row locks; LOCKSIZE PAGE or LOCKSIZE ANY will cause DB2 for z/OS to acquire page locks.

The mode of the lock (whether it will support READ or CHANGE) has been intentionally omitted from the discussion at this point.
## Lock Modes

### Table Lock Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Intent Share</td>
</tr>
<tr>
<td>IX</td>
<td>Intent Exclusive</td>
</tr>
<tr>
<td>S</td>
<td>Share</td>
</tr>
<tr>
<td>U</td>
<td>Update</td>
</tr>
<tr>
<td>X</td>
<td>Exclusive</td>
</tr>
</tbody>
</table>

### Row or Page Locking

Row or page locking is also used.

### Strict Table Locking

- **Row or Page Lock**
  - S - Share
  - U - Update
  - X - Exclusive

- **Minimum Supporting Table Lock**
  - S - IS
  - U - IX
  - X - IX

---

**Notes:**

### Table Lock Modes

- The lock modes listed at the top of the graphic are some of the ones used by DB2 at the table level:
  - **IS - Intent Share:** The lock owner can read any data in the table if an S lock can be obtained on the target rows or pages.
  - **IX - Intent Exclusive:** The lock owner can read or change any data in the table provided an X lock can be obtained on rows or pages to be changed and a U or S lock can be obtained on rows or pages to be read.
  - **S - Share:** The lock owner can read any data in the table and will not obtain row or page locks.
  - **U - Update:** The lock owner can read any data in the table and may change data if an X lock on the table can be obtained. No row or page locks are obtained.
  - **X - Exclusive:** The lock owner can read or update any data in the table. Row or page locks are not obtained.
• The modes of **IS, and IX** are used at the table level to **SUPPORT** row or page locks. They permit row-level or page-level locking while preventing more exclusive locks on the table by other applications.

• The following examples are used to further clarify the lock modes of **IS, and IX**:

  - An application obtains an IS lock on a table. That application may acquire a lock on a row or page for read only. Other applications can also **READ** the same row or page. In addition, other applications can **CHANGE** data on other rows in the table.
  
  - An application obtains an IX lock on a table. That application may acquire a lock on a page for change. Other applications can **READ/CHANGE** data on other pages in the table. Note: page locks would only be acquired in a DB2 for z/OS database if LOCKSIZE PAGE or LOCKSIZE ANY is specified; row locks would be obtained in a DB2 for Linux, UNIX, and Windows database.

• The modes of **S, U, and X** are used at the table level to enforce the strict table locking strategy. **No row-level or page-level locking** is used by applications that possess one of these modes.

• The following examples are used to further clarify the lock modes of **S, U, and X**:

  - An application obtains an S lock on a table. That application can read any data in that table. It will allow other applications to obtain locks that support read-only requests for any data in the entire table. No application can **CHANGE** any data in the table until the S lock is released.

  - An application obtains a U lock on a table. That application can read any data in that table, and may eventually change data in that table by obtaining an X lock. Other applications can only **READ** data in the table.

  - An application obtains an X lock on a table. That application can read and change any or all of the data in the table. No other application can access data in the entire table for **READ** or **CHANGE**.

* Denotes an exception to a given application scenario. Applications that utilize Uncommitted Read can read rows or pages that have been changed.

**Row or Page Lock Modes**

The bottom of the graphic shows row or page lock modes. The definitions are similar to those for corresponding table locks, except that the object of the lock is a row or a page.

• **S - Share**: The row or page is being **READ** by one application and is available for **READ ONLY** by other applications.

• **U - Update**: The row or page is being **READ** by one application but is possibly to be changed by that application. The row or page is available for **READ ONLY** by other applications. The major difference between the U lock and the S lock is the **INTENT TO UPDATE**. The U lock will support cursors that are opened with the **FOR UPDATE OF** clause. Only one application can possess a U lock on a row or page.

• **X - Exclusive**: The row or page is being changed by one application and is not available for other applications, except those that permit Uncommitted Read.
Row locks or page locks are only requested by applications that have supporting locks at the table level. These supporting locks are the INTENT locks such as IS, and IX.

**Minimum Supporting Table Lock** - Denotes the least restrictive lock necessary. However, this does not imply that the table lock listed is the only table lock that supports the row or page lock listed. For example, an application that possesses an IX table lock could possess S, U, or X locks on rows or pages.
Table Lock versus Row or Page Locking

Notes:

• The diagram above illustrates the degree of concurrency permitted among different locking strategies. The applications on the left side of the diagram are not concurrently accessing the target table due to the global nature of the lock obtained by the first application. A strict table lock can be requested by an application by issuing one of the following statements:

  EXEC SQL LOCK TABLE xxx IN SHARE MODE
  EXEC SQL LOCK TABLE xxx IN EXCLUSIVE MODE

  The lock requested is **S** if the first statement is issued and **X** if the second table is issued.

  It may be appropriate for batch applications to issue LOCK TABLE statements for performance reasons. The application that obtains a strict table lock will not incur the overhead of obtaining row locks. The cost is concurrency of other applications.

• The applications on the right side of the diagram are concurrently accessing the target table. The **Intent** strategies used (IX and IS) permit such concurrency. Any locking
conflict occurs at the row or page level. This strategy is typical for transaction environments as well as batch programs that are designed to commit periodically and support restart.
Notes:

- The isolation level chosen for an application can impact both the lock strategy and the duration of row or page locks. The isolation level can be specified during program preparation or bind. Cursor stability is the default.

- This visual details the impact of isolation level on row or page lock duration. For the following notes, assume that row locks are being obtained and that the rows shown in the sample table are being accessed through a query.

  - **Read Stability (RS):** Read row locks are held until the next commit or rollback. If a row is read multiple times within a unit of work, the value for the row will not change. However, the result set can change if a cursor that has been processed is subsequently re-opened with the same search criteria within a unit of work. This is because Read Stability does not prevent new rows from being added to the result set via insert or update activity.

    This isolation level reduces concurrency at the row level, but may be useful for applications that require examination of multiple rows before a processing decision can be made.
- **Repeatable Read (RR):** Read row locks are held until the next commit or rollback. If a row is read multiple times within a unit of work, the value for the row will not change. Also, the result set will not change if a cursor that has been processed is subsequently re-opened using the same search criteria within a unit of work.

This isolation level reduces concurrency at the row level, but may be useful for applications that require examination of multiple rows before a processing decision can be made and also requires the result set to be static if the same criteria is used to open a cursor a second time within a unit of work.

- **Cursor Stability (CS):** Read row locks are held only while the cursor is positioned on the row. Rows that have been previously examined within a unit of work are not locked.

This isolation level increases concurrency at the row level, and it is normally preferred for most application requirements.

- **Uncommitted Read (UR):** Read row locks are not obtained when single rows are read or when READ-ONLY cursors are OPENED and FETCHED. A cursor is READ-ONLY if:

  — The result table for the select statement is read-only. For example, the select statement contains an ORDER BY clause. (Other syntax constructs that cause a result table to be read-only are documented in the *SQL Reference*.)
  — The cursor is declared with FOR FETCH ONLY or FOR READ ONLY.

For non read-only cursors, the locking behavior is the same as cursor stability.

This isolation level may improve performance of applications since waits for read row locks are eliminated. However, the application programmer must be aware of any data integrity risks associated with this isolation level because rows that have X locks on them to support change activity of other applications will be read.

- Isolation level applies only to rows that are read. Rows that are changed require an X lock to be acquired by the application. **X locks are not released until rollback or commit, regardless of the application's isolation level.**

**Note:** If temporary tables are needed to satisfy a given request and LONG FIELD or BLOB data is accessed, the isolation level will be designated as Repeatable Read, regardless of the bind parameter. Also, if a temporary table is needed to support an application using Cursor Stability, the read row locks are obtained and released during the OPEN of the cursor. FETCH from the temporary table may therefore present a row to the application that has since been changed or deleted in the underlying base table. Temporary tables may be used for read-only cursors that require sort activity. The use of temporary tables is externalized via the EXPLAIN function.

Repeatable Read (RR) and Read Stability (RS) both lock all the rows an application retrieves within a unit of work.

- Using repeatable read, a SELECT statement issued by an application twice within a unit of work gives the same result each time. No other applications can update, delete, or
insert a row that would affect the result table until the unit of work completes. Every row that is referenced is locked, not just the rows that are retrieved. This prevents phantom rows from occurring. This means that if you scan 10,000 rows and apply predicates to them, all 10,000 rows are protected from change, even though only 10 rows qualify.

- Using read stability, if you scan 10,000 rows but only 10 rows qualify, only those 10 rows are protected from change. If your application issues the same query more than once, you may see additional phantom rows. A phantom row can occur in the following situation:

1. Application 1 reads the set of n rows that satisfy a search condition.
2. Application 2 then inserts one or more rows that satisfy the search condition.
3. Application 1 reads the set of rows again with the same search condition and obtains both the original rows and the rows inserted by application 2.

z/OS

The ACQUIRE and RELEASE options

Effects: The ACQUIRE and RELEASE options of bind determine when DB2 locks an object (table, partition, or table space) your application uses and when it releases the lock. (The ACQUIRE and RELEASE options do not affect page, row, or LOB locks.) The options apply to static SQL statements, which are bound before your program executes.

Options and their effect

ACQUIRE(ALLOCATE) Acquires the lock when the object is allocated. This option is not allowed for BIND or REBIND PACKAGE.

ACQUIRE(USE) Acquires the lock when the object is first accessed.

RELEASE(DEALLOCATE) Releases the lock when the object is deallocated (the application ends). RELEASE(COMMIT) Releases the lock at the next commit point, unless there are held cursors or held locators. If the application accesses the object again, it must acquire the lock again.
WITH RR/RS/CS/UR

**Notes:**

The WITH isolation-level clause allows you to override the isolation level of the plan/package.

If you combine the WITH isolation level with the FOR UPDATE clause, the latter will take precedence and the isolation level will be upgraded to CS.

The WITH (followed by isolation level) clause can be added to a simple SQL SELECT, INSERT, UPDATE or DELETE statement. Also the SET (CURRENT) ISOLATION command can be used on a session level. Both of these add further flexibility.
U-Lock Usage

```
DECLARE CX CURSOR FOR
SELECT
    EMPNO, LASTNAME
FROM TEMPL
WHERE
DEPTNO=:DPT
FOR UPDATE
OF . . .
```

**Notes:**

- The use of U-locks prevents two invocations of the same transaction from deadlocking on the same row.
- The intention for update prevents two invocations from examining the same row at the same time since both invocations have stated a possible need to obtain an X-lock through the cursor definition.
- The LANGLEVEL precompiler option of MIA or SQL92E can be used to permit cursor-positioned updates against cursors that have not been declared FOR UPDATE OF. However, use of this technique may lead to row deadlocking.
5.3 Connection Considerations (Linux, UNIX, and Windows)
Prep to Bind

prep myapp.sqc bindfile using mybind.bnd
package using mypack

Notes:
- It is generally recommended to use the defaults for bindfile and package names.
CONNECT (Type 1) and CONNECT (Type 2)

- CONNECT (Type 1)
  - Single database per unit of work
  - Remote Unit of Work

- CONNECT (Type 2)
  - Multiple databases per unit of work
  - Distributed Unit of Work

Notes:

- A unit of work is a single logical transaction. It consists of a sequence of SQL statements in which either all of the operations are successfully performed or the sequence as a whole is considered unsuccessful.

- DB2 will support two types of connections. The connection type will determine how an application can work with remote databases and whether it can work with more than one database at a time.
Remote Unit of Work - Type 1 Connect

Figure 5-19. Remote Unit of Work - Type 1 Connect (Linux, UNIX, Windows)

Notes:

- Remote unit of work lets a user or application read or update data at one location per unit of work. It supports access to one database within a unit of work. While an application program can access several remote databases, it can only access one database within a unit of work.

- Remote unit of work has the following characteristics:
  - Multiple requests per unit of work are supported.
  - Each unit of work can access only one database.
  - The application program either commits or rolls back the unit of work. In certain error circumstances, the server may roll back the unit of work.
Notes:

- Distributed unit of work (DUOW) allows an application to access more than one database within a unit of work; that is, the application can switch between databases before committing the data. This gives an application programmer the ability to do work involving multiple databases, local and remote, at the same time.

