Introduction to IBM DB2 Universal Database
Course Description

The course trains the student on an in-depth knowledge of basic SQL, DML, DDL, and an understanding of database objects (tables, views, etc.) in the DB2 database. A basic understanding of the concepts and features of DCL, database creation, data types, stored procedures, programming, components and packaging (SDK, EEE, ESE, etc.), protocols supported, installation of products, GUI tools, CLP, basic isolation levels, and import/export is also covered.

During the course the student will build and run a DB2 UDB database using data made available in unload format. The database and data are simple, but will illustrate the basic create and load tasks.

Objectives

At the end of this course, you will be able to:

- Plan and implement an installation
- Implement instances and their security
- Create and access DB2 databases
- Use SQL statements
- Describe various database objects
- Explain database concurrency

Prerequisites

To maximize the benefits of this course, we require that you have met the following prerequisites:

- UNIX/Linux systems knowledge
- Microsoft Windows knowledge
Acknowledgments

Course Developer ......................................................... Bob Bernard
Contributing Developer .................................................. Gene Rebman
Contributing Editors .................. Paul C Zikopoulos, Jennifer Gibbs, George Baklarz,
........................................ Roman B. Melnyk, Hana Curtis, Roger E. Sanders, Glen Mules
Courseware Production Editor ............................... Susan Dykman
Technical Review Team ..................................................... Harold Luse

Parts of this document were first published on IBM developerWorks (www.ibm.com/developerWorks/).

Further Information

To find out more about IBM education solutions and resources, please visit the IBM Education website at http://www-3.ibm.com/software/info/education.

Additional information about IBM Data Management education and certification can be found at http://www-3.ibm.com/software/data/education.html.

To obtain further information regarding IBM Informix training, please visit the IBM Informix Education Services website at http://www-3.ibm.com/software/data/informix/education.

Comments or Suggestions

Thank you for attending this training class. We strive to build the best possible courses, and we value your feedback. Help us to develop even better material by sending comments, suggestions and compliments to dmedu@us.ibm.com.
## Table of Contents

### Module 1  Installation and Planning

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives</td>
<td>1-2</td>
</tr>
<tr>
<td>DB2 UDB Product Family</td>
<td>1-3</td>
</tr>
<tr>
<td>Installation Exercise</td>
<td>1-5</td>
</tr>
<tr>
<td>DB2 UDB Component Packages</td>
<td>1-7</td>
</tr>
<tr>
<td>DB2 UDB Clients: Run-Time Client</td>
<td>1-9</td>
</tr>
<tr>
<td>DB2 UDB Clients: Administration Client</td>
<td>1-10</td>
</tr>
<tr>
<td>Control Center</td>
<td>1-13</td>
</tr>
<tr>
<td>Command Center</td>
<td>1-15</td>
</tr>
<tr>
<td>Command Center: Script</td>
<td>1-17</td>
</tr>
<tr>
<td>Command Center: Query Results</td>
<td>1-18</td>
</tr>
<tr>
<td>Command Center: Access Plan</td>
<td>1-19</td>
</tr>
<tr>
<td>Script Center</td>
<td>1-20</td>
</tr>
<tr>
<td>Alert Center</td>
<td>1-21</td>
</tr>
<tr>
<td>Journal</td>
<td>1-22</td>
</tr>
<tr>
<td>License Center</td>
<td>1-23</td>
</tr>
<tr>
<td>Stored Procedure Builder</td>
<td>1-24</td>
</tr>
<tr>
<td>Stored Procedure Builder (cont.)</td>
<td>1-25</td>
</tr>
<tr>
<td>Information Center</td>
<td>1-26</td>
</tr>
<tr>
<td>The Client Configuration Assistant</td>
<td>1-27</td>
</tr>
<tr>
<td>Tool Settings Notebook</td>
<td>1-28</td>
</tr>
<tr>
<td>Performance Monitor</td>
<td>1-29</td>
</tr>
<tr>
<td>First Steps</td>
<td>1-30</td>
</tr>
<tr>
<td>The Command Line Processor (CLP)</td>
<td>1-31</td>
</tr>
<tr>
<td>DB2 Products for Data Warehousing</td>
<td>1-32</td>
</tr>
<tr>
<td>Data Warehouse Center</td>
<td>1-33</td>
</tr>
<tr>
<td>Information Catalog and Query Patroller</td>
<td>1-34</td>
</tr>
<tr>
<td>Query Management Facility (QMF)</td>
<td>1-36</td>
</tr>
<tr>
<td>Online Analytical Processing (OLAP)</td>
<td>1-37</td>
</tr>
<tr>
<td>Summary</td>
<td>1-39</td>
</tr>
</tbody>
</table>
Module 2  Security and Instances

Objectives ...................................................... 2-2
DAS ................................................................. 2-3
DAS (cont.) ....................................................... 2-4
Creating the DAS ................................................. 2-5
DAS Configuration File ......................................... 2-6
Why Drop the DAS? .............................................. 2-7
DB2 Security Mechanisms ....................................... 2-8
Clients, Servers, Gateways and Hosts ....................... 2-9
DB2 Authentication ............................................... 2-10
DB2 Authentication Types ........................................ 2-12
Setting Authentication ............................................ 2-14
Setting Authentication on the Client ......................... 2-16
Dealing With Untrusted Clients ............................... 2-18
CLIENT Authentication ........................................... 2-20
DB2 Authorities ..................................................... 2-22
Requirements to Create an Instance ......................... 2-24
Obtaining SYSADM Authority ................................. 2-25
Fenced User ......................................................... 2-26
Obtaining SYSCTRL Authority ................................. 2-27
Obtaining SYSCONT Authority ................................. 2-28
Obtaining DBADM Authority ................................. 2-29
Obtaining LOAD Authority ................................. 2-31
DB2 Privileges ...................................................... 2-34
Explicit Privileges ............................................... 2-36
Implicit Privileges ............................................... 2-38
Indirect Privileges ............................................... 2-40
Creating Instances ............................................... 2-41
Creating and Dropping Instances ............................. 2-42
The db2icrt Command in Detail ............................... 2-43
Initializing the Instance ......................................... 2-44
Other Instance Level Commands .............................. 2-45
Summary ......................................................... 2-46

Module 3  Creating and Accessing DB2 Databases

Objectives ...................................................... 3-2
Parts of a DB2 Database ......................................... 3-3
DB2 Storage Model ............................................... 3-4
Tables, Indexes, Long Fields and Table Spaces ............ 3-6
Module 4 | Basic SQL

| Objectives | 4-2 |
| SQL Statement Types | 4-3 |
| Unit of Work | 4-4 |
| The “Parts of Speech” of SQL | 4-5 |
| Data Types — Numerics | 4-7 |
| Data Types — Character Strings | 4-9 |
| Data Types — Graphic and Binary Strings | 4-11 |
| Data Types — Date/Time/Timestamp | 4-13 |
| Special Registers | 4-16 |
| The System Catalog | 4-17 |
| Data Control Language, DCL | 4-19 |
| Authorities and Privileges | 4-21 |
| Schemas | 4-23 |
| Authorization ID and Authorization Name | 4-24 |
| Using GRANT | 4-25 |
| Using REVOKE | 4-26 |
| Using CREATE to Make New Objects | 4-27 |
| Using DECLARE to Make New Objects | 4-28 |
| Using ALTER to Change Objects | 4-29 |
| Using DROP to Delete Objects | 4-30 |
| Summary | 4-31 |

Module 5 | Intermediate SQL

| Objectives | 5-2 |
| Using DDL: Creating Objects | 5-3 |
| Creating Tables and Adding Columns | 5-4 |
| Creating Constraints | 5-7 |
| Creating Indexes | 5-8 |
| Referential Integrity — Keys | 5-9 |
| Referential Integrity — Delete Rules | 5-11 |
| Alter Table — Adding Referential Constraints | 5-12 |
| Creating Triggers | 5-13 |
| Creating Views | 5-15 |
| View Materialization | 5-16 |
| Using SELECT to Retrieve Data | 5-17 |
| Relational Operators | 5-19 |
| Character Search Operators | 5-20 |
| Using WHERE to Limit Data Returned | 5-21 |
Module 6 Database Objects

Objectives ............................................. 6-2
Database Objects ..................................... 6-3
Categories of Data Types ......................... 6-4
Numeric Data Types ................................. 6-5
String Data Types ................................... 6-7
Date-time Data Types ............................... 6-9
User-defined Data Types ......................... 6-11
DB2 Extenders ........................................ 6-13
What are Tables? .................................... 6-14
Creating a Table ..................................... 6-15
Where is a Table Stored in the Database? .... 6-16
Altering a Table ..................................... 6-17
How Locks are Acquired ........................................ 7-29
Concurrency and Granularity ............................... 7-31
Transaction Processing ..................................... 7-32
Data Access Paths .......................................... 7-33
Summary ..................................................... 7-34

Module 8  
Course Summary
Course Summary .............................................. 8-2
Where to Go From Here .................................... 8-3
Description of DB2 UDB Courses ....................... 8-4
Description of DB2 UDB Advanced Courses .......... 8-6
To Enroll in Courses ...................................... 8-9
DB2 UDB Technical Documents ......................... 8-10
Additional Technical Documents ....................... 8-11
Evaluation Sheet .......................................... 8-12

Appendix A  
Additional Reference Information
Appendix B  
Example Configuration Parameters
Module 1

Installation and Planning
Objectives

At the end of this module, you will be able to:

- Install DB2 Universal Database and DB2 Clients & Developer’s Edition
- Describe and use the DB2 UDB GUI tools and Command Line Processor
- Explain the concepts of data warehousing and OLAP issues
- Describe the DB2 tools
IBM provides a variety of DB2 Universal Database products according to your needs. Platforms include Windows 2000, OS/2, UNIX, LINUX, and several IBM environments. The IBM family of DB2 Universal Database products includes the following:

- **DB2 Enterprise Server Edition (ESE)** — A multiuser object-relational database management system for complex configurations and large database needs for platforms ranging from Intel to UNIX to SMPs. ESE offers connectivity and integration for other enterprise DB2 and Informix data sources.
- **DB2 Workgroup Server Unlimited Edition (WSUE)** — A simplified per processor licensing model for deployment in a departmental or small business environment that has Internet users or number of users that makes per processor licensing more attractive than the WSE licensing model.
- **DB2 Universal Database Personal Edition (PE)** — A single-user, object-relational database management system for use on a PC.
- **DB2 Universal Developer's Edition** — Contains all the tools needed to develop client-server applications for any machine in a DB2 UDB environment.
- **DB2 Personal Developer's Edition (PDE)** — A single-license package that includes all the tools needed to develop desktop tools and applications for a DB2 environment
- **DB2 Universal Database Express Edition V8.1** — A specially tailored full feature relational database for small and medium business

V8 DB2 UDB Enterprise Edition (EE) and DB2 UDB Enterprise-Extended Edition (EEE) have been merged into a single product, DB2 UDB Enterprise Server Edition (ESE). The ability to create and manage multiple database partitions is part of the ESE product.

For this class, you will install and use DB2 Universal Database Enterprise Edition (EE) version 7.2, or DB2 Universal Database Enterprise Server Edition (ESE) version 8.1 on your classroom PC.

1. Log on to your workstation as Administrator using the password provided by your instructor.
2. Insert the product CD into the CD-ROM drive of your computer.
3. If the startup script does not automatically start after inserting the disk in the drive, then choose Start/Run, and enter the command: \"drive_letter\"\stsetup.exe, where \"drive_letter\" is your CD-ROM drive.

**Warning!**

The steps shown below may differ for you, depending on what release and FixPak level of the software you are using. The steps below are used as a guide only. Please follow the instructions given on the installation media.
4. Unless otherwise indicated in the instructions that follow, use the default settings during installation.

5. Click on the install products icon on the left of the splash screen. When asked for the products to install, check the boxes for **DB2 Enterprise Edition** and **DB2 Administration Client**. When asked for the type of installation, choose to use the **Typical** installation.

6. When prompted for an administration user and password, leave the default user as *db2admin* and use *db2admin* as the password.

7. Installation will take several minutes to complete. Once the installation program begins copying files, direct your attention to your instructor as you continue with the module. You will complete the installation process in an exercise at the end of this module.

---

**V8**

If you are installing version 8, replace step 5 above with the following:

When asked for the products to install, check the box for **DB2 Enterprise Server Edition**. When asked for the type of installation, choose to use the **Typical** installation.

---

**Installing Fixpaks**

If you need to install a fixpak to the DB2 database software, follow these steps:

1. Download the latest version of the fixpak needed from:
   
   http://www-3.ibm.com/software/data/support/

   Select **DB2 Universal Database**, then select **Download FixPaks and Clients**

   Finally, select the product and fixpak version you want to download. (for example, **FixPak 8 WR21314**).

2. Decompress the fixpak file (a .zip if for Windows, and .tar if for Unix)

3. Read the **FixpakReadme.txt** file

4. Follow the instructions in the file to apply the fixpak. Generally, you would:
   
   4.1. Determine the version and level of the currently installed DB2

   4.2. Stop the instances and the DAS

   4.3. Install the Fixpak

   4.4. Rebind any files needing binding

   4.5. Restart the DAS and instances
The component packages that are included with your DB2 UDB product depend on which edition you have. This list of components includes:

- **DB2 Universal Database** — This is the database management system itself.
- **Run-Time Client** — Provides the connectivity between client and database server.
- **Administration Client** — Provides GUI tools for managing the database server and the databases within; these tools will be introduced in this module.
- **DB2 Extenders** — Includes additional data types and built-in functions.
- **DB2 XML Extender** — Provides new data types and additional functions to support the use of XML documents.
- **DB2 OLAP Starter Kit** (not available in v8) — Includes tools for creating complex analytical queries in a data warehousing environment.
- **Application Development Client** — Includes tools for developing client applications that connect to the DB2 UDB server. Java, C++, Perl, and Lotus are among the programming environments that are supported.
- **Net.Data** — Features a full-featured and easy-to-learn scripting language that allows you to create Web applications.
- **Websphere Application Server (separate install)**
- **QMF**
- Websphere Application Server — A Web application server that offers a single code base with multiple options, ranging from a simple single-server configuration to a clustered, high-volume environment (e.g. Domino, IBM HTTP server). This is separately installed from DB2 UDB
- QMF — A tightly integrated, powerful, and reliable query and reporting tool set provided to handle complex data warehousing analysis

Developer editions include additional packages to support the development of client applications.
DB2 UDB Clients: Run-Time Client

The Run-Time Client includes:

- ODBC Support
- JDBC Support
- Command Line Processor (CLP)

DB2 UDB Run-Time Client

The DB2 UDB Run-Time Client must be installed on every client workstation used to access the database server. It contains the support necessary to connect to the server using ODBC, JDBC, and the Command Line Processor (CLP). The supported communication protocols are APPC, IPX/SPX, named pipes, NetBIOS, and TCP/IP.

The following functions have been removed from the DB2 UDB Runtime Client:

- Client Configuration Assistant
- Command Center

The command line processor is included and can be used to administer the DB2 UDB Runtime Client.
The Administration Client is installed on a client workstation and consists of a suite of GUI tools that provide remote administration of databases and instances. The tools provided by the Administration Client depends on the edition of DB2 UDB installed. The administration tools include the following:

- Control Center — The primary screen from which the tools are accessed.
- Data Warehouse Center — Provides a central control point from which to manage the extraction and transformation of data for your data warehouse
- Command Center — A GUI command and statement processor screen
- Script Center — Allows you to create, save, schedule, and execute administration scripts
- Alert Center — The central point for alert notification
- Journal — Allows you to monitor scripted jobs, server history, messages, and alerts
- License Center — Allows you to display license status and usage history for DB2 UDB products installed on your system
- Stored Procedure Builder - A GUI tool for building and maintaining stored procedures
- Satellite Administration Center — Used to configure, manage, and synchronize satellite clients (clients with DB2 servers that are not always connected)
The Administration Client connects to the DAS on the remote server in order to connect to the instances and databases.

**V8**

The Administration tools in v8 include the following:

- **Information Center** — Enables you to access information
- **Data Warehouse Center** — Provides a central control point from which to manage the extraction and transformation of data for your data warehouse
- **Information Catalog Center** — Manages the business metadata
- **Command Center** — A GUI command and statement processor screen
- **Development Center** — A GUI tool for building and maintaining stored procedures and UDFs
- **Project Deployment Tool** — GUI for deploying objects that you exported to a file from the Development Center
- **Control Center** — The primary screen from which the tools are accessed
- **Journal** — Allows you to monitor scripted jobs, server history, messages, and alerts
- **Replication Center** — Interface tool used to set up and administer your replication environment
- **Task Center** — Allows you to create, save, schedule, and execute administration scripts
- **Event Analyzer** — View information on database activities being collected by an event monitor
- **Health Center** — Graphical interface to the Health Monitor and alert state indicators of your instances and database objects
- **Indoubt Transaction Manager** — GUI used to re-sync indoubt transactions
- **Memory Visualizer** — Visual displays and plotted graphs of memory components and their relationships to one another
- **Configuration Assistant** — Previously known as the Client Configuration Assistant
- **Satellite Synchronizer** — Used to configure, manage, and synchronize satellite clients (clients with DB2 servers that are not always connected)

---

**Note**

To determine which version of DB2 you currently have installed on your machine, execute the `db2level` command in DB2 Command Window.
You can use the `db2level` utility to determine the version and code level of DB2.

```
db2level
```

**Example of `db2level` for DB2 7.2 with FP7:**

```
DB21085I  Instance "DB2" uses DB2 code release "SQL07020" with level identifier "03010105" and informational tokens "DB2 v7.1.0.40", "n010415" and "WR21254".
```

Information in the above format is:

- The FixPak numbers for v7.2 will appear as "WR21254" (FP 3 for Windows 32), "WR21311" (FP 7), "WR21314" (FP 8), etc.
- To interpret these codes you may need to visit the DB2 FixPak page, accessible as a link from [www.ibm.com/software/data/db2/udb/support.html](http://www.ibm.com/software/data/db2/udb/support.html).

**Example of `db2level` for DB2 v8.1, initial release (i.e., GA, with no FixPaks):**

```
DB21085I  Instance "DB2" uses "32" bits and DB2 code release "SQL08010" with level identifier "01010106". Informational tokens are "DB2 v8.1.0.36", "s021023", ",", and FixPak "0".
Product is installed at "F:\IBM\DB2\SQLLIB".
```

Information in the above format is:

- The current instance name ("DB2")
- The code release level ("v8.1.0.36") (similar to the signature returned when you CONNECT to a database)
- The date that IBM compiled the installed level of DB2 ("2002-10-23", interpreted from "s021023")
- The FixPak number (here v8.1 FP 0).
- The location of the installed software ("F:\IBM\DB2\SQLLIB") -- not provided here prior to v8.1.
The Control Center is the central management point for a DB2 server. To start the Control Center on Windows, select it through the **Start** menu on Windows, or enter the `db2cc` command at any command prompt (for any operating system).

The Control Center offers database administrators the following features (and more):

- Manage database objects — Create, alter, and drop databases, table spaces, tables, views, indexes, triggers, and schemas. You can also manage systems, instances, users, groups, aliases, user-defined types (UDT), user-defined functions (UDF), packages, and replication objects.
- Manage data — Load, import, or export data, reorganize data, and collect statistics.
- Schedule jobs to run unattended
- Back up and restore databases
- Configure instances and databases
- Analyze queries (through embedded tools like the Performance Monitor and Visual Explain)
- Monitor and tune performance — Run statistics, look at the execution path of a query, start event and snapshot monitoring, generate SQL or DDL of database objects or commands, and view relationships between DB2 objects.
- Troubleshoot
- Manage data replication
- Manage database connections, such as DB2 Connect servers and subsystems
- Manage applications
- Change the font used for displaying menus and text throughout the Control Center
- Manage remote DB2 servers on S/390 or zSeries hardware
- Launch other DB2 centers

The Control Center is shown above. It uses a 2-3 pane layout with objects on the left and details on the right, much like other Windows tools.

The Control Center relies on the Database Administration Server (DAS). DAS helps the Control Center schedule jobs to run against a database server, manage objects, remove database servers, and more. You will learn more about this DAS in the next module.

To see everything that you can do with the Control Center, right-click the mouse on an object from the object tree. A popup menu shows all the functions you can perform on the selected object. For example, on a Tables folder, you can create a new table with or without the help of a wizard, monitor the performance of tables, and filter which tables appear in the contents pane. The tasks that you can perform depend on the object that you select.

From the toolbar located just below the menu at the top of the window, you can launch any other DB2 centers that are integrated into the Control Center. The Control Center toolbar is shown here:

A similar toolbar appears in each Administration Client tool. You can also access these tools by selecting them from the Tools menu.
Start the Command Center by clicking the Command Center icon on the toolbar, or entering the `db2cmdctr.bat` command. In a Windows environment, you can also start the Command Center from the **Start** menu. The Command Center lets you:

- See the results of SQL statements and DB2 commands in a results window. In addition, the Command Center maintains a history of commands and statements for the current session.
- Scroll through the results and generate a report.
- Create and save command scripts to the Script Center. You can also edit a command script to create a new script.
- Run SQL statements, DB2 commands, and operating system commands. When you run DB2 commands from the Command Center, you do not have to precede the command with the `db2` keyword.
- Get quick access to the DB2 administration tools, such as the Control Center, from the main toolbar.
- See the access plan and statistics associated with an SQL statement before execution.
Three additional icons appear in the Command Center toolbar, as shown here:

To execute the query in the **Command** window, press the Execute icon. View the results returned from the last query by clicking on the Retrieve Data icon. To view a graphical display of the query plan that is used for an SQL statement, click on the Create Access Plan icon.

**Interactive Section**

The **Interactive** section of the Command Center, as shown in the slide on page 1-15, allows you to run simple SQL commands. Only one command at a time can be executed, however. Enter the SQL command you want to run in the Command field and click on the Execute icon.

To run more than one command at a time, select the **Script** tab.
You can execute multiple SQL commands by choosing the **Script** tab in the Command Center. Once the commands have been entered, execute the script by pressing the Execute icon. You can also save your script, open a new script, or load an existing script by selecting commands from the **Script** menu.

**Tip**

The default termination character for an SQL statement is a carriage return. If you write scripts that use more than one line for a single command, you can change the termination character by going to the Tools Settings notebook and checking the box for **Statement termination character**.
When you execute a query, the results of the query are displayed under the Query Results tab. You can choose to display the results of the query at any time after executing the query by pressing the Retrieve Table Data icon from the toolbar, or by simply selecting the Query Results tab located beneath the toolbar.
The Visual Explain feature of DB2 UDB lets you view an access plan for explained SQL statements as a flowchart. You can use the information available from the chart to tune your SQL queries for better performance. Visual Explain also lets you dynamically explain an SQL statement and view the resulting access plan chart.

The DB2 optimizer chooses an access plan and Visual Explain displays the information as an access plan chart in which tables and indexes—and each operation on them—are represented as nodes, and the flow of data is represented by the links between the nodes.

The best part of Visual Explain is that you don't have to run the query to get the information you are looking for. For example, if you have a written query that you suspect is inefficiently written, you can use Visual Explain to graphically look at the query cost without actually running it.

You can view the access plan created by Visual Explain by pressing the Create Access Plan icon in the toolbar, or by selecting the Access Plan tab located below the toolbar. For detailed access plan information, double-click on one of the symbols in the chart.
The Script Center is a tool that helps you manage scripts containing SQL statements, DB2 commands, or operating systems commands. You can create new scripts, import scripts that you created earlier, copy scripts, edit existing scripts, remove scripts, and, of course, run scripts. There are also options that let you schedule scripts to run at specific times.

If you run a script from the Script Center (instead of from a command prompt or from the Command Center), the results are logged in the Journal. In the Journal, you can see the jobs that use a particular script, or the status of all scheduled jobs. A job is a Journal entry that is created whenever you schedule a script or run a script immediately.

The Task Center replaces the Script Center. Task information such as:
- Commands to be run
- Schedule, notification, and completion actions associated with the task
- Run results

are stored in a set of tables called the Tools Catalog.
The Alert Center monitors your system to warn you about potential problems. For example, you can set the Alert Center to automatically open and display any monitored objects that have exceeded their threshold.

You can set up thresholds using the Performance Monitor, which you launch from the Control Center by right-clicking on an object. The color of an object indicates the severity of the warning. A red icon indicates an alarm and a yellow icon indicates a warning. The data returned for the performance variable is displayed.

The Health Center now encompasses the Alert Center functionality. Performance monitoring and alert thresholds are handled within the Health Center.
The Journal enables you to monitor jobs and review results. You can start the Journal by selecting the icon from any toolbar. From the Journal, you can also display the recovery history and DB2 messages. The Journal allows you to monitor pending jobs, running jobs, and job histories; review results; display recovery history and alert messages; and show the log of DB2 messages.
The License Center displays the status of your DB2 license and usage information for DB2 products installed on your system. In addition, you can use the License Center to add new licenses, set a concurrent user policy, upgrade a *try-and-buy* license to a *production* license, and much more. You can also control DB2 licenses through the command line using the `db2licm` command.
The DB2 Stored Procedure Builder (DB2 SPB) is an easy-to-use development environment for creating, installing, and testing stored procedures. These procedures can run on the entire DB2 database family, ranging from DB2 for a single workstation to DB2 Universal Database for zSeries. DB2 SPB allows you to focus on creating your stored procedure logic rather than the details of registering, building, and installing stored procedures on a DB2 server.

The DB2 SPB is an optional component of the DB2 Application Development Client.

When you choose the Stored Procedure Builder icon, the above window is displayed to give you an opportunity to create a new stored-procedure project or to modify an existing project.

The Development Center replaces the Stored Procedure Builder. Stored procedures and UDFs are developed with this tool.
Whether you choose to create a new stored procedure project or open an existing project, the **Stored Procedure Builder — Project** window is displayed, as shown above. Through this tool, you can add new stored procedures or modify existing procedures.
Use the Information Center to find information about tasks, books, reference material, troubleshooting, sample programs, and related Web sites. Start the Information Center from the Help menu in any administration tool, or from the Start menu in a Windows environment.

The Information Center provides six types of information:

- **Tasks** — Key tasks you are likely to perform using DB2
- **Books** — DB2 product library
- **Reference** — DB2 reference information, such as keywords, commands, and APIs
- **Troubleshooting** — Categories of error messages and their recovery actions
- **Sample Programs** — Examples that come with the DB2 Application Development Client. If you did not install a DB2 Application Development Client, this tab is not displayed.
- **Web** — DB2 information on the Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the type of information you select.

The Information Center also provides a Find feature, so you can look for a specific topic without browsing the lists.
The DB2 Client Configuration Assistant (CCA) lets you maintain a list of databases to which your applications can connect. It catalogs nodes and databases while shielding you from the inherent complexities of these tasks, but it can do much more. From the main window, you can work with existing DB2 database entries, create new ones, bind applications, set database manager configuration parameters, and import and export configuration information.

The CCA is only available on Windows and OS/2-based DB2 systems. You start this tool by entering the `db2cca` command at a command prompt or by selecting it from the Start menu. Since this is a client tool, it is not part of the Control Center.

The Configuration Assistant replaces the Client Configuration Assistant. You can also start this tool with the `db2ca` command.
The Tools Settings notebook allows you to customize the DB2 graphical tools and some of their options. You can use this notebook to:

- Set online help properties.
- Set DB2 Universal Database for zSeries Control Center properties.
- Change the fonts for menus and text.
- Obtain DB2 diagnostic information.
- Specify a command statement termination character. The statement termination character is important when you want to run multiple statements.
- Set Alert Center properties.
- Invoke an action when the status of a node changes to *down* or *unknown*.

Layout of the Tool Settings panel tabs are slightly different in v8
The Performance Monitor provides information about the state of DB2 and the data that it controls. It is a graphical utility that can be customized for your database environment. You can define thresholds or zones that trigger warnings or alarms when the values being collected by the Performance Monitor are not within acceptable ranges.

You can monitor DB2 objects such as instances, databases, tables, table spaces, and connections by right-clicking an object in the **Object Tree** pane or in the **Contents** pane in the Control Center. From there, you can choose to start monitoring activity.

When an object is being monitored, the color of the icon appears green, yellow, or red to indicate the status of the monitor. The colors represent the severity of the problems as defined by the thresholds that you have set. Green signifies that the monitor is running and everything is fine. Yellow is a warning and signifies that the monitor is approaching the thresholds that you have set. Red indicates an alarm: the monitor has reached or exceeded the threshold. You can use the predefined monitors that are included with DB2, or you can create your own monitors.

The Performance Monitor displays information from two monitoring facilities in DB2: Snapshot Monitor and Event Monitor.

Use Health Monitor (in Health Center) to create monitors.
First Steps

- Create Sample Databases
- Work With Sample Databases
- Work With DB2 UDB Business Intelligence Tutorial
- View the DB2 Product Information Library
- Launch DB2 UDB Quick Tour

First Steps is a graphical tool that helps get you started with DB2. First Steps has a number of options; all are available by clicking on the icon next to the desired action.

With First Steps, you can create sample databases, launch the DB2 tools for management and connectivity, work with the business intelligence tutorials, view the product library, and take a quick tour of all the new features in DB2 Version 7.x or 8.1.

Tip

Check this internet web page for data management white papers:

The Command Line Processor (CLP) is a set of commands that are used to dynamically execute SQL requests and/or DB2 commands. CLP commands can be used to access local workstation databases, remote workstation databases, or remote DRDA Application Server databases (using DB2 Connect Personal Edition or DB2 Connect Enterprise Edition).
The Data Warehouse Manager supports a wide variety of relational and non relational data sources. You can fill your DB2 Universal Database warehouse with data from the most common relational databases (like any member of the DB2 family, Oracle, Sybase, Informix, Microsoft SQL Server), flat files, non relational data stores, and 40 other data sources!

DB2 WM helps you move data directly from source to target and controls the server on which transformations—as well as a host of other data warehousing related activities—take place with distributed warehouse agents.

DB2 WM is really a suite consisting of the four products listed above.

DB2 WM adds the following features to DB2 for data warehousing:

- Increase the scalability of data warehouses
- Fast deployment of data marts
- Comprehensive manageability and resource controls for DBAs
- Easy access to data and metadata by end users
- Enterprise reporting to develop and deliver reports to unlimited clients

Currently the DB2 WM is generally available for: AIX, Windows NT, Windows 2000, Solaris, IBM OS/2, zSeries, and iSeries.
DB2 on Windows comes with the IBM Data Warehouse Center. This tool offers a subset of the features of the Data Warehouse Manager for Windows-based databases. It is free of charge and allows DBAs a chance to sample some of the powerful features that they can leverage with Data Warehouse Manager.

The Data Warehouse Center offers:

- Registration and access to disparate data sources
- Definition of data extraction and transformation steps
- Population and the automation of warehouse management processes

The Data Warehouse Center supports full refresh and incremental update data movement options including leveraging the power of IBM integrated data replication functions. The Warehouse Launch pad simplifies the job of populating a warehouse by leading you through the related tasks.

The Data Warehouse Center is a selectable component of a DB2 installation. If you select this component, the Control database is created to help manage your data warehouse.
The effective management of metadata adds value by:

- Making the integration of solutions easier
- Freeing skilled resources
- Improving ability to rapidly respond to changing systems and business requirements
- Helping to locate and understand the data needed for decision making

Information Catalog

The Information Catalog empowers DBAs with a rich and powerful toolset that allows them to build, define, and administer metadata information. Metadata is really information about the information. Think of metadata as descriptors of the data. For example, if an aggregation was created in the data warehouse that summarized sales for root beer, a metadata entry might relate this information to the business analyst.

Query Patroller

DB2 Query Patroller provides DBAs with a facility for proactive query governing and workload management. Essentially, Query Patroller acts as a police force that can limit the resource consumption of a query, based on a user's profile. Management tools like this are becoming a necessity in data warehousing with the proliferation of ad-hoc query tools that allows you to ask business questions of that data (which isn’t always the most efficient SQL).
IBM plans to release Version 8 of DB2 Query Patroller, which is intended for use with DB2 Universal Database Version 8 databases. However, DB2 Query Patroller Version 8 is not being made available at this time.

Customers using Version 7.2 of the DB2 Query Patroller Version 7.2, which shipped with DB2 Warehouse Manager Version 7.2, should not upgrade to DB2 Universal Database Version 8 until Version 8 of DB2 Query Patroller is available.

DB2 Query Patroller Version 8 will deliver enhanced functionality to better manage and control all aspects of query submission.
Query Management Facility (QMF)

QMF allows you to:

- Build queries and reports easily by using the QMF quick-start interface
- Use the new Java-based query facility to launch queries from your favorite browser
- Easily integrate query results with desktop tools such as spreadsheets and personal databases
- Rapidly build data access and update applications
- Fully exploit DB2 performance, SQL syntax, and advanced database performance techniques such as static SQL

Query Management Facility (QMF) for Windows is included with DB2 WM to provide a multipurpose query tool for business reporting, data sharing, server resource protection, robust application development, and native connectivity to all of the DB2 workstation platforms.

QMF is not supplied with DB2 Universal Database Version 8, but is available separately.
Since the early 1990s, this multidimensional view of data has enhanced the ability to leverage information in decision-making. The technology is characterized by easy navigation among aggregated and derived information, delivering results with consistent, speed of thought response times.

Tools that employ OLAP technology, such as DB2, allow you to ask intuitive and complex ad hoc questions about your business, such as, “What is my profitability for the third quarter across the southeast region for my focus products?” Such a question requires multiple perspectives on the data, such as time, regions, and products. These perspectives are called dimensions.

OLAP analysis is about accessing business data with numerous descriptors. Most business analysts are accustomed to viewing their business two dimensions at a time. Think about it for a minute, how many of us have looked at a Lotus 1-2-3 spreadsheet? A spreadsheet represents facts in two dimensions. However, businesses usually don’t operate in just two dimensions. In fact, businesses usually operate within multiple dimensions, hence the need for a technology that can provide multidimensional analysis.

This type of analysis cannot be provided by just a spreadsheet. For this reason, when we talk about multidimensional analysis, we talk about prepared cubes of information, as in the above figure.
The intersections of the axes in a cube are called facts and they each represent a fact about the business. You can have hierarchies of facts in dimensions, and OLAP provides the ability to navigate these hierarchies. So, in our example, if the Time dimension was aggregated to quarters in the cube (as it is in our example above, since Time has four columns), you can imagine being able to look more closely at quarter one and see the members for the quarter: the months January, February, and March. This type of cube navigation is referred to as drill down. Other navigation OLAP techniques include slicing, dicing, pivoting, or drill up. Simply stated, OLAP can help your business make better, faster, and more informed decisions.

**DB2 as an OLAP Server**

There are several categories of OLAP with the most common being multidimensional OLAP (MOLAP) and relational OLAP (ROLAP). DB2 can do them both! MOLAP is handled with the DB2 OLAP Server or DB2 OLAP Starter Kit, and ROLAP with the native data stores.

An OLAP server processes multidimensional requests that calculate, consolidate, and retrieve information from a multidimensional database, a relational database, or both.

Some vendors store facts in proprietary formats, often called multidimensional databases, while others store them in a star-schema model in a relational database or cube. A sophisticated OLAP server offering such as IBM's DB2 OLAP Server can access and store both multidimensional and relational databases.

OLAP services are provided to you through the DB2 OLAP server products. DB2 OLAP Server leverages the industry leading Hyperion Essbase OLAP engine.

OLAP-like access to DB2 tables can be provided by SQL—this is ROLAP. A variety of tools can generate this SQL, including DB2. DB2 includes the latest OLAP SQL (now part of SQL-99) including rank, rownumber, moving aggregates, covariance, and various linear regression functions.

In DB2 Version 7.x, a limited trial copy of DB2 OLAP Server is included in every DB2 server. The DB2 OLAP Starter Kit gives you DB2 OLAP Server functionality to a maximum of three users. As businesses experiment with the DB2 OLAP Starter Kit and realize its value, they can purchase the rich feature set of DB2 OLAP Server.

**Note**

This course does not cover data warehouse concepts in further detail (see module 7, Summary, of this course for recommended courses as follow-up study).
## Summary

You should now be able to:

- Install DB2 Universal Database and DB2 Clients & Developer’s Edition
- Describe and use the DB2 UDB GUI tools and Command Line Processor
- Explain the concepts of data warehousing and OLAP issues
- Describe the DB2 tools
Exercises
Multiple Choice

For the following questions, choose the best answer from the choices provided.

1.1 In which of the following centers would you use the Visual Explain feature?
   a. Command Center
   b. Control Center
   c. Data Warehouse Center
   d. Script Center

1.2 Which of the following are included in the DB2 Extenders package?
   a. Additional data types and network protocols
   b. Additional data types and built-in functions
   c. Additional built-in functions and an easy-to-learn scripting language
   d. Additional built-in functions and network protocols
   e. Additional data types and an easy-to-learn scripting language

1.3 Which of the following is an advantage of running a script from the Script Center instead of from a command prompt?
   a. Access-path flowcharts are automatically generated in the Script Center
   b. Scripts run faster in the Script Center
   c. The Script Center has a built-in debugger
   d. Script Center results are logged in the Journal

1.4 Through which tool can you run a DB2 UDB tutorial?
   a. First Steps and the Information Center
   b. Client Configuration Assistant
   c. Information Center
   d. Performance Monitor
   e. Script Center
1.5 Pressing which of the following toolbar icons opens the Command Center?

a.  

b.  

c.  

d.  

e.  

1.6 Which of the following best describes Online Analytical Processing (OLAP)?

a. A way that performance can be monitored in the DB2 UDB server  
b. A technology that allows intuitive and complex questions to be asked about large volumes of data  
c. A means by which SQL statements and other commands can be executed through the command line  
d. A means by which the access path for a query can be displayed graphically

1.7 How would you set up the Alert Center to display warnings when a threshold has been reached by some database server resource?

a. Set up the thresholds using the Performance Monitor  
b. Execute statements to define the resource thresholds through the Command Line Processor  
c. Modify settings in the Command Center  
d. Change alarm settings in the Alert Center

1.8 Which of the following is a feature of the Query Management Facility (QMF)?

a. Includes a Java-based query facility to let you execute queries through a browser  
b. Organizes queries based on the databases and tables they reference  
c. Logs query results in the Journal  
d. Maintains lists of databases to which your applications can connect
1.9 The Control Center, Data Warehouse Center, Script Center and Alert Center are all tools found in what DB2 UDB package?
   a. Application Development Client
   b. Administration Client
   c. Run-Time Client
   d. Websphere Application Server

1.10 Support for ODBC and JDBC, and a Command Line Processor are included in which DB2 UDB package?
   a. Application Development Client
   b. Administration Client
   c. Run-Time Client
   d. Websphere Application Server
Completing the Installation

In this exercise, you will complete the process of installation that you started at the beginning of this module.

You probably have found that the installation of DB2 Universal Database that you started at the beginning of this module has not quite completed. Go ahead and complete the installation by performing the following steps:

2.1 Choose default options when prompted for additional information, except as indicated below.

2.2 When the First Steps window appears at the end of the installation process, select the Create Sample Databases option. Choose to create only the DB2 UDB Sample database.

2.3 On Windows 2000, verify that the DB2 software has been installed by looking at the list of services that are operational. To find the list of services, run: Start > Settings > Control Panel > Administrative Tools > Services. List the DB2 services that are listed that have a Status of Started and a Startup Type of Automatic.

2.4 Using the Windows 2000 Task Manager (Press Ctrl-Alt-Delete, then press the Task Manager button), select the Processes tab and click on the Image Name column header to display the processes in alphabetic order. List the DB2 processes that are displayed. Compare these processes with the services listed in 2.3.
Exercise 3

Using the Administration Tools

In this exercise, you will create a sample database and use some of the Administration Client tools to perform a series of tasks.

For each of the following tasks, determine which tool you need to perform the task and attempt to so as described.

3.1 Determine the names of the database instances that are configured on your computer and the names of the databases in those instances.

3.2 Go to the Tools Settings and ensure that the statement terminator character is set to a semicolon.

3.3 Run two simple SQL commands in the Interactive section of the Command Center tool:
   ```sql
   CONNECT TO sample ;
   SELECT * FROM staff
   WHERE job = 'Clerk'
   ORDER BY Dept, Name ;
   ```

3.4 Create a script that contains these SQL commands in Script section of the Command Center. Name the script, `staff1`.

3.5 Determine what access path is used to execute the above SELECT statement (enter the CONNECT TO sample as a separate command).

3.6 Close the Command Center window and open the Script (Task) Center. Locate the script you created in 3.4, execute it, and then monitor the execute job by opening the Journal.

3.7 Run the SELECT statement provided in 3.3 above using the Command Line Processor.

3.8 Open a DB2 Command Window (Start > IBM DB2 > Command Window, or Start > IBM DB2 > Command Line Tools > Command Window if v8) and execute the command `db2level` to find what level software is installed.
   Level __________
Solutions
Multiple Choice

For the following questions, choose the best answer from the choices provided.

1.1 In which of the following centers would you use the Visual Explain feature?
   a. Command Center
   b. Control Center
   c. Data Warehouse Center
   d. Script Center

1.2 Which of the following are included in the DB2 Extenders package?
   a. Additional data types and network protocols
   b. Additional data types and built-in functions
   c. Additional built-in functions and an easy-to-learn scripting language
   d. Additional built-in functions and network protocols
   e. Additional data types and an easy-to-learn scripting language

1.3 Which of the following is an advantage of running a script from the Script Center instead of from a command prompt?
   a. Access-path flowcharts are automatically generated in the Script Center
   b. Scripts run faster in the Script Center
   c. The Script Center has a built-in debugger
   d. Script Center results are logged in the Journal

1.4 Through which tool can you run a DB2 UDB tutorial?
   a. First Steps and the Information Center
   b. Client Configuration Assistant
   c. Information Center
   d. Performance Monitor
   e. Script Center

Solution 1
1.5 Pressing which of the following toolbar icons opens the Command Center?

a.  

b.  

c.  

d.  

e.  

1.6 Which of the following best describes Online Analytical Processing (OLAP)?

a. A way that performance can be monitored in the DB2 UDB server

b. **A technology that allows intuitive and complex questions to be asked about large volumes of data**

c. A means by which SQL statements and other commands can be executed through the command line

d. A means by which the access path for a query can be displayed graphically

1.7 How would you set up the Alert Center to display warnings when a threshold has been reached by some database server resource?

a. **Set up the thresholds using the Performance Monitor**

b. Execute statements to define the resource thresholds through the Command Line Processor

c. Modify settings in the Command Center

d. Change alarm settings in the Alert Center

1.8 Which of the following is a feature of the Query Management Facility (QMF)?

a. **Includes a Java-based query facility to let you execute queries through a browser**

b. Organizes queries based on the databases and tables they reference

c. Logs query results in the Journal

d. Maintains lists of databases to which your applications can connect
1.9 The Control Center, Data Warehouse Center, Script Center and Alert Center are all tools found in what DB2 UDB package?
   a. Application Development Client
   b. **Administration Client**
   c. Run-Time Client
   d. Websphere Application Server

1.10 Support for ODBC and JDBC, and a Command Line Processor are included in which DB2 UDB package?
   a. Application Development Client
   b. Administration Client
   c. **Run-Time Client**
   d. Websphere Application Server
Completing the Installation

You probably have found that the installation of DB2 Universal Database that you started at the beginning of this module has not quite completed. Go ahead and complete the installation by performing the following steps:

2.1 Choose default options when prompted for additional information, except as indicated below.

2.2 When the First Steps window appears at the end of the installation process, select the Create Sample Databases option. Choose only to create the DB2 UDB Sample database.

2.3 On Windows 2000, verify that the DB2 software has been installed by looking at the list of services that are operational. To find the list of services, run: Start > Settings > Control Panel > Administrative Tools > Services. List the DB2 services that have a Status of Started and a Startup Type of Automatic.

You should see the following DB2 services that were started automatically:

<table>
<thead>
<tr>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 Security Server</td>
</tr>
<tr>
<td>DB2-DB2</td>
</tr>
<tr>
<td>DB2-DB2CTLSV</td>
</tr>
<tr>
<td>DB2-DB2DAS00</td>
</tr>
<tr>
<td>DB2 Governor</td>
</tr>
<tr>
<td>DB2 JDBC Applet Server</td>
</tr>
<tr>
<td>DB2 JDBC Applet Server - Control Center</td>
</tr>
<tr>
<td>DB2 License Server</td>
</tr>
<tr>
<td>Warehouse logger</td>
</tr>
<tr>
<td>Warehouse server</td>
</tr>
</tbody>
</table>

This is sample list. You may or may not have these and other services listed, depending on what DB2 functionality you installed.
2.4 Using the Windows 2000 Task Manager (Press Ctrl-Alt-Delete, then press the Task Manager button), select the Processes tab and click on the Image Name column header to display the processes in alphabetic order. List the DB2 processes that are displayed. Compare these processes with the services listed in 2.3.

You should see the following processes listed:

<table>
<thead>
<tr>
<th>Process</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2sec.exe</td>
<td>DB2 Security Server</td>
</tr>
<tr>
<td>db2syscs.exe</td>
<td>DB2-DB2</td>
</tr>
<tr>
<td>db2syscs.exe</td>
<td>DB2-DB2CTLSV</td>
</tr>
<tr>
<td>db2syscs.exe</td>
<td>DB2-DB2DAS00 (db2admin.exe momentarily used to start and stop the DAS)</td>
</tr>
<tr>
<td>db2govds.exe</td>
<td>DB2 Governor</td>
</tr>
<tr>
<td>db2jds.exe</td>
<td>DB2 JDBC Applet Server</td>
</tr>
<tr>
<td>db2ccs.exe</td>
<td>DB2 JDBC Applet Server - Control Center</td>
</tr>
<tr>
<td>db2licd.exe</td>
<td>DB2 License Server</td>
</tr>
<tr>
<td>iwh2log.exe</td>
<td>Warehouse logger</td>
</tr>
<tr>
<td>iwh2serv.exe</td>
<td>Warehouse server</td>
</tr>
<tr>
<td>db2bp.exe</td>
<td>back-end process to support a db connection</td>
</tr>
</tbody>
</table>

This is sample list. You may or may not have these and other processes listed, depending on what DB2 functionality you installed.
Using the Administration Tools

3.1 Determine the names of the database instances that are configured on your computer and the names of the databases in those instances.

**Start the Control Center by selecting** Start > Programs > IBM DB2 > Control Center.

**Start the Control Center by selecting** Start > Programs > IBM DB2 > General Administration Tools > Control Center.

Double-click on your system name in the left half of the window. This displays the Instances folder. Double-click on this folder to see the instances available on your machine. Double-click on the DB2 instance to display a list of folders. Double-click on the Databases folder to see the list of databases on this instance. The Control Center should look something like this:

![Control Center screenshot]

There are two instances in this example: DB2 and DB2CTLSV (with v8, you may only see DB2). The only database shown above is SAMPLE.
3.2 Go to the Tools Settings and ensure that the statement terminator character is set to a semicolon.

Click on the Tools Settings tool from the toolbar. Check the box with the label, Use statement termination character, as shown in the figure below. A semicolon should be in the field next to this label. You can close Tools Settings by pressing the close icon in the top-right corner of the window.

![Tools Settings](image)

3.3 Run two simple SQL commands in the Interactive section of the Command Center tool:

```sql
CONNECT TO sample ;
SELECT * FROM staff
   WHERE job = 'Clerk'
ORDER BY Dept, Name ;
```

Start the Command Center by clicking on the the appropriate tool in the toolbar or by choosing the Command Center option from the Tools menu. You can only enter one command at a time through the Interactive tab, so enter each of the above commands into the Command field and press the Execute icon in the toolbar. Information about the database is displayed in the field below when the CONNECT TO command is executed. The Query Results tab is displayed to show the rows from the SELECT.

3.4 Create a script that contains these SQL commands in the Script section of the Command Center. Name the script, `staff1`.

Select the Script tab in the Command Center and enter the above script, exactly as shown above, into the Script field. You can test the script by pressing the Execute icon in the toolbar. To save the script, choose Script > Save Script As from the command menu. Enter the name of the script, `staff1`, in the Path field. Set the Instance to DB2 and for the Script description, enter a short description of the query. Click OK to save the script. Here is an example:
3.5 Determine what access path is used to execute the above SELECT statement.

Return to the Interactive tag and press the Create Access Plan icon to generate the access plan flowchart for the query. The chart should look something like this:
3.6 Close the Command Center window and open the Script (Task) Center. Locate the script you created in 3.4, execute it, and then monitor the execute job by opening the Journal.

In the Script Center, choose your system name from the pull down menu and highlight the script you created. Choose Selected > Run Now to execute the script. To view the Journal, you can click on the Journal icon from the toolbar. You can also display the Journal by choosing Show Pending Jobs, Show Running Jobs, or Show Job History from the Selected menu. The job you started will appear under Pending Jobs, Running Jobs, or Job History according to the state of the job at any given time.

3.7 Run the SELECT statement provided in 3.3 above using the Command Line Processor.

Start the CLP by running Start > Programs > IBM DB2 > Command Line Processor. At the command prompt, enter the SELECT statement on a single line. Do not end the statement with a semicolon. Press <Return> to execute the command.

![V8](Start the CLP by running Start > Programs > IBM DB2 > Command Line Tools > Command Line Processor.)

3.8 Open a DB2 Command Window (Start > IBM DB2 > Command Window, or Start > IBM DB2 > Command Line Tools > Command Window if v8) and execute the command db2level to find what level software is installed.

Level DB2 v7.1.0.72

![V8](Level DB2 v8.1.0.0)
Module 2

Security and Instances
Objectives

At the end of this module, you will be able to:
- Describe the functionality of the Administration Server (DAS)
- Provide users with authority on DB objects
- List the requirements to create an instance
- Specify the authorization level needed to create an instance
- Specify user authority levels for an instance
- Create a DB2 instance
DAS

- DAS — *Database Administration Server*
  - Special DB2 instance that performs administration tasks
  - Created automatically at installation time
  - Executes administration and monitoring tasks requested by remote clients
  - Executes scheduled jobs
  - Collects information for DB2 Discovery

- DAS is a process in v8, not an instance

---

DAS

The *DB2 Administration Server* (DAS) is a special instance of DB2 that keeps track of other DB2 instances. It is automatically created and configured when DB2 is initially installed on the host computer and is automatically started whenever the host computer is booted. The DAS provides the following specific functions:

- Enables remote administration and monitoring of DB2 instances.
- Provides a scheduler that is used to execute user-defined jobs. These jobs may include operating system commands.
- Allows DB2 Discovery to return information to remote clients.
- Queries the operating system for user and group information.
Although the DAS is automatically created at installation time and is automatically started when the system boots, you can also manually create, start, stop, list, and remove the DAS.

The DAS is the connection between the GUI tools on the client and those on the server. If the DAS is not installed or has been stopped, you cannot connect to the database(s) using the GUI tools. Since there can be only one DAS running on the server, multiple versions of DB2, such as v6.1 or v7.2, connect through the same DAS.

Some users do not need to use the GUI administration tools as their applications connect through JDBC (the GUI uses ODBC), so they do not create the DAS.

By default, the DAS name is DB2DAS00 on Windows and DB2AS on UNIX.
### Creating the DAS

During DB2 installation, a DAS is created, requiring a SYSADM user for the DAS.

To remove the DAS:
- `dasidrop` (UNIX)
- `db2admin drop` (INTEL)

To create the DAS:
- `dasicrt` (UNIX)
- `db2admin create` (INTEL)

If this installation of DB2 is a new installation, then a Database Administration Server (DAS) is created along with the database manager instance. During the installation process, you are asked to provide a name for the DAS. The user name that you provide becomes the name of the DAS and the installing user has SYSADM authority on the DAS. In addition, the registry variable `DB2ADMINSERVER` is set to the name of the DAS. If you do not plan on using the GUI administration tools, the DAS is not needed and can be dropped after the database manager instance has been created.
The DAS also has an Administration Server configuration file. It is similar to the Database Manager Configuration file but stores considerably fewer parameters. The ADMIN CFG and DBM CFG both have a DIAGLEVEL configuration parameter. The following commands illustrate how to list the ADMIN CFG parameters and how to change the DIAGLEVEL parameter from 3 to 4. The DIAGLEVEL parameter dictates the amount of diagnostic information to be written to the instances db2diag.log file (by default this is set to 3).

```
   db2 "GET ADMIN CFG"
db2 "UPDATE ADMIN CFG USING DIAGLEVEL 4"
db2admin stop
db2admin start
```
The DAS provides:

- remote users the opportunity to DISCOVER instances and databases.
- the ability to schedule jobs.
- remote access of the server by the GUI tools on the client.

Why drop the DAS?

- The DAS may not be needed (no GUI interface, no scheduling of jobs, etc.).
- You need to reduce memory footprint.
- You don’t want anyone using DISCOVER (perhaps for security reasons) to find other servers and databases on the network.

What if you do drop and recreate the DAS?

If you DROP the DAS and recreate it, your other instances and all your databases are untouched.

You are exactly back to where you were (except for some GET ADMIN CFG settings), including all other instances and databases being intact.
There are three main mechanisms within DB2 that allow a DBA to implement a database security plan: Authentication, Authorization, and Privileges.

**Authentication**

Authentication is the first security feature encountered when an attempt to access a DB2 instance or a DB2 database is made. DB2 authentication works closely with the security features of the underlying operating system to verify user IDs and passwords. DB2 also has the ability to work with security protocols like DCE and Kerberos for authenticating users.

**Authorization**

Authorization involves assigning DB2 roles to users and/or groups. Each role holds a certain level of authority to execute commands against a particular database and/or the objects within it. The five different authorities within DB2 are SYSADM, SYSCTRL, SYSMAINT, DBADM, and LOAD.

**Privileges**

Privileges are a bit more granular than authorities and can be assigned to users and/or groups. Privileges help define which objects a user can create or drop. They also define which commands a user can use to access objects like tables, views, indexes, and packages.

There are three main mechanisms within DB2 that allow a DBA to implement a database security plan: *Authentication*, *Authorization*, and *Privileges*.
It is particularly important that the terms *client*, *server*, *gateway*, and *host* are understood when thinking about the security of the entire database environment. A database environment often consists of several different computers. It is important to safeguard the database at any potential data access point. The concept of clients, servers, gateways, and hosts is particularly important when dealing with DB2 Authentication.

Consider the database server as the computer (or computers in the case of DB2 EEE or ESE) that the database physically resides on. The DB2 distributed database server can run on several operating systems including UNIX, Linux, Windows, and OS/2. The DB2 database clients are computers that are configured via the `CATALOG DATABASE` command to run queries against the database on the server. These clients can be local, meaning they reside on the same physical computer as the database server, or they can be remote, meaning they reside on a separate computer.

The host or host server is a database residing on a mainframe computer running an operating system like AS/400 or OS/390.

A gateway is a computer running the DB2 Connect product. Through the gateway, DB2 client computers can connect to a DB2 database that resides on a host computer. The gateway is also referred to as the DB2 Connect Server.
When DB2 authenticates

Authentication is done with the help of the underlying operating system security features whenever an ATTACH or CONNECT command is issued. An ATTACH command is used to connect to the DB2 instance, whereas a CONNECT command is used to connect to a database within a DB2 instance. The examples below show you different ways that DB2 will authenticate a user issuing these commands.

By logging onto the DB2 machine with the user ID used to create the `db2inst1` instance, the following commands can be issued:

1. `db2 "ATTACH TO db2inst1"`
   Authentication is done implicitly. The user ID and password used to log onto the machine are assumed and verified by the operating system.

2. `db2 "CONNECT TO sample USER tst1 USING mypass"`
   Database Connection Information
   - Database server = DB2/NT 7.2.4
   - SQL authorization ID = TST1
   - Local database alias = SAMPLE

   Authentication is done explicitly. The user `tst1` with password `mypass` is verified by the operating system. User `tst1` is successfully connected to the sample database.

DB2 Authentication

DB2 Authentication controls the following aspects of a database security plan:

- Who is allowed access to the instance and/or database
- Where and how a user's password will be verified
3. `db2 "CONNECT TO sample USER tst1 USING mypass
   NEW chgpass CONFIRM chgpass"

The user ID **tst1** with password **mypass** is verified by the operating system as in example 2. The password for **tst1** is then changed by the operating system from **mypass** to **chgpass**. As a result, the command in example 2 will fail if reissued.
Authentication types are used by DB2 to determine where authentication is to take place. For example, in a client-server environment, will the client or the server computer verify the user's ID and password? In a client-gateway-host environment, will the client, gateway (aka Connect Server) or host computer verify the ID and password?

The following table summarizes the available DB2 Authentication types. Notice that none of the authentication options are actually set on the host computer. We will discuss setting these options in more detail throughout this module. To clarify, gateway authentication only comes into play when an attempt to access a database on the host is made.

<table>
<thead>
<tr>
<th>Type</th>
<th>Location Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER</td>
<td>Server, Client, Gateway</td>
<td>Authentication takes place on the server using local operating system security.</td>
</tr>
<tr>
<td>SERVER_ENCRYPT</td>
<td>Server, Client, Gateway</td>
<td>Authentication takes place on the server by passing a user ID and password encrypted at the client computer.</td>
</tr>
<tr>
<td>Type</td>
<td>Location Set</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLIENT</td>
<td>Server, Client, Gateway</td>
<td>Authentication takes place on the database partition where the application is invoked using operating system security.</td>
</tr>
<tr>
<td>DCE</td>
<td>Server, Client, Gateway</td>
<td>Authentication is performed by DCE Security Services.</td>
</tr>
<tr>
<td>DCE_SERVER_ENCRYPT</td>
<td>Server</td>
<td>DB2 uses the authentication setting of DCE, SERVER, or SERVER_ENCRYPT.</td>
</tr>
<tr>
<td>*KERBEROS</td>
<td>Server, Client</td>
<td>Authentication is performed by Kerberos security protocol.</td>
</tr>
<tr>
<td>*KRB_SERVER_ENCRYPT</td>
<td>Server</td>
<td>Authentication is performed by Kerberos security protocol or SERVER_ENCRYPT.</td>
</tr>
<tr>
<td>DCS</td>
<td>Client, Gateway</td>
<td>Authentication takes place on the host database server.</td>
</tr>
<tr>
<td>DCS_ENCRYPT</td>
<td>Client, Gateway</td>
<td>Authentication takes place on the host database server and the password is encrypted.</td>
</tr>
</tbody>
</table>

*This setting is valid only for Windows 2000 clients and servers.

**V8** Support for DCE security has been removed in response to the industry move towards Kerberos as the mechanism for secure network authentication and single sign-on. In a future delivery of DB2 UDB Version 8 will extend Kerberos support to UNIX and Linux servers and clients.

Version 8 removes the ability to authenticate a user at the DB2 Connect gateway. Authentication can only be done at the client (using CLIENT authentication) or at the server (using SERVER or SERVER_ENCRYPT). These options must be cataloged at the client in the database directory, or left as NOT_SPEC.

DCS and DCS_ENCRYPT now have exactly the same meaning as SERVER and SERVER_ENCRYPT.
Authentication is set on the database server within the Database Manager Configuration file using the AUTHENTICATION parameter. Remember that the DBM CFG file is an instance-level configuration file. So, the authentication parameter affects all databases within the instance. The following commands illustrate how this parameter can be altered.

To view the authentication parameter in the configuration file:

```
db2 "GET DBM CFG"
```

To alter the authentication parameter to SERVER_ENCRYPT:

```
db2 "UPDATE DBM CFG USING AUTHENTICATION SERVER_ENCRYPT"
db2stop
db2start
```

### Setting Authentication on the Gateway

Authentication is set on the gateway using the CATALOG DATABASE command. On the gateway, this command is required when setting up gateway-to-host database communication.
To set the gateway authentication type to DCS, the following command would be issued on the gateway computer (nd1):

```
    db2 "CATALOG DATABASE myhostdb AT NODE nd1 AUTHENTICATION DCS"
```

Where the host database name is `myhostdb`. 
Consider two separate client computers. One is being configured to connect to a database on a server (DB2 distributed platform) and the other is being configured to connect to a database on a host (DB2 for OS/390, for example).

### Client to Server Database

Example of cataloging the server database named `sample` to server authentication type of `SERVER`:

```plaintext
DB2 "CATALOG DATABASE sample AT NODE nd1 AUTHENTICATION SERVER"
```

### Client to Host Database

If the authentication type on the gateway is `SERVER`:

```plaintext
DB2 "CATALOG DATABASE myhostdb AT NODE nd1 AUTHENTICATION SERVER_ENCRYPT"
```

If the authentication type on the gateway is `DCS_ENCRYPT`:

```plaintext
DB2 "CATALOG DATABASE myhostdb AT NODE nd1 AUTHENTICATION DCS_ENCRYPT"
```

Consider two separate client computers. One is being configured to connect to a database on a server (DB2 distributed platform) and the other is being configured to connect to a database on a host (DB2 for OS/390, for example).

### Client to Server Database

With the exception of `DCE_SERVER_ENCRYPT` and `KRB_SERVER_ENCRYPT`, the client authentication setting must match that of the database server the client is connecting to. Client authentication is set using the `CATALOG DATABASE` command. Again, this command is a required step in setting up client to server communication.

Assume the server authentication type is set to `SERVER`. The following command would then be issued to catalog the server database named `sample`:

```plaintext
DB2 "CATALOG DATABASE sample AT NODE nd1 AUTHENTICATION SERVER"
```
Client to Host Database

Assume the authentication type set on the gateway is set to SERVER. Authentication will take place on the gateway. The following command issued from the client causes the password to be encrypted on the client before being sent to the gateway.

```
db2 "CATALOG DATABASE myhostdb AT NODE nd1
    AUTHENTICATION SERVER_ENCRYPT"
```

For SERVER authentication, the user name and password are validated at the DB2 Connect workstation, but the transferred passwords are encrypted at the client and decrypted at the DB2 Connect workstation.

Now assume authentication is set to DCS_ENCRYPT on the gateway. Authentication will take place on the host. The following command issued on the client causes the password to be encrypted on the client before being sent to the gateway and encrypted on the gateway before being sent to the host computer.

```
db2 "CATALOG DATABASE myhostdb AT NODE nd1
    AUTHENTICATION DCS_ENCRYPT"
```

<table>
<thead>
<tr>
<th>Authentication at Client</th>
<th>Authentication at Gateway</th>
<th>Authentication Location</th>
<th>Client-Gateway Encryption?</th>
<th>Gateway-Server Encryption?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_ENCRYPT</td>
<td>SERVER</td>
<td>gateway</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>DCS_ENCRYPT</td>
<td>DCS_ENCRYPT</td>
<td>server</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Dealing With Untrusted Clients

Untrusted clients have no native security features as part of their operating systems. These include:


If the remote instance has CLIENT authentication, two additional parameters are used to determine where authentication should take place — TRUST_ALLCLNTS and TRUST_CLNTAUTH.