- This is sometimes referred to as Application-Directed Distributed Unit of Work.

- You can use DUOW to read and update multiple DB2 databases within a unit of work. If you have installed the DDCS product, you can also use DUOW with other DRDA application servers, such as DB2 for z/OS.

- A transaction manager coordinates the commit among multiple databases. If you use a transaction processing (TP) Monitor environment such as CICS/6000, the TP Monitor uses its own transaction manager.

- The Transaction Manager supplied with DB2 can be used to coordinate DUOW transactions if all resource managers are DB2 databases.
Releasing A Connection

**Notes:**

- RELEASE specifies a connection to be released upon successful COMMIT.

  **Note:** The release of connections at COMMIT is also affected by a precompile option that will be discussed later.

  Arguments to RELEASE are ALL, CURRENT, a specific database name, or a host variable containing a specific database name.

- DISCONNECT indicates that connections should be released. It is only valid outside of a logical unit of work.

  Arguments to DISCONNECT are ALL, CURRENT, a specific database name, or a host variable containing a specific database name.

  **Note:** A ROLLBACK will not release connections marked for release.
Multisite Update Wizard

Notes:

A Multisite Update Wizard is available in DB2 UDB. This tool assists an administrator in configuring and testing the multisite update environment.

Invoke the Wizard from the instance icon. Right-click the instance from the pop-up menu and select Multisite Update.

Use the Configure Multisite Update Wizard to configure an instance to perform a multisite update. You configure whether a sync-point manager is required for a multisite update. You use the Wizard in cases when an application uses multiple databases, where it is critical that distributed data be consistent, and where the best method to ensure the data’s uniformity is to update it simultaneously.

You can use the Test for:

- Before using the Configure Multisite Update: confirm if the multisite update environment is configured properly.
- After using the Configure Multisite Update: confirm that the environment has been configured properly.
Connection-Oriented Precompile Options

**Notes:**

- Command options:
  - CONNECT - Specifies whether Remote Unit of Work (1) or Distributed Unit of Work (2) should be supported.
  - DISCONNECT - Specifies how database connections should be handled at COMMIT.
  - SQLRULES - Specifies whether CONNECT will be allowed to reestablish a dormant connection (DB2) or if SET CONNECTION TO will be required (STD).
  - SYNCPOINT - Specifies how commits or rollbacks are coordinated among multiple database connections.
Syncpoint Considerations

**Notes:**

- The requirements of the application will impact the choice for the precompiler option of SYNCPOINT.
- SYNCPOINT ONEPHASE entails less overhead. It should be used if the needs of the application are satisfied.
Multisite Bind

myapp.bnd

EXEC SQL
   CONNECT TO DALLAS;

EXEC SQL
   DELETE FROM TABLE1
       WHERE ID=:hv;

EXEC SQL
   CONNECT TO LA;

EXEC SQL
   UPDATE TABLE2
       SET PAY=PAY* 1.10;

EXEC SQL
   COMMIT;

EXEC SQL
   BIND
       DB2 DALLAS
       TABLE1
       TABLE2

EXEC SQL
   BIND
       DB2 LA
       TABLE1
       TABLE2

Notes:

• An application that accesses multiple databases must be bound on each database.

• All objects referenced by the application must be defined on each database. The actual data within an object may only exist at one site.

QUESTION - What could you do to avoid creating definitions of empty objects to support applications that must connect to multiple sites?
Checkpoint

Exercise — Unit 5 Checkpoint

1. If an application has an IS lock on the table level, what kind of activity is it doing to the rows of the table?

___________________________________________________________________
Unit Summary

Having completed this unit, you should be able to:

- Define a unit of recovery
- Identify the basic locking strategies used by DB2

Notes:
Unit 6. Dynamic SQL Introduction

What This Unit Is About

This unit describes the differences between static and dynamic SQL and the programming techniques used for dynamic SQL.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Describe the difference between static and dynamic SQL
- List the types of dynamic statements
- Code dynamic SQL in a program

References

DB2 UDB for UNIX and Windows
SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849 Call Level Interface Guide and Reference, Volume 1
SC09-4850 Call Level Interface Guide and Reference, Volume 2
SC09-4828 Command Reference
SC09-4844 SQL Reference - Volume 1
SC09-4845 SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

- Describe the difference between static and dynamic SQL
- List the types of dynamic statements
- Code dynamic SQL in a program

Notes:
6.1 Embedded Dynamic SQL
Types of Dynamic SQL Statements

Supporting code varies depending on whether:

1. The statement returns data

2. The statement contains incomplete search values

Types

★ Non-SELECT statement
  • Complete
  • Parameterized

★ SELECT statement
  • Fixed SELECT list
  • Varying SELECT list

★ UNKNOWN statement

Notes:

Dynamic SQL programs read the DML statements that manipulate the database from an external source (a file, a terminal, and so forth), or program logic is used to formulate the statement. These statements are not available when the program is prepared for execution, so the BIND process cannot create a PACKAGE for the SQL statement. The BIND for the DML statements must be dynamic, that is, completed during the actual execution of the program.

Note: SQL statements that support the preparation and/or execution of externally supplied or program formulated SQL statements are included in the source code and must be processed through a BIND step. The syntax and function of these dynamic statements will be discussed in subsequent visuals.

The above visual illustrates the relative complexity of the different types of dynamic SQL statements.

SELECT statements will have to be treated differently from all the other statements because they return data, and the program must be made ready to receive the data. Since
the columns are unknown at BIND time, storage will have to be allocated to receive the column data at execution time. For the NON-SELECT statements, this isn't necessary.

Parameterized statements require more complex coding techniques than Complete statements because the parameters must be assigned values before the execution of the statement.

Of course, there are also cases where you simply don't know which statement type you will have to execute. This will be the last case discussed.
Complete Non-SELECT Statement

Notes:
This is the easiest case. Once the statement is known, it is prepared and executed. Preparing a statement means asking DB2 to find the access path (like a run-time BIND). EXECUTE IMMEDIATE performs both functions at once.
Complete Non-SELECT Statement - C

char stmt [255];

read SQL statement into stmt

EXEC SQL EXECUTE IMMEDIATE :stmt;

check SQLCA

Notes:
Easy, isn't it?

• The dynamic BIND for a single statement will complete syntax and authorization checks. If both checks are successful, the prepared statement will be built.

• EXECUTE IMMEDIATE :stmt causes the text within stmt to be processed by a dynamic BIND, and the resulting prepared statement to be EXECUTEd.

Note: :stmt is used in the above example as a host variable. It should therefore be declared within an SQL DECLARE section. This is not shown on the visual for space considerations.

• The prepared statement created as a result of EXECUTE IMMEDIATE is not stored; however, it can be cached. If the same statement is issued, a recompilation is avoided. If not, the new statement is compiled.

• The SQLCA remains an important communication vehicle between DB2 and the application. It must be checked after executable SQL statements are processed.
**Note:** Checks of the SQLCA are not shown in every example of this unit due to space considerations.
Parameterized Non-SELECT Statement

**Notes:**

Sometimes, a statement needs to be executed several times with different values for the column values at each execution. An example is the SQL statement, UPDATE table WHERE EMPNO = value, where table is given to the program in the beginning followed by the different values for value. In order to execute this statement, you need two steps:

- First, prepare the completed statement (completed with the table name but with value still unknown) to create an access path.
- Then, execute the program providing a value at each execution.

Note that a dynamic SQL statement cannot contain a host variable.
Parameterized Non-SELECT Statement - C

char stmt[255];
char x[4];
char y[11];

read statement into stmt

DETERM FROM EMP
WHERE DEPTNO=?
AND HIREDATE=?

EXEC SQL PREPARE PRST FROM :stmt;
check SQLCA

get values for X and Y

X D17  Y 2000-11-03

EXEC SQL EXECUTE PRST USING :x, :y;
check SQLCA

Notes:
A little bit more complicated but still very easy, isn't it?

- The PREPARE statement causes the contents of :stmt to be processed by a dynamic BIND. If the BIND is successful, the result is a prepared statement that can be referenced later by the name PRST.
- Each prepared statement with a unique name is cached until a new statement is prepared using the same statement name in the application.
- The values for the parameter markers are provided positionally on the EXECUTE statement.
SELECT Statement with Fixed List - C

```
char stmt[255];
char x[11];
char name[16];
char phone[5];

EXEC SQL INCLUDE SQLCA;
read statement into stmt

stmt

EXEC SQL DECLARE C1 CURSOR FOR PRST;
EXEC SQL PREPARE PRST FROM :stmt;
get value into X
EXEC SQL OPEN C1 USING :x;
while SQLCODE <> 100
    EXEC SQL FETCH C1 INTO :name, :phone;
process
EXEC SQL CLOSE C1;
```

---

**Notes:**

If at program preparation time, the number of columns that will be accessed and their data types are known, dealing with a parameterized SELECT statement is not any more complicated than with static SQL.

Note again that where you have a host variable in static SQL, you must use a question mark (parameter marker) in dynamic SQL.

All languages that support DB2 also support this technique.
SELECT Statement with Varying List

**Notes:**

This is where it gets harder.

Since you don't know which columns will be accessed, you cannot pre-allocate any storage for host variables until you know that information. This will be the first thing to do.

If you have any idea **how many** columns you have to access, it will simplify your task.

In order to know the data types and names of the columns, you will have to ask DB2 to DESCRIBE the table you access. DB2 needs storage (SQLDA) to be able to do this DESCRIBE. If you know how many columns you will need, you can at least allocate enough storage for DB2 to be able to do the DESCRIBE. Once this is done, you can allocate storage for the host variables and execute the statement.
Unit Summary

Having completed this unit, you should be able to:

- Describe the difference between static and dynamic SQL
- List the types of dynamic statements
- Code dynamic SQL in a program

Notes:
Unit 7. Managing Test Data

What This Unit Is About

This unit describes the setup of a test data environment. Subjects such as loading data, collecting statistics for BIND and maintaining tables using REORG are covered.

What You Should Be Able to Do

After completing this unit, you should be able to:

• Identify methods to insert data into a table
• Use the LOAD or IMPORT utility
• Identify the purpose of the RUNSTATS utility
• Identify the purpose of the REORG utility

References

DB2 UDB for UNIX and Windows
SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849 Call Level Interface Guide and Reference, Volume 1
SC09-4850 Call Level Interface Guide and Reference, Volume 2
SC09-4828 Command Reference
SC09-4844 SQL Reference - Volume 1
SC09-4845 SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

• Identify methods to insert data into a table
• Use the LOAD or IMPORT utility
• Identify the purpose of the RUNSTATS utility
• Identify the purpose of the REORG utility

Notes:
7.1 Managing Test Data
Single Row INSERT

<table>
<thead>
<tr>
<th>PROJNO</th>
<th>PROJNAME</th>
<th>DEPTNO</th>
<th>RESPEMP</th>
<th>PRSTAFF</th>
<th>PRSTDATE</th>
<th>PRENDATE</th>
<th>MAJPROJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP2011</td>
<td>SCP SYSTEMS SUPPORT</td>
<td>E21</td>
<td>000320</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-02-01</td>
<td>OP2010</td>
</tr>
<tr>
<td>OP2012</td>
<td>APPLICATIONS SUPPORT</td>
<td>E21</td>
<td>000330</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-02-01</td>
<td>OP2010</td>
</tr>
<tr>
<td>OP2013</td>
<td>DB/DC SUPPORT</td>
<td>E21</td>
<td>000340</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-02-01</td>
<td>OP2010</td>
</tr>
</tbody>
</table>

INSERT INTO PROJ

(Projno, Projname, Deptno, Majproj, Resemp)
VALUES ('MA2114', ' ', 'B01', ' ', '')

INSERT INTO PROJ
VALUES ('MA2114', ' ', 'B01', ' ', NULL, NULL, NULL, '')

Notes:

Once you have created your test tables, you must populate them with test data. An obvious way is to use singleton INSERTs, but this isn't feasible for large amounts of data.
Multiple Row INSERT

TESTPROJ

<table>
<thead>
<tr>
<th>PROJNO</th>
<th>PROJNAME</th>
<th>DEPTNO</th>
<th>RESPEMP</th>
<th>PRSTAFF</th>
<th>PRSTDATE</th>
<th>PRENDATE</th>
<th>MAJPROJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL2100</td>
<td>WELD LINE PLANNING OPERATION</td>
<td>B01</td>
<td>000020</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-01-01</td>
<td>MA2100</td>
</tr>
<tr>
<td>OP1010</td>
<td></td>
<td>E11</td>
<td>000090</td>
<td>5.00</td>
<td>1994-01-01</td>
<td>1994-01-01</td>
<td>OP1000</td>
</tr>
<tr>
<td>OP2011</td>
<td>SCP SYSTEMS SUPPORT</td>
<td>E21</td>
<td>000320</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-01-01</td>
<td>OP2010</td>
</tr>
<tr>
<td>OP2013</td>
<td>DB/DC SUPPORT</td>
<td>E21</td>
<td>000340</td>
<td>1.00</td>
<td>1994-01-01</td>
<td>1994-01-01</td>
<td>OP2010</td>
</tr>
<tr>
<td>MA2114</td>
<td></td>
<td>B01</td>
<td>000330</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OP2010</td>
</tr>
</tbody>
</table>

INSERT INTO TESTPROJ
(PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, MAJPROJ)
SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, MAJPROJ
FROM PROJ
WHERE DEPTNO LIKE 'B%' OR DEPTNO IN ('E11', 'E21', 'E31')

Notes:
If you want to copy data from one table to another, you can use the multiple row insert.
(For a self-referencing table, this is not allowed (RI restriction). You will have to find ways around it.)

Multiple rows can be inserted using the VALUES clause of the insert statement on Linux/UNIX/Windows as shown in the following example:

```
INSERT INTO TESTPROJ
VALUES ('AB2114', 'B01', 'NULL', 'NULL', 'NULL', 'NULL');
```

Multiple rows cannot be inserted using the above syntax on z/OS. You can only insert 1 row at a time using the values clause on z/OS. For example, on z/OS you would need two insert statements as follows:

```
INSERT INTO TESTPROJ
VALUES ('AB2114', 'B01', 'NULL', 'NULL', 'NULL', 'NULL');
```

© Copyright IBM Corporation 2005
Data Movement Utilities

Figure 7-4. Data Movement Utilities

Notes:

- **Load** can be used in either DB2 for z/OS or DB2 for Linux, UNIX, and Windows.
- The **Import** utility is available in DB2 for Linux, UNIX, and Windows, and might be a simpler option to insert data into tables.
Notes:

The IMPORT utility may be used to insert data from an input file into a table, with the input file containing data from another database or spreadsheet program.

The EXPORT utility may be used to copy data from a table to an output file for use by another database or spreadsheet program. This file can be used to load tables, providing a convenient method of migrating data from one database to another.

The IMPORT and EXPORT utilities may be used to move data between databases which exist on DB2 Universal Database platforms.

The IMPORT and EXPORT utilities may be used to move data between DB2 and DRDA host databases if DB2 Connect is installed.

Import and Export File Formats

- **DEL** - Delimited ASCII is commonly used for storing data that separates column values with a special delimiting character.
• **ASC** - Non-delimited ASCII may be used for importing data from other applications which create flat text files with aligned column data. ASC may not be used with **EXPORT**.

• **WSF** - Work-Sheet Format is used for exchange with products such as Lotus 1-2-3 and Symphony.

• **IXF** - PC version of the Integrated Exchange Format. This is the preferred method for exchange within the database manager. Use PC/IXF to export data from a table so it can be imported later into the same table or another table.

For IXF data file formats, the table does not need to exist before beginning the import. The table can automatically be created when the data is imported. For DEL, WSF, and ASC data file formats, the table including its column names and data types must be defined before the file can be imported.

For **EXPORT** and **IMPORT**, you can select LOB column types and have the data stored in separate files if the **MODIFIED BY** option LOBSINFILE is specified. The same path as the data file will be used.
Import Utility Example

```
db2 connect to musicdb
db2 import from artexprt of ixf
  messages artmsg [create insert insert_update replace replace_create]
  into artists
.....
db2 import from artexprt of ixf
  messages ...
```

- Imports data from a file to a database table.
- Check message for error or warning messages.

**Notes:**

There are several IMPORT options available to load a table.

- **CREATE** - if table does not exist already, the table is created using PC/IXF file information before the data is inserted.
- **INSERT** - data is inserted into an existing table without changing the existing table data.
- **INSERT_UPDATE** - add rows of imported data to the target table or updates existing rows (of the target table) with matching primary keys.
- **REPLACE** - data in the existing table is deleted before the new data is inserted.
- **REPLACE_CREATE** - old data in an existing table is deleted before the new data is inserted. If the specified table does not already exist, then the table is created using PC/IXF file information before the data is inserted.

**MESSAGES** with no message file specified will result in messages written to standard output.
**COMMITCOUNT n** performs a commit every n records. This option can help to keep log sizes manageable. **RESTARTCOUNT n** specifies that an import is to be started at record n + 1. The first n records are skipped.

**METHOD L** specifies the start and end column numbers from which to import data. This option must be used for ASC files. **METHOD N** specifies the names of the columns to be imported. **METHOD P** specifies the order of column numbers to be imported.

To import using INSERT, you must have SYSADM, DBADM, or CONTROL on the table, or INSERT and SELECT on the table. To import to an existing table using INSERT_UPDATE, REPLACE, or REPLACE_CREATE, you must have SYSADM, DBADM, or CONTROL on the table.

To import to a table that does not exist using CREATE or REPLACE_CREATE, you must have SYSADM, DBADM, or CREATETAB privilege and either IMPLICIT_SCHEMA or CREATEIN privilege.

When using the IMPORT utility, REPLACE and REPLACE_CREATE functions are not allowed if the object table has any dependents other than itself.

The success of importing into a table with self-referencing constraints depends on the order in which the rows are inserted.

The MODIFIED BY option allows specification of additional information. Documentation of specific options is provided in the Command Reference. The MODIFIED BY options allow you to specify different items depending on the filetype being created. For example, for delimited output data, you can specify the character string delimiter and the column delimiter. Another option which is not tied to filetype is **COMPOUND n**, which allows n statements to be sent as a block using non-atomic compound SQL. The COMPOUND value ranges from 1 to 100. COMPOUND can greatly reduce the overhead associated with importing data.

The export command can be used to export data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement.

Filetypes that are supported include:

- **DEL** (delimited ASCII format) which is used by a variety of database manager and file manager programs.
- **WSF** (work sheet format) which is used by programs such as Lotus 1-2-3 and Lotus Symphony.
- **IXF** (integrated exchange format, PC version) in which most of the table attributes, as well as any existing indexes, are saved in the IXF file, except when columns are specified in the SELECT statement. With this format, the table can be recreated, while with the other file formats, the table must already exist before data can be imported into it.

Refer to the *Command Reference* for details.
Notes:

The purpose of RUNSTATS is twofold:

- Provide the statistics for BIND/prepare (keyword ACCESSPATH) and for other performance related issues.
- Provide the DBA information about space and freespace usage.
Figure 7-8. REORG

**Notes:**

The REORG utility is used to compact data, to reorganize the data within the table to eliminate fragmentation and overflow rows, to reestablish free space which may have been used by insert processing, and to rebuild indexes into a compact format.
Unit Summary

Having completed this unit, you should be able to:

- Identify methods to insert data into a table
- Use the LOAD or IMPORT utility
- Identify the purpose of the RUNSTATS utility
- Identify the purpose of the REORG utility
Unit 8. Performance Considerations

What This Unit Is About

In an SQL statement, you specify what data you want from the DB2 system. It is DB2 which decides how this data will be accessed from the database.

Normally, if you are creating a query and using it only once, you do not need to know what kind of access DB2 has used. But, for queries that are embedded in an application program, it is important that you understand what kind of access DB2 will use to satisfy the SQL request.

Therefore, as an application programmer responsible for retrieving or changing data in a DB2 database, you should consider performance issues related to the database. By knowing such issues, you will be more likely to write code that executes efficiently. Sometimes, analysis of SQL statements will identify data model changes that are required. In that case, you and your DBA will need to work together to come up with a solution.

What You Should Be Able to Do

After completing this unit, you should be able to:

- Use programming techniques that enhance DB2 application performance by following general guidelines, using indexable predicates, and avoiding unnecessary sorts
- Identify the access paths available to DB2
- List common causes of deadlocks and avoid such causes when possible
- Use the EXPLAIN tools as aids to develop applications that emphasize performance

How You Will Check Your Progress

Accountability:

- Paper lab
- Machine lab
- Checkpoint questions
References

**DB2 UDB for UNIX and Windows**

SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
SC09-4824 Administrative API Reference
SC09-4825 Application Development Guide: Building and Running Applications
SC09-4826 Application Development Guide: Programming Client Applications
SC09-4827 Application Development Guide: Programming Server Applications
SC09-4849 Call Level Interface Guide and Reference, Volume 1
SC09-4850 Call Level Interface Guide and Reference, Volume 2
SC09-4828 Command Reference
SC09-4844 SQL Reference - Volume 1
SC09-4845 SQL Reference - Volume 2
Unit Objectives

After completing this unit, you should be able to:

- Use programming techniques that enhance DB2 application performance by following general guidelines, using indexable predicates, and avoiding unnecessary sorts
- Identify access paths available to DB2
- List common causes of deadlocks and avoid such causes when possible
- Use the EXPLAIN tools as aids to develop applications that emphasize performance

Notes:
8.1 Index Concepts
DB2 Optimizer

Notes:

- When compiling an SQL statement, the SQL optimizer estimates the execution cost of different ways of satisfying your request.
- Based on this evaluation, the database manager selects what it believes to be the optimal access plan. An access plan specifies the order of operations required to resolve an SQL statement.
- When an application program is bound, a package is created. This package contains access plans for all of the static SQL statements in that application program.
Index Concepts

Notes:

- In concept, indexes are used by DB2 in much the same way as indexes in books are used by people. The index contains two major parts - the value of interest and a pointer that identifies where the corresponding row can be found in the table. This identifier is known as the RID, which is an acronym for Row ID.

- The index is structured in ascending or descending sequence on one or more columns. A given value of interest can be located quickly in the index because of this ascending or descending structure. This again is similar to searching the index in the back of a book. For example, assume you need to look in the index of a book for any information concerning configuration. Because the index is ordered, you need not scan through the entire index. You can quickly find if any entries match your search criteria by finding the c's in the index and applying your knowledge of the alphabet.
Typical Access Paths

- Relation scan
- Matching index scan
- Index-only access
- Non-matching index scan

Notes:

The optimizer in DB2 must determine an access path for every executable SQL statement. To do this, it looks at several things, such as the catalog statistics. It takes into consideration how large a table is, whether there is an index on the table, and so forth.

You are capable of predicting and influencing the access path by the way you write your SQL statement.
Relation Scan

Notes:
This access path is referred to as a relation scan (table space scan in z/OS; table scan or relation scan in Linux, UNIX, and Windows). All data must be read.
Matching Index Scan / Range Delimiting

SELECT LASTNAME, FIRSTNME, PHONENO
FROM PHONEBOOK
WHERE LASTNAME LIKE 'S%'

Notes:

In this example, DB2 can use the index on PHONEBOOK to access the data. This is a matching index scan because DB2 exploits the tree structure of the index; that is, it reads the root page and only the part of the tree needed to find the row. DB2 knows where to begin its search in the index; it does not have to scan the entire index.

In this type of search, DB2 must know the leading part of the indexed key. In addition, since DB2 is using a multicolumn index, the leading column (LASTNAME) is very important. If the leading column is not specified in the SQL query, a matching index scan is not used.

In this example, there is one matching column because one column (LASTNAME) can be used to limit the range of the search through the index. Because the SELECT requests information not in the index, DB2 uses the index information to access the data pages.

- z/OS: Matching Index Scan
- Linux, UNIX, and Windows: Range-Delimiting Predicate
Index-Only Access

SELECT FIRSTNME
FROM PHONEBOOK
WHERE LASTNAME LIKE 'S%'

Notes:
This is a matching index scan in which the SQL request can be satisfied without accessing the data pages. All of the required data is available in the index.

This type of access minimizes:

- The number of I/Os
- Data transfer
- CPU cost

For DB2 to choose this plan, all of the predicates must be able to be evaluated via the index, and only index columns must be selected.