Two other factors when the server or gateway authentication type is CLIENT:

- Whether a user ID and password were explicitly supplied

If the server or gateway computer has authentication set to CLIENT, this implies that the client is expected to authenticate a user's ID and password. However, some client computers may not have native security features as part of their operating systems. These clients are considered untrusted and include DB2 clients running on Windows 3.x, Windows 95, Windows 98, Windows Millennium Edition and Macintosh.

There are two additional parameters in the DBM CFG file used to determine where authentication should take place when the server or gateway authentication method is set to client and untrusted clients are attempting to connect to the database or attach to the DB2 instance. These are the TRUST_ALLCLNTS and TRUST_CLNTAUTH parameters.

There are two other factors that come into play in additional to the TRUST_ALLCLNTS and TRUST_CLNTAUTH parameters when the server or gateway authentication type is CLIENT. The first being whether a user ID and password were explicitly supplied and the second being the type of client connecting. The three DB2 clients are:

- Untrusted clients: As described above.
- Host clients: Clients running on host operating systems like OS/390.
- Trusted clients: Clients running non-host operating systems that have native security features.
**TRUST_ALLCLNTS**

When the authentication type of CLIENT has been selected, an additional option may be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the TRUST_ALLCLNTS parameter to NO. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the TRUST_ALLCLNTS configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES.

**TRUST_CLNTAUTH**

You may also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the TRUST_CLNTAUTH configuration parameter. The default for this parameter is CLIENT.
The table below summarizes where authentication will take place when a connect or attach command is issued by each type of client to a server whose authentication type is set to CLIENT.

### Table 1 — Client Authentication

<table>
<thead>
<tr>
<th>User ID/Password Supplied?</th>
<th>TRUST_ALLCLNTS</th>
<th>TRUST_CLNTAUTH</th>
<th>Untrusted Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUST_ALLCLNTS</td>
<td>YES (default)</td>
<td>NO</td>
<td>DRDAONLY</td>
</tr>
<tr>
<td>Untrusted Clients</td>
<td>see table 2</td>
<td>SERVER</td>
<td>SERVER</td>
</tr>
<tr>
<td>Trusted Clients</td>
<td>see table 2</td>
<td>see table 2</td>
<td>SERVER</td>
</tr>
<tr>
<td>Host Clients</td>
<td>see table 2</td>
<td>see table 2</td>
<td>see table 2</td>
</tr>
</tbody>
</table>

Setting authentication on the server:

```
db2 "UPDATE DBM CFG USING AUTHENTICATION CLIENT"
db2 "UPDATE DBM CFG USING TRUST_ALLCLNTS YES"
db2 "UPDATE DBM CFG USING TRUST_CLNTAUTH SERVER"
db2stop
```

Setting authentication on the client:

```
db2 "CATALOG DATABASE sample AT NODE nd1 AUTHENTICATION CLIENT"
```
In the examples above:

- If the command `db2 "CONNECT TO sample"` was issued from any client, authentication would take place on the client.
- If the command `db2 "CONNECT TO sample USER tst1 USING mypass"` from any client, authentication would take place on the server.

### Table 2 — Trusted Client Authentication

<table>
<thead>
<tr>
<th>User ID/Password Supplied?</th>
<th>TRUST_ALLCLNTS</th>
<th>TRUST_CLNTAUTH</th>
<th>Untrusted Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUST_CLNTAUTH</td>
<td>CLIENT (default)</td>
<td></td>
<td>SERVER</td>
</tr>
<tr>
<td>No password</td>
<td></td>
<td>CLIENT</td>
<td>CLIENT</td>
</tr>
<tr>
<td>(no USING clause)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password supplied</td>
<td></td>
<td>CLIENT</td>
<td>SERVER</td>
</tr>
<tr>
<td>(via USING clause)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A DB2 instance is a logical repository for the DB2 executables and for any DB2 databases you create. A single computer may contain one or more instances, and a single instance may contain one or more databases. An instance may also be referred to as a Database Manager. The user ID that is used to create the instance is referred to as the Instance Owner ID. By default, the DB2 executables and libraries used by an instance are contained in a directory called SQLLIB.

Of the five authorities available in DB2, SYSADM, SYSCTRL and SYSMAINT are instance-level authorities. This means that their scope includes instance-level commands as well as commands against all the databases within the instance. These authorities can only be assigned to a group and are done so through the Database Manager Configuration file.

The DBADM and LOAD authorities are assigned to a user or group for a particular database. This can be done explicitly using the GRANT command.

The following pages will explain how each authority is assigned and what commands users with that authority are allowed to perform. Note that any reference to group membership implies that the user and group names have already been defined at the operating system level.

A user can determine what authorities and database level privileges they have by issuing the db2 "GET AUTHORIZATIONS" command.
New authorities have been defined to explicitly control who can register external routines (stored procedures, UDFs, and methods).

The CREATE_EXTERNAL_ROUTINE authority is required to register external routines.

The CREATE_NOT_FENCED_ROUTINE authority is required to register NOT FENCED routines.
Prior to creating a Database Manager instance, the following users must exist:

- SYSADM user
- Fenced user
- DAS user

Each of these will be discussed in detail in the following pages.
SYSADM authority in DB2 is comparable to:
- **root** authority (UNIX)
- **Administrator** authority (Windows)

SYSADM users can:
- Issue any DB2 commands against that DB2 instance
- Access data within the databases
- Grant or revoke privileges and authorities
- Update the DBM CFG file

Example of granting SYSADM authority to the group **grp1**:
```
db2 "UPDATE DBM CFG USING SYSADM_GROUP grp1"
```
- Requires an instance restart

SYSADM authority in DB2 is comparable to **root** authority on UNIX or **Administrator** authority on Windows. Users with SYSADM authority are able to issue any DB2 commands against that DB2 instance, any databases within the instance, and any objects within those databases. They also have the ability to access data within the databases and grant or revoke privileges and authorities. SYSADM users are the only users allowed to update the Database Manager Configuration (DBM CFG) file.

SYSADM authority is controlled in the DBM CFG via the SYSADM_GROUP parameter. When the instance is created this parameter is set to Administrator on Windows (although it appears blank if you issue the command `db2 "GET DBM CFG"`). On UNIX, it is set to the primary group of the user who created the instance.

Since SYSADM users are the only users allowed to update the DBM CFG, they are also the only ones allowed to grant any of the SYS* authorities to other groups. The following example illustrates how to grant SYSADM authority to the group **grp1**.

```
db2 "UPDATE DBM CFG USING SYSADM_GROUP grp1"
```

This will not take effect until the instance is stopped and then restarted. Also, beware that if you change this to a group that the ID you are currently using is not a member of, you may not have authority to restart the instance! You would have to log in with an ID that is a member of that group or add your current ID to group **grp1**.
Before a Database Manager instance can be created, a user must exist that can run any user-defined functions (UDFs) and stored procedures in a fenced mode. This user is necessary since UDFs are created using the C programming language, which can use pointers to reference memory addresses outside of their defined memory space. To prevent a poorly written UDF from corrupting the DB2 memory, UDFs are commonly run in a fenced section of memory, which prohibits references to memory addresses outside of the fence.

The fenced user:

- Allows user-defined functions to run in fenced mode
- Prevents a poorly written UDF from corrupting the DB2 memory structures
Users with SYSCTRL authority can perform all administrative and maintenance commands within the instance. However, unlike SYSADM users, they cannot access any data within the databases unless they are granted the privileges required to do so. Examples of commands a SYSCTRL user can perform against any database in the instance are:

```!
  db2start
  db2stop
  db2 "CREATE DATABASE db_name"
  db2 "DROP DATABASE db_name"
  db2 "CREATE TABLESPACE tblspace_name"
  db2 "DROP TABLESPACE tblspace_name"
  db2 "BACKUP DATABASE db_name"
  db2 "RESTORE DATABASE db_name"
  db2 "ROLLFORWARD DATABASE db_name"
  db2 "RUNSTATS ON TABLE table_name"
  db2 "UPDATE DB CFG FOR DATABASE dbname"
```

SYSCTRL is assigned to a group by a user with SYSADM authority using the command

db2 "UPDATE DBM CFG USING SYSCTRL_GROUP group_name".
Obtaining SYSMAINT Authority

SYSMAINT users can issue a subset of commands allowed for SYSCTRL authority—tasks that are considered “maintenance” related.

Examples:

- `db2start`
- `db2stop`
- `db2 "BACKUP DATABASE db_name"`
- `db2 "RESTORE DATABASE db_name"`
- `db2 "ROLLFORWARD DATABASE db_name"`
- `db2 "RUNSTATS ON TABLE table_name"`
- `db2 "UPDATE DB CFG FOR DATABASE dbname"`

A SYSADM user can assign SYSMAINT to a group by:

`db2 "UPDATE DBM CFG USING SYSMAINT_GROUP group_name"

The commands that a user with SYSMAINT authority can issue are a subset of those allowed for SYSCTRL authority—just the tasks that are considered “maintenance” related. The following are examples:

- `db2start`
- `db2stop`
- `db2 "BACKUP DATABASE db_name"
- `db2 "RESTORE DATABASE db_name"
- `db2 "ROLLFORWARD DATABASE db_name"
- `db2 "RUNSTATS ON TABLE table_name"
- `db2 "UPDATE DB CFG FOR DATABASE dbname"

Notice that users with SYSMAINT cannot create or drop databases or table spaces. They also cannot access any data within the databases unless they are granted the privileges required to do so.

SYSMAINT is assigned to a group by a user with SYSADM authority using the command `db2 "UPDATE DBM CFG USING SYSMAINT_GROUP group_name"`. 
DBADM authority is a database-level authority and can be assigned to both users and groups.

DBADM users:
- Have almost complete control over the database
- Cannot perform maintenance or administrative tasks
- Can perform:
  - db2 "CREATE TABLE table_name ..."
  - db2 "DROP TABLE table_name"
  - db2 "GRANT SELECT ON TABLE table_name TO USER user_name"
  - db2 "REVOKE SELECT ON TABLE table_name FROM USER user_name"
  - db2 "RUNSTATS ON TABLE table_name"

DBADM authority is a database-level authority rather than an instance-level authority. In summary, DBADM users have almost complete control over the database. DBADM users cannot perform any maintenance or administrative tasks like drop database, drop/create table space, backup/restore database or UPDATE DB CFG FOR DATABASE db_name.

DBADM users are also automatically granted all privileges to the database objects and their contents. Since DBADM authority is a database-level authority, it can be assigned to both users and groups. The following examples illustrate how DBADM authority can be given to the user tst1 who is a member of the grp1.

This command gives implicit DBADM authority on database test to the user that issued the command.

   db2 "CREATE DATABASE test"

This command can only be issued by SYSADM users. The user would have to be connected to the database on which the authority is to be held.

   db2 "GRANT DBADM ON DATABASE TO USER tst1"
For example, if the following commands were issued first, DBADM would have been granted to **tst1** on the **sample** database:

```
   db2 "CONNECT TO sample"
   db2 "GRANT DBADM ON DATABASE TO GROUP grp1"
```

Again, this command can only be issued by SYSADM users.
Obtaining LOAD Authority

LOAD authority is a database-level authority and can be assigned to both users and groups (new as of DB2 v7).

- Users with either SYSADM or DBADM authority can grant or revoke LOAD authority to users or groups

LOAD users:

- Allows users to issue the LOAD command against a table
- Specific privileges on the table may also be required

LOAD authority is also considered a database-level authority and can therefore be granted to both users and groups. This authority is new as of DB2 Version 7. As the name implies, LOAD authority allows users to issue the LOAD command against a table. The LOAD command is typically used as a faster alternative to insert or import commands when populating a table with large amounts of data. Depending on the type of LOAD you wish to perform, having LOAD authority alone may not be sufficient. Specific privileges on the table may also be required. The following commands can be run by users with LOAD authority:

```
db2 "QUIESCE TABLESPACES FOR TABLE table_name"
db2 "LIST TABLESPACES"
db2 "RUNSTATS ON TABLE table_name"
```

Must have insert privilege on table `table_name`:

```
db2 "LOAD FROM filename OF DEL INSERT INTO table_name"
db2 "LOAD FROM filename OF DEL RESTART INSERT INTO table_name"
db2 "LOAD FROM filename OF DEL TERMINATE INSERT INTO table_name"
```

Must have insert and delete privilege on table `table_name`:

```
db2 "LOAD FROM filename OF DEL REPLACE INTO table_name"
db2 "LOAD FROM filename OF DEL RESTART REPLACE INTO table_name"
db2 "LOAD FROM filename OF DEL TERMINATE REPLACE INTO table_name"
```
Only users with either SYSADM or DBADM authority are permitted to grant or revoke LOAD authority to users or groups. The following examples illustrate how LOAD authority can allow user `tst1` to LOAD data into a table called `sales`. Assume the command `db2 "CONNECT TO sample"` was already issued.

```
db2 "GRANT LOAD ON DATABASE TO USER tst1"
db2 "GRANT INSERT ON TABLE sales TO USER tst1"
```

With LOAD authority and insert privilege, `tst1` could issue a LOAD INSERT or a LOAD RESTART or TERMINATE after a LOAD INSERT against the `sales` table.

```
db2 "GRANT LOAD ON DATABASE TO GROUP grp1"
db2 "GRANT DELETE ON TABLE sales TO GROUP grp1"
db2 "GRANT INSERT ON TABLE sales TO GROUP grp1"
```

With LOAD authority, delete and insert privileges, any member of the `grp1` group could issue a LOAD REPLACE or a LOAD RESTART or TERMINATE after a LOAD REPLACE against the `sales` table.
## System and Database Authority Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>SYSADM</th>
<th>SYSCTRL</th>
<th>SYSMAINT</th>
<th>DBADM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrate database</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update DBM configuration</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant/revoke DBADM</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update <strong>db/node/dcs</strong> directories</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force users off system</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create/drop databases</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create/drop/alter table spaces</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore to new database</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update DB configuration</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Backup database or table space</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Restore to existing database</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Perform rollforward recovery</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Start/stop database instance</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Restore to table space</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Run trace</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Get snapshots</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Query table space state</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Update log history files</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Quiesce table space</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REORG table</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Run RUNSTATS utility</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Read log files</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Create/activate/drop event monitors</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
# DB2 Privileges

Database-level privileges — span all objects within the database

Object-level privileges — associated with a specific object

Database-level privileges a user might be given:

- **CREATETAB**: Users are allowed to create tables within the database.
- **BINDADD**: Users are allowed to create packages in the database using the bind command.
- **CONNECT**: Users are allowed to connect to the database.
- **CREATE_NOT_FENCED**: Users are allowed to create unfenced user-defined functions (UDFs).
- **IMPLICIT_SCHEMA**: Users can implicitly create schemas within the database without using the CREATE SCHEMA command.

---

## Database and object privileges

The preceding pages briefly touched on the concept of privileges. Privileges can generally be placed into two main categories: database-level privileges, meaning they span all objects within the database; and object-level privileges, meaning they are associated with a specific object. The database-level privileges that a user might be given are as follows:

- **CREATETAB** — Users are allowed to create tables within the database.
- **BINDADD** — Users are allowed to create packages in the database using the bind command.
- **CONNECT** — Users are allowed to connect to the database.
- **CREATE_NOT_FENCED** — Users are allowed to create unfenced user-defined functions (UDFs).
- **IMPLICIT_SCHEMA** — Users can implicitly create schemas within the database without using the CREATE SCHEMA command.

Database objects include tables, views, indexes, schema and packages. Fortunately, most of the object-level privileges are self explanatory.
The following table summarizes these privileges.

<table>
<thead>
<tr>
<th>Privilege name</th>
<th>Relevant object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL</td>
<td>Table, View, Index,</td>
<td>Provides full authority on the object. Users can grant or revoke privileges on the object to others.</td>
</tr>
<tr>
<td></td>
<td>Package</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>Table, View</td>
<td>Allows users to delete records from the object.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Table, View</td>
<td>Allows users to insert or import records into the object.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Table, View</td>
<td>Provides the ability to view the contents of the records using the SELECT statement.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Table, View</td>
<td>Allows users to modify records within the object using the UPDATE statement.</td>
</tr>
<tr>
<td>ALTER</td>
<td>Table</td>
<td>Allows users to alter the object definition using the ALTER TABLE statement.</td>
</tr>
<tr>
<td>INDEX</td>
<td>Table</td>
<td>Allows users to create indexes on the object using the CREATE INDEX statement.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Table</td>
<td>Provides the ability to create or drop foreign constraints on an object.</td>
</tr>
<tr>
<td>BIND</td>
<td>Package</td>
<td>Allows users to rebind existing packages.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Package</td>
<td>Allows users to execute packages.</td>
</tr>
<tr>
<td>ALTERIN</td>
<td>Schema</td>
<td>Allows users to modify definitions of objects within the schema.</td>
</tr>
<tr>
<td>CREATEIN</td>
<td>Schema</td>
<td>Allows users to create objects within the schema.</td>
</tr>
<tr>
<td>DROPIN</td>
<td>Schema</td>
<td>Allows users to drop objects within the schema.</td>
</tr>
</tbody>
</table>

Information on object level privileges is stored in the system catalog views. The view names are `syscat.tabauth`, `syscat.indexauth`, `syscat.schemaauth` and `syscat.packageauth`.

**V8** New privilege for invoking routines:

The routine EXECUTE privilege has been defined to explicitly control who can invoke routines (stored procedures, UDFs, and methods).

When the routine is used in an SQL statement, the routine definer must have the EXECUTE privilege on any packages used by the routine.
Explicit Privileges

Privileges on objects can be explicitly granted and revoked to users or groups using:

- GRANT
- REVOKE

Examples:

```sql
db2 "GRANT SELECT ON TABLE inst101.org TO USER tst1"
```
```sql
db2 "REVOKE INSERT ON TABLE inst101.org FROM GROUP grp1"
```

Privileges can be explicitly granted and revoked to users or groups using the GRANT and REVOKE commands. The following examples illustrate the use of these commands on various objects.

Imagine this scenario:

If logged in as a user with Administrator authority on Windows, bring up two DB2 command windows. Ensure that the DB2INSTANCE variable is set to `db2inst1` in both windows!

As Administrator, start command window 1 and issue:

```sql
db2 "CONNECT TO sample"
```

As Administrator, start command window 2 and issue:

```sql
db2 "CONNECT TO sample USER tst1 USING passwd"
```

The commands in Window 1 are being issued by a user (`inst101`) with SYSADM authority. The commands in Window 2 are being issued by a user (`tst1`) with no specific authority or privileges on the sample database. Note that the schema name associated with the tables in your sample database will be the name of the user that issued the `db2 "CONNECT TO sample"` command. In these examples, it is `inst101`. 

2-36  Security and Instances
In Window 2:

db2 "SELECT * FROM inst101.org"

SQL0551N "TST1" does not have the privilege to perform operation "SELECT" on object "INST101.ORG".

In Window 1:

db2 "GRANT SELECT ON TABLE inst101.org TO USER tst1"

The command in Window 2 should now complete successfully.

In Window 2:

db2 "INSERT INTO inst101.org VALUES
    (100, 'Tutorial', 1, 'Eastern', 'Toronto')"

SQL0551N "TST1" does not have the privilege to perform operation "INSERT" on object "INST101.ORG".

In Window 1:

db2 "GRANT INSERT ON TABLE inst101.org TO GROUP grp1, USER mytest"

The command in Window 2 should now complete successfully because tst1 is a member of group grp1.

In Window 2:

db2 "DROP TABLE inst101.emp_photo"

SQL0551N "TST1" does not have the privilege to perform operation "DROP TABLE" on object "INST101.EMP_PHOTO".

In Window 1:

db2 "GRANT DROPIN ON SCHEMA inst101 TO ALL"

The command in Window 2 should now complete successfully

The following commands, issued from Window 1, would revoke the privileges granted in the above examples:

    db2 "REVOKE SELECT ON TABLE inst101.org FROM USER tst1"
    db2 "REVOKE INSERT ON TABLE inst101.org FROM GROUP grp1"
    db2 "REVOKE DROPIN ON SCHEMA inst101 FROM ALL"
DB2 may grant privileges automatically when certain commands are issued, without the need for an explicit GRANT statement to be issued, as shown previously. The table below summarizes some commands that result in privileges being implicitly granted by the database manager. Note that these privileges are implicitly revoked when the object created is dropped.

<table>
<thead>
<tr>
<th>Command issued</th>
<th>Privilege granted</th>
<th>To whom it is granted</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE <em>mytable</em></td>
<td>CONTROL on <em>mytable</em></td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE SCHEMA <em>myschema</em></td>
<td>CREATEIN, ALTERIN, DROPIN on <em>myschema</em>, plus ability to grant these to others</td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE VIEW <em>myview</em></td>
<td>CONTROL on <em>myview</em> only if CONTROL is held on all tables and views referenced in the definition of <em>myview</em></td>
<td>User issuing command</td>
</tr>
<tr>
<td>CREATE DATABASE <em>mydb</em></td>
<td>SELECT on <em>mydb</em>'s system catalog tables, IMPLICIT_SCHEMA on <em>mydb</em></td>
<td>PUBLIC*</td>
</tr>
</tbody>
</table>

*PUBLIC is a special DB2 group that includes all users of a particular database. Unlike the other groups referenced thus far, PUBLIC does not have to be defined at the operating system.
level. This group can also be used with the GRANT and REVOKE commands as illustrated here:

```sql
db2 "GRANT SELECT ON TABLE sysibm.systables TO PUBLIC"
db2 "REVOKE SELECT ON TABLE sysibm.systables FROM PUBLIC"
```
Privileges can be obtained indirectly when packages are executed by the database manager.

A user requires EXECUTE privilege on the package if all the statements in the package are static.

A user with EXECUTE privilege on `db2package1` would indirectly be granted SELECT privilege on table `org` and INSERT privilege on table `test` if `db2package1` executes the following static SQL statements:

```
db2 "SELECT * FROM org"
db2 "INSERT INTO test VALUES (1, 2, 3)"
```

Privileges can be obtained indirectly when packages are executed by the database manager. To summarize, a package contains one or more SQL statements that have been converted into a format that DB2 uses internally to execute them. In other words, a package contains multiple SQL statements in an executable format. If all the statements in the package are static, a user would only require EXECUTE privilege on the package to successfully execute the statements in the package.
There are two ways you can create a Database Manager instance. You can use the Administration Client GUI tools, or you can use the DB2 command line utility. Both methods actually execute the `db2icrt` program to cause an instance to be created.

You will learn how to use the GUI tools in another module, therefore, the next few pages discuss the command line method of creating instances.
Creating and Dropping Instances

For Unix command line:

- Use the `db2icrt` command to create a database manager instance
- Normally, only `root` can execute this command

Syntax:
```
db2icrt -u fenced_user sysadm_user
```

Example:
```
db2icrt -u fence101 inst101
```

The fenced user is not used in a Windows environment

The DB2 command line command used to create the database manager instance is `db2icrt`. In the above example a database manager instance is created with the name `inst101`, and a fenced user is created named `fence101`. The user `inst101` is the owner of the instance and is assigned SYSADM authority over the instance.

In addition, all files associated with the instance, plus any default SMS table spaces, are created in the `$HOME` directory for user `inst101` on UNIX, and in the `drive:\DB2\NODE0000` directory on Windows.

![V8]

If the default method (using the menu) is used to create an instance, it will be created as multinode enabled (for database partitioning in ESE). To create an instance that is not multinode capable, use this command method:
```
db2icrt <instancename>
```

You may need to drop (`db2idrop`) an instance and re-create it manually.
The db2icrt Command in Detail

The db2icrt command:

- Creates the database manager instance
- Sets the environment variables DB2INSTANCE and PATH
- Creates the /sqlib subdirectory in the $HOME directory of the SYSADM on UNIX
- Configures communications based on the server's available protocols
- Creates the db2profile and userprofile files (UNIX)

The db2icrt command installs and configures the database manager instance on the server. Normally only the user root (UNIX) or Administrator (Windows) has authority to run this command, but in our classroom environment, the student logins have been given authority to run this command.

The environment variable DB2INSTANCE is set to the name of the Database Manager instance and PATH is set to include the path to the DB2 binary files. A new directory, sqlib, is created in the SHOME directory of the user specified as the SYSADM.

The communications protocols that are supported on the server are examined and entries are made in the operating system services file to allow communications with the DAS and the database manager instance.

Finally, the files necessary to set environment variables are created. In UNIX, the first of these two files is db2profile (or db2bashrc or db2cshrc, depending on your shell), which sets the default environment variables. This file is often overwritten by new versions of DB2 or by fixpacks, and you should not make any changes to it. The second file is called userprofile and is provided for your use to set environment variables unique to your installation. It will not be overwritten by new versions of DB2 or by fixpacks.
Now that an instance is installed, it can be started. When the `db2start` command is executed, the system reads the value of DB2INSTANCE and starts the specified database manager instance. The process of starting consists of reading the DBM configuration file and setting up UNIX processes (called *agents*) and memory control structures to allow communication with the instance.

The DB2 ATTACH command assigns agents to an application so that instance-level utilities and commands, such as CREATE DATABASE, can work. Use the `db2 "ATTACH TO instance_name"` command to establish this attachment.

At this point there is still no connection to any of the databases on the instance. The CONNECT command creates a connection to a database and is discussed in a later module.

Two of the most important instance commands are those that start and stop the DB2 processes used to run DB2 itself. The DB2 instance can be started using the `db2start` command and stopped using the `db2stop` command.
The `db2ilist` command lists the DB2 instances on the computer. Note that an instance called DB2 is also listed. This instance is usually created automatically when DB2 is installed. On Windows, the active instance is identified by the `DB2INSTANCE` environment variable.

The `db2idrop` command drops a DB2 instance on the computer.

Every DB2 instance has a Database Manager Configuration file, or DBM CFG file for short. This file is used to hold parameters related to authentication, monitoring levels, diagnostic levels and memory and process resources across the instance. The instance must be stopped and started in order for any parameter changes in the DBM CFG file to become effective.

To view the configuration file:
```
  db2 "GET DBM CFG"
```

To alter this file's `DIAGLEVEL` parameter, which dictates the amount of diagnostic information to be written to the instances `db2diag.log` file (by default this is set to 3):
```
  db2 "UPDATE DBM CFG USING DIAGLEVEL 4"
  db2stop
  db2start
```
Summary

You should now be able to:

- Describe the functionality of the Administration Server (DAS)
- Provide users with authority on DB objects
- List the requirements to create an instance
- Specify the authorization level needed to create an instance
- Specify user authority levels for an instance
- Create a DB2 instance
Exercises
The following exercise is designed to test your knowledge on DB2 security issues.

1.1 The Database Administration Server is often referred to as the ___ and is used by the __________ to help ______ your database environment.

1.2 In a default DB2 for Windows installation, which user account is associated with the Administration Server instance?
   A. das  
   B. db2admin  
   C. dasadmin  
   D. administrator

1.3 The First Steps tool makes it easy to create which three types of sample databases?
   ___________________
   ___________________
   ___________________

1.4 Open a DB2 Command Window (Start > Programs > IBM DB2 > Command Window) to look at your instance information. Run db2admin (no parameters needed) to determine the name of your DAS:
   ___________

1.5 Run db2ilist (no parameters needed) to determine the names of the instances that are currently available on your machine:
   ___________
1.6 Run db2 LIST ACTIVE DATABASES to determine the names of the databases that are currently available on your machine:

1.7 Use Windows administration tools to determine what DB2 users were created on your system when the installation was performed:
Record the users found on the system:
On Windows 2K Pro this requires the following sequence of steps:
Start > Settings > Control Panel > Administrative Tools > Computer Management > Local Users and Groups > Users.
Exercise 2

This exercise tests you on your ability to access the GUI Administration Client tools.

Use the installed software and your new database to locate—where feasible—all the screen shots illustrated in Module 1.

2.1 Run the DB2 Control Center.

On Windows 2K Pro this requires the following sequence of steps:

Start > Programs > IBM DB2 > Control Center

DB2 Command Window (Start > Programs > IBM DB2 > General Administration Tools > Command Window)
At this point, you should have the DB2 product installed. During that installation, the DAS instance was created, and a Database Manager instance was also created, in which you created the sample database.

**Note** Although there are usernames and instance names indicated explicitly in the various exercises of this course, the instructor may assign different ones to each student, depending on the computer system used for this course.

In this exercise, you will create your database instance manually, which you will use in later modules to create databases and database objects.

When prompted for an administration user and password, use **db2admin** for the login and **db2admin** for the password.

3.1 Log in to the Windows workstation.

3.2 Open a DB2 Command Window and run **db2icrt inst101** at the command line to create another instance (**inst101**).

3.3 Run **db2ilist** to list all instances available on your workstation:

________

________

3.4 Restart your DB2 Control Center. Does the new instance show in the Control Center when you open the Instances entry in the left-hand pane?

____

3.5 If the new instance does not show, add it to your Control Center view, then click Refresh (hint: right click on **Instances** and choose **Add** — but before you click on **OK**, choose **Show Command**).

________________

________________

________________
3.6 What is the command for putting a new entry in the catalog of available instances?

3.7 If your workstation needed to support remote database users and access DRDA databases, what product would you need to install?
Solutions
The following exercise is designed to test your knowledge on DB2 security issues.

1.1 The Database Administration Server us often referred to as the **DAS** and is used by the **DB2 Tools** to help **manage** your database environment.

1.2 In a default DB2 for Windows installation, which user account is associated with the Administration Server instance?
   - A. das
   - B. db2admin
   - C. dasadmin
   - D. administer

1.3 The First Steps tool makes it easy to create which three types of sample databases?
   - **DB2 UDB Sample**
   - **OLAP Sample**
   - **Data Warehousing Sample**

1.4 Open a DB2 Command Window (**Start > Programs > IBM DB2 > Command Window**) to look at your instance information. Run **db2admin** (no parameters needed) to determine the name of your DAS:

   **DB2DAS00**

1.5 Run **db2ilist** (no parameters needed) to determine the names of the instances that are currently available on your machine:

   **DB2**
1.6 Run `db2 LIST ACTIVE DATABASES` to determine the names of the databases that are currently available on your machine:

**SAMPLE**

1.7 Use Windows administration tools to determine what DB2 users were created on your system when the installation was performed:

Record the users found on the system:

On Windows 2K Pro this requires the following sequence of steps:

**Start > Settings > Control Panel > Administrative Tools > Computer Management > Local Users and Groups > Users.**

`db2admin` (at least)
This exercise tests you on your ability to access the GUI Administration Client tools.

2.1 Ensure that you can navigate through the various GUI screens discussed in Module 1, and list the screen names you find.
At this point, you should have the DB2 product installed. During that installation, the DAS instance was created, and a Database Manager instance was also created, in which you created the sample database.

**Note**

Although there are usernames and instance names indicated explicitly in the various exercises of this course, the instructor may assign different ones to each student, depending on the computer system used for this course.

In this exercise, you will create your database instance manually, which you will use in later modules to create databases and database objects.