In some cases, performance can be improved by including columns in the index data which is not used to access the data rows, thus exploiting index-only access.
Non-Matching Index Scan

SELECT LASTNAME, FIRSTNME, PHONENO
FROM PHONEBOOK
WHERE FIRSTNME = 'ABE'

*INDEX ON LASTNAME, FIRSTNME*  
© Copyright IBM Corporation 2005

**Notes:**

In this strategy, DB2 does not exploit the tree structure of the index. It only reads the leaf pages to find the requested data rows. The reason that DB2 uses this strategy is because the leading column of the index was not included as a search criteria on the query.

In this example, DB2 can use the (LASTNAME, FIRSTNME) index, but it cannot use the FIRSTNME column to limit its search. It must start at the beginning of the index and search through the index contents. This is not as efficient as a matching index scan, but it may be faster than a relation scan. Since the index should be a smaller physical object than the table, it should be faster to access the data through an index search (as long as the percentage of rows that satisfy the predicate is not large - which would require most of the data pages to be accessed anyway).

In this case, this query is not index-only, because the information selected is not all found in the index. However, if the query was:

```
SELECT LASTNAME, FIRSTNME FROM PHONEBOOK WHERE FIRSTNME = 'ABE'
```
then all of the predicates (one) can be evaluated using the index, and only index columns are selected. This would be an example of Index-Only Non-Matching Index Scan.

- z/OS: Non-Matching Index Scan
- Linux, UNIX, and Windows: Index-SARGable Predicate
## Access Path Exercise

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>FIRSTNME</th>
<th>MID</th>
<th>RID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abell</td>
<td>Jim</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Cambell</td>
<td>Debbie</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Cheng</td>
<td>David</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Ann</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Bernie</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>David</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Debbie</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Dennis</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Diane</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Donald</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Donna</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Doreen</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Free</td>
<td>Doug</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Klonne</td>
<td>Dennis</td>
<td>L</td>
<td></td>
</tr>
</tbody>
</table>

1. **SELECT** *
   FROM PHONEBOOK
   WHERE
   LASTNAME = 'Free'

2. **SELECT** LASTNAME
   FROM PHONEBOOK
   WHERE
   FIRSTNME = 'David'

3. **SELECT** *
   FROM PHONEBOOK
   WHERE
   LASTNAME = 'Free' AND FIRSTNME > 'Diane'

4. **SELECT** *
   FROM PHONEBOOK
   WHERE
   LASTNAME = 'Free' AND MID > 'J'

### Notes:

Determine whether the given queries will use a matching (range-delimiting) or non-matching (index-sargable) index scan. If matching, determine how many columns can be used to limit the index search. Also determine if the query can use index-only access.
Clustered and Non-Clustered Indexes

**Notes:**

- An index is structured as a B+ tree. This structure enables the database manager to find values in the index rapidly.
- The root node is at the top of this structure. In the illustration, the root node would contain an entry for each non-leaf, or intermediate node. The entry in the root node consists of the high value contained on the intermediate node and a pointer to this node.
- The intermediate nodes are similar in structure to the root node, except that the range of values addressed is more specific. The intermediate node contains an entry for each of the leaf nodes addressed. The entry consists of the high value contained on the leaf node and a pointer to this leaf node.
- The leaf nodes contain the value/RID pairs themselves. The leaf nodes collectively address the entire table.
• The visual above is only one example of index structure. It is possible that an index contains more intermediate levels. Such an index would be required if the number of leaf nodes could not be addressed by one level of intermediate nodes.

• Some insight regarding the structure of an actual index can be obtained by examining SYSCAT.INDEXES. The column NLEVELS indicates the number of root, intermediate, and leaf node levels. The column NLEAF indicates the number of leaf pages.

• The degree to which the data rows correspond in order to the values on the leaf nodes is regarded as the clustering of the data. Therefore, the concept of clustering is only applicable when discussing a table and a given index.

• The index on the left side of the visual illustrates a clustering index because the rows in the data pages are physically ordered according to the index sequence. The index on the right is not a clustering index. The rows in the data pages do not seem to correspond to the order of the index entries.

• During REORG, an index can be identified by which to cluster the data. If no index is specified on the REORG, and the table has a clustering index defined, DB2 will order the data based on the clustering index. Clustering benefits range type searches, such as BETWEEN or EQUAL for a non-unique key. This is due to a reduction in I/O costs.

• INSERT, UPDATE, and DELETE activity can change the degree of clustering. The CLUSTERRATIO column of SYSCAT.INDEXES will indicate the degree of clustering for each index defined on the table if non-detailed index statistics are gathered. Detailed statistics will use the columns CLUSTERFACTOR and PAGE_FETCH_PAIRS, which provides a finer measurement of clustering.
8.2 General Guidelines
General Guidelines - Avoid Unnecessary Conversions

Notes:

- Unnecessary conversions cause additional processing and should be avoided whenever possible.
- If system-defined Referential Integrity (RI) is used, foreign keys should be identical to the primary key referenced in terms of data type, size, and scale.
- If the columns are not identical, DB2 may be unable to use indexes to check referential constraints.
- It is the responsibility of the CREATOR of tables (usually the DBA) to ensure that columns used in JOINS will not cause conversions.
- Note that use of a character string in C language that has an additional byte for the null terminator in comparison with the character column in DB2 will not force a conversion routine to be invoked.
General Guidelines - Select What You Need

Explicitly SELECT only the columns to be used

Notes:

- Each column requested from DB2 will require additional processing. Make sure that no unused columns are selected.
- SELECT * in application programs is not usually desirable.
  - Implicit coding does not provide documentation for the maintenance programmer.
  - The asterisk is resolved at BIND. Therefore, SELECT * is dependent on the catalog description of the table or view at BIND. Changes to the table or view can cause warnings after REBIND for a package with SELECT *. How these warning are handled by the application may be an issue.
General Guidelines - Minimize DB2 Sorts

EXEC SQL

A FEW KEYSTROKES

I/O

CPU

Notes:

• Don’t request sequencing unless it is necessary. The ease of coding the ORDER BY clause sometimes leads to misuse.

  Sorting increases CPU requirements and possibly additional I/O to DB2’s sort work areas.

• Allow DB2 to consider using an INDEX to avoid a sort. Data must be accessed in sequence to support DISTINCT, GROUP BY, ORDER BY, UNION, EXCEPT, INTERSECT, and certain JOINS. If an INDEX that contains the appropriate columns is available, DB2 may access the data using the INDEX and avoid an internal sort.
General Guidelines - Use Views

Notes:

**USE, BUT DON'T ABUSE VIEWS!**

- The above view could be created via the following statement:

  ```sql
  CREATE VIEW name AS SELECT EMPNO, DEPTNAME
  FROM TEMPL, TDEPT
  WHERE WKDEPT = DEPTNO
  ```

- Assume that WKDEPT and DEPTNO are indexed.

  Programmers, DBAs, and end users would not need to be concerned about the join criteria and the results of the view could be obtained quickly via the INDEXes. The possibilities of Cartesian product joins are minimized. In this manner, a view could be considered a performance aid.

  However, if the requirement is to obtain information that can be found in only a single table, the use of the above view would cause unnecessary processing to join the two tables. A programmer should be aware of the definition of the objects being referenced so that views are not used arbitrarily.
• In general, coding alternatives that emphasize performance may be defined as views by knowledgeable programmers for common table accesses. All individuals using such views would benefit from the expertise of the performance experts.
**Notes:**

- Most transactions, queries and small batch jobs favor the intent table locking with row locking strategy in order to maximize concurrency.

- Large batch jobs that access a large percentage of the table may benefit from a more restrictive locking strategy. If the LOCK TABLE statement is issued, an S or X lock is obtained at the TABLE level and no row locking is required. The overhead of obtaining and releasing row locks is eliminated. The cost is concurrency.

- The lock requested by the LOCK TABLE statement is acquired at statement execution.

- The lock acquired by the LOCK TABLE statement is released at:
  - ROLLBACK
  - COMMIT, if the lock is not supporting a cursor declared WITH HOLD.

- The LOCK TABLE statement provides the application programmer with the flexibility to override row locking for a given process.
Notes:

- A static statement that references host variables is optimized without knowledge of the actual value of the host. Default filter factors are used. At times, the default filter may be too optimistic. Range predicates are especially sensitive to this problem. For example, consider an application with the predicate LASTNAME BETWEEN :host1 AND :host2. Assume an index on LASTNAME is chosen to support this predicate, based on default filtering. If :host1 is set to Abell and :host2 is set to Brown during execution, using the index would be appropriate. However, if :host1 is set to Abell and :host2 is set to Williams during execution, using the index would be less efficient than a table scan.

- Dynamic SQL is bound and optimized during execution and all values are considered during the selection of access strategy. More accurate filter factors can be used. An access strategy that is specific to the current execution of the statement can be selected if a complete statement is provided during preparation.
General Guidelines - Scalar Functions

- RANGE DELIMITING INDEX ON SERIAL NOT USED

```sql
EXEC SQL
SELECT CUSTINFO
FROM SALES
WHERE SUBSTR (SERIAL, 1, 4) = '1333'
```

- RANGE DELIMITING INDEX ON SERIAL CONSIDERED

```sql
EXEC SQL
SELECT CUSTINFO
FROM SALES
WHERE SERIAL LIKE '1333%'
```

**Notes:**

- SCALAR functions are considered expressions. Therefore, the use of a SCALAR function may not allow an index to be considered.

- This restriction also applies to using SCALAR functions against host variables within an SQL statement. For example:

  ```sql
  WHERE INDCOL = SUBSTR (:HV, 1, 10)
  ```

- Many application languages provide data manipulative functions that could be applied BEFORE execution of the SQL statement.

- As a general rule, an application programmer should investigate alternatives to coding expressions in the SQL statement WHERE clause. Following this rule may also aid in enhancing performance of SQL that is executed by database servers with optimizers that are not as sophisticated as the optimizer provided with DB2.
General Guidelines - Statements

- Use local predicates
- Write joins if equivalent to subquery
- Remember UNION ALL

Notes:

- Local predicates can allow the database manager to limit the number of rows that are used in multiple-step access paths. They can be extremely significant in determining which table to access first in join processes.
- In general, joins can outperform subqueries since there are many access strategies available to support the join. If you have logically equivalent join and subquery statements, choose the join.
- UNION ALL is more efficient than UNION. UNION ALL does not require sorting to eliminate duplicate results. If duplicate results are desired or do not impact other parts of the application adversely, UNION ALL should be exploited.
General Guidelines - Exploit Blocking (1 of 2)

Non-Blocking Operation

```
OPEN_CURSOR
FETCH
FETCH
FETCH

:  
CLOSE_CURSOR
```

generate result set*

return 1 row
return 1 row
return 1 row

:  

destroy result set

Client

Server

Notes:

- A non-blocking operation entails a high communication expense.
- Each FETCH request is sent from the client to the server, and one row is returned for each request until the answer set is exhausted.

Note: The cursor may or may not be materialized at the server. However, the concept of non-blocking operation is applicable to either case.
General Guidelines - Exploit Blocking (2 of 2)

**Notes:**

- A blocking operation reduces the communication expense.
- When record blocking is used, a single FETCH request from the client will result in a multiple row transfer between the server and the client. The results are fetched one at a time from a buffer allocated at the client.
- After the client buffer is emptied, the next FETCH will cause an additional block of rows to be returned from the server.
- The application program logic is not changed to enable or use blocking, with the possible exception of cursor declarations. However, other factors will determine when blocking is used. These other factors are the subject of the following pages.

**Note:** The cursor may or may not be materialized at the server. However, the concept of blocking operation is applicable to either case.

- To favor blocking:
  - Use FOR FETCH ONLY or FOR READ ONLY on DECLARE of cursors that you intend to use for read-only.
- Specify BLOCKING ALL when preparing or binding an application.

- Use the OPTIMIZE FOR clause to specify the number of rows that you expect from the output of a SELECT statement. This clause affects the number of rows that are blocked in the communication buffer.
What Is a Deadlock?

UNIT OF WORK -
DELETE SOME CEREAL AND MILK

Notes:

A deadlock occurs when applications cannot complete a unit of work due to conflicting lock requirements that cannot be resolved until the unit of work is completed.

The visual illustrates the concept of deadlocks. The unit of work that both application A and application B need to complete before committing is to get a bowl of cereal with milk. For the sake of simplicity, assume there is only enough milk and cereal left for a single bowl. (Another way for the scenario to work is to assume the milk represents a single row and the cereal represents a single row.)

1. Application A obtains an X lock on the cereal.
2. Application B obtains an X lock on the milk.
3. Application A wants an X lock on the milk but cannot obtain it until application B commits.
4. Application B wants an X lock on the cereal but cannot obtain it until application A commits.

Neither application can proceed to a commit point.
This type of deadlock, caused by accessing objects in reverse order, can be reduced by establishing rules of access at an installation for highly used objects. In this particular example, if both applications access the cereal first, followed by the milk, the first application to process would obtain an X lock on the cereal and prevent the other from continuing at that point. The application possessing the X lock on the cereal could proceed to get the X lock on the milk and complete the unit of work.

Deadlocks are handled by a background process called the **deadlock detector**. If a deadlock is detected, a victim is selected, then the victim is automatically rolled back and returned a negative SQL code (-911). Rolling back the victim releases locks and should allow other processes to continue.
General Guidelines - Reduce Deadlocks

- Repeatable Read
- Lock Escalation
- Lock Conversion
- Inconsistent Access
- Catalog Modification
- Referential Constraint Enforcement

Notes:

Although some deadlocks will occur in systems that provide data integrity and the capability to change multiple items within a unit of work, there are certain conditions that are more prone to causing deadlocks.

Repeatable Read Isolation Level

An application running in repeatable read continues to hold S locks on all rows that are referenced until the unit of work is completed. This greater number of locks increases the chance for another application to require a resource held by the repeatable read application. If such an application holds a non-compatible lock that the repeatable read application requires, a deadlock occurs.

Relational Scans Under Repeatable Read

A repeatable read application that must perform a relational scan will use strict table locking. Such a strict lock will be required on each table scanned. If more than one table is accessed via the relational scan during a unit of work, the conditions are prime for deadlock. For example, assume application A obtains an S lock on table W to support a...
relational scan. Application B is presently updating data in table Y using an IX table lock and X row locks. At this point, application A attempts to acquire an S lock on table Y. It cannot proceed due to the IX lock posted by Application B. Application B attempts to update one row via a unique index in table W. It requires an IX lock on the table. Since application A holds an S lock on the table, the request for the IX lock causes a deadlock.

**Lock Escalation for an Application**

If an application is using repeatable read, or is changing a large number of rows, it may run into the limit on the number of locks allowed for an application. Lock escalation causes the application to obtain a higher level table lock, potentially on multiple tables.

**Lock Conversion**

This case was addressed with regard to U-locks with cursors.

**Inconsistent Access**

This cause of deadlock was highlighted on the previous visual. In general, these types of deadlocks can be minimized if a consistent protocol for modifying different objects is followed.

**Modifying System Catalog Tables**

There are three ways in which catalog tables are modified:

- DDL (creating, dropping, or altering tables, indexes, views or plans)
- Authorization (granting or revoking privileges)
- Binding access plans

These types of operations can create locking problems among themselves and with applications doing normal database processing.

**Enforcement of Referential Integrity**

Usually, deadlocks due to constraint enforcement can be minimized by following guidelines for consistent access order on the parent table.
Limit Fetch Size

- **OPTIMIZE FOR n ROWS**
  - DB2 must plan to access all data
  - Use to determine best communications buffer size
  - Can be used to favor an index with low cluster ratio

- **FETCH FIRST k ROWS ONLY**
  - Provides optimizer with ability to plan to process only a portion of data
  - Specify if small number of rows in result set

```sql
SELECT EMPNAME, SALARY FROM EMPLOYEE
ORDER BY SALARY DESC
FETCH FIRST 100 ROWS ONLY
OPTIMIZE FOR 20 ROWS
```

**Notes:**

For example, assume an application issues the following statement:

```sql
SELECT NAME, SALARY
FROM EMPLOYEE
ORDER BY NAME
```

Assume there is an index on NAME but the data is clustered according to a different index. The optimizer may choose a table scan followed by a sort as the best strategy to return all the data. However, if the application will only fetch the first 10 rows and display them to a terminal, it would be more efficient to use the index to avoid the scan + sort, regardless of clustering. The application can be coded to indicate this intent to the optimizer by including the clause OPTIMIZE FOR 10 ROWS.

The OPTIMIZE FOR n ROWS clause should never be used in batch applications that will access the entire answer set.

The OPTIMIZE FOR n ROWS will also help in minimizing the amount of data transmitted across the network.
When the number of rows to be retrieved is small, there is no need to specify the OPTIMIZE FOR n ROWS clause in addition to the FETCH FIRST k ROWS ONLY clause. However, if k is large and you want optimize by getting the first n rows quickly with a possible delay for the subsequent rows, specify both. The communication buffers are sized based on the lesser of n and k.

If the actual rows required from a query is only a subset of the whole result set of the query, the query can be optimized by specifying the expected (smaller) number of rows.

When retrieving rows using a SELECT statement, consider using the FETCH FIRST k ROWS ONLY clause. This clause sets the maximum number of rows that can be retrieved from within a SELECT statement. Limiting the result table to the first several rows can improve performance. Unlike OPTIMIZE FOR n ROWS, the database manager ceases processing the query once the specified number of rows have been retrieved when FETCH FIRST k ROWS ONLY is specified.

The OPTIMIZE FOR n ROWS clause requests special processing of the SELECT statement. If the clause is omitted, it is assumed that all rows of the result table will be retrieved. If it is specified, it is assumed that the number of rows retrieved will probably not exceed n. The value of n must be a positive integer, and it will determine a suitable communication buffer size to improve the performance of blocked cursors.

This clause does not limit the number of rows that can be fetched or affect the result in any other way than performance. Using OPTIMIZE FOR n ROWS can improve the performance if no more than n rows are retrieved, but may degrade performance if more than n rows are retrieved.
8.3 Optimizer
Notes:

- During the bind process, the optimizer will consider several factors to determine the access strategy.

- The syntax constructs of the SQL statement itself will be analyzed. For example, consider the unique index on EMPNO in the EMP table. A predicate of the form \texttt{EMPNO} \texttt{<> value} cannot take advantage of the index via a matching index scan, while a predicate of the form \texttt{EMPNO} = \texttt{value} could.

- The size of tables, the number of unique values in columns and indexes, and the second highest and second lowest values in columns are just some examples of another key input for access strategy determination - catalog statistics. RUNSTATS is the utility that gathers these statistics.

- The indexes available for consideration are available to the optimizer via the catalog. As a reminder, an index may be used to access particular data values or to avoid a sort.
- The size of configurable parameters, which are controlled by the DBA or Systems Administrator, will be considered during optimization as well. More information on these items are addressed in the appropriate performance course for your platform.
Factors Considered during Optimization

- Distribution statistics
- Query rewrites
- Join techniques
- Access methods

Notes:
- The optimizer considers various factors to determine the best access strategy. These factors can be examined in more detail in the reference section.

For more information, see the Performance Appendix
Optimization Classes

- QUERYOPT optimization level
  - 0 - Use a minimal amount of optimization
  - 1 - Use a degree of optimization roughly comparable to DB2/6000 Version 1, plus some additional low cost features not found in Version 1
  - 2 - Use features of opt level 5, but simplified join algorithm
  - 3 - Perform a moderate amount of optimization; similar to the query optimization characteristics of DB2 for z/OS
  - 5 - Use a significant amount of optimization; with Heuristic rules (default)
  - 7 - Use a significant amount of optimization; without Heuristic rules
  - 9 - Use all available optimization techniques

Notes:

- The optimization class chosen will affect the number of strategies evaluated to determine the access plan. It may affect the efficiency of the access plan that is developed.
- QUERYOPT = 0 directs the optimizer to use a minimal amount of optimization to generate an access plan.
- QUERYOPT = 1 directs the optimizer to use a degree of optimization which is roughly comparable to DB2 Version 1, plus some additional low cost features not found in Version 1.
- QUERYOPT = 2 directs the optimizer to use a degree of optimization which significantly improves upon that of class 1, while keeping the compilation cost significantly lower than class 3 and above for complex queries.
- QUERYOPT = 3 requests that a moderate amount of optimization be performed to generate an access plan. This class comes closest to matching the query optimization characteristics of DB2 for z/OS.
• QUERYOPT = 5 directs the optimizer to use a significant amount of optimization to generate an access plan. However, optimization may be reduced if machine resources are not available. This allows class 5 to be useful across varying hardware configurations, such as a laptop with relatively small amounts of memory or a UNIX machine with relatively large amounts of memory. Class 5 is the default query optimization class.

• QUERYOPT = 7 is the same as class 5, except that reducing optimization algorithms based on hardware resources are not applied.

• QUERYOPT = 9 directs the optimizer to use all available optimization techniques.
8.4 Explain
Using Explain to Optimize Performance

Notes:

- The diagram above illustrates the use of EXPLAIN as part of an iterative process in order to code application programs that perform adequately. This process will assist the programmer in coding the best possible solution with respect to performance.

- The key elements of the process involve running EXPLAIN, ANALYZING the results, and implementing CHANGES indicated by the ANALYSIS.

Running EXPLAIN and implementing CHANGES are generally routine tasks. ANALYZING the results of EXPLAIN and proposing CHANGES to eliminate undesirable access strategies requires knowledge of DB2 performance factors and detailed knowledge of the application and the data being accessed (size of tables, isolation/locking requirements, INDEXes available, fragmentation of data, and so forth).

- The programmer can use this iterative process until the pinnacle is reached - the best possible performance for the given requirement.

- There are various forms of EXPLAIN available in DB2.
Forms of Explain

- Visual Explain
- z/OS
  - EXPLAIN (PLAN_TABLE)
- UNIX/Windows
  - Character-Based Detailed Explain (db2exfmt)
  - Explain Tool (db2expln)

Notes:
There are several techniques available to obtain information about the access strategies determined by the optimizer.

- Visual Explain
  Visual Explain is a graphical workstation feature of DB2 that provides:
  - An easy-to-understand graphical display of a selected access path
  - An ability to invoke EXPLAIN for dynamic SQL statements

- (z/OS) EXPLAIN (PLAN_TABLE)
  - EXPLAIN is a monitoring tool that produces information about a plan, package, or SQL statement when it is bound. The output appears in a table under your userid called PLAN_TABLE which is referred to as a plan_table.

- (Linux, UNIX, Windows) Character-Based Detailed Explain (db2exfmt)
  - Uses EXPLAIN tables to describe access strategies via character output.
- Invoked through the db2exfmt command.
- Used for problem analysis and detailed examination when Visual Explain is not available or desired.

**• (Linux, UNIX, Windows) Explain Tool (db2expln)**
- Uses prepared static statements to display access strategies in a character based format.
- Invoked via db2expln command.
- Used for quick, high-level examination.

Regardless of the form of Explain used, it will enable you to get information about the access strategies of SQL statements. This information can help you:

- Assist in designing application programs
- Determine when an application should be rebound
- Determine the execution plan chosen for a query
- Assist in database design.

Explain output for SQL SELECT statements tends to be the most varied, and it will tend to be the focus of analysis. Explain for SELECT statements will detail sort requirements, order of access for multiple table joins, and so forth. To contrast, explain output for a DELETE statement will typically be analyzed to determine if an index is used or not. Certainly, the preceding statement is not meant to be globally applied. At times, the problem statement in an application will be a non-SELECT.
Visual Explain

- Graphical format of Explain output
- Available for Static and Dynamic SQL
- Analyzed statement can be one of the following:
  - Previously bound static package
  - Previously Explain-ed dynamic SQL
- You can customize amount of information
- Access Visual Explain by:
  - Control Center
  - Command Center

Notes:

Visual Explain is used to graphically display, store and analyze the access plan and decision criteria used by the optimizer to resolve an SQL statement. Visual Explain provides more detailed information on the access plan than the non-graphical form of Explain. It is displayed via a graphical interface.

The amount of information displayed by Visual Explain can be customized by the user depending on his/her needs. For example, you can display general information, debug type of information and so on.

You can access the Visual Explain through the Control Center or Command Editor.

z/OS

Visual Explain is a non-priced feature of DB2 for z/OS that lets you graphically analyze the access paths that DB2 chooses, which eliminates the need to manually interpret the plan table output.