When prompted for an administration user and password, use db2admin for the login and db2admin for the password.

3.1 Log in to the Windows workstation.

3.2 Open a DB2 Command Window and run `db2icrt inst101` at the command line to create another instance (inst101).

3.3 Run `db2ilist` to list all instances available on your workstation:

   INST101

   DB2

3.4 Restart your DB2 Control Center. Does the new instance show in the Control Center when you open the Instances entry in the left-hand pane?

   No

3.5 If the new instance does not show, add it to your Control Center view, then click Refresh (hint: right click on Instances and choose Add — but before you click on OK, choose Show Command).

   Remote Instance: inst101

   Instance Name: inst101

   [Show Command]

   OK
3.6 What is the command for putting a new entry in the catalog of available instances?

CATALOG LOCAL NODE inst101 INSTANCE inst101 SYSTEM xxxx OSTYPE NT

3.7 If your workstation needed to support remote database users and access DRDA databases, what product would you need to install?

DB2 UDB Enterprise Edition

DB2 UDB Enterprise Server Edition
Module 3

Creating and Accessing DB2 Databases
Objectives

At the end of this module, you will be able to:

- Create a DB2 database
- Catalog a remote database or local database or DRDA
- Use the DB2 GUI tools to create, access and manipulate DB2 objects
- Create a table
A DB2 database is actually made up of a collection of objects. From the user's perspective, a database is a collection of tables that are (hopefully) related in some way.

Some of these items relate to how the data is organized, like tables or views. Other items refer to the physical implementation of these objects, like table spaces. Finally, some objects only deal with how the database performance is managed through buffer pools and other memory objects.

Rather than dwell on all possible combinations of parameters and objects, you should first concentrate on the physical implementation of the database. How do you create a database and allocate the disk storage required for it? To properly answer that question, you need to know about the basic objects in the database and how they get mapped to physical disk storage.
DB2 Storage Model

Logical storage model:
- Tables are made up of columns and rows
- Tables are placed into *table spaces*
- Table spaces can contain more than one table

Physical storage model:
- *Container* is a physical storage device
- Can be a directory, a device, or a file
- Container is assigned to a table space

DB2 has both a logical and physical storage model to handle data. The actual data that users deal with are found in tables. While tables may be made up of columns and rows, the user has no knowledge of the physical representation of the data. This is sometimes referred to as the *physical independence* of the data.

The tables themselves are placed into *table spaces*. A table space is used as a layer between the database and the container objects that hold the actual table data. A table space can contain more than one table.

A *container* is a physical storage device. It can be identified by a directory name, a device name, or a file name. A container is assigned to a table space and a table space can span many containers. The ability to have multiple containers assigned to a table space gets around operating system limitations that may restrict the amount of data that one container can have. The relationship between all of these objects is illustrated in the chart below.
Although a table is the basic object that is placed into a table space, you must be aware of additional objects within the DB2 system and how they get mapped to a table space.
Tables, Indexes, Long Fields and Table Spaces

A table:
- Is an unordered set of data records
- Consists of columns and rows, generally known as records

Table types are:
- Permanent (base), temporary (declared), temporary (derived)

An index is a physical object that is associated with a single table.

A long field (BLOB) refers to a type of data found within a table:
- Typically contains unstructured data (image, document, audio)
- Not stored directly in the row of the table
  - A pointer is stored that links to a spot in a long field table space

Tables, indexes and long fields (sometimes called Binary Large Objects or BLOBs) are objects that are created within a DB2 database. These objects get mapped to a table space that is itself mapped to physical disk storage.

Table

A table is an unordered set of data records. It consists of columns and rows that are generally known as records. These tables can be either permanent (base) tables, temporary (declared) tables, or temporary (derived) tables. From a DBA perspective, space is allocated for each one of these table objects, but in different table spaces. Derived temporary tables, such as those used as intermediate tables during sort operations, reside in the system temporary table space. Declared temporary tables reside in user temporary table spaces.

Index

An index is a physical object that is associated with a single table. Indexes are used to enforce uniqueness in a table (make sure there are no duplicate values) and to improve performance when retrieving information. Indexes are not required to run your Structured Query Language
(SQL) statements. However, your users will appreciate your foresight in creating a few of these objects to speed up the query processing!

**Long field (BLOB)**

A *long field* (BLOB) refers to a type of data found within a table. This data type typically contains unstructured data (image, document, audio) and usually contains a significant amount of information. Storing this type of data within a table would lead to excessive overhead when deleting, inserting and manipulating these objects. Instead of storing them directly in the row of the table, a pointer is stored that links to a spot in a long field table space. You need to be aware of this datatype so that appropriate table spaces can be created to contain these objects.

Armed with the knowledge of these different object types, you are now ready to determine which type of space you need to allocate.
Table spaces are the logical layer between the database and the tables stored in that database. Table spaces are created within a database and tables are created within table spaces. DB2 supports two kinds of table spaces:

- **System Managed Space (SMS)**
  - Managed by the operating system
  - Requires very little maintenance
- **Database Managed Space (DMS)**
  - Managed by DB2
  - Usually better performance

From a DBA’s perspective, SMS table spaces require very little maintenance. The trade-off from an ease-of-use perspective is potentially lower performance and fewer optimization options.
So which type of table space should you use? Although this is not a complete list, above are some things for you to consider when deciding on DMS or SMS table spaces:

Now that you are an expert on the different types of table spaces, it's time to create your first database. The next few pages will show you how.
Creating a DB2 database from a command line is relatively simple. To create a database, you must use the DB2 Command Line Processor (CLP). This can be accomplished by either selecting **Command Line Processor** from the DB2 Program Group, or executing the command `db2cmd db2` from a command line.

The syntax for creating a DB2 database is:

```
CREATE DATABASE my1stdb
```

“That's it!” you say. Yes, the only element that is required as part of a CREATE DATABASE command is the name of the database. Of course, there are many more options that are available.

The rules for a database name are:

- The database name can consist of the following characters: a-z, A-Z, 0-9, @, #, and $.
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.
- A database name or database alias is a unique character string containing from one to eight letters, numbers, or keyboard characters from the set described above.

Let’s examine what actually happened as a result of this command.
A number of files were created by DB2 as part of the CREATE DATABASE command. These files include log files, configuration information, history files, and three table spaces. These three table spaces are:

- **SYSCATSPACE**
  This is where the DB2 system catalog, which tracks all of the metadata associated with DB2 objects, is kept.

- **TEMPSPACE1**
  A temporary work area for DB2 to place intermediate results.

- **USERSPACE1**
  A place where all user objects (tables, indexes) reside by default.

All of these files are placed into the DB2 directory found on your default drive. The default drive is typically the same drive where you installed the DB2 product.

For simple applications, this default configuration may be sufficient for your needs. However, you may want to change the location of your database files, or change the way DB2 manages these objects. The next few pages explores the CREATE DATABASE command in more detail.
The CREATE DATABASE Command

The full syntax of the DB2 CREATE DATABASE command can be found in the *DB2 Command Reference*.

The railroad diagram below contains the majority of options that would be of interest to you.
Create Database options

- ON
  - path
  - drive
- ALIAS database_alias
- USING CODESET codeset TERRITORY
- COLLATE USING SYSTEM
  - COMPATIBILITY
  - IDENTITY
  - NLSCHAR

Table Space options

- CATALOG USER TEMPORARY TABLESPACE
- MANAGED BY
  - SYSTEM USING (‘container_string’)
  - DATABASE USING (‘container_string’ - #_pages)
  - FILE DEVICE
  - EXTENT SIZE #_pages
  - PREFETCH SIZE #_pages

The following pages describe what these various options are and how you would use them.
The ON *path/drive* option tells DB2 where to create the database.

- On UNIX, this specifies the path for the database
  - If not specified, the database is created based on the DFTDBPATH parameter
- On OS/2 or the Windows operating systems, this specifies the letter of the drive for the database

One of the parameters of the CREATE DATABASE command is the ON path/drive option. This option tells DB2 where you want to create the database.

On UNIX-based systems, this specifies the path on which to create the database. If a path is not specified, the database is created on the default database path specified in the Database Manager Configuration File (DFTDBPATH parameter).

On OS/2 or the Windows operating systems, this specifies the letter of the drive on which to create the database.

For example, the following CREATE DATABASE command places the database on the D: drive on a Windows operating system:

```sql
CREATE DATABASE mydb ON D:
```
A character code page is associated with all DB2 character data types (CHAR, VARCHAR, CLOB, DBCLOB). A code page can be considered as a reference table that is used to convert alphanumeric data to binary data. A DB2 database can only use a single code page. The code page is established during the CREATE DATABASE command using the options CODESET and TERRITORY. The code page can use a single byte to represent an alphanumeric (a single byte can represent 256 unique elements) or multiple bytes.

Languages like English contain relatively few unique characters; therefore, a single-byte code page is sufficient to store data. Languages like Japanese require more than 256 elements to represent all of the unique characters; therefore, a multi-byte code page (usually a double-byte code page) is required.

By default, the collating sequence of a database is defined according to the code set used in the CREATE DATABASE command. If you specify the option COLLATE USING SYSTEM, the data values are compared based on the TERRITORY specified for the database. If the option COLLATE USING IDENTITY is used, all values are compared using their binary representation in a byte-to-byte manner.

The DB2 Administration Guide: Planning lists the various code pages that are available when creating a database. In most instances, you would let this value default to the same code page as the operating system that the database is currently running on.
Each of the three table spaces (SYSCATSPACE, TEMPSPACE1, USERSPACE1) are created automatically in the default directory unless you specify their location (ON keyword). For each table space, the DBA can specify the characteristics of the file system that the table space uses.

The three table spaces are defined using the following syntax:

```
Table Space
Options

CATALOG
USER
TEMPORARY
TABLESPACE
```

If any of these keywords are omitted, DB2 will use the default values to generate the table spaces. The table space definition follows these options and has the following syntax:

```
Table Space Definitions

The table space definition indicates what type of table space to use and where its containers are placed.

The MANAGED BY options tell DB2 to generate these table spaces and determine how the space will be managed.

- SMS table spaces use the SYSTEM USING ('container_string') specification
- DMS table spaces use the DATABASE USING (FILE|DEVICE 'container_string' number_of_pages) specification

Other parameters include:
- EXTENTSIZEx number_of_pages
- PREFETCHSIZE number_of_pages
```
Table Space options

MANAGED BY

SYSTEM USING ( 'container_string' )

DATABASE USING

FILE 'container_string' # pages

DEVICE 'container_string' # pages

EXTENT SIZE # pages

PREFETCH SIZE # pages

The MANAGED BY options tell DB2 to generate these table spaces and determine how the space will be managed. SMS (System Managed Storage) table spaces use the SYSTEM USING keywords, while DMS (Database Managed Storage) table spaces are defined with the DATABASE USING keywords.

- SYSTEM USING ( 'container_string' )
  For an SMS table space, the container_string identifies one or more containers that will belong to the table space and into which the table space's data will be stored.

  Each container string can be an absolute or relative directory name. The directory name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. The format of container_string is dependent on the operating system.

- DATABASE USING (FILE/DEVICE 'container_string' number_of_pages)
  For a DMS table space, the container_string identifies one or more containers that belong to the table space and into which the table space's data will be stored. The type of the container (either FILE or DEVICE) and its size (in PAGESIZE pages, without a K, M, or G qualifier) are specified. The size can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes). A mixture of FILE and DEVICE containers can be specified.

  For a FILE container, the container string must be an absolute or relative file name. The file name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. If the file does not exist, it will be created and initialized to the specified size by the database manager. For a DEVICE container, the container string must be a device name and the device must already exist.
One important note: All containers must be unique across all databases; a container can belong to only one table space.

- **EXTENTSIZE** `number_of_pages`
  
  EXTENTSIZE specifies the number of PAGESIZE pages that are written to a container before skipping to the next container. The extent size can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes). The database manager cycles repeatedly through the containers as data is stored.

- **PREFETCHSIZE** `number_of_pages`
  
  PREFETCHSIZE specifies the number of PAGESIZE pages that are read from the table space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes).
  
  Prefetching reads in data needed by a query prior to it being referenced by the query so that the query need not wait for I/O to be performed.
Above is an example of a CREATE DATABASE command that uses many of the options previously discussed.

1. CREATE DATABASE my1stdb
   This statement defines the name of the database we are creating.
2. DFT_EXTENT_SZ 4
   This parameter tells DB2 that the default extent size is four pages, unless explicitly stated.
3. CATALOG TABLESPACE MANAGED BY DATABASE USING
   The DB2 catalog space will be managed by the database.
4. FILE 'C:\CAT\CATALOG.DAT' 2000, FILE D:\CAT\CATALOG.DAT' 2000)
5. EXTENTSIZEx 8
6. PREFETCHSIZE 16
7. TEMPORARY TABLESPACE MANAGED BY SYSTEM USING
   ('C:\TEMPTS','D:\TEMPTS')
8. USER TABLESPACE MANAGED BY DATABASE USING
   (FILE 'C:\TS\USERTS.DAT' 121)
9. EXTENTSIZEx 24
10. PREFETCHSIZE 48
7. **TEMPORARY TABLESPACE MANAGED BY SYSTEM USING**
   The temporary space used by DB2 will be handled by the operating system.
8. 'C:\TEMPTS' ...
    The temporary space will be split across two files whose size is automatically adjusted
during DB2 execution.
9. **USER TABLESPACE MANAGED BY DATABASE USING**
    The user space (where the real tables are placed) will be managed by DB2 directly.
10. FILE 'C:\TS\...'
    There is only one container for this space and it consists of 121 pages.
11. **EXTENTS SIZE 24**
    The EXTENTS SIZE for the USER table space will be 24 pages.
12. **PREFETCH SIZE 48**
    Queries will prefetch 48 pages at one time.

In most cases, the default values of the CREATE DATABASE command will give you a
database that you can use for development and testing. However, time spent planning the data
placement and the table space definitions used by DB2 may result in a database that is easier to
manage with potentially better performance.
Why Catalog a Database?

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

So why does a database have to be cataloged? Without this information, an application cannot connect to a database! DB2 has multiple directories that are used to access databases. These directories allow DB2 to find databases known to it whether they are on the local system or a remote system. The system database directory contains a list and pointer to where all the known databases can be found. The node directory contains information relating to how and where remote systems or instances can be found. To put an entry into any of these directories, use CATALOG.
To remove an entry, use UNCATALOG.
Normally, cataloging a database is not required when you have created a database.

However, this may be required if you:
- Had uncataloged the database
- Want to set up an *alias* (alternate name) for this database
- Need to access this database from a client

Cataloging a database is relatively straightforward. The CATALOG command is shown below.
You can use the `db2"LIST DATABASE DIRECTORY"` command to view cataloged databases.
When a user needs to connect to a DB2 database, they need to catalog the database on their local workstation. In order to do this, the user would use the CATALOG command or the DB2 Client Configuration Assistant (CCA). The CCA lets you maintain a list of databases to which your applications can connect. It catalogs nodes and databases while shielding the user from the inherent complexities of these tasks.

From a client perspective, cataloging databases using a profile or discovery is the easiest way to do this. Manual configuration requires knowledge of the database location and characteristics in order to successfully run the command.

Using either of the automated configurations requires that the DBA either generate profiles for their users or set up discovery services within the DB2 database. This tutorial does not go into the details of creating either of these facilities. Read the *DB2 Administration Guide: Implementation* for more details on these features.
Automated Configuration Using Discovery

Discovery works in one of two ways:
- Search discovery — the DB2 client searches for DB2 servers on the network
- Known discovery — one particular server is queried for information about the instances and their databases

This method is normally used to configure small numbers of clients.

If you use this type of automated configuration, you do not need to provide any detailed communications information to enable the DB2 client to contact the DB2 server.

You can use either the Client Configuration Assistant or the Control Center to exploit discovery-based configuration. This method is normally used to configure small numbers of clients.
Access profiles are another automated method to configure a DB2 client to access remote DB2 servers and their databases. An access profile contains the information that a client needs to catalog databases to a DB2 server.

As with discovery, when using access profiles, you do not need to provide any detailed communications information to enable the DB2 client to contact the DB2 server.

Use of access profiles is typical for configuring a large number of clients. The DB2 Control Center can be used to export and then import a server access profile system. A client access profile is exported, then imported using the Client Configuration Assistant (CCA).

V8 The Client Configuration Assistant is now named Configuration Assistant.

If you have a large number of clients to configure, you should also consider making use of DCE Directory Services or LDAP (Lightweight Directory Access Protocol). These features allow you to store catalog information in one centralized location. Each client just needs to know the centralized location to be able to connect to any database that has been made available in the network. See the DB2 Administration Guide: Implementation for more details about LDAP.
It is also possible to manually configure a database connection. To do this, you need to know the details of the communications setup between the client and the server.

There are two ways to manually configure connections, as shown above.

In either case, manual configuration must be used to exploit some advanced options not available using automation.
Using The CA To Catalog A Database

The following panels show the steps required to manually catalog a database using the Configuration Assistant (CA).

The user needs the following information:

- One of the protocols supported by the server instance containing the database
- The protocol connection information required to configure the connection to the server instance
- The server name
- The name of the database on the remote server

The initial CA screen is invoked by selecting Configuration Assistant from the Windows DB2 program menu.
Once the user has selected this program, the following screen will be displayed.

The top section of this screen gives the user a list of databases that are currently cataloged on the system. There are various options here that allow a user to optimize the client configuration and performance of the connection to the database. A DBA would be interested in the Add button, which allows you to add a database to the system. Clicking Add brings up the Add Database Wizard.

The following panels will describe each step required in configuring the client.

The functionality is the same on the new Configuration Assistant window. The only difference is that the button functions have been moved to pull-down menus along the top of the screen.
The first panel of the CA Wizard asks the user which method they want to use to catalog the database.

The Wizard gives the user three possible ways of cataloging a database.

- Use a profile
- Search the network
- Manually configure a connection

We are interested in the manual configuration option. As the previous pages described, using a profile or searching the network are much easier methods for a user who wants to catalog a database on their client. Once the manual configuration option is selected, the user has to select the communication protocol they want.

Selecting **Add Database Using Wizard** from a drop-down menu produces the same functionality as the next few screens shown.
In this panel, you select the protocol that you will use to connect to the database. The contents of the Protocol Parameters box changes according to the protocol. Here are the protocols you can choose along with their parameters:

- **TCP/IP** — Server hostname/IP address, port number
- **NetBIOS** — Server workstation name, adapter number
- **Named Pipe** — Server computer name, instance
- **APPC** — Server symbolic destination name
- **IPX/SPX** — Server internetwork address, socket number. If using file server addressing, file server name and object name.
- **LOCAL** — A local database on this machine

All of the protocols that DB2 supports are listed here, including APPC. If you have chosen APPC, your operating system choices will be: LAN-based, OS/390 or MVS, OS/400, VM, or VSE. If you choose TCP/IP, your choices are: LAN-based, OS/390, OS/400, or VM.

You should check that the machine is properly configured on the network before clicking **Next**.
At this point, you enter the communication details of the database that you want to catalog. This screen is different for each communication protocol. The example shown above is for a database that is found on the same machine as the client.
At this point, you enter the details of the database that you want to catalog. Enter the name of the database (as known at the server) in the **Database name** field. You can accept the same name as the local alias for the database, or change the alias to a name of your choice. You can also enter a description.
You can register the database as an ODBC data source. By default the **Register this database for ODBC** check box is selected. You can choose an application from the **Optimize for Application** selection box to optimize the ODBC settings for that application.

If you plan to run Windows applications against this database, you should become familiar with the various optimization settings available in this environment.
Two additional panels are available to enter:

- Options relating to the remote node the database is on
- Type of security being used

Fill in the Node information if you plan to use the Control Center.

- Affects the behavior of this tool

Two additional panels are available to allow you to enter options relating to the remote node that the database is on and the type of security being used.

Fill in the Node information if you plan to use the Control Center since this affects the behavior of this tool. The system and instance names are given by the values of DB2SYSTEM and DB2INSTANCE at the server. You should also select the operating system of the remote system.

You can also specify where authentication of the user takes place: at the server (the default), at the client, on a host or OS/400, or on a DCE server. You can also choose to use SOCKS security for TCP/IP connections, which allows you to access a remote database outside of your firewall.
Once you click **Finish** on the CA Catalog screen, the program attempts to catalog the database.

At this point, you can test the connection to make sure that the client can communicate with the database. If the test is successful, you can use an application to access the DB2 database.
The Control Center is the central point of administration for DB2. The Control Center provides the user with the tools necessary to perform typical database administration tasks. It allows easy access to other server administration tools, gives a clear overview of the entire system, enables remote database management, and provides step-by-step assistance for complex tasks.

So why did we go through all that pain of learning how to create databases using a command? Although the control center makes your life easier, there are times when you want to create scripts that automatically create objects or invoke database maintenance. The Control Center can help you generate, manage, and schedule these scripts, but they are all run as DB2 commands. In some cases, the Control Center may not be available on the operating system you are using, so you have no alternative but to use DB2 commands.

The screen that is displayed will be similar to one on the next page.
The Systems object represents both local and remote machines. To display all the DB2 systems that your system has cataloged, expand the object tree by clicking on the plus sign (+) next to the Systems folder. The left portion of the screen lists available DB2 systems (local and remote). From this example, the system BOB-LTOP contains a DB2 instance, DB2, in which the database SAMPLE is located. When the Tables folder is highlighted, details about each system is shown in the contents pane. A number of the existing tables in the SAMPLE database are displayed.

The main components of the Control Center are listed below:

- **Menu bar** — used to access Control Center functions and online help.
- **Tool bar** — used to access the other administration tools.
- **Objects pane** — this is shown on the left-hand side of the Control Center window. It contains all the objects that can be managed from the Control Center as well as their relationship to each other.
- **Contents pane** — this is found on the right side of the Control Center window and contains the objects that belong or correspond to the object selected on the objects pane.
- **Contents pane toolbar** — these icons are used to tailor the view of the objects and information in the contents pane. These functions can also be selected in the **View** menu.
Hover help is also available in the Control Center, providing a short description for each icon on the toolbar as you move the mouse pointer over the icon.

Clicking the View button provides a way to create and save a customized view of columns selected, and filtering and sort criteria.
The Control Center can be used to create and manage your databases. Remember the CREATE DATABASE command you used earlier to create your first database? Here's how you would go about doing that with the Control Center.

On the left side of the Control Center (the Objects Pane), right click on Database. This brings up a menu of options that are available for Databases. In this case, you would select Create, then Database Using Wizard.

Once you have selected this option, DB2 will present a series of panels that you need to fill in to create a database. The next series of panels shows you how to use these wizards to simplify some of the common DBA tasks.
The **Create Database Wizard** prompts you through a number of steps to generate a database. The first screen asks for the name of the database, the default drive for creating it, and an alias name. You can also add a comment about contents of the database.
A table space is a logical storage structure where the data for a database is stored. It consists of one or more physical storage containers, and is associated with one memory structure called a bufferpool.

Table spaces are categorized by the method used to access the data:

- **System-managed space (SMS)**—This type of table space is managed by the operating system and utilizes the O/S disk processes and data buffers. Therefore, the data access time can be slower, but this type of table space is relatively easy to manage.

- **Database-managed space (DMS)**—This type of space is managed directly by the DB2 database manager and bypasses the O/S system data buffers. Therefore, the access time can be faster, but this type of table space is potentially more difficult to manage.

Table spaces are divided up in terms of pages and extents. A page is the smallest quantity of data that can be retrieved from disk in one I/O operation. An extent is a set of pages grouped contiguously to minimize I/O operations and improve performance. Both page size and extent size are defined when a table space is created and cannot be changed.
The next three panels ask the user to fill in information on how they would like the USERSPACE1, SYSCATSPACE, and TEMPSPACE1 table spaces to be created. The Low Maintenance option creates SMS table spaces for you, while High Performance requires that you specify the devices and file systems that you plan to use for these table spaces.

For SMS, you can specify which containers (directories) that you want to allocate to that table space. For DMS, you can specify which containers (files, or devices) that you want to allocate to that table space. Click Add to define the containers being used.
If you do not specify containers or files for your table spaces, DB2 automatically generates one for you on the default drive that you specified on an earlier panel.
There are two performance parameters that can affect how your database will perform. The first one is EXTENTSIZE parameter and the other is the PREFETCHSIZE parameter.

- **EXTENTSIZE**

  An *extent* is a contiguous unit of space within a container of a table space. Database objects are stored in pages within DB2 (except for LOBs and LONG VARCHARs). These pages are grouped into extents. The extent size is defined at the table space level. Once the extent size is established for the table space, it cannot be altered. A database configuration parameter DFT_EXTENT_SZ specifies the default extent size for all table spaces in the database. The range this value can take is from 2 pages to 256 pages, so, for example, 8 KB to 1024 KB for 4 KB pages or 16 KB to 2048 KB for 8 KB pages. During table space create time, this figure can be overridden by using the EXTENTSIZE parameter in the CREATE TABLESPACE statement.

- **PREFETCHSIZE**

  Sequential prefetching is the ability of the database manager to read pages in advance of those pages being referenced by a query in anticipation of being required by the query. This asynchronous retrieval can reduce execution times significantly. You can control how aggressively the prefetching is performed by changing the PREFETCHSIZE parameter on the CREATE TABLESPACE statement. By default, this value is set to the DFT_PREFETCH_SZ database configuration parameter. This value represents how
many pages are read at a time when a prefetch request is triggered by DB2. By setting this value to a multiple of the extent size, multiple extents can be read in parallel. This function is even more effective when the containers for the table space are on separate hard disks.

The default values for these parameters are adequate for many applications, but you may want to experiment with higher PREFETCHSIZE settings for applications that do heavy queries or analyze large amounts of data.
The next option is the code page and collating sequence screen.

When a DB2 application is bound to a DB2 database, the application and database code page are compared. If the code pages are not equal, code page conversion is attempted for each SQL statement. If you are using a code page other than that of the database you are accessing, it is important to ensure that the code pages are compatible and conversion can be accomplished.

By default, the collating sequence of a database is defined according to the code set used in the CREATE DATABASE command. If you specify the option Collate using system, the data values are compared based on the territory specified for the database. If the option to collate using identity is used, all values are compared using their binary representation in a byte-to-byte manner. When you need to store data in its native (binary) format, avoid using data types with code pages. It is generally advantageous to have the application and the database code page the same to avoid the code page conversion process.
Once all of your parameters have been entered into the system, the Create Database Wizard will present a summary screen with all of the selections that you have made.

One extremely useful feature of this summary page is the ability to Show Command. Clicking Show Command displays the DB2 command that will be used to CREATE the database.

You can save this command for later execution, or cut and paste it into a script that you might be developing. If you are satisfied with the parameters that you have entered into the system, click Finish on the Summary screen to have the database created.
While the Control Center is certainly useful for creating databases, it has a lot of additional functionality that lets you create, modify, or delete almost any database object. Let's take a look at what else the Control Center has in it.

On the right size of the screen we have the objects pane. If you want to create a new object, place your mouse on the object type (table) and right click. For most objects this presents a menu of options, including one for creating the object with a wizard.
You should become familiar with these wizards. Initially these wizards can be extremely useful in determining how the DB2 commands are generated. Taking advantage of **Show Command** can be a great learning tool. Even experienced DBAs aren't ashamed to use the Control Center to generate seldom used commands!
New objects are created by right clicking on the object name in the objects pane. If you need to modify or delete an object, you need to display it in the right side of the Control Center, or the contents pane.

In this example panel, the tables found in the sample database are viewed in the contents pane. Now that these tables are displayed, you can modify these objects by right clicking on the object you want to modify.
3-52 Creating and Accessing DB2 Databases

For table objects, there are a large number of actions that can be done. Some of the more common actions are:

- Viewing the sample contents of the table
- Altering the table
- Reorganizing the table

Two commands of interest are the **Show Related** and **Generate DDL** commands. The **Show Related** displays all database objects that are related to this table, like indexes, other tables, or table spaces. The **Generate DDL** command reverse engineers the definition of this table so that you can recreate the table design in the event of a failure. (Of course, you always backup your database, correct?)

Additional information on the DB2 **Control Center** can be found in the online help supplied with the tool. Also, the **DB2 Administration Guide** volumes and the **DB2 Command Reference** hold a wealth of information on database features and functions, and how to design a database for the best performance. These books are an excellent reference and should be kept close at hand when designing your databases!
Summary

You should now be able to:
- Create a DB2 database
- Catalog a remote database or local database or DRDA
- Use the DB2 GUI tools to create, access and manipulate DB2 objects
- Create a table
Exercises
In this exercise, you will create a database that will be used as the sample database for the remainder of the course. After the database has been created, you will explore the default settings.

1. The student will create a simple default database storesdb and use it to explore the default settings.
2. The DB2 command line interface will be used to examine the configuration file (db2 "GET DB CFG") and set various database level configuration parameters (db2 "UPDATE …").
3. The student will use the GUI tools to create a test database, add one table to that database, and alter that table.

The student will compare the information in the system catalog tables — especially syscat.tables (view) — with container and file names at the operating system level.

1.1 In this exercise, you will use the DB2 Command Line Processor. This program runs in a DOS window and provides the environment and settings needed to connect to your server instance running on the Windows workstation. Start this interface now:

   Start > Programs > IBM DB2 > DB2 Command Line Processor

V8

What prompt do you get?

1.2 At this prompt, enter the CREATE DATABASE storesdb statement to create a database named storesdb using the default location and parameters. Note: this will take probably 40 or 50 seconds to complete since the database has to be set up and catalog tables created.

1.3 Connect to this database with CONNECT TO storesdb — this will be the first connection to this database. We will use this connection shortly to explore the database and its settings.

1.4 List basics of the current connection with the LIST ACTIVE DATABASES command. This will give information to you on your current connection.

   In what directory is your database? Note: this is relative to the installation directory (here D:\DB2); your information may be different.
1.5 Use a Windows Explorer window to open this directory. How many subdirectories are there? What do you find in each of these subdirectories? Can you specify the purpose of each directory?

What are the suffixes of the files in the directory that holds the database catalog tables?

Create a table \( t \) at the DB2 prompt using the following syntax:

```
CREATE TABLE t (i INTEGER).
```

Find whether a file was created to hold this table and where it is located.

1.6 Use the GET DBM CONFIG command at the DB2 prompt to list your server instance configuration settings.

1.7 Use the GET DB CONFIG FOR \texttt{storesdb} command at the DB2 prompt to list your database configuration settings for this database.

1.8 How many log files are configured (LOGPRIMARY) for this database?

1.9 Create a directory that you will use for a backup. Name the directory \texttt{db2backup}.

Open a DB2 Command Window and create the backup directory.

1.10 Update the configuration for this database to the following settings:

```
LOGPRIMARY = 5
LOGRETAIN = ON
```

These are non-dynamic database parameters, so you must also take a backup at this point to gain access to the database.

In a DB2 Command Window, execute a backup command as shown:

```
db2 "backup db storesdb to C:\db2backup"
```

Verify the new values by rerunning the DB2 GET DB CONFIG command.

Did this change the number of files in the SQLOGDIR subdirectory?

What is the new value for LOGRETAIN?

Disconnect from your current database. Reconnect. Look again at the number of files in the SQLOGDIR directory.
Exercise 2

In this exercise you will use the GUI tools to create a test database, add one table to that database, and alter that table. You will look at the system catalog tables — especially `sysibm.systables` (view) — and compare the information found here with the container and file names at the operating system level that we found in exercise 1.