The Visual Explain tool can be downloaded from the Web at:
http://www-3.ibm.com/software/data/db2/os390/db2ve/
8.5 Reading Linux, UNIX, and Windows Explain Output
What Does the Explain Tool Do?

- EXPLAIN
  - Reads a stored package from the catalog
  - Interprets the contents of one or more sections

- EXPLAIN answers the question:
  - How will a DML statement be executed?

Notes:

It is important to note that the Explain Tool analyzes SQL sections that have been bound. When using this tool to investigate the impact of changes to your design, you must bind the packages and reexecute the tool to determine if any change had an impact.

Since the Explain Tool evaluates existing SQL sections, using the tool against production packages will produce output that corresponds to the actual access strategy being used.
Using the Explain Applet

- Bind your program with option = package
  
  ```
  prep - - - - - BINDFILE - - - - - PACKAGE
  ```

- Syntax

  ```
  - db2expln -d SAMPLE -p % -c % -s 0 -o sample.expln
  ```

- Examine output file

Notes:

The Explain Tool can be invoked in two ways. The technique shown above could be considered the primary technique. The other technique is addressed on a subsequent visual.

The `db2expln` command will invoke the Explain Tool. (The specification of creator (-c) and package name (-p) parameters can include the wildcard search characters underscore (_) and percent sign (%). These wildcards have the same meaning assigned to them in the SQL language when using the LIKE predicate.)
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection Options</strong></td>
<td></td>
</tr>
<tr>
<td>-d &lt;db&gt;</td>
<td>Connect to the database name &lt;db&gt;.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-database &lt;db&gt;</td>
<td></td>
</tr>
<tr>
<td>-u &lt;name&gt; &lt;pw&gt;</td>
<td>Connect as user &lt;name&gt; with password &lt;pw&gt;.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-u &lt;name&gt; &lt;pw&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
</tr>
<tr>
<td>-o &lt;file&gt;</td>
<td>Output file name.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-output &lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-t</td>
<td>Send output to the terminal.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-terminal</td>
<td></td>
</tr>
<tr>
<td><strong>Help Options</strong></td>
<td></td>
</tr>
<tr>
<td>-h</td>
<td>This option requests help for the option parameters. If it is specified, all other options are overridden.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-help</td>
<td></td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-?</td>
<td></td>
</tr>
<tr>
<td><strong>Package Options</strong></td>
<td></td>
</tr>
<tr>
<td>-schema &lt;pattern&gt;</td>
<td>The package creator must match &lt;pattern&gt;.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-c &lt;pattern&gt;</td>
<td></td>
</tr>
<tr>
<td>-package &lt;pattern&gt;</td>
<td>The package name must match &lt;pattern&gt;. If not specified, then the package with version &quot;&quot; (the empty string) will be explained.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-p &lt;pattern&gt;</td>
<td></td>
</tr>
<tr>
<td>-section &lt;number&gt;</td>
<td>The section number is &lt;number&gt;. Use 0 (zero) for all sections in the package.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-s &lt;number&gt;</td>
<td></td>
</tr>
<tr>
<td>-escape &lt;character&gt;</td>
<td>Use &lt;character&gt; as the escape character when matching patterns.</td>
</tr>
<tr>
<td>(or)</td>
<td></td>
</tr>
<tr>
<td>-e &lt;character&gt;</td>
<td></td>
</tr>
</tbody>
</table>
The creator and package information must be specified unless dynamic SQL is being explained. If the section information is not specified, then all sections will be displayed.
The <pattern> for creator, package, and version is in LIKE predicate form, which allows the percent sign (%) and underscore (_) as pattern matching characters. This allows multiple packages to be explained with one invocation of db2expln. The escape character can be used to force the % and _ characters to be treated literally. (See the SQL Reference for more information on the LIKE predicate.) If multiple packages may be matched, the section number is automatically set to 0 (all sections).

**Dynamic Statement Options**

- **-statement <statement>** (or) `-q <sql>`
The dynamic statement (statement) will be explained.

- **-stmtfile <file>** (or) `-f <file>`
The dynamic statements contained in the file <file> will be explained. <File> must exist at the client.

- **-terminator <character>** (or) `-z <character>`
Each dynamic statement ends at <character>. If this option is not specified, then each statement is assumed to be one line long.

- **-noenv**
By default, db2expln will invoke each dynamic SET statement after it has been explained. This option prevents the execution of these statements.

**Explain Options**

- **-graph** (or) `-g`
Reconstruct the original optimizer plan graph (as presented by Visual Explain). Note that the reconstructed graph may not exactly match the original plan.

- **-opids** (or) `-i`
Show the operator ID numbers.
Using Dynamic Explain

db2expln -d <dbname> -q <SQL statement> -o <output-file>

Notes:
The product also provides a tool that can be used to explain an SQL statement entered on the command line. The tool supports this function by:

1. Building a small C program that contains the statement submitted.
2. Connecting to the database specified.
3. Preparing the C program and creating a package.
4. Running the Explain Tool against the resultant package.
5. Discarding the C program and object code.
Figure 8-33. Explain Output Example 1 (Linux, UNIX, Windows)

Notes:

- This output was generated by issuing:

  ```
  db2expln -d sample -g -i -o expln.out -q "select lastname, salary from employee where lastname like ‘H%’ "
  ```

- The information produced by the explain tool is shown:

  DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
  Licensed Material - Program Property of IBM
  IBM DB2 Universal Database SQL Explain Tool

  *********************************** DYNAMIC ***********************************

  *********************************** STATEMENT ***********************************

  Isolation Level = Cursor Stability
  Blocking = Block Unambiguous Cursors
  Query Optimization Class = 5
Partition Parallel = No
Intra-Partition Parallel = No

SQL Path = "SYSIBM", "SYSFUN", "SYSPROC", "PATTIC"

SQL Statement:

select lastname, salary
from employee
where lastname like 'H%'

Estimated Cost = 50.144180
Estimated Cardinality = 1.347575

(  2) Access Table Name = PATTIC.EMPLOYEE  ID = 2,5
    | #Columns = 2
    | Relation Scan
    |   | Prefetch: Eligible
    | Lock Intents
    |   | Table: Intent Share
    |   | Row : Next Key Share
    | Sargable Predicate(s)
    |   | #Predicates = 1
(  2) | Return Data to Application
    |   | #Columns = 2
(  1) Return Data Completion

End of section

Optimizer Plan:

RETURN
(  1)
|
TBSCAN
(  2)
|
Table:
PATTIC
EMPLOYEE

This information can be valuable when analyzing an application that contains multiple SQL statements
• The explain output describes the access plan determined by the optimizer.
• The output is summarized:
  - The table is accessed via a Relation Scan (table scan).
- This scan is eligible for prefetch, or for DB2 UDB to bring the data into memory ahead of the application’s direct request for it.
- The locking is an Intent Share lock at the table level and Next Key Share (a type of Share lock) on the row level - not very powerful locks.
- The optimizer has one predicate to apply - it is a data sargable predicate (not applied via an index).

- Assistance with detailed interpretation of Explain output is provided in the Administration Guide. Also, predicate transformation can impact the output.
- The key is to be able to read the basic output initially, and grow skills in the more obscure areas.
- Note that no index access is used. Why might this be the case?
Figure 8-34. Explain Output Example 2 (Linux, UNIX, Windows)  CF108.1

Notes:

- The information produced by the explain tool is shown:

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DB2 Universal Database SQL Explain Tool

*****************************************************************************
*****************************************************************************

------------------- STATEMENT -------------------

Isolation Level = Cursor Stability
Blocking = Block Unambiguous Cursors
Query Optimization Class = 5
Partition Parallel = No
Intra-Partition Parallel = No
SQL Path = "SYSIBM", "SYSFUN", "SYSPROC", "PATTIC"
SQL Statement:

```
select lastname, salary
from employee
where lastname like 'H%'
```

Estimated Cost = 26.417871
Estimated Cardinality = 1.347575

( 2) Access Table Name = PATTIC.EMPLOYEE  ID = 2,5
    #Columns = 2
    Index Scan:  Name = PATTIC.LAST  ID = 1
    Regular Index (Not Clustered)
    Index Columns:
    |  1: LASTNAME (Ascending)
    #Key Columns = 1
    |  Start Key: Inclusive Value
    |  |  1: 'H
    |  Stop Key: Inclusive Value
    |  |  1: 'ZZZZZZZZZZZZZZZ'
    |  Data Prefetch: None
    |  Index Prefetch: None
    Lock Intents
    |  Table: Intent Share
    |  Row : Next Key Share
( 2) Return Data to Application
    #Columns = 2
( 1) Return Data Completion

End of section

Optimizer Plan:

```
RETURN
  ( 1)

|  FETCH
  ( 2)
/ \ IXSCAN Table:
  ( 2) PATTIC
   |  EMPLOYEE
Index:
PATTIC
LAST
```
The key difference between the previous statement and this one is that the data model was changed and an index on the LASTNAME column was added. The output is summarized:

- The EMPA table is accessed using the index based on the LASTNAME column. The optimizer indicates that it is doing a Range Delimiting search by indicating a Start Key and a Stop Key value. Note that the optimizer knows that it needs to start searching for rows that have a LASTNAME of “H” followed by 19 blanks (the length of the LASTNAME column is 20), and that it has found all rows of interest when it reaches the LASTNAME of H followed by 19 Zs.

- The index is used to identify the candidate rows, but the data pages still must be accessed. This is because the SALARY column is in the select list, but not in the index.
Explain Output Example 3

(  2) Access Table Name = PATTIC.EMPLOYEE
   ID = 2,5
   | #Columns = 2
   | Index Scan: Name = PATTIC.LASTSAL  ID = 2
   | | Regular Index (Not Clustered)
   | | Index Columns:
   | | | 1: LASTNAME (Ascending)
   | | | 2: SALARY (Ascending)
   | | #Key Columns = 1
   | | | Start Key: Inclusive Value
   | | | | 1: 'H
   | | | Stop Key: Inclusive Value
   | | | 1: 'HZZZZZZZZZZZZZZZ'
   | | Index-Only Access
   | | Index Prefetch: None
   (  2) | | | Return Data to Application
   | | | #Columns = 2
   | Lock Intents
   | | Table: Intent Share
   | | Row : Next Key Share
(  1) Return Data Completion

Notes:

- The information produced by the explain tool is shown:

  DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
  Licensed Material - Program Property of IBM
  IBM DB2 Universal Database SQL Explain Tool

  ********************** DYNAMIC *******************************

  *****************************************************************

  Isolation Level = Cursor Stability
  Blocking = Block Unambiguous Cursors
  Query Optimization Class = 5
  Partition Parallel = No
  Intra-Partition Parallel = No
  SQL Path = "SYSIBM", "SYSFUN", "SYSPROC", "PATTIC"
SQL Statement:

```
select lastname, salary
from employee
where lastname like 'H%'
```

Estimated Cost = 0.047204
Estimated Cardinality = 1.347575

Optimizer Plan:

```
RETURN
 ( 1)
 |
 IXSCAN
 ( 2)
/   \ Index:  Table:
PATTIC  PATTIC
LASTSAL  EMPLOYEE
```
• The key difference between the previous statement and this one is that the data model was changed and an index on the (LASTNAME, SALARY) columns was added. The output is summarized:

- The EMPA table is accessed using the index based on the LASTNAME, SALARY column. The optimizer again indicates that it is doing a Range Delimiting search by indicating a Start Key and a Stop Key value. Note that the optimizer knows that it needs to start searching for rows that have a LASTNAME of “H” followed by 19 blanks (the length of the LASTNAME column is 20), and that it has found all rows of interest when it reaches the LASTNAME of “H” followed by 19 “Z”s.

- Note that in contrast to the previous example, this example specifically indicates that it will be accomplishing Index-Only Access - DB2 will not have to read the data pages to access the column data. All the data of interest will all be found in the index.
Spotting Bad Access Paths

Look for:

- Relation Scan (TBSCAN)
- Index-Sargable (IXSCAN with Sargable Predicates)

Potential causes:
- No INDEX
- INDEX but not used or badly used
  - Statistics problem
  - Predicate too complex
  - "OR" used in WHERE clause
  - Data type conversion needed
  - Bad index design

Notes:
When DB2 determines an access path for an SQL statement, it will read statistics from the catalog to make its choice. If no indexes exist or if the index cannot be used in an efficient way (wrong column order leading to inefficient matching columns, for example), DB2 will not be able to pick an efficient access path.

DB2 can only be efficient if you give it the right tools (indexes) to be efficient! Your DBA can use a tool called the Index Advisor to evaluate a set of SQL and suggest optimal indexes given the use of the tables.
Populating Explain Tables

Precompile/Bind Options

**EXPLAIN**

- **NO** - Explain information not captured
- **YES** - Explain tables will be populated
- **ALL** - Explain tables will be populated for static and dynamic SQL

**EXPLSNAP**

- **NO** - Explain snapshot not captured
- **YES** - Explain snapshot will be populated
- **ALL** - Explain snapshot will be populated for static and dynamic SQL

**Notes:**

Explain tables can be populated at precompile or bind time by using the options of EXPLAIN or EXPLSNAP.

**EXPLAIN**

- **NO** Explain information will not be captured.
- **YES** Explain tables will be populated with information about the chosen access strategy.
- **ALL** Explain information for each eligible static SQL statement will be placed in the Explain tables. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time.

**EXPLSNAP**

- **NO** Explain snapshot information will not be captured.
- **YES** An Explain snapshot for each eligible static SQL statement will be placed in the Explain tables.
**ALL**  
An Explain snapshot for each eligible static SQL statement will be placed in the Explain tables. In addition, Explain snapshot information will be gathered for eligible dynamic SQL statements at run time.
8.6 Visual Explain
Notes:

• The statement being explained is:

```
select lastname, salary
from employee where lastname like 'H%'
```

No index currently exists on the LASTNAME column.

• More information about the operands (the tables and indexes displayed) and the operators (the functions being carried out, like SORT, TBSCAN, and so forth.) can be determined by double-clicking the graphics in the Visual Explain output.
Visual Explain Output Example 2

Figure 8-39. Visual Explain Output Example 2

Notes:

- The statement being explained is the same one from the first example:
  
  ```sql
  select lastname, salary
  from employee
  where lastname like 'H%'
  
  but now an index has been added.
Notes:

- The same query is being used:
  
  ```sql
  select lastname, salary
  from employee
  where lastname like 'H%
  
  but now the index contains both LASTNAME and SALARY, so DB2 can employ index-only access.
Checkpoint

Exercise — Unit 8 Checkpoint

1. What is a deadlock?

___________________________________________________________________
___________________________________________________________________

2. What tool can be used to see what access strategy DB2 has chosen?

___________________________________________________________________
Unit Summary

Having completed this unit, you should be able to:

- Use programming techniques that enhance DB2 application performance by following general guidelines, using indexable predicates, and avoiding unnecessary sorts
- Identify access paths available to DB2
- List common causes of deadlocks and avoid such causes when possible
- Use the EXPLAIN tools as aids to develop applications that emphasize performance

Notes:
Appendix A. Checkpoint Solutions

Unit 1

1. Some of the platforms where DB2 is available are:
   - AIX on RISC System/6000
   - z/OS
   - VSE and VM
   - OS/400
   - Solaris
   - HP-UX

2. True or False? A trigger may be defined on a table to enforce referential integrity.
   True

3. True or False? Check constraints are defined only at the column definition of a table.
   False

4. True or False? A stored procedure can only be defined in a client server environment when called from a Java program.
   False

5. True or False? DB2 Call Level Interface is a callable SQL programming interface for C/C++ programming language.
   True

6. True or False? DB2 will determine the access strategy for dynamic SQL within an application program before execution time.
   False

Unit 2

1. What are the delimiters to identify an SQL statement in a C program?
   EXEC SQL to begin it, semicolon to end it

2. How do you indicate a host variable in the program? Declare it in the DECLARE SECTION.
   Put a colon (:) in front of it in an SQL statement

3. How can you find out if the value of a column is NULL?
   Check the indicator variable for a negative value

4. What should you check after every executable SQL statement?
   SQLCODE or SQLSTATE
Unit 3

1. What command do you need to run against your source code before you compile it?
   
   **Precompile or Prep**

2. What option should you use on the precompile command so you can separately bind the program?
   
   **BINDFILE**

3. Should RUNSTATS be run before or after you bind a program on which you want optimum performance?
   
   **Before**

Unit 4

1. What do you need to declare in order to be able to retrieve multiple rows in a SELECT statement?
   
   **A cursor**

2. What is the statement used to get the next row in an answer set?
   
   **FETCH**

Unit 5

1. If an application has an IS lock on the table level, what kind of activity is it doing to the rows of the table?
   
   **Reading rows**

Unit 8

1. What is a deadlock?
   
   **When two applications each hold the resource that the other needs to continue**

2. What tool can be used to see what access strategy DB2 has chosen?
   
   **EXPLAIN**
Appendix B. Reference Section: Performance Considerations
Factors Considered during Optimization

- Distribution statistics
- Query rewrites
- Join techniques
- Access methods

For more information, see the Performance Appendix

Notes:
The optimizer considers various factors to determine the best access strategy. These factors can be examined in more detail in this reference section.
Notes:

- The database manager can collect, maintain, and use frequent-value statistics and quantiles, two types of statistics that describe, in a concise way, the distribution of the data values in a column. Use of these statistics by the optimizer can lead to significantly more accurate estimates of the number of rows in a column that satisfy given equality or range predicates. These more accurate estimates in turn increase the likelihood that the optimizer will choose an optimal plan.

  In DB2 UDB for z/OS, you may collect statistics about the distribution of these data values with the COLGROUP parameter of the RUNSTAT utility for a group of columns or a specific column.

- You may collect statistics about the distribution of these data values by using the WITH DISTRIBUTION clause on the RUNSTATS command (Linux, UNIX, Windows). While collecting these additional statistics results in additional overhead for the RUNSTATS utility, the SQL compiler can use this information to help ensure the best access plan is chosen.

- To determine if distribution statistics should be kept for a given table:
- Distribution statistics are only useful for dynamic SQL and static SQL that does not use host variables. When using SQL with host variables, the optimizer does not use distribution statistics.

- Keeping distribution statistics is advisable if at least one column in the table has a highly non-uniform distribution of data and the column appears frequently in equality or range predicates. Non-uniformity may be:
  - Data which is clustered in some subinterval (like the Grade column above, clustered in the range (85,89)).
  - Certain data values have a much higher frequency than other data values (Year).
**Notes:**

The SQL compiler includes a query rewrite stage which transforms SQL statements into forms that can be optimized more easily, and as a result, can improve the access path chosen. Rewriting queries is particularly important for queries which are very complex, including those queries with many subqueries or many joins.

There are three major categories of rewriting that the SQL compiler may perform:

- **Operation Merging - Examples:**
  - View Merges - using views in a SELECT statement can restrict the join order of the table and can also introduce redundant joining of tables. By merging the views during query rewrite, these restrictions can be lifted.
  - Subquery to Join Transformations - use of subqueries in a SELECT statement force a join method and the selection of inner and outer tables for the join. During query rewrite, the subquery can be merged into the main query as a join, which gives the optimizer more choices to choose the most efficient access plan.
- Redundant Join Elimination - during query rewrite, redundant joins can be removed to further simplify the SELECT statement that will be optimized.

- Operation Movement - Examples:
  - DISTINCT Elimination - during query rewrite, the optimizer can move where the DISTINCT operation is performed, to reduce the cost of this operation.
  - General Predicate Pushdown - during query rewrite, the order of applying predicates can be changed so that more selective predicates are applied to the query as early as possible.

- Predicate Translation - Examples:
  - Addition of Implied Predicates - during query rewrite, predicates can be added to the query to allow the optimizer to consider additional table joins when selecting the best access plan for the query.
  - OR to IN Transformations - during query rewrite, an OR predicate can be translated into an IN predicate to allow for a more efficient access plan to be chosen. The SQL compiler can also translate an IN predicate into an OR predicate if this transformation would allow a more efficient access plan to be chosen.

An important point to note is that transformations can occur and examining a statement directly may not yield all possible predicates evaluated by the optimizer.

References:

SC09-4820 Administration Guide: Implementation
SC09-4821 Administration Guide: Performance
SC09-4822 Administration Guide: Planning
Nested Loop Join

**Notes:**

- The nested loop join is one of two methods that can be used by DB2 to satisfy join requirements. In this method, the inner table is examined once for each row that is qualified in the outer table. The determination of which table will be considered outer versus inner is based on cost estimations performed by the optimizer.

- As the diagram illustrates, the access to the inner table can be done via a relation scan or through an index on the join column or columns. Obviously, if the tables are of any significant size, the use of an index would be favored if such an index exists.

- This technique is efficient for small tables, tables with clustered indexes on the join columns, joins for which few rows in the outer table qualify and require examination of the inner table, and joins that use multicolored indexes.

**Note:** A Cartesian product join (one that does not use a join condition) will always use the nested loop technique.
### Merge Scan Join

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>JOB_ID</th>
<th></th>
<th></th>
<th>JOB_ID</th>
<th>FIRSTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>David</td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>Doug</td>
</tr>
<tr>
<td>...</td>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td>Donna</td>
</tr>
<tr>
<td>...</td>
<td>4</td>
<td></td>
<td></td>
<td>3</td>
<td>Donald</td>
</tr>
<tr>
<td>...</td>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td>Doreen</td>
</tr>
<tr>
<td>...</td>
<td>6</td>
<td></td>
<td></td>
<td>5</td>
<td>Debbie</td>
</tr>
<tr>
<td>...</td>
<td>7</td>
<td></td>
<td></td>
<td>6</td>
<td>Dennis</td>
</tr>
<tr>
<td>...</td>
<td>8</td>
<td></td>
<td></td>
<td>7</td>
<td>Diane</td>
</tr>
<tr>
<td>...</td>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
<td>Debbie</td>
</tr>
<tr>
<td>...</td>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
<td>Tad</td>
</tr>
</tbody>
</table>

- If Duplicate Join Value

SELECT *
FROM SIBLING S,
OCCUPATIONS O
WHERE
S.JOB_ID = O.JOB_ID
JOIN SATISFIED THROUGH
TABLES ORDERED ON
JOIN COLUMNS, INDEXES
COULD REDUCE SORT COSTS.

**Notes:**

- The second join method available to DB2 is the merge scan join. In this method, both of the tables must be in join column order. This can be accomplished by accessing the tables via an index on the join column or by sorting the tables. In either case, the inner table of the merge scan join will always be a temporary table, since the join process uses RIDs and these cannot change due to update activity.

- The merge scan join minimizes the amount of the inner table that needs to be examined because the tables are known to be in join column order. DB2 keeps track of positioning on the inner table, and does not need to reexamine rows that have a join value less than the current target row on the outer table.

- All possible combinations for joins must be presented in relational systems. Row 8 in the visual is intended to highlight that DB2 meets this requirement when a merge scan join is performed. If a subset of rows on the inner table needs to be joined for multiple occurrences of the join value in the outer table, the merge scan join will reexamine this subset. This reexamination is still less expensive than scanning the entire table.
The merge scan join may be chosen when no clustered indexes exist on the join columns and the inner table is large enough to justify the cost of sorting. The cost of the sort (or sorts) may offset the cost of the repeated scans of the inner table that would occur if a nested loop join technique were used.

**Note:** In many cases, the join technique used will **NOT** be of particular concern to the programmer. If the appropriate indexes exist on the tables and the database administration tasks for data maintenance (reorganization and statistics gathering) have been completed, the join technique should normally be considered the best for the particular SQL statement and database design. This is the job of the optimizer. However, some knowledge of the basic methods can be useful for analyzing exceptions to this general rule or to assist in determining if additional indexes are necessary to support business requirements.
HASH Join

Notes:

Hash join requires one or more predicates of the form table1.columnX = table2.columnY and the column types to be 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.

First, one table (called the INNER table) is scanned and the rows 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. If the size of the first table exceeds the available sort heap space, buffers from selected partitions are written to temporary tables. After finishing the processing of the INNER table, the second table (called the OUTER table) is scanned. Rows of the OUTER table are matched to rows from the INNER table by first comparing a hash code generated from the columns of the join predicates. Then, if the hash code of the OUTER row matches the hash code of the INNER row, the actual join predicate columns are compared.