2.1 Start the DB2 Control Center GUI. Connect to your existing `storesdb` database (from exercise 1).

2.2 Right-click on **Databases** and select **Create > Database Using Wizard** and use this interface to create a new database named **test**. The only parameter that you will need to enter is the name of the database ("test") — use defaults for the other possibilities. See what your choices are, but just select **Next** in each case. There are seven screens. Note: it may take over a minute for the actual database to be created after you click **Finish**.

2.3 Left-click and open the explorer-style [+] mark next to your new database ("test") and left-click on tables. You will see the system catalog tables for this database.

What schemas are currently in use for tables in this database?

In what table spaces are the tables?

Which directory does this correspond to on disk (from Exercise 1.4)?

2.4 Right-click on **Tables** in the left-pane. Select **Create**. You will use the GUI interface to create your own table (**mytable**) in the **test** database. Create the table with at least one column, e.g., `i` with data type INTEGER and constraint NOT NULL (uncheck the box "nullable").

What is your choice of table spaces at this time? What director(ies) do these tables space(s) correspond to?

Finish creating the table, but before clicking on the **OK** button, choose **Show SQL**. Write down the SQL that the wizard uses for creating the table.
Review the information about the table in the Control Center.

Find the file on disk using Windows Explorer. Verify the date and time of creation against the current clock setting. What is the full path name of the file (yours may be different from the one shown)?

The database catalog table `sysibm.systables` contains information about all of the tables in the database. There is a view created on this table for your use with the `syscat` schema. To view table information as requested above, use the following select command syntax on the `syscat.tables` view:

```
SELECT tabname, tableid, tbspaceid
FROM syscat.tables WHERE tabname = your_tablename
```

2.5 Use the Control Center to alter this table. Add another column `c` that is CHAR(10) and allows NULLs. Again use Show SQL before you commit the update. Is the file name the same after the change?

2.6 Add another table using the Control Center. Right-click on Tables in the left-pane. Select Create. Create table `yourtable` in the `test` database. Create the table with at least one column, e.g., `i` with data type INTEGER and constraint NOT NULL (uncheck the box "nullable"). Again use Show SQL before you commit the update.

Find the file on disk using Windows Explorer. Verify the date and time of creation against the current clock setting. What is the full path name of the file (yours may be different from the one shown)?
Solutions
In this exercise, you will create a database that will be used as the sample database for the remainder of the course. After the database has been created, you will explore the default settings.

1. The student will create a simple default database `storesdb` and use it to explore the default settings.

2. The DB2 command line interface will be used to examine the configuration file (`db2 "GET DB CFG"`) and set various database level configuration parameters (`db2 "UPDATE ..."`).

3. The student will use the GUI tools to create a test database, add one table to that database, and alter that table.

The student will compare the information in the system catalog tables—especially `syscat.tables` (view)—with container and file names at the operating system level.

1.1 In this exercise, you will use the DB2 Command Line Processor. This program runs in a DOS window and provides the environment and settings needed to connect to your server instance running on the Windows workstation. Start this interfaced now:

```
Start > Programs > IBM DB2 > DB2 Command Line Processor
```

What prompt do you get?

```
db2 =>
```

1.2 At this prompt, enter the CREATE DATABASE `storesdb` statement to create a database named `storesdb` using the default location and parameters. Note: this will take probably 40 or 50 seconds to complete since the database has to be set up and catalog tables created.

```
db2 => CREATE DATABASE storesdb
DB20000I  The CREATE DATABASE command completed successfully.
```
1.3 Connect to this database with CONNECT TO storesdb — this will be the first connection to this database. We will use this connection shortly to explore the database and its settings.

```sql
db2 => CONNECT TO storesdb
```

Database Connection Information

- Database server = DB2/NT 7.2.5
- SQL authorization ID = DB2ADMIN
- Local database alias = STORESDB

1.4 List basics of the current connection with the LIST ACTIVE DATABASES command. This will give information to you on your current connection.

```sql
db2 => LIST ACTIVE DATABASES
```

Active Databases

- Database name = STORESDB
- Applications connected currently = 1
- Database path = D:\DB2\NODE0000\SQL00001\n
In what directory is your database? Note: this is relative to the installation directory (here D:\DB2); your information may be different.

```
D:\DB2\NODE0000\SQL00001\n```

1.5 Use a Windows Explorer window to open this directory. How many subdirectories are there? What do you find in each of these subdirectories? Can you specify the purpose of each directory?

- DB2EVENT - event information; currently empty.
- SQLOGDIR - permanent logs; there are currently three (the detail - see below Q1.7).
- SQLT0000.0 - database catalog tables; 187 files (DB2 UDB 7.2 fixpack 7).
- SQLT0001.0 - temporary files; currently empty.
- SQLT0002.0 - user created tables; currently empty.
What are the suffixes of the files in the directory that holds the database catalog tables?

-.DAT
-.INX
-.LB
-.LBA
-.LF
-.NAM

Create a table $t$ at the DB2 prompt using the following syntax:

```
CREATE TABLE t (i INTEGER);
```

Find whether a file was created to hold this table and where it is located.

**The file created is in the SQLT0002.0 subdirectory.**

1.6 Use the GET DBM CONFIG command at the DB2 prompt to list your server instance configuration settings.

```
db2 => GET DBM CONFIG
```

**Database Manager Configuration**

Node type = Database Server with local clients

<table>
<thead>
<tr>
<th>Database manager configuration release level</th>
<th>0x0900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum total of files open (MAXTOTFILOP)</td>
<td>16000</td>
</tr>
<tr>
<td>CPU speed (millisec/instruction) (CPUSPEED)</td>
<td>7.478784e-007</td>
</tr>
</tbody>
</table>

| Max number of concurrently active databases (NUMDB) | 3 |
| Data Links support (DATALINKS) | NO |
| Federated Database System Support (FEDERATED) | YES |
| Transaction processor monitor name (TP_MON_NAME) | = |

| Default charge-back account (DFT_ACCOUNT_STR) | = |

| Java Development Kit 1.1 installation path (JDK11_PATH) | = |

| Diagnostic error capture level (DIAGLEVEL) | 3 |
| Notify Level (NOTIFYLEVEL) | 2 |
| Diagnostic data directory path (DIAGPATH) | = |

<table>
<thead>
<tr>
<th>Default database monitor switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer pool (DFT_MON_BUFPOOL)</td>
</tr>
<tr>
<td>Lock (DFT_MON_LOCK)</td>
</tr>
<tr>
<td>Sort (DFT_MON_SORT)</td>
</tr>
<tr>
<td>Statement (DFT_MON_STMT)</td>
</tr>
<tr>
<td>Table (DFT_MON_TABLE)</td>
</tr>
<tr>
<td>Unit of work (DFT_MON_UOW)</td>
</tr>
</tbody>
</table>
SYSADM group name                        (SYSADM_GROUP) =
SYSCTRL group name                      (SYSCTRL_GROUP) =
SYSMAINT group name                    (SYSMAINT_GROUP) =

Database manager authentication        (AUTHENTICATION) = SERVER
Cataloging allowed without authority   (CATALOG_NOAUTH) = YES
Trust all clients                      (TRUST_ALLCLNTS) = YES
Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT

Default database path                       (DFTDBPATH) = D:

Database monitor heap size (4KB)           (MON_HEAP_SZ) = 12
UDF shared memory set size (4KB)           (UDF_MEM_SZ) = 256
Java Virtual Machine heap size (4KB)      (JAVA_HEAP_SZ) = 2048
Audit buffer size (4KB)                   (AUDIT_BUF_SZ) = 0

Backup buffer default size (4KB)           (BACKBUFSZ) = 1024
Restore buffer default size (4KB)          (RESTBUFSZ) = 1024

Agent stack size                         (AGENT_STACK_SZ) = 16
Minimum committed private memory (4KB)     (MIN_PRIV_MEM) = 32
Private memory threshold (4KB)            (PRIV_MEM_THRESH) = 1296

Sort heap threshold (4KB)                  (SHEAPTHRES) = 10000

Directory cache support                   (DIR_CACHE) = YES

Application support layer heap size (4KB) (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)      (RQRIOLBK) = 32767
DOS requester I/O block size (bytes)      (DOS_RQRIOLBK) = 4096
Query heap size (4KB)                     (QUERY_HEAP_SZ) = 1000
DRDA services heap size (4KB)             (DRDA_HEAP_SZ) = 128

Priority of agents                       (AGENTPRI) = SYSTEM
Max number of existing agents             (MAXAGENTS) = 200
Agent pool size                          (NUM_POOLAGENTS) = 4 (calculated)
Initial number of agents in pool          (NUM_INITAGENTS) = 0
Max number of coordinating agents         (MAX_COORDAGENTS) = MAXAGENTS
Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
Max number of logical agents              (MAX_LOGICAGENTS) = MAX_COORDAGENTS

Keep DARI process                        (KEEPDARI) = YES
Max number of DARI processes             (MAXDARI) = MAX_COORDAGENTS
Initialize DARI process with JVM          (INITDARI_JVM) = NO
Initial number of fenced DARI process     (NUM_INITDARIS) = 0

Index re-creation time                   (INDEXREC) = ACCESS

Transaction manager database name         (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec)         (RESYNC_INTERVAL) = 180

SPM name                                 (SPM_NAME) = IBM_VC5J
SPM log size                             (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit                   (SPM_MAX_RESYNC) = 20
SPM log path                             (SPM_LOG_PATH) =
NetBIOS Workstation name (NNAME) =

TCP/IP Service name (SVCENAME) = db2cDB2
APPC Transaction program name (TPNAME) =
IPX/SPX File server name (FILESERVER) =
IPX/SPX DB2 server object name (OBJECTNAME) =
IPX/SPX Socket number (IPX_SOCKET) = 879E

Discovery mode (DISCOVER) = SEARCH
Discovery communication protocols (DISCOVER_COMM) = TCPIP
Discover server instance (DISCOVER_INST) = ENABLE

Directory services type (DIR_TYPE) = NONE
Directory path name (DIR_PATH_NAME) = /./subsys/database/
Directory object name (DIR_OBJ_NAME) =
Routing information object name (ROUTE_OBJ_NAME) =
Default client comm. protocols (DFT_CLIENT_COMM) =
Default client adapter number (DFT_CLIENT_ADPT) = 0

Maximum query degree of parallelism (MAX_QUERYDEGREE) = ANY
Enable intra-partition parallelism (INTRA_PARALLEL) = NO

No. of int. communication buffers(4KB) (FCM_NUM_BUFFERS) = 512
Number of FCM request blocks (FCM_NUM_RQB) = 256
Number of FCM connection entries (FCM_NUM_CONNECT) = (FCM_NUM_RQB * 0.75)
Number of FCM message anchors (FCM_NUM_ANCHORS) = (FCM_NUM_RQB * 0.75)

1.7 Use the GET DB CONFIG FOR storesdb command at the DB2 prompt to list your database configuration settings for this database.

   db2 => GET DB CONFIG FOR storesdb

   Database Configuration for Database storesdb

   Database configuration release level = 0x0900
   Database release level = 0x0900
   Database territory = US
   Database code page = 1252
   Database code set = IBM-1252
   Database country code = 1
   Dynamic SQL Query management (DYN_QUERY_MGMT) = DISABLE
   Directory object name (DIR_OBJ_NAME) =
   Discovery support for this database (DISCOVER_DB) = ENABLE
   Default query optimization class (DFT_QUERYOPT) = 5
   Degree of parallelism (DFT_DEGREE) = 1
   Continue upon arithmetic exceptions (DFT_SQLMATHWARN) = NO
   Default refresh age (DFT_REFRESH_AGE) = 0
   Number of frequent values retained (NUM_FREQVALUES) = 10
   Number of quantiles retained (NUM_QUANTILES) = 20
Backup pending = NO
Database is consistent = YES
Rollforward pending = NO
Restore pending = NO

Multi-page file allocation enabled = NO

Log retain for recovery status = NO
User exit for logging status = NO

Data Links Token Expiry Interval (sec) (DL_EXPINT) = 60
Data Links Number of Copies (DL_NUM_COPIES) = 1
Data Links Time after Drop (days) (DL_TIME_DROP) = 1
Data Links Token in Uppercase (DL_UPPER) = NO
Data Links Token Algorithm (DL_TOKEN) = MAC0

Database heap (4KB) (DBHEAP) = 300
Catalog cache size (4KB) (CATALOGCACHE_SZ) = 16
Log buffer size (4KB) (LOGBUFSZ) = 8
Utilities heap size (4KB) (UTIL_HEAP_SZ) = 5000
Buffer pool size (pages) (BUFFPAGE) = 250
Extended storage segments size (4KB) (ESTORE_SEG_SZ) = 16000
Number of extended storage segments (NUM_ESTORE_SEGS) = 0
Max storage for lock list (4KB) (LOCKLIST) = 25

Max appl. control heap size (4KB) (APP_CTL_HEAP_SZ) = 64

Sort list heap (4KB) (SORTHEAP) = 256
SQL statement heap (4KB) (STMTHEAP) = 2048
Default application heap (4KB) (APPLHEAPSZ) = 128
Package cache size (4KB) (PCKCACHESZ) = (MAXAPPLS*8)
Statistics heap size (4KB) (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms) (DLCHKTIME) = 10000
Percent. of lock lists per application (MAXLOCKS) = 22
Lock timeout (sec) (LOCKTIMEOUT) = -1

Changed pages threshold (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners (NUM_IOCLEANERS) = 1
Number of I/O servers (NUM_IOSERVERS) = 3
Index sort flag (INDEXSORT) = YES
Sequential detect flag (SEQDETECT) = YES
Default prefetch size (pages) (DFT_PREFETCH_SZ) = 16

Track modified pages (TRACKMOD) = OFF

Default number of containers = 1
Default tablespace extentsize (pages) (DFT_EXTENT_SZ) = 32

Max number of active applications (MAXAPPLS) = 40
Average number of active applications (AVG_APPLS) = 1
Max DB files open per application (MAXFILOP) = 64
Log file size (4KB) (LOGFILSZ) = 250
Number of primary log files (LOGPRIMARY) = 3
Number of secondary log files (LOGSECOND) = 2
Changed path to log files (NEWLOGPATH) =
Path to log files =
D:\DB2\NODE0000\SQL00001\SQLOGDIR
First active log file =

Group commit count (MINCOMMIT) = 1
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100
Log retain for recovery enabled (LOGRETAIN) = OFF
User exit for logging enabled (USEREXIT) = OFF

Auto restart enabled (AUTORESTART) = ON
Index re-creation time (INDEXREC) = SYSTEM (ACCESS)
Default number of loadrec sessions (DFT_LOADREC SES) = 1
Number of database backups to retain (NUM_DB_BACKUPS) = 12
Recovery history retention (days) (REC_HIS_RETENTN) = 366

TSM management class (TSM_MGMTCLASS) =
TSM node name (TSM_NODENAME) =
TSM owner (TSM_OWNER) =
TSM password (TSM_PASSWORD) =

1.8 How many log files are configured (LOGPRIMARY) for this database?
   (LOGPRIMARY) = 3

1.9 Create a directory that you will use for a backup. Name the directory db2backup.
   Open a DB2 Command Window and create the backup directory.
   mkdir C:\db2backup

1.10 Update the configuration for this database to the following settings:
    LOGPRIMARY = 5
    LOGRETAIN = ON

   db2 => UPDATE DB CONFIG FOR storesdb USING LOGPRIMARY 5
   db2 => UPDATE DB CONFIG FOR storesdb USING LOGRETAIN on
   These are non-dynamic database parameters, so you must also take a backup at this point
to gain access to the database.
   In the DB2 Command Window, execute a backup command as shown:
   db2 "backup db storesdb to C:\db2backup"
Verify the new values by rerunning the DB2 GET DB CONFIG command.

Number of primary log files (LOGPRIMARY) = 5
Log retain for recovery enabled (LOGRETAI) = ON

Did this change the number of files in the SQLOGDIR subdirectory?
No.

What is the new value for LOGRETAI?
**LOGRETAI = RECOVERY**

Disconnect from your current database. Reconnect. Look again at the number of files in the SQLOGDIR directory.

```
db2 => CONNECT RESET
db2 => CONNECT TO storesdb
```

The number of primary log files is now 5. (Some configuration changes require that you disconnect all users and restart the database).

**FYI**
You may need to close the database for an offline backup. This is done by issuing the following commands:

```
db2 terminate
db2 force applications all
```
In this exercise you will use the GUI tools to create a test database, add one table to that database, and alter that table. You will look at the system catalog tables — especially `sysibm.systables` (view) — and compare the information found here with the container and file names at the operating system level that we found in exercise 1.

2.1 Start the DB2 Control Center GUI. Connect to your existing `storesdb` database (from exercise 1).

Start > Programs > IBM DB2 > Control Center

Systems > Instances > DB2 > Databases > storesdb

2.2 Right-click on Databases and select Create > Database Using Wizard and use this interface to create a new database named `test`. The only parameter that you will need to enter is the name of the database ("test") — use defaults for the other possibilities. See what your choices are, but just select Next in each case. There are seven screens. Note: it may take over a minute for the actual database to be created after you click Finish.

2.3 Left-click and open the explorer-style [+] mark next to your new database ("test") and left-click on tables. You will see the system catalog tables for this database.

What schemas are currently in use for tables in this database?
SYSIIBM

In what table spaces are the tables?
SYSCATSPACE

Which directory does this correspond to on disk (from Exercise 1.4)?
SQLT0000.0 - database catalog tables with 187 files (DB2 UDB 7.2 fixpak 7).

FYI The solution information in the labs of this course are representative of possible solutions and not definitive results. Your result may vary, depending on the software version, Fixpak level, operating system, etc.
2.4 Right-click on Tables in the left-pane. Select Create. You will use the GUI interface to create your own table (mytable) in the test database. Create the table with at least one column, e.g., i with data type INTEGER and constraint NOT NULL (uncheck the box "nullable").

What is your choice of table spaces at this time? What director(ies) do these tables space(s) correspond to?

**USERSPACE1 corresponding to subdirectory SQLT0002.0**

Finish creating the table, but before clicking on the OK button, choose Show SQL. Write down the SQL that the wizard uses for creating the table.

```
CREATE TABLE mytable
  ('I' INTEGER NOT NULL) DATA CAPTURE NONE
```

Review the information about the table in the Control Center.

Find the file on disk using Windows Explorer. Verify the date and time of creation against the current clock setting. What is the full path name of the file (yours may be different from the one shown)?

The database catalog table `sysibm.systables` contains information about all of the tables in the database. There is a view created on this table for your use with the `syscat` schema. To view table information as requested above, use the following select command syntax on the `syscat.tables` view:

```
SELECT tabname, tableid, tbspaceid
  FROM syscat.tables WHERE tabname = your_tablename
```

**On the Control Center, with Table Spaces selected in your test database, double-click on USERSPACE1 in the right pane.**

The table will be created as SQL000002.DAT in subdirectory SQLT0002.0 -- the full path name of the file is:

```
D:\DB2\NODE0000\SQL00002\SQLT0002.0\SQL000002.DAT
```

Use this SQL statement to find which file number is being used to store data for the mytable table:

```
SELECT tabname, tableid, tbspaceid
  FROM syscat.tables WHERE tabname = 'MYTABLE'
```

Tableid = 2 would mean the file is SQL000002.DAT
2.5 Use the Control Center to alter this table. Add another column c that is CHAR(10) and allows NULLs. Again use Show SQL before you commit the update. Is the file name the same after the change?

Yes.

2.6 Add another table using the Control Center. Right-click on Tables in the left-pane. Select Create. Create table yourtable in the test database. Create the table with at least one column, e.g., i with data type INTEGER and constraint NOT NULL (uncheck the box "nullable"). Again use Show SQL before you commit the update.

Find the file on disk using Windows Explorer. Verify the date and time of creation against the current clock setting. What is the full path name of the file (yours may be different from the one shown)?

**On the Control Center, with Table Spaces selected in your test database, double-click on USERSPACE1 in the right pane.**

The table will be created as SQL000003.DAT in subdirectory SQLT0002.0 — the full path name of the file is:

D:\DB2\NODE0000\SQL00002\SQLT0002.0\SQL000003.DAT

Use this SQL statement to find which file number is being used to store data for the yourtable table:

```sql
SELECT tabname, tableid, tbspaceid
FROM syscat.tables WHERE tabname = 'YOURTABLE'
```

Tableid = 3 would mean the file is SQL000003.DAT
Module 4

Basic SQL
At the end of this module, you will be able to:

- Explain the use of DDL SQL, DML SQL, and DCL SQL statements
- Determine which data type to use for object structures
- Use special registers
- Specify authorities and privileges
- Create and drop database objects
The fundamentals of SQL are based on a variety of statement types. These types are:

- Data Control Language (DCL) — used to provide access control to database objects
- Data Definition Language (DDL) — used to create, modify, or drop database objects
- Data Manipulation Language (DML) — used to select, insert, update, or delete data
A unit of work (transaction) has specific start and end boundaries. In DB2, it is assumed that, during an SQL statement, you are always in a unit of work. Therefore there is no begin unit-of-work statement.

To terminate a unit of work, use either the COMMIT statement or ROLLBACK statement.

- COMMIT
  Forces all data changes within the unit of work to be committed to the database storage files.
- ROLLBACK
  Abandons data changes in the unit of work, and returns changed data to its original values.

Executing either the COMMIT or ROLLBACK statements causes the current unit of work to be closed and a new unit of work to be established.
SQL is a language used to define and manipulate database objects. SQL is the language that you use to define a database table, insert data into the table, change the data in the table, and retrieve data from the table. Like all languages, SQL has a defined syntax and set of language elements.

Most SQL statements contain one or more of the following language elements:

- **Characters**
  - Single-byte characters can be a letter (A-Z, a-z, $, #, and @, or a member of an extended character set), a digit (0-9), or a special character (including comma, asterisk, plus sign, percent sign, ampersand, and several others).

- **Tokens**
  - A token is a sequence of one or more characters. A token cannot contain blank characters unless it is a delimited identifier (one or more characters enclosed by double quotation marks), or a string constant.

- **Identifiers**
  - An SQL identifier is a token that is used to form a name.

- **Data types**
  - The data type of a value determines how DB2 interprets the value. DB2 supports a large number of built-in data types and also supports user-defined data types (UDTs).
- **Constants**

  A constant specifies a value. Constants are classified as character, graphic, or hexadecimal string constants, or integer, decimal, or floating-point numeric constants.

- **Functions**

  A function is a relationship between a set of input data values and a set of result values. Database functions can be either built in or user defined. The argument of a column function is a collection of like values; a column function returns a single value. Examples of built-in column functions are AVG, MIN, MAX, and SUM. The arguments of a scalar function are individual scalar values; a scalar function returns a single value. Examples of built-in scalar functions are CHAR, DATE, LCASE, and SUBSTR.

- **Expressions**

  An expression specifies a value. There are string expressions, arithmetic expressions, and case expressions, which can be used to specify a particular result based on the evaluation of one or more conditions.

- **Predicates**

  A predicate specifies a condition that is true, false, or unknown about a given row or group:
  - A basic predicate compares two values (for example, \( x > y \)).
  - The BETWEEN predicate compares a value with a range of values.
  - The EXISTS predicate tests for the existence of certain rows.
  - The IN predicate compares one or more values with a collection of values.
  - The LIKE predicate searches for strings that have a certain pattern.
  - The NULL predicate tests for null values.

  The NOT qualifier can be used with the above predicates to negate their meaning.
The built-in data types can be classified as numeric, character string, graphic string, binary string, or date-time. There is also a special data type named DATALINK. A DATALINK value contains a logical reference to a file stored outside of the database.

Numeric data types include SMALLINT, INTEGER, BIGINT, DECIMAL\((p,s)\), REAL, and DOUBLE. All numbers have a sign and a precision. The precision is the number of bits or digits excluding the sign. The sign is considered positive if the value of a number is zero or more.

- **Small integer — SMALLINT**
  A small integer is a two-byte integer with a precision of five digits. The range of small integers is -32,768 to 32,767.

- **Large integer — INTEGER, or INT**
  A large integer is a four-byte integer with a precision of 10 digits. The range of large integers is -2,147,483,648 to 2,147,483,647.

- **Big integer — BIGINT**
  A big integer is an eight-byte integer with a precision of 19 digits. The range of big integers is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
■ Decimal — DECIMAL\((p,s)\), DEC\((p,s)\), NUMERIC\((p,s)\), or NUM\((p,s)\)

A decimal value is a packed decimal number with an implicit decimal point. Packed decimal numbers are stored in a variation of binary-coded decimal (BCD) notation. The position of the decimal point depends on the precision \((p)\) and the scale \((s)\) of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits. The range of decimals is \(-10^{31}+1\) to \(10^{31}-1\).

■ Single-precision floating-point — REAL

A single-precision floating-point number is a 32-bit approximation of a real number. The number can be zero or can range from \(-3.402E+38\) to \(-1.175E-37\), or from \(1.175E-37\) to \(3.402E+38\).

■ Double-precision floating-point — DOUBLE, DOUBLE PRECISION, or FLOAT

A double-precision floating-point number is a 64-bit approximation of a real number. The number can be zero or can range from \(-1.79769E+308\) to \(-2.225E-307\), or from \(2.225E-307\) to \(1.79769E+308\).
A character string is a sequence of bytes. Character strings include fixed-length character strings of type CHAR(n), and varying-length character strings of type VARCHAR(n), LONG VARCHAR, or CLOB(n). The length of the string is the number of bytes in the sequence.

- **Fixed-length character string — CHARACTER(n) or CHAR(n)**
  A fixed-length character string is between 1 and 254 bytes long. If a length is not specified, it is assumed to be one.

- **Varying-length character string — VARCHAR(n), CHARACTER VARYING(n), or CHAR VARYING(n)**
  VARCHAR(n) type strings are varying-length character strings that can be up to 32,672 bytes long.

- **Varying-length character string — LONG VARCHAR**
  LONG VARCHAR type strings are varying-length character strings that can be up to 32,700 bytes long.
Character large object string — CLOB($n[K|M|G]$

A character large object is a varying-length character string that can be up to 2,147,483,647 bytes long. If only $n$ is specified, the value of $n$ is the maximum length. If $nK$ is specified, the maximum length is $n*1,024$ (with a maximum value for $n$ of 2,097,152). If $nM$ is specified, the maximum length is $n*1,048,576$ (with a maximum value for $n$ of 2,048). If $nG$ is specified, the maximum length is $n*1,073,741,824$ (with a maximum value for $n$ of 2). A CLOB is used to store large single-byte character set (SBCS) character-based data or mixed (multibyte character set (MBCS) and SBCS) character-based data.
A graphic string is a sequence of bytes that represents double-byte character data. Graphic strings include fixed-length graphic strings of type GRAPHIC(n) and varying-length graphic strings of type VARGRAPHIC(n), LONG VARGRAPHIC, and DBCLOB(n). The length of the string is the number of double-byte characters in the sequence.

- **Fixed-length graphic string — GRAPHIC(n)**
  A fixed-length graphic string is between 1 and 127 double-byte characters long. If a length is not specified, it is assumed to be 1.

- **Varying-length graphic string — VARGRAPHIC(n)**
  VARGRAPHIC(n) type strings are varying-length graphic strings that can be up to 16,336 double-byte characters long.

- **Varying-length graphic string — LONG VARGRAPHIC**
  LONG VARGRAPHIC type strings are varying-length graphic strings that can be up to 16,350 double-byte characters long.
Double-byte character large object string — DBCLOB($n[K|M|G]$)

A double-byte character large object is a varying-length graphic string of double-byte characters that can be up to 1,073,741,823 characters long. If only $n$ is specified, $n$ is the maximum length. If $nK$ is specified, the maximum length is $n\times1,024$ (with a maximum value for $n$ of 1,048,576). If $nM$ is specified, the maximum length is $n\times1,048,576$ (with a maximum value for $n$ of 1,024). If $nG$ is specified, the maximum length is $n\times1,073,741,823$ (with a maximum value for $n$ of 1). A DBCLOB is used to store large DBCS (double-byte character set) character-based data.

Binary Strings

A binary string is a sequence of bytes. Binary strings include varying-length strings of type BLOB($n$), which are used to hold non-traditional data such as pictures, voice, or mixed media, and which can also hold structured data for user-defined types and user-defined functions.

Binary Large Object — BLOB($n[K|M|G]$)

A binary large object is a varying-length string that can be up to 2,147,483,647 bytes long. If only $n$ is specified, $n$ is the maximum length. If $nK$ is specified, the maximum length is $n\times1,024$ (with a maximum value for $n$ of 2,097,152). If $nM$ is specified, the maximum length is $n\times1,048,576$ (with a maximum value for $n$ of 2,048). If $nG$ is specified, the maximum length is $n\times1,073,741,824$ (with a maximum value for $n$ of 2).
Date-time data types include DATE, TIME, and TIMESTAMP. Date-time values can be used in certain arithmetic and string operations, and they are compatible with certain strings, but they are neither strings nor numbers.

- **DATE**
  A DATE is a three-part value (year, month, and day). The range of the year part is 0001 to 9999. The range of the month part is 1 to 12. The range of the day part is 1 to \( n \), where the value of \( n \) depends on the month. The length of a DATE column is 10 bytes.

Examples:
Assume that the column RECEIVED (timestamp) has an internal value equivalent to '1988-12-25-17.12.30.000000'.
- This example results in an internal representation of '1988-12-25'.
  \( \text{DATE}(\text{RECEIVED}) \)
- This example results in an internal representation of '1988-12-25'.
  \( \text{DATE}(\text{'1988-12-25'}) \)
- This example results in an internal representation of '1988-12-25'.
  \texttt{DATE('25.12.1988')}
- This example results in an internal representation of '0001-02-04'.
  \texttt{DATE(35)}

**TIME**

A TIME is a three-part value (hour, minute, and second). The range of the hour part is 0 to 24. The range of both the minute part and the second part is 0 to 59. If the hour is 24, the minute value and the second value are both zero. The length of a TIME column is eight bytes.

Example:

- Select all notes from the IN\_TRAY sample table that were received at least one hour later in the day (any day) than the current time.
  
  \begin{verbatim}
  SELECT * FROM IN\_TRAY
  WHERE TIME(RECEIVED) \geq\ CURRENT TIME + 1 HOUR
  \end{verbatim}

**TIMESTAMP**

A TIMESTAMP is a seven-part value (year, month, day, hour, minute, second, and microsecond). The range of the year part is 0001 to 9999. The range of the month part is 1 to 12. The range of the day part is 1 to \(n\), where the value of \(n\) depends on the month. The range of the hour part is 0 to 24. The range of both the minute part and the second part is 0 to 59. The range of the microsecond part is 000000 to 999999. If the hour is 24, the minute value, the second value, and the microsecond value are all zero. The length of a TIMESTAMP column is 26 bytes.

Example:

- Assume the column START\_DATE (date) has a value equivalent to 1988-12-25, and the column START\_TIME (time) has a value equivalent to 17.12.30.\n  
  \begin{verbatim}
  TIMESTAMP(START\_DATE,\ START\_TIME)
  \end{verbatim}
  
  Returns the value '1988-12-25-17.12.30.000000'.

**String representations of date-time values**

Although internal representations of DATE, TIME, and TIMESTAMP values are transparent to the user, dates, times, and timestamps can be represented by character strings, and the \texttt{CHAR} scalar function can be used to create a string representation of a date-time value.

- A string representation of a date value is a character string that starts with a digit and has a length of at least eight characters. Leading zeros can be omitted from the month and the day parts of the date value.
A string representation of a time value is a character string that starts with a digit and has a length of at least four characters. A leading zero can be omitted from the hour part of the time value, and seconds can be omitted entirely. If a value for seconds is not specified, it is assumed to be 0.

A string representation of a timestamp value is a character string that starts with a digit and has a length of at least 16 characters. The complete string representation of a timestamp has the form `yyyy-mm-dd hh.mm.ss.nnnnnn`. Leading zeros can be omitted from the month, day, or hour part of the timestamp value, and microseconds can be truncated or omitted entirely. If any trailing zeros are omitted from the microseconds part of the timestamp value, a value of 0 is assumed for the missing digits.
A special register is a storage area that is defined for an application process by the database manager and is used to store information that can be referenced in SQL statements. To illustrate how useful special registers can be, consider the `sales` table, which has a column named `sales_date`. The following statement inserts a new row into the `sales` table, setting the sales date to the value of the CURRENT DATE special register:

```
INSERT INTO sales (sales_date, sales_person, region, sales)
VALUES (CURRENT DATE, 'SMITH', 'Manitoba', 100)
```

You will learn more about the INSERT statement in later material.
The database manager creates and maintains two sets of system catalog views. All the system catalog views are created when a database is created; they cannot be explicitly created or dropped. The system catalog views are updated during normal operation. Most of the system catalog views cannot be modified using data manipulation language. SELECT privilege on all of the system catalog views is granted to PUBLIC by default.

One set of system catalog views belongs to the SYSCAT schema and are not directly updatable.

A second set of views (the SYSSTAT schema) represent a subset of the SYSCAT schema and contain statistical information used by the optimizer.

The database manager creates and maintains two sets of system catalog views. All the system catalog views are created when a database is created; they cannot be explicitly created or dropped. The system catalog views are updated during normal operation. Most of the system catalog views cannot be modified using data manipulation language. SELECT privilege on all of the system catalog views is granted to PUBLIC by default.

One set of system catalog views belongs to the SYSCAT schema; these views cannot be directly updated. A second set of views, representing a subset of views belonging to the SYSCAT schema, contain statistical information that is used by the optimizer. The optimizer is a component of the SQL compiler that chooses an access plan for a DML statement (see Data Manipulation Language, DML on a later page) by modeling the execution cost of alternative access plans and choosing the one with the lowest estimated cost. Views belonging to the SYSSTAT schema contain some columns that can be updated.

The system catalog views can be queried to obtain useful information about a database. For example, the following statement uses the NOT LIKE predicate (see page 4-5) to return the name of each user-defined table with an entry in syscat.tables, along with the number of columns in each of those tables, and table status (N = normal; C = check pending):
SELECT tabname, colcount, status
FROM syscat.tables
WHERE tabschema NOT LIKE 'SYS%'
ORDER BY tabname

Following is a partial result set returned by this query:

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>COLCOUNT</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_SCHED</td>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>DEPARTMENT</td>
<td>5</td>
<td>N</td>
</tr>
<tr>
<td>EMP_ACT</td>
<td>6</td>
<td>C</td>
</tr>
</tbody>
</table>
## Data Control Language, DCL

DCL is a subset of SQL that is used to provide access control to database objects.

Two levels of security control access to database objects:

- **The first level: authentication (using ATTACH)**
  
  Controls access to the DB2 instance and is managed by the operating system via a valid user ID and password.

- **The second level: authorities (using CONNECT)**
  
  Controls access to a database on the server via the SQL CONNECT statement.
  
  - The user is again authenticated with the user ID and password

---

DCL is a subset of SQL that is used to provide access control to database objects. There are two levels of security to control access to database objects. The first level (ATTACH), which controls access to the DB2 instance, is managed by the operating system. This level is called authentication, and involves verifying a user's identity through a valid user ID and password. The second level of security controls access to a database on the server (CONNECT), checking the user’s authority in that database.

To access a database on the server, you must establish a connection between your DB2 client and the database using the SQL CONNECT statement. The syntax of this statement allows you to specify a user ID and password, and DB2 uses them to authenticate you and checks your authorities in that database. You can also request a password change by supplying your user ID, old password, and new password twice. For example:

```
CONNECT TO sample USER shaman USING mypassword
   NEW newpassword CONFIRM newpassword
```

If you do not specify a user ID and password with the CONNECT statement, DB2 may use the user ID and password you logged on with at the client for authentication, or it may prompt you to supply a user ID and password.
The following message tells you that you have made a successful connection:

Database Connection Information
  Database product = DB2/NT 7.2.4
  SQL authorization ID = SHAMAN
  Local database alias = SAMPLE
After you are connected you can access the database, its objects, and the data, if you have the authority or necessary privileges to do so.

Authorities are broad categories of user rights. They include:

- **SYSADM**
  This authority level gives a user the ability to run utilities, issue database and database manager commands, and control database objects throughout the database manager instance. In addition to all of the capabilities granted to the other authorities, users with SYSADM authority can migrate a database, modify the Database Manager Configuration file, and grant DBADM authority.

- **SYSCTRL**
  This authority level enables a user to perform all administrative tasks, such as running maintenance and utility operations against the database manager instance and its databases. A user with SYSCTRL authority cannot access data directly unless specifically granted additional privileges. In addition to the functions of the SYSMAINT authority, a user with SYSCTRL or higher authority can create or drop a database, force applications, restore to a new database, create, drop, or alter a table space, and update a database, node, or distributed connection services (DCS) directory.
- **SYSMAINT**

  This authority level enables a user to perform maintenance activities without the ability to access data within the database instance. A user with SYSMAINT or higher authority can update Database Configuration files, back up a database or table space, restore to an existing database, restore a table space, perform roll-forward recovery, stop or start a database instance, run traces, and take system monitor snapshots of an instance or its databases.

- **DBADM**

  This authority level applies to individual databases. A user with DBADM authority on a database can perform any administrative task on that database, such as creating database objects, loading data, and monitoring database activity. A user with DBADM authority can read log files, create, activate, and drop event monitors, query the state of a table space, update history files, **quiesce** a table space, reorganize a table, and collect catalog statistics using the RUNSTATS utility. The creator of a database automatically has DBADM authority on that database.

- **LOAD**

  This authority level enables a user to perform load operations on a particular database. If a user only has LOAD authority, the user must also have table-level privileges, such as the INSERT privilege, on a table to be able to load data into the table. If a load operation is to replace existing data in a table, the user must also have the DELETE privilege on the table.

Privileges are specific rights that can be granted to users, allowing them to work with specific objects in the database. A privilege is the right to create or access a database object. Objects on which privileges can be held include databases, schemas, table spaces, tables, views, nicknames, servers, packages, and indexes.

If you create an object, you have full access to that object. This is known as having CONTROL privilege on the object. A user with CONTROL privilege on an object can let other users have access to the object, and can give other users permission to grant privileges on the object. Privileges can be granted or revoked using the SQL GRANT or REVOKE statement (see Using the GRANT statement or Using the REVOKE statement). Individual privileges, such as SELECT, INSERT, DELETE, and UPDATE, allow a user to perform a specific function, sometimes on a specific object.

To grant privileges on database objects, you must have SYSADM authority, DBADM authority, CONTROL privilege, or have the WITH GRANT OPTION (a selectable option on the GRANT statement) on that object. You must have SYSADM or DBADM authority to grant CONTROL privilege to another user. You must have SYSADM authority to grant DBADM authority.
# Schemas

The schema:

- Is a database object
- Is a collection of named objects, such as tables, views, triggers, and functions
- Provides a logical classification of objects in the database
- Is used as the first part of a two-part object name
- Has privileges associated with it
  - Privileges allow the schema owner to control which users have the privilege to create, alter, and drop objects in the schema

A schema is a collection of named objects, such as tables, views, triggers, and functions. Schemas provide a logical classification of objects in the database. A schema name is used as the first part of a two-part object name. For example: `smith.staff`. In this example, the fully qualified name of the `staff` table includes the schema name, `smith`, to distinguish it from any other table named `staff` in the system catalog. Yes, you can have multiples of the same named table in the same database! (must have different schemas)

The schema itself is a database object. A schema can be explicitly created using the CREATE SCHEMA statement; it can also be implicitly created when another object is created, if the user creating the object has IMPLICIT_SCHEMA privilege.

When a database is created, all users have IMPLICIT_SCHEMA privilege. This allows any user to create objects in any schema not already in existence. An implicitly created schema allows any user to create other objects in this schema.

Schemas have privileges associated with them. This allows the schema owner to control which users have the privilege to create, alter, and drop objects in the schema. A schema owner is initially given all of these privileges on the schema, with the ability to grant them to others. An implicitly created schema is owned by the system, and all users are initially given the privilege to create objects in that schema. A user with SYSADM or DBADM authority can change the privileges held by users on any schema, even one that was implicitly created.
An authorization ID is a character string that is obtained by the database manager when a connection is established between the database manager and an application process. It designates a set of privileges. Authorization IDs are used by the database manager to provide authorization checking of SQL statements; an authorization ID applies to every SQL statement.

An authorization name specified in an SQL statement is simply an identifier that is used within that statement. For example, an authorization name is used in a CREATE SCHEMA statement to designate the owner of the schema, and an authorization name is used in the GRANT or the REVOKE statement to designate the target of the grant or the revoke operation (see Using GRANT or Using REVOKE on the next few pages).
Authorities and privileges can be explicitly granted to an individual user or to a group through invocation of the GRANT statement. Several flavors of the GRANT statement are available, depending on the object on which privileges are being granted. Objects on which privileges can be granted include databases, table spaces, tables, views, indexes, packages, and schemas. In general, the GRANT statement looks like this:

```
GRANT privilege ON object-type object-name
    TO [{USER | GROUP | PUBLIC}] authorization-name
    [WITH GRANT OPTION]
```

Specifying the keyword PUBLIC grants the privilege to all users. Some operating systems allow users and groups to have the same name. In such cases, you can specify the optional keyword USER or GROUP to distinguish them. For example:

```
GRANT INSERT, DELETE ON TABLE staff
    TO USER rosita WITH GRANT OPTION
```

The WITH GRANT OPTION gives Rosita the ability to grant the INSERT or DELETE privilege to other users. Of course, if another user already has SYSADM authority, or DBADM authority on the database that contains the staff table, or CONTROL privilege on the staff table, that user already has the authority to grant the INSERT or the DELETE privilege on the staff table.
The REVOKE statement is used to explicitly remove authorities and privileges from a user or a group. Authorities and privileges can be explicitly revoked from an individual user or a group through invocation of the REVOKE statement. As with the GRANT statement, several flavors of the REVOKE statement are available, depending on the object from which privileges are being revoked. Objects from which privileges can be revoked include databases, table spaces, tables, views, indexes, packages, and schemas. In general, the REVOKE statement looks like this:

```
REVOKE privilege ON object-type object-name
    FROM [(USER | GROUP | PUBLIC)] authorization-name
```

Specifying the keyword PUBLIC revokes the privilege from all users. Some operating systems allow users and groups to have the same name. In such cases, you can specify the optional keyword USER or GROUP to distinguish them. In the following example, the optional keyword ALL is used to revoke all privileges held by Joanna on the staff table:

```
REVOKE ALL PRIVILEGES ON TABLE staff FROM joanna
```

To revoke privileges on an object, you must have SYSADM authority, DBADM authority on the database that contains the object, or CONTROL privilege on the object. Having the WITH GRANT OPTION on an object does not allow you to revoke privileges on that object.

Revoking privileges on an object from a group does not guarantee that none of the members of the group will have privileges on the object. If any members of the group have been granted privileges as individuals or as members of another group, they retain these privileges unless additional REVOKE statements are issued.
Using CREATE to Make New Objects

The CREATE statement is used to create database objects, including:

- Buffer pools
- Event monitors
- Functions
- Indexes
- Schemas
- Stored procedures
- Tables
- Table spaces
- Triggers
- Views

The system catalog is updated whenever you create a database object.

Consider the CREATE TABLE statement. This statement has a large number of options that let you define precisely the kind of table you want to create. In its simplest form, the CREATE TABLE statement requires only that you specify one or more columns and associated data types. For example:

```sql
CREATE TABLE org (  
    deptnumb SMALLINT NOT NULL,  
    deptname VARCHAR(14),  
    manager SMALLINT,  
    division VARCHAR(10),  
    location VARCHAR(13))
```

This statement specifies the creation of a five-column table called `org`. The `org` table is actually part of the **sample** database that comes with DB2. Each column represents an attribute of the organization. Data pertaining to these attributes can be collected and stored in the table. The first column, `deptnumb`, cannot have NULL values, because this column will represent a unique key for the table: If each value in this column is unique, a specific value can be used to uniquely identify a row in the table.
The DECLARE statement is similar to the CREATE statement, except that it is used to create temporary tables that exist only for the duration of a database connection. A table is the only object that can be declared. The system catalog is not updated when you declare a temporary table. You can declare a temporary table by using the DECLARE GLOBAL TEMPORARY TABLE statement.

In the above example, the DECLARE GLOBAL TEMPORARY TABLE statement is used to declare a temporary table named temp1, located in an existing user temporary table space named mytempspace. (The user temporary table space must exist.) The columns in this table will have the same names and definitions as the columns in the employee table. The rows of the temporary table are preserved (not deleted) whenever a COMMIT statement is processed. Finally, changes to the temporary table are not logged (this is the only option).

Temporary tables must be explicitly (or will be implicitly) qualified by the schema name session, because each session that defines a declared table has its own (possibly unique) description of that temporary table.
Using ALTER to Change Objects

The ALTER statement is used to change some of the characteristics of an existing database object, including:

- Buffer pools
- Tables
- Table spaces
- Views

You cannot alter an index.

- To change an index, you must drop it and create it with a different definition

Following is an example of the ALTER TABLE statement. In this example, a new column is being added to the org table that was created earlier with the CREATE TABLE statement. The new state column will eventually contain a two-character state code for each department record.

```
ALTER TABLE org ADD state char(2)
```
The DROP statement removes object definitions from the system catalog and therefore from the database itself. Here is an example of the DROP TABLE statement:

```
DROP TABLE org
```

Because database objects can be dependent on other database objects, dropping an object can result in a related object becoming invalid.
You should now be able to:

- Explain the use of DDL SQL, DML SQL, and DCL SQL statements
- Determine which data type to use for object structures
- Use special registers
- Specify authorities and privileges
- Create and drop database objects
Exercises
Exercise 1

This exercise is designed to solidify your knowledge base on DB2 SQL statements.

1.1 In DB2 SQL, what statement begins a unit of work?

1.2 If you want to return changed data to its original condition (and therefore not permanently save it), what SQL statement should be used?

1.3 What statement would you use to cause your data changes to become permanent in the database?

1.4 What special register would you use to retrieve the operating systems date?

1.5 How would you retrieve the logon user name?

1.6 Which system catalog view contains the list of all tables in the database?

1.7 Which system catalog view contains a list of all columns defined in the database?

1.8 How would you connect to a database as a different user that the one you are logged in as?

1.9 What are the five authorities in DB2?

1.10 Given that you have the authority in the database to do so, what SQL statement would you use to give privileges to others?

1.11 What SQL statement (DDL) would you use to make a regular table in the database?
1.12 What DDL statement would you use to create a temporary table?

1.13 Can you use the ALTER TABLE statement to insert columns in the middle of a table? Can you use the ALTER TABLE statement to add a column at the end of a table?

1.14 Can you ALTER an index?

1.15 What happens to the data in a table when you drop that table?
Exercise 2

This lab provides you with some experience in creating database objects, specifically tables. You will use these tables in the next module.

2.1 In a DB2 command window, connect to the SAMPLE database and create a table called scores, with 5 fields that will store the t1name (char 15), t1score (integer), t2name (char 15), t2score (integer), and gamedate (date). No fields can contain NULLS.

2.2 Oops! You forgot to specify a field to store the game location. Alter the scores table, and add the location field (char 15) after the gamedate field.

2.3 Again in a DB2 command window (and connected to the SAMPLE database) create a table named players. This table should have the following structure: lname (char 10), fname (char 10), address1 (char 30), address2 (char 30), city (char 15), state (char 2), zip (char 5), teamname (char 15), and startdate (date). A player’s first name and last name, and the team name, MUST be required.

2.4 Create a team table, tracking teamname (char 15), location (char 15), and teamtype (char 2). All fields require data.
Solutions
This exercise is designed to solidify your knowledge base on DB2 SQL statements.

1.1 In DB2 SQL, what statement begins a unit of work?

   In DB2, it is assumed that when connected to a database, the application is always in a unit of work, so there is no specific statement that starts a unit of work.

1.2 If you want to return changed data to its original condition (and therefore not permanently save it), what SQL statement should be used?

   Use the ROLLBACK SQL statement to return data to its original state.

1.3 What statement would you use to cause your data changes to become permanent in the database?

   You would use the COMMIT SQL statement to permanently store your data changes.

1.4 What special register would you use to retrieve the operating systems date?

   Use the CURRENT DATE special register in your SQL statement. Example:
   ```sql
   SELECT CURRENT DATE FROM org
   ```

1.5 How would you retrieve the logon user name?

   You would use the USER special register to retrieve the logon user account name. Example:
   ```sql
   SELECT USER FROM sysibm.sysdummy1
   ```

1.6 Which system catalog view contains the list of all tables in the database?

   The syscat.tables view tracks all of the tables in the database. Important columns of the syscat.tables view are:

   TABSCHEMA, TABNAME, DEFINER, TYPE, STATUS, BASE_TABNAME, CREATE_TIME, STATS_TIME, COLCOUNT, TABLEID, TBSpace, LOCKSIZE.
1.7 Which system catalog view contains a list of all columns defined in the database?

The `syscat.columns` view tracks all of the columns in the database. Important columns of the `syscat.columns` view are:

TABSCHEMA, TABNAME, COLNAME, COLNO, TYPENAME, LENGTH, SCALE, DEFAULT, NULLS.

1.8 How would you connect to a database as a different user that the one you are logged in as?

You would precede the user name by the keyword USER and the password by the keyword USING.

```
db2 "CONNECT TO sample USER marty USING gonehome"
```

1.9 What are the five authorities in DB2?

SYSADM, SYSCTRL, SYSMAINT, DBADM, LOAD

1.10 Given that you have the authority in the database to do so, what SQL statement would you use to give privileges to others?

Use the GRANT statement to give privileges to others. Use REVOKE to rescind privileges.

1.11 What SQL statement (DDL) would you use to make a regular table in the database?

The CREATE TABLE statement. Example:

```
CREATE TABLE parts (  
    PartID integer NOT NULL,  
    PartName CHAR(50) NOT NULL,  
    PartDescription CHAR(500) NOT NULL,  
    PartLength DECIMAL(3,2) NOT NULL,  
    PartWidth DECIMAL(3,2) NOT NULL,  
    PartHeight DECIMAL(3,2) NOT NULL,  
    PartWeight DECIMAL(3,2) NOT NULL  
)  
```
1.12 What DDL statement would you use to create a temporary table?

You would use the DECLARE GLOBAL TEMPORARY TABLE statement.

Example:

    DECLARE GLOBAL TEMPORARY TABLE session.temp1
        LIKE sales
        ON COMMIT PRESERVE ROWS
        NOT LOGGED
        IN usertempspace1

1.13 Can you use the ALTER TABLE statement to insert columns in the middle of a table? Can you use the ALTER TABLE statement to add a column at the end of a table?

   No. You cannot add columns to a table except at the end of the table. If you need to re-order the columns, you must drop and re-create the table with the new design. There is no inherent meaning in a particular order to columns, and no practical impact except when doing a SELECT * FROM this table.

1.14 Can you ALTER an index?

   No, you cannot alter an index. To make changes to an index, you must drop and re-create it.

1.15 What happens to the data in a table when you drop that table?

   The data is destroyed. The only way to regain that data is to perform a restore and recovery from a backup of the data in question. When the table is dropped, all related objects (indexes, constraints, triggers, summary tables referencing the table, referential integrity constraints) are also dropped.
This lab provides you with some experience in creating database objects, specifically tables. You will use these tables in the next module.

2.1 In a DB2 command window, connect to the SAMPLE database and create a table called scores, with 5 fields that will store the t1name (char 15), t1score (integer), t2name (char 15), t2score (integer), and gamedate (date). No fields can contain NULLS.

```
db2 "CONNECT TO sample"
db2 "CREATE TABLE scores (
  t1name CHAR(15) NOT NULL,
  t1score INTEGER NOT NULL,
  t2name CHAR(15) NOT NULL,
  t2score INTEGER NOT NULL,
  gamedate DATE NOT NULL
)
"
db2 "terminate"
```

2.2 Oops! You forgot to specify a field to store the game location. Alter the scores table, and add the location field (char 15) after the gamedate field.

```
db2 "CONNECT TO sample"
db2 "ALTER TABLE scores ADD location CHAR(15)"
db2 "terminate"
```

2.3 Again in a DB2 command window (and connected to the SAMPLE database) create a table named players. This table should have the following structure: lname (char 10), fname (char 10), address1 (char 30), address2 (char 30), city (char 15), state (char 2), zip (char 5), teamname (char 15), and startdate (date). A player’s first name and last name, and the team name, MUST be required.

```
db2 "CONNECT TO sample"
db2 "CREATE TABLE players ( 
  lname CHAR (10) NOT NULL,
  fname CHAR (10) NOT NULL,
  address1 CHAR (30),
  address2 CHAR (30),
  city CHAR (15),
  state CHAR (2),
  zip CHAR (5),
)
```

Basic SQL 4-41
2.4 Create a **team** table, tracking **teamname** (char 15), **location** (char 15), and **teamtype** (char 2). All fields require data.

```sql
db2 "CONNECT TO sample"
db2 "CREATE TABLE team (  
    teamname CHAR (15) NOT NULL,  
    location CHAR (15) NOT NULL,  
    teamtype CHAR (2) NOT NULL
)"
```

db2 "terminate"
Module 5

Intermediate SQL
Objectives

At the end of this module, you will be able to:

• Create objects
• Use a variety of table joins
• Craft a subquery with basic predicate
• Use scalar functions
• Explain the purpose of user-defined types and user-defined functions
• Consider performance issues
• Explain the purpose of the optimizer
• Identify the affect and scope of a COMMIT or ROLLBACK statement
To store data and control its use, you need to create objects within the database. In the next few pages, you will learn the DDL SQL used to create some of the more important objects in a database.

Using DDL: Creating Objects

Creating Tables and Adding Columns
- Create Constraints
- Create Index
- Create Trigger
- Create View
- View Materialization
The syntax for creating a table is platform-dependent. In a DB2 UDB for UNIX, Windows and OS/2 environment, you can code CREATE TABLE.....IN tablespace-name. This identifies the table space in which the table will be created. If the IN clause is not specified, then the table is created in one of three possible default table spaces, depending on which of them exists. If none of the default table spaces exist, the CREATE fails.

In an z/OS or OS/390 environment, there are three valid approaches for creating a table:

- CREATE TABLE... IN database-name.tablespace-name. This approach puts the table in a specified table space of a specified database.
- CREATE TABLE... IN DATABASE database-name. This approach puts the table in a specified database and creates a simple table space within that database to hold the table.
- CREATE TABLE.... This approach puts the table in the default database, DSNDB04, and creates a simple table space within that database to hold the table.

Data types BLOB, CLOB, and DBCLOB are available in DB2 UDB for OS/390 Versions 6 as well as on DB2 UDB for UNIX, Windows and OS/2 and DB2 UDB for AS/400.

Generally, you specify the columns of the table when you create the table. If you need to add a column to the table later, you can use the ALTER TABLE statement with the ADD option. The

The CREATE TABLE statement:

```
CREATE TABLE order (
    order_no    INTEGER NOT NULL,
    order_date  DATE NOT NULL,
    cust_no     SMALLINT NOT NULL
)
```

The ALTER TABLE statement:

```
ALTER TABLE order
    ADD order_ref INTEGER
```
new column must be defined as nullable or with NOT NULL WITH DEFAULT since the table may already contain rows at that point in time. If the column was defined with NOT NULL, DB2 will not know which value to assign to the column for the existing rows. If the column is defined as nullable, DB2 assigns a value of null to the column for existing rows; if it is defined with NOT NULL WITH DEFAULT, the appropriate default value (system-defined or user-defined) is assigned to the column for existing rows.

**Note**

In general you cannot drop a column, change a column's name, change a column's data type, or change a column's null attribute with ALTER TABLE. You have to drop the whole table and redefine it. Since dropping the table results in the immediate drop of all dependent objects, including indexes, views, privileges, etc., it is very risky to drop an object unless you have stored the SQL necessary to recreate the dependent objects somewhere outside the DB2 catalog. Otherwise, it will be very difficult to re-create the dependent objects. In DB2 UDB for UNIX, Windows, OS/2, OS/390 and AS/400 you can rename a table using RENAME TABLE and you can change the length of a VARCHAR-column using ALTER TABLE. In DB2 UDB AS/400, you can even DROP a Column of a table.

**For More Information**

Please refer to the following:

- *DB2 UDB for OS/390 and z/OS V7 SQL Reference* (SC26-9944-02).
- *IBM DB2UDB SQL Reference SQL Reference Volume 1 Version 8* (SC09-4844-00) and *Volume 2 Version 8* (SC09-4845-00).

```sql
CREATE TABLE new_order
    LIKE OLD_ORDER;

CREATE TABLE new_staff
    AS
    (SELECT COL, COL2, COL5 FROM staff)
    DEFINITION ONLY;
```

The above shows alternative syntax for creating a new table by taking the definitions from an existing table.
- **CREATE TABLE**...LIKE creates a new table with the same column definitions (column name, data type and null attribute) as the existing table.
  - No unique constraints, foreign key constraints, triggers, or indexes are created.
  - The table is not populated.
  - This syntax is available in DB2 UDB z/OS and OS/390 and DB2 UDB for UNIX, Windows and OS/2. It is not available in DB2 UDB for OS/400 V4 R4.
  - **CREATE TABLE** --- DEFINITION ONLY uses a query to define a new table.
  - The table is not populated using the result of query and the REFRESH TABLE statement cannot be used.
  - When the CREATE TABLE statement is completed, the table is no longer considered a summary table.
  - The columns of the table are defined based on the definitions of the columns that result from the fullselect. If the fullselect references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, nullability characteristic and column default value of the referenced table.
  - **CREATE TABLE**... DEFINITION ONLY is available on DB2 UDB for UNIX, Windows and OS/2.
Check constraints

Single-column constraints can be defined at the column level or the table level.

Multi-column constraints must be defined at the table level.

- Check constraints can be defined on a new table with the CREATE TABLE statement.
- Check constraints can be added to an existing table with the ALTER TABLE statement.
- Constraints may include the basic WHERE clause constructs:
  - Basic comparisons (>, <, =, >=, etc.)
  - BETWEEN
  - LIKE
  - IN
- Constraints may NOT include:
  - Subqueries
  - Column functions
  - Special registers (such as CURRENT DATE)
  - On OS/390, the NOT logical operator
## Creating Indexes

**Example of creating an index:**

```sql
CREATE UNIQUE INDEX order_ind
    ON order(order_no);

ALTER TABLE order
    ADD PRIMARY KEY(order_no);
```

DB2 must read (scan) a complete table if no index is specified on a column used in a predicate of a WHERE clause.

An index is an ordered collection of the key values in the column on which the index is defined. Each key value has a RID (Row IDentifier) or RIDs that points to the actual data row(s) which contain the key value.

A table can have many indexes. Each index can span one column or several columns. Each index can be unique or non unique.

The usefulness of an index depends on its key. Columns that you use frequently in WHERE clauses, joins, grouping and ordering operations are good candidates for indexes.

DB2 indexes allow duplicate values by default. If you want to prevent duplicate values in a key column, use CREATE UNIQUE INDEX.

The example creates a unique index called ORDER_IND on the ORDER_NO column of the ORDER table.
A unique key is a key (set of columns) that is constraint so that no two values are equal. The columns of a unique key cannot contain null values.

A primary key is a special case of a unique key. It is normally used to refer to the rows of the appropriate table.

- A table can have multiple unique keys, but cannot have more than one primary key. A key can either be defined as unique key or as primary key, but not both. In DB2 UDB for UNIX, Windows and OS/2 and DB2 UDB for AS/400, CREATE TABLE and ALTER TABLE can be used to identify a key as unique key. In DB2 for OS/390, a key can only be defined as unique key by means of the CREATE TABLE statement.

- For both DB2 UDB for OS/390, DB2 UDB for UNIX, Windows and OS/2, and DB2 UDB for AS/400 primary keys can be defined using CREATE TABLE or ALTER TABLE.

By definition, the values of a foreign key always match the values of a primary key or unique key.

A referential constraint is a rule, enforced by the database manager, that controls the relationship between a primary key/unique key and a foreign key. Referential constraints are defined to DB2 when the foreign key is defined. They are enforced when rows containing a foreign-key value are inserted, updated, or loaded.
DB2 prevents the updating of primary key/unique key values if there are matching foreign key values. The new values are not propagated to the appropriate rows of the dependent table. In other words, a primary key/unique key value can only be updated if no foreign key value matches the original value of the primary/unique key.
Referential Integrity — Delete Rules

The problem:
- When deleting a primary/unique key, what should be done to rows that contain matching foreign keys?

Delete rules:
- CASCADE—Delete the rows with the matching values in the dependent table
- SET NULL—Change the matching values in the dependent table to NULL
- RESTRICT or NO ACTION—Disallow the DELETE in the parent table if matching values exit in the dependent table

The delete rules are defined in the REFERENCES clause of the foreign key definition in the CREATE TABLE or ALTER TABLE statement.

The only difference between NO ACTION and RESTRICT is when the referential constraint is enforced. RESTRICT enforces the rule immediately and NO ACTION enforces the rule at the end of the statement.
The examples illustrate how to implement the relationships between the ORDER table and the ORDER_ITEM table and between the ARTICLE table and the ORDER_ITEM table.

DB2 for OS/390 requires that a unique index has been defined for the columns for the primary key before the primary key can be added by means of the ALTER TABLE statement. In DB2 UDB for UNIX, Windows and OS/2 and DB2 UDB for AS/400, if not provided, the unique index is defined automatically.

ALTER TABLE can also be used to add or drop check constraints and to add a column to an existing table as we have seen before.