OUTER table rows corresponding to partitions not written to a temporary table are matched immediately with INNER table rows in memory. If the corresponding INNER table partition
was written to a temporary table, the OUTER row is also written to a temporary table. Finally, matching pairs of partitions from temporary tables are read and the hash codes of their rows are matched and join predicates checked.

In the case of a hash join, the inner table is kept in memory buffers. If there are too few memory buffers, then the hash join is obliged to spill to disk. The optimizer attempts to avoid this and so will pick the smaller of the two tables as the inner table, and the larger one as the outer table. Data elements are available for monitoring hash join overflows.
### HASH Join

**Outer/Larger Table (occupations)**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>JOB_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>... Civil Engineer</td>
<td>4</td>
</tr>
<tr>
<td>... Interpreter</td>
<td>1</td>
</tr>
<tr>
<td>... CPA</td>
<td>3</td>
</tr>
<tr>
<td>... Biotechnician</td>
<td>4</td>
</tr>
<tr>
<td>... Medical Technologist</td>
<td>5</td>
</tr>
<tr>
<td>... Police</td>
<td>6</td>
</tr>
<tr>
<td>... Nurse</td>
<td>7</td>
</tr>
<tr>
<td>... Computer Analyst</td>
<td>1</td>
</tr>
</tbody>
</table>

**Inner/Smaller Table (siblings)**

```
SELECT * 
FROM SIBLINGS S,
     OCCUPATIONS O 
WHERE 
  S.JOB_ID = O.JOB_ID
```

**Notes:**

- Reading several consecutive pages into the buffer pool using a single I/O operation can greatly reduce the overhead associated with running your application.

- If prefetch is determined to be beneficial during optimization, the optimizer will include prefetch as part of the access strategy. In cases such as table scans and table sorts, the optimizer can easily determine that sequential prefetch will improve I/O performance.

- There may be cases when the optimizer cannot determine that prefetch would be beneficial because sequential access cannot be determined at statement preparation. However, prefetch may be used by the database manager if the access is determined to be relatively sequential during statement execution. The DBA can prevent such run-time use of prefetch by setting the configuration parameter `seqdetect` to NO, but should do so only in special cases.
Sequential Prefetch

- **Look ahead reading**
  - Done Asynchronously

- **Helps performance for**
  - Table scans (data pages)
  - Clustered index scans (data pages)
  - Index only scans (index pages)

**Notes:**

List prefetch, or list sequential prefetch, is a way to access data pages efficiently, even when the data pages needed are not contiguous. List prefetch can be used in conjunction with either single or multiple index access. List prefetch has three steps:

1. **RID retrieval** - a list of needed data pages is compiled by performing matching index scans of one or more indexes and gathering a list of RIDs from the index leaf pages for the rows that need to be retrieved. A RID uniquely identifies a row of data and also identifies on which data pages the row resides. No data pages are read.

2. **RID sort** - the list of RIDs is then sorted in ascending page number order.

3. **Data retrieval** - the required data pages are accessed in order, using the sorted RID list. Multiple data pages may be read in a single I/O operation, but these pages are not necessarily contiguous.

Unlike a regular index scan with no list prefetch, an index scan with list prefetch does not preserve the ordering given by the index. Because the RIDs are sorted in page number order before accessing the data, the data is not in order by any column after retrieval. If the
data needs to be ordered because of an ORDER BY clause or any other reason, an additional sort of the data is needed.

References:

SC09-4820      Administration Guide: Implementation
SC09-4821      Administration Guide: Performance
SC09-4822      Administration Guide: Planning
Bibliography

Manuals:

DB2 UDB for UNIX and Windows

- SC09-4820: Administration Guide: Implementation
- SC09-4821: Administration Guide: Performance
- SC09-4822: Administration Guide: Planning
- SC09-4824: Administrative API Reference
- SC09-4825: Application Development Guide: Building and Running Applications
- SC09-4826: Application Development Guide: Programming Client Applications
- SC09-4827: Application Development Guide: Programming Server Applications
- SC09-4849: Call Level Interface Guide and Reference, Volume 1
- SC09-4850: Call Level Interface Guide and Reference, Volume 2
- SC09-4828: Command Reference
- SC09-4844: SQL Reference - Volume 1
- SC09-4845: SQL Reference - Volume 2
Index

A
access path 8-14
access paths 8-8
access strategy 3-26
Accessing DB2 1-12
ACQUIRE
   ACQUIRE(ALLOCATE) 5-30
   ACQUIRE(USE) 5-30
Administration Client 1-8
Administration Interfaces 1-13
answer set materialization 4-10
APIs 1-13
Application Alternatives 1-24
Application Development Client 1-8
Application Interfaces 1-13
Application Programming Interfaces 1-13
are 1-58
authorization 3-30

B
backup 5-15
bad access paths 8-64
bind 3-14, 3-16, 3-22, 3-30
bindadd 3-30
bindfile 3-11

C
Call Level Interface 1-13, 1-27
child table 1-55
CLI 1-27
CLP 1-13, 1-42, 1-43, 1-46
coding aids 2-48
Command Center 1-13, 1-45
Command Line Processor 1-13
commit 5-7, 5-8
Communication Support 1-7
compile 3-12, 3-13
compound SQL 4-28
   C 4-29
   restrictions 4-32
configuration files 1-15
CONNECT (Type 1) 5-35
CONNECT (Type 2) 5-35
Connect Personal Edition 1-8
Connect to Database 1-17
Connecting to DB2 UDB Database Servers 1-9
consistency token 3-20
Control Center 1-13
create object 1-48
create table 1-49
cursor
   close 4-19
   commit 4-19
   delete 4-15
      C 4-12
      PL/I 4-12
   fetch 4-19
   open 4-19
   reposition 4-22
      C 4-23
      PL/I 4-23
   update 4-15
      C 4-13
      PL/I 4-13
   with hold 4-20

D
data types 1-50
   character 1-50
   date 1-51
   numeric 1-50
   time 1-51
database configuration file 1-15
Database Engine 1-12
database manager configuration file 1-15
DB2 Family 1-6
DB2 Object Hierarchy 1-18
DB2 Product Components - UNO 1-7
DB2 Run-Time client 1-9
db2exfmt 8-45
db2expln 8-46, 8-51
DCL 1-22
dclgen 2-27
DDL 1-20
deadlock 8-30, 8-32
declared temporary table 1-53
defered bind 3-36
delimiters 2-8, 2-9
Development Center 1-38, 1-39
Distributed Relational Database Architecture 1-7
distributed unit of work 5-37
DML 1-21
DSNTIAR 2-48
DUOW 5-37
Dynamic SQL 1-25
dynamic SQL 6-4, 6-6, 6-7, 6-9, 6-10
   non-select
      complete 6-4, 6-6
      C 6-7
      parameterized 6-4, 6-9
      C 6-10
COBOL 6-9
   select
     fixed list 6-4, 6-11
     varying list 6-4, 6-12
     unknown statement 6-4
dynexpln 8-46, 8-54

E
Embedded SQL 1-13
embedded SQL 2-6
Embedded SQL for Java 1-13
error message formatting routine 2-50
explain 8-44, 8-45
   index-only access 8-61
   range delimiting predicate 8-58
   relation scan 8-55
   table scan 8-55
EXPLAIN precompile option 8-65
explain tool 8-50
explicit rebinding 3-29
EXPLSNAP precompile option 8-65
extension 3-7

F
fetch 4-6
   C 4-8
   PL/I 4-8
FETCH FIRST k ROWS ONLY 4-18, 8-34, 8-35
foreign key 1-55

G
general guideline
   dynamic versus static SQL 8-24
   examine lock strategy 8-23
   exploit blocking 8-26, 8-27, 8-28
   minimize DB2 sorts 8-20
   reduce deadlocks 8-32
   scalar functions 8-25
   select what you need 8-19
   statements 8-26
   use views 8-21
general guidelines
   avoid unnecessary conversions 8-18

H
host variable 2-12, 2-18
   C 2-14, 2-15, 2-17, 2-24, 2-26
   COBOL 2-18
   PL/I 2-14, 2-15, 2-17
   SELECT 2-16
I
implicit rebind 3-27
import 7-6
index
  clustered 8-15
  concepts 8-7
index sargable 8-12
index-only access 8-8, 8-11
  explain 8-61
  visual explain 8-70
indicator variable 2-37
  C 2-34
  c 2-35
  example 2-33
  numeric conversion 2-38
indicator variables 2-32
inline bind 3-34
insert
  multiple row 7-5
  single row 7-4
instance 1-15
isolation level 5-28

J
JAVA 1-35
Java 1-13
Java Applications 1-35
Java Database Connectivity 1-13, 1-35
JDBC 1-13, 1-35

L
limit fetch size 8-34
link 3-12, 3-13
load 7-6
load module 3-20
locking
  attributes 5-20
  deadlock 8-30, 8-32
  page locks 5-21, 5-23, 5-26
  row locks 5-21, 5-23, 5-26
  size 5-21, 5-23, 5-26
  strict table locking 5-21, 5-23, 5-26
  U locks 5-32
logical unit of work 5-6
Lotus Approach 1-14

M
matching index scan 8-8, 8-10
materialization 4-10
modified source file 3-34
multiple row insert 7-5
multiple rows 4-6
multisite bind 5-42
multisite update wizard 5-39

N
non-matching index scan 8-8, 8-12
null value on select 2-34
null value passing 2-35

O
object
creation 1-48
ODBC 1-13, 1-29
OLE DB 1-31
Open Database Connectivity 1-29
OPTIMIZE FOR n ROWS 4-16, 8-34
optimizer 3-25, 8-6, 8-38
classes 8-41
factors 8-40
owner 3-23

P
package 3-20, 3-26, 3-34
parent key 1-55
parent table 1-55
Personal Developer's Edition 1-8
Physical Environment 1-15
plan 3-26
PLAN_TABLE 8-45
precompile 3-8, 3-9, 3-22, 3-34
bindfile 3-36
connection options 5-40
EXPLAIN 8-65
EXPLSNAP 8-65
syncpoint 5-41
primary key 1-55
privileges 3-30
program preparation 3-6

Q
QMF 1-12, 1-14
qualifier 3-23, 3-24
Query Management Facility 1-12

R
range delimiting 8-10
range delimiting predicate
   explain 8-58
   visual explain 8-69
rebind 3-27, 3-30
rebinding 3-29
recovery 5-16
referential integrity 1-55, 1-56, 1-57
relation scan 8-8, 8-9
  explain 8-55
RELEASE
  RELEASE(COMMIT) 5-30
  RELEASE(DEALLOCATE) 5-30
releasing a connection 5-38
remote unit of work 5-36
rollback 5-7, 5-8
run 3-17, 3-18
Run-Time Client 1-7

S
savepoint 5-10
  considerations 5-13
  restrictions 5-12
security 3-32
single row insert 7-4
Sorts 8-20
source code 3-7
spotting bad access paths 8-64
SPUFI 1-12
sqb 3-7
sqC 3-7
sqc 3-7
sqf 3-7
SQL INCLUDE statement 2-52
SQL Interfaces 1-12
SQL Processor Using File Input 1-12
sqlaintp 2-48
SQLCA 2-40
  codes 2-42
  SQLERRRD 2-45
  warnings 2-44
SQLCODE 2-41
SQLERRRD 2-45
SQLJ 1-13, 1-35
sqlj 3-7
SQLSTATE 2-41
sqx 3-7
Static and Dynamic SQL 1-25
Static SQL 1-25
stored procedures 1-37
structures 2-30

T
table
  creation 1-49
table scan
  explain 8-55
visual explain 8-68

**U**

U locks 5-32
UDF 1-40
unique constraint 1-55
unique key 1-55
unit of recovery 5-6
Universal Developer's Edition 1-8
user-defined function 1-40

utilities
import 7-6, 7-7, 7-9
load 7-6
reorg 7-12
runstats 7-11

**V**

view 1-52
Visual Explain 8-47

visual explain
index-only access 8-70
range delimiting predicate 8-69
table scan 8-68

**W**

WHENEVER statement 2-48, 2-49
alternative 2-51
WITH HOLD cursor 4-20
WITH RR/RS/CS/UR 5-31