```sql
ALTER TABLE order
  ADD PRIMARY KEY (order_no);
ALTER TABLE article
  ADD PRIMARY KEY (art_no);
ALTER TABLE order_item
  ADD PRIMARY KEY (order_no, art_no);

ALTER TABLE order_item
  ADD CONSTRAINT in_order
    FOREIGN KEY (order_no)
    REFERENCES order
    ON DELETE CASCADE;
ALTER TABLE order_item
  ADD CONSTRAINT OF_ARTICLE
    FOREIGN KEY (art_no)
    REFERENCES ARTICLE
    ON DELETE RESTRICT;
```
Creating Triggers

Example of creating a trigger:

```sql
CREATE TRIGGER re_order
AFTER UPDATE OF stock_qty ON article
REFERENCING NEW AS N OLD AS O
FOR EACH ROW
MODE DB2SQL
WHEN (N.stock_qty < 50 AND O.stock_qty >= 50)
INSERT INTO sup_orders
VALUES(N.art_no, N.stock_qty)
```

A trigger is a set of actions that will be executed when a defined event occurs. The triggering events can be the following SQL statements.

- INSERT
- UPDATE
- DELETE

Triggers are defined for a specific table and once defined, a trigger is automatically active. Trigger definitions are stored in the system catalog tables.

Triggers can be defined with the CREATE TRIGGER statement.

In the example:

- AFTER UPDATE OF STOCK_QTY ON ARTICLE indicates that a triggered action should take place after the STOCK_QTY column in has been updated in the ARTICLE table. The triggered action can be made to occur either before or after an event. The event can be an INSERT, an UPDATE of a specific column or columns, an UPDATE of any column of a table, or a DELETE.
REFERENCING is used to correlate the event with the triggered action. In this case, we want the row that is being added to the SUP_ORDERS table in the triggered action to contain the values in the ART_NO and STOCK_QTY columns, but we want these to be the values after the update of the ARTICLE table. In other words, we want the new values, not the old values.

We use the REFERENCING clause to designate the new values with a correlation name of N. We also use the REFERENCING clause to designate the old values with a correlation name of O. We want to ensure that the trigger is fired only if the old stock quantity is greater than or equal to 50 and the new stock quantity is less than 50.

These variables defined by the REFERENCING clause are called transition variables.

- FOR EACH ROW indicates that the trigger should fire for each row inserted. FOR EACH STATEMENT, which causes the trigger to fire once per triggering statement, is another option.
- MODE DB2SQL identifies the mode of the trigger. DB2SQL is the only valid mode at present.
- WHEN restricts the triggered action. The action will only be executed if the condition of the WHEN evaluates to TRUE.
- INSTEAD OF specifies that the associated triggered action replaces the action against the subject view. Only one INSTEAD OF trigger is allowed for each kind of operation on a given subject view (SQLSTATE 428FP).
- INSERT INTO SUP_ORDERS... is the triggered action. Note that it takes advantage of the correlation name established with the REFERENCING clause.
A view is a named specification of a result table. The specification is an SQL SELECT statement that is effectively executed whenever the view is referenced in an SQL statement.

The data needed to satisfy queries is often stored in many large tables. However, we often need only a small subset of this data in order to create reports. A view can act like a filter so that a report contains only the information we need, rather than a lot of extra detail.

In the example above:

- Assume a user needs the last names of the employees in the EMPLOYEE table shown on the visual and, for each employee, also the name of his/her department. However, the user should not be able to see the salaries of the employees. The department name must be obtained by joining to the DEPARTMENT table.
- When a view is created, a name is assigned to the view. The name has the same format as table names. It can be used to refer to the view in other SQL statements.
- The column list, which appears in parentheses after the view name, can be used to give the columns from the base tables different names in the view or to name calculated columns.
- The user of a view needs only the authority to access the view. Access to the original table is not needed. It is possible to give a user SELECT authority on a view without granting any authority on the base table.
In some cases, a view cannot be merged with a SELECT statement because the two cannot be combined into a single statement. In these cases, the view is materialized, that is, written to a work file on disk to be processed like a table.

In the example above, the view applies the SUM() function to the SALARY column to build a list of the departments and their total salaries. The subsequent SELECT statement determines the largest of the sums returned by the function. The SELECT statement cannot determine the largest sum until all sums have been determined. This requires writing the results of the view to disk so that the SELECT statement can scan the view result to find the largest sum.

The need to write to disk can cause a negative impact on performance. Therefore, view merge is preferred over view materialization.
In its simplest form, the SELECT statement can be used to retrieve all the data in a table. For example, to retrieve all the staff data from the sample database, issue:

SELECT * FROM staff;

Here is a partial result set returned by this query:

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>DEPT</th>
<th>JOB</th>
<th>YEARS</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Sanders</td>
<td>20</td>
<td>Mgr</td>
<td>7</td>
<td>18357.50</td>
</tr>
<tr>
<td>20</td>
<td>Pernal</td>
<td>20</td>
<td>Sales</td>
<td>8</td>
<td>18171.25</td>
</tr>
<tr>
<td>30</td>
<td>Marenghi</td>
<td>38</td>
<td>Mgr</td>
<td>5</td>
<td>17506.75</td>
</tr>
</tbody>
</table>

To restrict the number of rows in a result set, use the FETCH FIRST clause. For example:

SELECT * FROM staff FETCH FIRST 10 ROWS ONLY;

You can retrieve specific columns from a table by specifying a select list of column names separated by commas. For example:

SELECT name, salary FROM staff;

Use the DISTINCT clause to eliminate duplicate rows in a result set. For example:

SELECT DISTINCT dept, job FROM staff;
Use the \texttt{AS} clause to assign a meaningful name to an expression or an item in the select list. For example:

\begin{verbatim}
SELECT name, salary + comm AS pay FROM staff;
\end{verbatim}

Without the \texttt{AS} clause, the column name would have been 2, meaning that the derived column is the second column in the result set.
When you specify comparison conditions in WHERE clauses, the relational operators listed on the slide above are valid.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=</code></td>
<td>equals</td>
</tr>
<tr>
<td><code>!=</code> or <code>&lt;&gt;</code></td>
<td>does not equal</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>
Use the keyword LIKE in a WHERE clause to perform *variable text* queries of character fields based on wildcard matching. The slide above shows the wildcard symbols that you can use with LIKE.

The database server returns an error if you use the LIKE search operators with non-character data types. The following example returns an error because `customer_num` is a numeric data type:

```sql
SELECT * FROM customer 
WHERE customer_num LIKE '103'
```

Rewrite the query using the equals (=) operator.

```sql
SELECT * FROM customer 
WHERE customer_num = 103
```
Use the WHERE clause to select specific rows from a table or view. When building search conditions, be sure to:

- Apply arithmetic operations only to numeric data types
- Make comparisons only among compatible data types
- Enclose character values within single quotation marks
- Specify character values exactly as they appear in the database

Example:

```
SELECT name, job, salary FROM staff
WHERE job <> 'Mgr'
AND salary > 20000
```

Use the WHERE clause to select specific rows from a table or view by specifying one or more selection criteria, or search conditions. A search condition consists of one or more predicates. A predicate specifies something about a row that is either true or false.

- List the name, job title, and salary of staff members who are not managers and whose salary is greater than $20,000.

  ```
  SELECT name, job, salary FROM staff
  WHERE job <> 'Mgr'
  AND salary > 20000
  ```

- Find all names that start with the letter S.

  ```
  SELECT name FROM staff
  WHERE name LIKE 'S%'
  ```

In this example, the percent sign (%) is a wild card character that represents a string of zero or more characters.

A subquery is a SELECT statement that appears within the WHERE clause of a main query, and feeds its result set to that WHERE clause. For example:

```
SELECT lastname FROM employee
WHERE lastname IN
  (SELECT sales_person FROM sales
   WHERE sales_date < '01/01/1996')
```
A correlation name is defined in the FROM clause of a query, and can serve as a convenient short name for a table. Correlation names also eliminate ambiguous references to identical column names from different tables. For example:

```sql
SELECT e.salary FROM employee e
WHERE e.salary <
 (SELECT AVG(s.salary) FROM staff s)
```

**Note**

If you are using the DB2 command line utility to input DB2 commands, you should enclose the statement within double quotation marks to prevent your operating system from misinterpreting special characters, such as >; the greater-than symbol could be interpreted as an output redirection request by the operating system. If you are using the Command Line Processor (CLP) or the GUI tools to enter DB2 commands, you would not need to use the double quotes.
### Using ORDER BY to Sort Results

Use the ORDER BY clause to sort the result set by values in one or more columns.

Example:

```sql
SELECT fname, lname, state FROM customer
WHERE state in ('CA','AZ','OR')
ORDER BY customer_num, lname desc
```

Use the ORDER BY clause to sort the result set by values in one or more columns. The column names that are specified in the ORDER BY clause do not have to be specified in the select list. You can sort the result set in descending order by specifying DESC in the ORDER BY clause (ORDER BY salary DESC).

For example:

```sql
SELECT name, salary FROM staff
WHERE salary > 20000
ORDER BY salary
```
A join is a query that combines data from two or more tables. It is often necessary to select information from two or more tables, because no one table contains all the required information. A join adds columns to the result set. For example, a full join of two three-column tables produces a result set with six columns. The simplest join is one in which there are no specified conditions. For example:

```
SELECT deptnumb, deptname, id AS manager_id, name AS manager
FROM org, staff
WHERE manager = id
ORDER BY deptnumb
```

A join is a query that combines data from two or more tables. It is often necessary to select information from two or more tables, because no one table contains all the required information. A join adds columns to the result set. For example, a full join of two three-column tables produces a result set with six columns. The simplest join is one in which there are no specified conditions. For example:

```
SELECT deptnumb, deptname, id AS manager_id, name AS manager
FROM org, staff
WHERE manager = id
ORDER BY deptnumb
```

This statement returns all combinations of rows (cartesian product) from the `org` table and the `staff` table. The first three columns come from the `org` table, and the last four columns come from the `staff` table. Such a result set (the cross product of the two tables) is not very useful. What is needed is a join condition to refine the result set.

Following is a partial result set returned by the query shown in the slide:

<table>
<thead>
<tr>
<th>DEPTNUMB</th>
<th>DEPTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Head Office</td>
</tr>
<tr>
<td>15</td>
<td>New England</td>
</tr>
<tr>
<td>20</td>
<td>Mid Atlantic</td>
</tr>
</tbody>
</table>
Inner joins return only rows from the cross product that meet the join condition.

- If a row exists in one table but not the other, it is not included in the result set.

To explicitly specify an inner join, the previous query can be rewritten with an INNER JOIN operator in the FROM clause:

```sql
SELECT deptnumb, deptname, id AS manager_id, name AS manager
FROM org INNER JOIN staff
ON manager = id
ORDER BY deptnumb
```

Inner joins

The statement on the previous page is an example of an inner join. Inner joins return only rows from the cross product that meet the join condition. If a row exists in one table but not the other, it is not included in the result set.

In the above statement, the keyword `ON` specifies the join conditions for the tables being joined. Remember that `deptnumb` and `deptname` are columns in the `org` table, and that `manager_id` and `manager` are based on columns (`id` and `name`) in the `staff` table. The result set for the inner join consists of rows that have matching values for the `manager` and `id` columns in the left table (`org`) and the right table (`staff`), respectively.

**Note** When you perform a join on two tables, you arbitrarily designate one table to be the *left* table and the other one to be the *right* table.
Outer joins return rows that are generated by an inner join operation, plus rows that would not be returned by the inner join operation. There are three types of outer joins:

- Left outer join
- Right outer join
- Full outer join

A left outer join includes the inner join plus the rows from the left table that are not returned by the inner join. This type of join uses the LEFT OUTER JOIN (or LEFT JOIN) operator in the FROM clause.

A right outer join includes the inner join plus the rows from the right table that are not returned by the inner join. This type of join uses the RIGHT OUTER JOIN (or RIGHT JOIN) operator in the FROM clause.

A full outer join includes the inner join plus the rows from both the left table and the right table that are not returned by the inner join. This type of join uses the FULL OUTER JOIN (or FULL JOIN) operator in the FROM clause.
More complex queries can be constructed to answer more difficult questions. The following query is designed to generate a list of employees who are responsible for projects, identifying those employees who are also managers by listing the departments that they manage:

```sql
SELECT empno, deptname, projname
FROM (employee
    LEFT OUTER JOIN project
    ON respemp = empno)
    LEFT OUTER JOIN department
    ON mgrno = empno
```

The first outer join gets the name of any project for which the employee is responsible; this outer join is enclosed by parentheses and is resolved first. The second outer join gets the name of the employee's department if that employee is a manager.
You can combine two or more queries into a single query by using the UNION, EXCEPT, or INTERSECT set operator. Set operators process the results of the queries, eliminate duplicates, and return the final result set.

- The UNION set operator generates a result table by combining two or more other result tables.
- The EXCEPT set operator generates a result table by including all rows that are returned by the first query, but not by the second and any subsequent queries.
- The INTERSECT set operator generates a result table by including only rows that are returned by all the queries.

Following is an example of a query that makes use of the UNION set operator. The same query could use the EXCEPT or the INTERSECT set operator by substituting the appropriate keyword for UNION.

```sql
SELECT sales_person FROM sales
  WHERE region = 'Ontario-South'
UNION
SELECT sales_person FROM sales
  WHERE sales > 3
```
A GROUP BY example:

```
SELECT sales_date, MAX(sales) AS max_sales FROM sales
GROUP BY sales_date
```

This statement returns a list of sales dates from the `sales` table. The `sales` table in the `sample` database contains sales data, including the number of successful transactions by a particular sales person on a particular date. There is typically more than one record per date. The GROUP BY clause groups the data by sales date, and the MAX function in this example returns the maximum number of sales recorded for each sales date.

A different flavor of the GROUP BY clause includes the specification of the GROUPING SETS clause. Grouping sets can be used to analyze data at different levels of aggregation in a single pass.

For example:

```
SELECT YEAR(sales_date) AS year, region,
      SUM(sales) AS tot_sales
FROM sales
GROUP BY GROUPING SETS (YEAR(sales_date), region, ())
```
Here, the YEAR function is used to return the year portion of date values, and the SUM function is used to return the total in each set of grouped sales figures. The grouping sets list specifies how the data is to be grouped or aggregated. A pair of empty parentheses is added to the grouping sets list to get a grand total in the result set. The statement returns:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>REGION</th>
<th>TOT_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>155</td>
</tr>
<tr>
<td>-</td>
<td>Manitoba</td>
<td>41</td>
</tr>
<tr>
<td>-</td>
<td>Ontario-North</td>
<td>9</td>
</tr>
<tr>
<td>-</td>
<td>Ontario-South</td>
<td>52</td>
</tr>
<tr>
<td>-</td>
<td>Quebec</td>
<td>53</td>
</tr>
<tr>
<td>1995</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>1996</td>
<td>-</td>
<td>147</td>
</tr>
</tbody>
</table>

The same statement, but specifying the ROLLUP clause or the CUBE clause instead of the GROUPING SETS clause, returns a result set that provides a more detailed perspective on the data, such as summaries by location or time.

The HAVING clause is often used with a GROUP BY clause to retrieve results for groups that satisfy only a specific condition. A HAVING clause can contain one or more predicates that compare some property of the group with another property of the group or a constant. For example:

```
SELECT sales_person, SUM(sales) AS total_sales FROM sales
GROUP BY sales_person
HAVING SUM(sales) > 25
```

This statement returns a list of sales persons whose sales totals exceed 25.

The statement returns:

<table>
<thead>
<tr>
<th>SALES_PERSON</th>
<th>TOTAL_SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOUNOT</td>
<td>50</td>
</tr>
<tr>
<td>LEE</td>
<td>91</td>
</tr>
</tbody>
</table>
A database function is a relationship between a set of input data values and a single result value. DB2 Universal Database provides many built-in functions, including column functions and scalar functions:

- Column functions operate on a set of values in a column. For example:
  - SUM(sales) returns the sum of the values in the sales column.
  - AVG(sales) returns the sum of the values in the sales column divided by the number of values in that column.
  - MIN(sales) returns the smallest value in the sales column.
  - MAX(sales) returns the largest value in the sales column.
  - COUNT(sales) returns the number of non-null values in the sales column.

- Scalar functions operate on a single value to return another single value.
  - ABS(), HEX(), LENGTH(), YEAR(), MONTH(), DAY(), LCASE() or LOWER(), UCASE() or UPPER()
Scalar functions operate on a single value to return another single value. For example:

- \( \text{ABS}(-5) \) returns the absolute value of -5, or 5.
- \( \text{HEX}(69) \) returns the hexadecimal representation of the number 69, or 45.
- \( \text{LENGTH}('\text{Pierre}') \) returns the number of bytes in the string 'Pierre', or 6. For a GRAPHIC string, the LENGTH function returns the number of double-byte characters.
- \( \text{YEAR}('03/14/2002') \) extracts the year portion of the date-time value, or 2002.
- \( \text{MONTH}('03/14/2002') \) extracts the month portion of the date-time value, or 3.
- \( \text{DAY}('03/14/2002') \) extracts the day portion of the date-time value, or 14.
- \( \text{LCASE}('\text{SHAMAN}') \) or \( \text{LOWER}('\text{SHAMAN}') \) returns a string in which all of the characters have been converted to lowercase characters, or shaman.
- \( \text{UCASE}('\text{shaman}') \) or \( \text{UPPER}('\text{shaman}') \) returns a string in which all of the characters have been converted to uppercase characters, or SHAMAN.

- \( \text{TO_CHAR (timestamp-expression, format-string)} \) returns a character representation of a timestamp that has been formatted using a character template (synonym for VARCHAR_FORMAT).
- \( \text{TO_DATE (string-expression, format-string)} \) returns a timestamp from a character string that has been interpreted using a character template.
The INSERT statement is used to add new rows to a table or a view. Examples:

```
INSERT INTO staff
VALUES
(1212,'Cerny',20,'Sales',3,90000.00,30000.00)
```

Equivalent:

```
INSERT INTO staff
(id, name, dept, job, years, salary, comm)
VALUES
(1212,'Cerny',20,'Sales',3,90000.00,30000.00),
(1213,'Wolfrum',20,'Sales',2,90000.00,10000.00)
```

A row can be inserted into a view that is defined using a UNION ALL if the row satisfies the check constraints of exactly one of the underlying base tables. If a row satisfies the check constraints of more than one table, or no table at all, an error is returned (SQLSTATE 23513).
The UPDATE statement is used to change the data in a table or a view. You can change the value of one or more columns for each row that satisfies the conditions specified by a WHERE clause.

For example:

```
UPDATE staff
    SET dept = 51, salary = 70000
    WHERE id = 750
```

or the equivalent:

```
UPDATE staff
    SET (dept, salary) = (51, 70000)
    WHERE id = 750
```

The UPDATE statement is used to change the data in a table or a view. You can change the value of one or more columns for each row that satisfies the conditions specified by a WHERE clause.

**Warning!**

If you don't specify a WHERE clause, DB2 updates each row in the table or view!
The DELETE statement is used to delete entire rows of data from a table.

For example:

```sql
DELETE FROM staff
    WHERE id IN (1212, 1213)
```

Warning!

If you don't specify a WHERE clause, DB2 deletes all the rows in the table or view!
**CASE Expressions In SELECT**

**CASE expression example:**

```sql
SELECT empno, lastname,
    CASE
        WHEN salary < 25000 THEN 'LOW'
        WHEN salary BETWEEN 25000 AND 39999 THEN 'AVERAGE'
        ELSE 'HIGH'
    END AS salary_class,
    CASE SUBSTR(workdept,1,1)
        WHEN 'A' THEN 'ADMINISTRATION'
        WHEN 'C' THEN 'CUSTOMER SERVICE'
        WHEN 'D' THEN 'DEVELOPMENT' ELSE NULL
    END AS area_type
FROM employee
```

CASE expressions allow a user to code IF-THEN-ELSE logic instead of simple values.

- DB2 evaluates the conditions in order from top to bottom. The value returned from the CASE expression is the result of the first true condition. If no case is satisfied, the ELSE value is used. If no ELSE is coded, the value returned is NULL.
- CASE expressions can appear in various places in an SQL statement, including the SELECT clause, the WHERE clause, the GROUP BY clause, and the HAVING clause. It can also appear in IN and VALUES clauses.

The examples illustrate the use of CASE expressions in the SELECT clause. The first CASE expression returns a salary classification for the employees rather than the salary itself. The salary classification is based on the range of the salary. If the salary is less than 25000, the salary classification is 'LOW'. It is 'AVERAGE' if the salary is equal to or higher than 25000, but is lower than 40000. Salaries of 40000 and higher are considered 'HIGH' as specified via the ELSE keyword. The appropriate column of the result table is called SALARY_CLASS. This example, which has no expression between the CASE keyword and the first THEN keyword, is called a searched WHEN clause.
The second example, which has an expression between the CASE keyword and the first THEN keyword, is called a simple WHEN clause. The expression uses the SUBSTR() scalar function to determine the first character of the WORKDEPT column and the THEN clauses are evaluated for each expected value of that character. If the value is 'A', 'ADMINISTRATION' is displayed in the result, if the value is 'C', 'CUSTOMER SERVICE' is displayed, and, if the value is 'D', 'DEVELOPMENT' is displayed. Null is displayed for all other values.

**CASE expression in WHERE clause**

This is an example of CASE statement usage to protect from division by 0 errors.

It finds the employees who earn more than 8% of their income from commission, but are not fully paid on commission.

```sql
SELECT empno, lastname, salary + comm AS tot_sal
FROM employee
WHERE (CASE WHEN salary = 0 THEN NULL
            ELSE comm/salary
            END) > 0.08
```

**CASE expression in function**

This is an example of CASE in a function.

It finds the minimum salary for all employees except those with a salary equal to zero.

```sql
SELECT MIN(CASE
            WHEN salary = 0 THEN NULL
            ELSE salary END) AS min_sal
FROM employee
```

The same query could also be written:

```sql
SELECT MIN(salary)
FROM employee
WHERE NOT salary = 0
```
**Nested CASE expression**

This is an example of a nested CASE expression.

It computes a different salary increase depending on the salary and the employee number.

```
SELECT empno,
       CASE WHEN workdept LIKE 'A%' THEN
           salary + 100
       WHEN workdept LIKE 'C%' THEN
           CASE WHEN empno = '000130' THEN
               salary + 200
           ELSE salary + 50
           END
       ELSE salary
       END AS new_sal
FROM employee
WHERE salary < 50000
```
CAST Specifications

Cast example:

```
SELECT empno, comm/salary AS col2,
       CAST(comm/salary AS DEC(9,2)) AS col3
FROM employee
WHERE empno = '000140'
```

DB2 supports the CAST syntax defined by SQL92 Standard.

Casting is often used in programming languages to refer to the process of changing a value from one data type to another. Casting in SQL has the same meaning.

- In the example above, the division of COMM/SALARY in COL2 gets a default datatype depending on the data types of the columns COMM and SALARY. With CAST, the division can be presented with different data type, in this case DECIMAL(9,2). In practice this limits the number of places to the right of the decimal point as shown in COL3 to two.

CAST is also useful when a value of a particular data type is needed as the parameter of a function.

One special use of CAST is to cast a NULL value to a particular data type:

```
SELECT empno, CAST(NULL AS SMALLINT)
FROM employee
WHERE empno < '000050'
```
Automatic Summary Tables

Automatic Summary Tables provide:

- Materialized views
- Aggregate Aware Optimization
- Contains pre-computed results
- Queries can reuse Summary Tables without accessing base tables
- Improved performance

DB2 UDB on UNIX, Windows and OS/2 supports summary tables since Version 5.2.

A Summary table is a table whose definition is based on the result of a query. As such, the summary table typically contains pre-computed results based on the data existing in the table or tables on which its definition is based.

If the optimizer determines that a dynamic query will run more efficiently against a summary table than the base table, the query executes against the summary table, and we should receive the result of the query faster than access directly to the base tables. The query rewrite function of the optimizer will access a summary table if it determines that the query can be answered by using the data in the summary table instead of accessing the base table or tables.

On DB2 UDB UNIX, Windows and OS/2, you can set the optimization level. For the optimizer to consider the use of summary tables, the optimization level has to be 2 or greater.
In the example above:

- **REFRESH** - Indicates how the data in the table is maintained.
- **DEFERRED** - The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query at the time of the REFRESH TABLE statement is processed (a snapshot).
- **IMMEDIATE** - The changes made to the underlying tables as part of a DELETE, INSERT or UPDATE are cascaded to the summary table. In this case, the contents of the table, at any point-in-time, are the same as the specified subselect where to be processed.
- When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the fullselect cannot include:
  - References to a view or a summary table
  - Functions that have external action
  - Table of view references to system objects (explain tables also should not be specified)
When REFRESH IMMEDIATE is specified,

• The fullselect must be a subselect and cannot include:
  • Functions that are not deterministic
  • Scalar fullselects
  • Predicates with fullselects
  • Special registers

• A GROUP BY must be included in the subselect

• The select list must have a COUNT(*) function and no DISTINCT

• Only SUM (of not nullable columns), or COUNT column functions are allowed in the select list (without DISTINCT) and the other select list items must be included in the GROUP BY clause

• All GROUP BY items must be included in the select list

• A HAVING clause is not allowed

To exploit Summary table and ensure that we are not using 'old' data from the Summary table:
SET CURRENT REFRESH AGE.
A *basic* predicate is one that uses $=, <, >, \leq$, or any combination of these symbols. A subquery in a basic predicate must specify a single result column and cannot return more than one value.

The result of the predicate can be true, false, or unknown.

- The predicate result is unknown if:
  - the value of the expression on the left side of the predicate is null. For example, if the predicate is `WHERE BONUS + COMM > 50000`, and COMM is null in the row being read by the outer query, the expression `BONUS + COMM` evaluates to null and the value of the entire predicate is unknown for this row of the outer query. In other words, it is not known whether or not the row currently being read by the outer satisfies the predicate.
  - the subquery result set is empty or null. For example, if the predicate is `WHERE salary > (SELECT salary FROM employee WHERE empno = '000405')` and there is no employee 000405, the result of the subquery will be empty and the entire predicate is unknown for any row of the outer query.
The predicate result is false if the row in the outer query does not satisfy the operator of the subquery. The value returned by the subquery in the example on the visual is 52750.00. If the value in the current row of the outer query is 38250.00, the predicate value is false.

In all other cases, the predicate value is true.

The row from the outer query only appears in the final result if the predicate value is true for that row.

The subquery in the example determines the largest SALARY value in the EMPLOYEE table. The outer query then determines employee number, last name, and salary of the persons - or people - whose salary is equal to the one returned by the subquery.

**Note**

Several people may earn the same salary. If several people earn the maximum salary, the subquery will still return just that one value, but the outer query will return all people who earn the maximum salary.
Noncorrelated subqueries execute the subquery once at the beginning and use the result to control the rows returned by the outer query. A correlated query works differently: a row is read by the outer query, then a value from that row is passed to the subquery and is used to control which rows are returned by the subquery. Then, the predicate in the outer query is used to determine if the outer table row will appear in the result set. This process is repeated for each row of the outer table until each one has been examined and either written to the result set or omitted from it.

In the example above, the outer query reads a row in the EMPLOYEE table and gets the employee number, last name, salary, and department (WORKDEPT). The (work) department for the employee is passed to the subquery. The subquery calculates the average salary for all employees in the work department that was passed from the outer row. Then, the result of the subquery is compared to the individual salary of the employee whose row was read by the outer query. If the individual salary exceeds the result of the subquery, the outer table row is written to the result set; otherwise, it is ignored. This process is repeated for each row of the outer table.

Example correlated subquery:
```
SELECT empno, lastname, salary
FROM employee e
WHERE salary >
    (SELECT AVG(salary)
     FROM employee
     WHERE workdept = e.workdept)
```
The SUBSTR() function is used to extract a portion of a string.

COALESCE() and VALUE() perform exactly the same function. The function returns the first argument of its argument list that is not null. If all arguments are null, the function returns a null.

The DECIMAL() function returns a decimal representation of the value in its argument in the format DECIMAL(p,s), where \( p \) and \( s \) are the second and third argument. \( p \) is the precision of the decimal number and \( s \) the scale, that is, the number of decimal places.

The ROUND() function returns expression1 rounded to expression2 places right of the decimal point.

The DIGITS() function returns a character string representation of its argument.

The CHAR() function returns a character-string representation of an expression.

The LENGTH() function returns the length of its argument.

The LTRIM (Left TRIM) and RTRIM (Right TRIM) functions remove leading or trailing blanks in an expression.

The date-related functions include DATE(), YEAR(), MONTH(), DAY(), DAYS(), DAYOFYEAR(), DAYOFMONTH(), DAYOFWEEK and JULIAN_DAY. The first four functions extract a portion of their argument, namely, the full date, the year, the month, or the
The DAYS() function also works with dates, but in a different way than the other date functions.

The DAYS() function determines the difference between the date in the argument and the date of January 1, 0001 and expresses the difference in days.

The DAYOFYEAR function returns the day of the year in argument as an integer value in the range 1-366.

The WEEK() function returns the week of the year of the argument as an integer value in the range 1-54. The week starts with Sunday.

The DAYOFWEEK() function returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Sunday.

The WEEK_ISO() function returns the week of the year of the argument as an integer value in range 1-53. The week starts with Monday. Week 1 is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4.

The DAYOFWEEK_ISO() function returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Monday.

The JULIAN_DAY() function returns an integer value representing a continuous count of days and fractions since noon Universal Time on January 1, 4713 BCE (on the Julian calendar) to the date value specified in the argument.

The time and timestamp related scalar functions include TIME(), HOUR(), MINUTE(), SECOND(), MICROSECOND(), and TIMESTAMP(). The first five functions extract a portion of their argument, namely, the time, the hour, the minute, the second, or the microseconds, respectively.

The TIMESTAMP() function returns a timestamp derived from its arguments.
User-Defined Types

CREATE DISTINCT TYPE kph AS INTEGER
   WITH COMPARISONS

CREATE DISTINCT TYPE mph AS INTEGER
   WITH COMPARISONS

CREATE TABLE speed_limits
   ( route_num SMALLINT NOT NULL,
     canada_sl kph NOT NULL,
     us_sl mph NOT NULL )

Standard data types, like INTEGER and CHAR, are not always adequate for every business need. A company sometimes needs data types that specifically reflect the use of data. DB2 allows users to create their own data types. These are called *User-Defined Distinct Types*, or UDTs for short.

UDTs allow DB2 itself, rather than applications, to ensure that only valid comparisons are made. This is especially useful for cases where users are writing ad-hoc queries and may not realize that one column is not logically comparable to another.

UDTs are always based on existing built-in data types like CHAR and SMALLINT. UDTs cannot be used as source types for newer UDTs.

The WITH COMPARISONS clause tells DB2 to generate =, <>, <, <=, >, and >= operators for comparing different instances of the UDT. This clause is required if the source data type supports these comparisons and is prohibited if the source data type does not support comparisons.
Although many functions are already built into DB2 and more may emerge over time, many users want more functions. DB2 supports User-Defined Functions, UDFs for short, as a way for users to develop their own functions and have DB2 manage them.

UDFs are all registered in DB2. This ensures that they always give consistent answers.

UDFs can return a single value or a whole table. UDFs can be based on other UDFs that already exist.
Indexes are used by DB2 in much the same way as indexes in books are used by people. The index contains two major parts - a desired key value 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 IDentifier.

The index is structured in strict ascending or descending sequence on the columns in the index. The desired value can be located quickly in the index because of this ascending or descending sequence.

**Predicate**

A predicate is a single search condition.
The first and second predicates in the graphic, 'A = B' and 'C = D', are often referred to as “equal predicates”. The third search criteria, 'J > K', is a “range predicate”.

The part of DB2 that executes predicates is called the Optimizer.

There are three main parts in the Optimizer: the Index Manager, Data Management Services, and Relational Data Services. Each is responsible for different types of predicates.

Predicates that can be evaluated through the use of an index are evaluated by the Index Manager. These predicates include range-delimiting and index-sargable predicates.

- A range-delimiting predicate allows the index manager to narrow the search of index pages. If DEPTNO was the leftmost column of an index and the query contains the predicate 'DEPTNO = 10', this predicate is considered range-delimiting. DB2 can use this index to limit the number of index pages it examines so that only pages with a DEPTNO value of 10 are examined. This will most likely save considerable I/O with respect to reading the entire index or table.

- An index-sargable predicate does not limit the range of pages examined, but it can help find the data faster. If SALARY was a column of an index, but was not one of the leading columns and the values in the leading columns was unknown, the predicate 'SALARY > 50000' would be considered index-sargable. The predicate could not be used to reduce the number of index pages read, but it could still be used to reduce the number of data rows which the query would have to read.
Predicates that cannot be evaluated by the index manager, but do not require the more complex algorithms of relational data services, are called data-sargable predicates. An example of a data-sargable predicate is DEPTNO = 'D10' when DEPTNO is not in an index. These predicates do not reduce the number of index rows read nor the number of data rows read, but they do reduce the number of data rows returned to the result set.

Predicates that require more complex evaluation require relational data services and are called residual (remaining) predicates. In DB2 for OS/390, they are also referred to as Stage 2 or non-sargable predicates. Correlated subqueries, quantified subqueries, and predicates that use LONG VARCHAR columns are all examples of residual predicates. The term residual also infers that these predicates are evaluated after the other predicates have been applied. The presence of a residual predicate is not in itself cause for concern. However, limiting the number of residual predicates can enhance performance.

Although there are always exceptional cases, the best performance is obtained from range-delimiting predicates and the worst from residual predicates.

The user or programmer does not need to analyze each predicate manually in order to predict performance. The EXPLAIN facility provided with DB2 will generate an explanation of the access paths that DB2 will use to resolve each part of the SQL statement.

DB2 ignores the sequence in which you write the predicates, except for the join sequence when more than two tables are involved. It executes predicates in the sequence that will enable DB2 to do the work most efficiently. For example, DB2 will perform local predicates (those that operate on only one table) before it does joins so that the join is concatenating the fewest possible rows. By the same token, DB2 will defer performing residual predicates to the last possible moment to ensure that the residual predicates are working with the smallest amount of data possible.

The EXPLAIN facility gives the user information on the sequence and cost of each stage of the processing. The EXPLAIN facility is documented in the product manuals. It is also covered in the programming classes for each platform.
The Optimizer

The Optimizer:
- Chooses the access path for all SQL statements
- Calculates the cost of each possible access path for a given query and uses the path that is the cheapest

Each code version and operating system for DB2 has its own distinct Optimizer.

Each Optimizer is capable of rewriting some queries to improve performance.

Poorly written SQL can dramatically hurt the performance of a given query.

The Optimizer is used to calculate access paths for SQL regardless of its source:

- SPUFI, QMF, the Command Center, the command line, or an application.

The Optimizer will not use an index if the amount of data is very small: in this case, the Optimizer may calculate that it is cheaper to read all the rows with a table scan and discard rows that do not satisfy the predicates in the query.

UNIX, Windows and OS/2 versions of DB2 UDB all share the same code base so they also share the same Optimizer. Mainframe versions of the Optimizer have also changed with virtually every release.

The Optimizer can rewrite queries in a variety of different ways. For example, the Optimizer can:

- Rewrite subqueries as joins
- Merge views
- Eliminate redundant DISTINCTs
- Convert correlated subqueries to non correlated subqueries
- Convert multiple OR predicates into an IN predicate
- Convert a 'NOT <' predicate to a '>=' predicate

Some queries provide poor performance and cannot be rewritten. For example, a '<>'-predicate cannot be turned into a positive predicate.
Recursive SQL is used to work on tables that contain component breakdowns where each component is broken down into subcomponents and each subcomponent is broken down again into sub-subcomponents, etc. Problems involving these kinds of tables are often called "Bill of Materials" problems. A table that represented the parts in a computer would be an example of a Bill of Materials: the major components, the monitor, system unit, and printer, all contain subassemblies like the hard drive, the mother board, and the print head, each of which is composed of other subassemblies, etc., etc.

Recursive SQL involves defining a common table expression that references itself. The common table expression consists of two distinct components, an initialization select and an iterative select. The initialization select is the first SELECT in the table expression and the iterative select is the second SELECT in the table expression. The iterative select is combined with the initialization select by means of UNION ALL.

The recursive common table expression in the example is named `rpl`. It is defined within the parentheses.
# Column Functions

By themselves, column functions work on the complete set of matching rows. One can use a GROUP BY expression to limit them to a subset of matching rows. One can also use them in an OLAP function to treat individual rows differently.

**Warning!**

Be very careful when using either a column function, or the DISTINCT clause, in a join. If the join is incorrectly coded, and does some form of Cartesian Product, the column function may get rid of the all the extra (wrong) rows so that it becomes very hard to confirm that the answer is incorrect. Likewise, be appropriately suspicious whenever you see that someone (else) has used a DISTINCT statement in a join. Sometimes, users add the DISTINCT clause to get rid of duplicate rows that they didn't anticipate and don't understand.

- **AVG**—gets the average (mean) value of a set of non-null rows.
- **CORRELATION**—returns the coefficient of correlation of a set of number pairs.
- **COUNT**—gets the number of values in a set of rows.

---

<table>
<thead>
<tr>
<th>AVG</th>
<th>CORRELATION</th>
<th>COUNT</th>
<th>COUNT_BIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVARIANCE</td>
<td>GROUPING</td>
<td>MAX</td>
<td>MIN</td>
</tr>
<tr>
<td>REGR_AVGX</td>
<td>REGR_AVGY</td>
<td>REGR_COUNT</td>
<td>REGR_INTERCEPT</td>
</tr>
<tr>
<td>REGR_R2</td>
<td>REGR_SLOPE</td>
<td>REGR_SXX</td>
<td>REGR_SXY</td>
</tr>
<tr>
<td>REGR_SYY</td>
<td>STDDEV</td>
<td>SUM</td>
<td>VAR or VARIANCE</td>
</tr>
</tbody>
</table>
COUNT_BIG—gets the number of rows or distinct values in a set of rows. Use this function if the result is too large for the COUNT function. Example:

```sql
SELECT COUNT_BIG(*) AS C1,
      COUNT_BIG(DEPT) AS C2,
      COUNT_BIG(DISTINCT DEPT) AS C3,
      COUNT_BIG(DISTINCT DEPT/10) AS C4,
      COUNT_BIG(DISTINCT DEPT)/10 AS C5
FROM STAFF;
```

ANSWER

```
C1  C2  C3  C4  C5
--- --- --- --- ---
35  35  8  7  0
```

COVARIANCE—returns the covariance of a set of number pairs.

GROUPING—used in CUBE, ROLLUP, and GROUPING SETS statements to identify what rows come from which particular GROUPING SET. Example:

```sql
SELECT DEPT,
      AVG(SALARY) AS SALARY,
      GROUPING(DEPT) AS DF
FROM STAFF
GROUP BY ROLLUP(DEPT)
ORDER BY DEPT;
```

ANSWER

```
DEPT  SALARY  DF
------ -------- --
10  20865.86  0
15  15482.33  0
20  16071.52  0
38  15457.11  0
42  14592.26  0
51  17218.16  0
66  17215.24  0
84  16536.75  0
-  16675.64  1
```

MAX—gets the maximum value of a set of rows.

MIN—gets the minimum value of a set of rows.
- **REGRESSION**—various regression functions support the fitting of an ordinary-least-squares regression line of the form \( y = a \times x + b \) to a set of number pairs.
  - `REGR_AVGX` returns a quantity that than can be used to compute the validity of the regression model. The output is of type float.
  - `REGR_AVGY` (see `REGR_AVGX`).
  - `REGR_COUNT` returns the number of matching non-null pairs. The output is integer.
  - `REGR_INTERCEPT` returns the \( y \)-intercept of the regression line.
  - `REGR_R2` returns the coefficient of determination for the regression.
  - `REGR_SLOPE` returns the slope of the line.
  - `REGR_SXX` (see `REGR_AVGX`).
  - `REGR_SXY` (see `REGR_AVGX`).
  - `REGR_SYY` (see `REGR_AVGX`).

**Example:**

```sql
SELECT
  DEC(REGR_SLOPE(BONUS, SALARY), 7, 5) AS R_SLOPE,
  DEC(REGR_INTERCEPT(BONUS, SALARY), 7, 3) AS R_ICPT,
  INT(REGR_COUNT(BONUS, SALARY)) AS R_COUNT,
  INT(REGR_AVGX(BONUS, SALARY)) AS R_AVGX,
  INT(REGR_AVGY(BONUS, SALARY)) AS R_AVGY,
  INT(REGR_SXX(BONUS, SALARY)) AS R_SXX,
  INT(REGR_SXY(BONUS, SALARY)) AS R_SXY,
  INT(REGR_SYY(BONUS, SALARY)) AS R_SYY
FROM EMPLOYEE
WHERE WORKDEPT = 'A00';
```

**ANSWERS**

```
0.01710
100.871
3
42833
833
296291666
5066666
86666
```

- **STDDEV**—gets the standard deviation of a set of numeric values.
- **SUM**—gets the sum of a set of numeric values.
- **VAR or VARIANCE**—gets the variance of a set of numeric values.
The OLAP (Online Analytical Processing) functions enable one to sequence and rank query rows. They are especially useful when the calling program is very simple.

- The RANK and DENSE_RANK functions enable one to rank the rows returned by a query.
  - The RANK function returns the number of proceeding rows, plus one.
  - The DENSE_RANK function returns the number of proceeding distinct values, plus one.

```sql
SELECT ID, YEARS, SALARY,
  RANK() OVER(ORDER BY YEARS) AS RANK#,
  DENSE_RANK() OVER(ORDER BY YEARS) AS DENSE#,
  ROW_NUMBER() OVER(ORDER BY YEARS) AS ROW#
FROM STAFF
WHERE ID < 100
AND YEARS IS NOT NULL
ORDER BY YEARS;
```
The PARTITION phrase lets one rank the data by subsets of the rows returned.

- The ROW_NUMBER function lets one number the rows being returned.
- The RANGE phrase limits the aggregation result to a range of numeric values - defined relative to the value of the current row being processed.
You should now be able to:

- Create objects
- Use a variety of table joins
- Craft a subquery with basic predicate
- Use scalar functions
- Explain the purpose of user-defined types and user-defined functions
- Consider performance issues
- Explain the purpose of the optimizer
- Identify the affect and scope of a COMMIT or ROLLBACK statement
Exercises
5-62 Intermediate SQL

Exercise 1

This exercise is designed to solidify your knowledge base on DB2 SQL statements.

1.1 In a query type of DML statement, what are the two REQUIRED keywords?

1.2 What is the multi-character wildcard symbol in DB2?

1.3 What is the single-character wildcard symbol in DB2?

1.4 What are the two most important reasons for using the WHERE clause in an SQL query statement?

1.5 If unspecified, what is the default direction of the sort in an ORDER BY clause?

1.6 In the following statement, what does the ON keyword provide?

```sql
SELECT deptnum, deptname, id, manager_id
FROM org INNER JOIN staff
ON manager = id
ORDER BY deptnum
```

1.7 What is an INNER join?

1.8 What is a LEFT OUTER join?

1.9 What is a RIGHT OUTER join?

1.10 What is a FULL OUTER join?

1.11 What are the three set operators you can use to combine tables together?
1.12 When the SELECT list contains an aggregate, like MIN or MAX, and several other columns, what clause must be used to collect like data together?

1.13 What does the HAVING clause provide to the GROUP BY clause?

1.14 What function can you use in an SQL query to determine the number of rows in a table?

1.15 What functions could you use to perform a case-insensitive search of data?

1.16 What DML statement would you use to put new data rows into a table? What are the three required keywords on this type of statement?

1.17 What is the DML statement that is used to changed existing data in a table? What is the syntax of that statement?

1.18 How would you remove data from a table? What is the danger with this statement?
Exercise 2

This exercise gives you some hands-on experience executing various SQL statements.

2.1 You will be using the Command Center to input and execute your SQL statements, but you need to set the termination character first.

   a. **On your workstation, select** Start > Programs > IBM DB2 > Command Center (**in v8, use:** Start > Programs > IBM DB2 > Command Line Tools > Command Center).

   b. **Click on the Tools menu at the top of the panel, and select** Tools Settings.

   c. **If there is no check in the** Use statement termination character **check box, put a check mark there. This will use the semicolon as the termination character. Close the Tools Settings panel by clicking on the X in the upper right corner of the panel.**

2.2 Using the **Command Center’s Interactive** panel to run your SQL statements. Connect to the sample database by typing your statement in the Command pane, then execute it by clicking on the Execute icon, which is the left-most icon on the tool bar. You should see confirmation of your database connect command in the response pane under the Command pane.

   ```sql
   CONNECT TO sample;
   ```

2.3 Using SQL and the sample database, find all the people who are members of department D21. Query the employee table for this.

2.4 You want a list of all employees, sorted by their hire dates. You need employee number, first name, last name, department, hire date, job, and sex, all from the employee table.

2.5 Now find just one of the highest paid employees (there may be more than one).

2.6 Did you account for all of the employee’s income in the last question? How would you?

2.7 Now you need the employee number, last name, department, phone number, job name, the region, and the sales made for all employees, regardless if they are sales people or not.
2.8 Find all of the projects involving department D21. You need the project number and project name, but you will also need to reference the department table.

2.9 You need the name and phone number of the salesman with the highest sales in the Quebec region.

2.10 What is the total project staffing for all of the projects that are assigned to department D21?

2.11 How many rows are there in the project table?

2.12 How many different departments are assigned to projects?

2.13 Extract the month of each sale made from the sales table. Include the sales person’s name, and number of sales.

2.14 Sales person Lee has made 4 new sales, and they must be entered into the sales table. Use the INSERT DML statement to enter this data. The information to be entered is:

<table>
<thead>
<tr>
<th>Sales date</th>
<th>Sales person</th>
<th>Region</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002-03-22</td>
<td>LEE</td>
<td>Manitoba</td>
<td>1</td>
</tr>
<tr>
<td>2001-11-20</td>
<td>LEE</td>
<td>Quebec</td>
<td>2</td>
</tr>
<tr>
<td>2002-01-09</td>
<td>LEE</td>
<td>Quebec</td>
<td>1</td>
</tr>
</tbody>
</table>

2.15 You made an error in the number of sales inserted for Lee on 2002-03-22. The correct number of sales on that date for Lee is 3. Update that row in the sales table to reflect the change.

2.16 Another error in reporting sales requires that the sale for LEE for the date of 2001-01-09 be deleted from the sales table. Do this, but make sure you delete only that row!

Tip

If you have time left in the this exercise, it is suggested that you go back through the pages of this module and try some of the SQL statements explained throughout.
This exercise is designed to solidify your knowledge base on DB2 SQL statements.

1.1 In a query type of DML statement, what are the two REQUIRED keywords?

You must use **SELECT**, to indicate the column select list, and **FROM**, to indicated which table to query.

1.2 What is the multi-character wildcard symbol in DB2?

To wildcard multiple characters, use the `%` symbol.

1.3 What is the single-character wildcard symbol in DB2?

To wildcard a single character, use the `_` symbol.

1.4 What are the two most important reasons for using the WHERE clause in an SQL query statement?

Use the WHERE clause to limit the number of rows returned in the query. You can also use the WHERE clause to provide a join condition for joining multiple tables in a query (this helps prevent the results being returned as a Cartesian product).

1.5 If unspecified, what is the default direction of the sort in an ORDER BY clause?

An ORDER BY clause causes an ascending sort, unless otherwise specified.

1.6 In the following statement, what does the ON keyword provide?

```
SELECT deptnum, deptname, id, manager_id
FROM org INNER JOIN staff
    ON manager = id
ORDER BY deptnum
```

The **ON** keyword is providing the join condition between the **org** and the **staff** tables. This is used when the JOIN keyword is used to signify a table join.
1.7 What is an INNER join?

An INNER join returns only the rows of a cross-product specified by the join condition.

1.8 What is a LEFT OUTER join?

A LEFT OUTER join returns the rows that an INNER join would return, plus all the rows from the left table.

1.9 What is a RIGHT OUTER join?

A RIGHT OUTER join returns the rows that an INNER join would return, plus all the rows from the right table.

1.10 What is a FULL OUTER join?

A FULL OUTER join returns the rows that an INNER join would return, plus all the rows from the both the left and right tables.

1.11 What are the three set operators you can use to combine tables together?

UNION — result set combines two or more result sets
EXCEPT — result set includes all rows from the first query, but none from any subsequent query
INTERSECT — result set includes only the rows returned by all queries

1.12 When the SELECT list contains an aggregate, like MIN or MAX, and several other columns, what clause must be used to collect like data together?

The GROUP BY clause is used to group like data together. Example:

```
SELECT sales_date,
       MAX(sales) AS max_sales
FROM sales
GROUP BY sales_date
```

1.13 What does the HAVING clause provide to the GROUP BY clause?

The HAVING clause is used as a conditional to the GROUP BY, in a similar manner that the WHERE clause is a conditional to the SELECT. Example:
SELECT sales_person, SUM(sales) AS total_sales
FROM sales
GROUP BY sales_person
HAVING SUM(sales) > 25

1.14 What function can you use in an SQL query to determine the number of rows in a table?

You can use the COUNT(*) function to retrieve the number of rows in a table.

1.15 What functions could you use to perform a case-insensitive search of data?

You can use either the LCASE or UCASE functions to test for case in the predicate of a WHERE clause. Example:

SELECT * FROM staff
WHERE UCASE(name) = 'SMITH'

LCASE has a counterpart, LOWER, and UCASE has a counterpart, UPPER.

1.16 What DML statement would you use to put new data rows into a table? What are the three required keywords on this type of statement?

The INSERT statement is used to insert new data rows into a table. You must use the INSERT INTO keywords to indicate the table being changed, and the VALUES keyword to indicate the list of values being inserted into each row.

1.17 What is the DML statement that is used to changed existing data in a table? What is the syntax of that statement?

The UPDATE statement is used to changed existing data. The syntax of an UPDATE statement is:

UPDATE tablename
SET columnname = value

or

UPDATE tablename
SET (columnname,columnname) = (value,value)

Note: Without the WHERE clause, as shown here, all rows will be updated. If you want just certain rows to be updated, you must include a WHERE clause.
1.18 How would you remove data from a table? What is the danger with this statement?

You would use the DELETE statement to remove data from a table. The danger is that you will remove all data from the table, unless you use the WHERE clause to qualify the rows you want removed. Example to remove some rows:

DELETE FROM orders
  WHERE order_num BETWEEN 100 and 1000

To remove all rows from a table:

DELETE FROM orders

Note: DB2 does not currently have a TRUNCATE TABLE statement, though an alternative would be to LOAD with the REPLACE option, using a zero byte data file as input.
This exercise gives you some hands-on experience executing various SQL statements.

2.1 You will be using the Command Center to input and execute your SQL statements, but you need to set the termination character first.
   
a. **On your workstation, select** Start > Programs > IBM DB2 > Command Center *(in v8, use: Start > Programs > IBM DB2 > Command Line Tools > Command Center).*

b. **Click on the Tools menu at the top of the panel, and select** Tools Settings.

c. **If there is no check in the** Use statement termination character check box, **put a check mark there. This will use the semicolon as the termination character. Close the Tools Settings panel by clicking on the X in the upper right corner of the panel.**

2.2 Using the **Command Center’s Interactive** panel to run your SQL statements. Connect to the **sample** database by typing your statement in the **Command** pane, then execute it by clicking on the **Execute** icon, which is the left-most icon on the tool bar. You should see confirmation of your database connect command in the response pane under the **Command** pane.

   ```sql
   CONNECT TO sample;
   ```

2.3 Using SQL and the **sample** database, find all the people who are members of department **D21**. Query the **employee** table for this.

   ```sql
   SELECT *
   FROM employee
   WHERE workdept = 'D21';
   ```

2.4 You want a list of all employees, sorted by their hire dates. You need employee number, first name, last name, department, hire date, job, and sex, all from the **employee** table.

   ```sql
   SELECT empno, firstnme, lastname, workdept, hiredate, job, sex
   FROM employee
   ORDER BY hiredate;
   ```
2.5 Now find just one of the highest paid employees (there may be more than one).

```
SELECT empno, lastname, workdept, salary
FROM employee
ORDER BY salary DESC
FETCH FIRST 1 ROWS ONLY;
```

2.6 Did you account for all of the employee’s income in the last question? How would you?

No.

```
SELECT empno, lastname, workdept, 
   (salary + bonus + comm) AS total_income
FROM employee
ORDER BY (salary + bonus + comm) DESC
FETCH FIRST 1 ROWS ONLY;
```

2.7 Now you need the employee number, last name, department, phone number, job name, the region, and the sales made for all employees, regardless if they are sales people or not.

```
SELECT empno, lastname, workdept, phoneno, 
   job, sales_date, region, sales
FROM employee LEFT OUTER JOIN sales
ON lastname = sales_person
ORDER BY lastname;
```

This will produce the same result set:

```
SELECT empno, lastname, workdept, phoneno, 
   job, sales_date, region, sales
FROM sales RIGHT OUTER JOIN employee
ON lastname = sales_person
ORDER BY lastname;
```

2.8 Find all of the projects involving department D21. You need the project number and project name, but you will also need to reference the department table.

```
SELECT projno, projname
FROM project, department
WHERE project.deptno = department.deptno
AND department.deptno = 'D21';
```

or
SELECT projno, projname
FROM project JOIN department
ON project.deptno = department.deptno
WHERE department.deptno = 'D21';

2.9 You need the name and phone number of the salesman with the highest sales in the Quebec region.

SELECT sales_person, phoneno, SUM(sales)
FROM sales, employee
WHERE region = 'Quebec'
AND sales.sales_person = employee.lastname
GROUP BY sales_person, phoneno
ORDER BY SUM(sales) DESC
FETCH FIRST 1 ROWS ONLY;

2.10 What is the total project staffing for all of the projects that are assigned to department D21?

SELECT projno, SUM(prstaff) AS staffing
FROM project
WHERE deptno = 'D21'
GROUP BY projno;

2.11 How many rows are there in the project table?

SELECT COUNT(*) AS number_of_projects
FROM project;

2.12 How many different departments are assigned to projects?

SELECT COUNT(DISTINCT deptno) AS number_of_dept
FROM project;

2.13 Extract the month of each sale made from the sales table. Include the sales person’s name, and number of sales.

SELECT sales_person, sales,
    MONTH(sales_date) AS month_of_sales
FROM sales
ORDER BY sales_person;
2.14 Sales person Lee has made 4 new sales, and they must be entered into the sales table. Use the INSERT DML statement to enter this data. The information to be entered is:

<table>
<thead>
<tr>
<th>Sales date</th>
<th>Sales person</th>
<th>Region</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002-03-22</td>
<td>LEE</td>
<td>Manitoba</td>
<td>1</td>
</tr>
<tr>
<td>2001-11-20</td>
<td>LEE</td>
<td>Quebec</td>
<td>2</td>
</tr>
<tr>
<td>2001-01-09</td>
<td>LEE</td>
<td>Quebec</td>
<td>1</td>
</tr>
</tbody>
</table>

```
INSERT INTO sales
VALUES
('2002-03-22', 'LEE', 'Manitoba', 1);

INSERT INTO sales
VALUES
('2001-11-20', 'LEE', 'Quebec', 2),
('2001-01-09', 'LEE', 'Quebec', 1);
```

2.15 You made an error in the number of sales inserted for Lee on 2002-03-22. The correct number of sales on that date for Lee is 3. Update that row in the sales table to reflect the change.

```
UPDATE sales
SET sales = 3
WHERE sales_person = 'LEE'
AND sales_date = '2002-03-22';
```

2.16 Another error in reporting sales requires that the sale for LEE for the date of 2001-01-09 be deleted from the sales table. Do this, but make sure you delete only that row!

```
DELETE from sales
WHERE sales_date = '2002-02-09'
AND sales_person = 'LEE';
```

Tip
If you have time left in the this exercise, it is suggested that you go back through the pages of this module and try some of the SQL statements explained throughout.
This lab gives you some practice with referential integrity.

3.1 In the sample database, alter your players and team tables to include referential integrity, using referential constraints. Consider the team table as the parent table and the players table as the dependent table.

```
ALTER TABLE team
  ADD PRIMARY KEY (teamname);

ALTER TABLE players
  ADD CONSTRAINT (on_team)
  FOREIGN KEY (teamname)
  REFERENCES team
  ON DELETE RESTRICT;
```

3.2 Where you forced to perform the alters in a specific order? Why or why not?

No. Usually you would establish the primary key first, but because there is already a unique index on what would be the primary key column in the team table, the foreign key in the players table can reference the teamname column in the team table without declaring it the primary key.

3.3 Why is it considered very inefficient to code a primary-foreign key relationship like you did in the team and players tables?

The columns used for the relationship in this case are char(15) data type. This means that every combination of every character position in both columns are parsed and sorted to be used as the connector between the two tables. Very inefficient!

3.4 What would have been a better way to code the above relationship?

A better method would have been to codify the team name, such that it is either an integer, or a short (char(2)) character column. This would minimize the parsing and sorting needed to make the table connection.
Module 6

Database Objects
## Objectives

At the end of this module, you will be able to:

- Demonstrate usage of DB2 data types
- Identify when referential constraints are used
- Identify methods of data validation
- Identify characteristics of a table, view or index
- Explain the concept of extenders
Database objects may be created and/or modified through the use of DB2 command line commands, or through Wizards in the Administration Client GUI tools.

Some of the objects are shown above.

Although DB2 data types are not considered objects, you will learn more about them in this module.
DB2 provides a rich and flexible assortment of data types. DB2 comes with basic built-in data types such as INTEGER, CHAR and DATE and also facilities to create user-defined data types, which give the programmer the ability to create complex, non-traditional data types suited to today's complex programming environments. Choosing which type to use depends on the type and range of information that will be stored in the column.
There are three categories of numeric data types. These types vary in the range and precision of numeric data they can store.

- **Integer**
  SMALLINT, INTEGER and BIGINT are used to store numbers which are integers. For example, an inventory count could be defined as INTEGER. SMALLINT can store integers from -32768 to 32767 in two bytes. INTEGER can store integers from -2,147,483,648 to 2,147,483,647 in four bytes. BIGINT can store integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 in eight bytes.

- **Decimal**
  DECIMAL is used to store numbers which fractional parts. To define this data type, you must specify a precision\(p\), total number of digits, and scale\(s\), the number of digits to the right of the decimal. For example, currency could be represented by DECIMAL(10,2). A column defined by DECIMAL(10,2) would hold values up to 10 million dollars. The amount of storage required in the database depends on the precision and is calculated by the formula \(p/2 + 1\). So, DECIMAL(10,2) would require 10/2 + 1 or 6 bytes.
Floating Point

REAL and DOUBLE are used to store approximations of numbers. For example, very small or very large scientific measurements could be defined as REAL. REAL can be defined with a length between 1 and 24 and requires 4 bytes of storage. DOUBLE can be defined with a length of between 25 and 53 and requires 8 bytes of storage. FLOAT can be used as a synonym for REAL or DOUBLE.
DB2 provides several data types for storing character data or strings. Which data type you use depends on the size of the string you are going to store and what data will be in the string.

The following data types are used to store single-byte character strings:

- **CHAR**
  
  CHAR or CHARACTER is used to store fixed-length character strings up to 254 bytes. For example, a part identifier may be defined with a specific length of eight characters and therefore stored in the database as a column of CHAR(8).

- **VARCHAR**
  
  VARCHAR is used to store variable-length character strings. For example, a part description may have a different length depending on the part and may be defined as a VARCHAR(100). The maximum length of a VARCHAR column is 32,672 bytes. In the database, VARCHAR data only takes as much space as required.

The following data types are used to store double-byte character strings:

- **GRAPHIC**
  
  GRAPHIC is used to store fixed length double-byte character strings. The maximum length of a GRAPHIC column is 127 characters.
VARGRAPHIC

VARGRAPHIC is used to store variable length double-byte character strings. The maximum length of a VARGRAPHIC column is 16,336 characters.

DB2 also provides data types to store very long strings of data. All long string data types have similar characteristics. First, the data is not stored physically with the row data in the database, which means that additional processing is required to access this data. Long data types can be defined up to 2 gigabytes in length. However, only the space required is actually used. Long data types are:

- LONG VARCHAR
- CLOB or character large object
- LONG VARGRAPHIC
- DBCLOB or double byte character large object
- BLOB or binary large objects
The values of these data types are stored in the database in an internal format, however, you manipulate them as strings in programs. When any of these data types is retrieved it is represented as a character string. When updating these data types, you must enclose the value in quotation marks.

DB2 provides built-in functions to manipulate date-time values. For example, you can determine the day of the week of a date value using the DAYOFWEEK or DAYNAME functions. You can use the DAYS function to calculate how many days between two dates. DB2 also provides special registers that can be used to generate the current date, time or timestamp based on the time of day clock. For example, CURRENT DATE returns a string representing the current date on the system.

The format of the date and time values depends on the country code of the database, which is specified when the database is created. There are several formats available: ISO, USA, EUR and JIS. For example, if your database is using the USA format, the format of date values would be "MM/DD/YYYY". You can change the format by using the DATETIME option of the BIND command when creating your application.

There is a single format for the TIMESTAMP data type. The string representation is YYYY-MM-DD-HH.MM.SS.NNNNNN.
**DATALINK**

DB2 provides the DATALINK data type to manage external files. A DATALINK column allows you to store a reference to a file outside the database. These files can reside in a file system on the same server or on a remote server. DB2 provides facilities that allow applications to access these files securely.

To insert values into a DATALINK column, you must use the built-in function DLVALUE. DLVALUE requires several parameters which tell DB2 the file name and where the file is stored. To retrieve data from the DATALINK column, DB2 provides several functions depending on what information is required.
DB2 allows you to define data types that suit your application. There are three user-defined data types:

- **User-defined distinct types**
  You can define a new data type based on a built-in type but ensures that only values of the same type are compared. For example, you can define a Canadian dollar type (candol) and a USA dollar type (usadol) both based on DECIMAL(10,2). Both types are based on the same built-in type, but they cannot be compared unless a conversion function is applied.

These are CREATE TYPE statements to create the candol and usadol UDTs:

```sql
CREATE DISTINCT TYPE candol AS DECIMAL(10,2) WITH COMPARISONS
CREATE DISTINCT TYPE usadol AS DECIMAL(10,2) WITH COMPARISONS
```

DB2 automatically generates functions to perform casting between the base type and the distinct type, and comparison operators for comparing instances of the distinct type. The following statements show how to create a table with a column of candol type and insert data into the table using the candol casting function.

```sql
CREATE TABLE items (ITEMID CHAR(5), PRICE candol)
INSERT INTO items VALUES('ABC11', candol(30.50))
```
- User-defined structured types
  This support allows you to create a type that consists of several columns of built-in types. You can then use this structured type when creating a table. For example, you can create a structured type named address that contains data for street number, street name, city, etc. Then you can use this type when defining other tables such as employees or suppliers since the same data is required for both. Also, structured types can have subtypes in a hierarchical structure. This allows objects that belong to a hierarchy to be stored in the database.

- User-defined reference types
  When using structured types, you can define references to rows in another table using reference types. These references appear similar to referential constraints, however, they do not enforce relationships between the tables. References in tables allow you to specify queries in a different way.

User-defined structured and reference types are an advanced topic and this information serves only as an introduction to these types.
DB2 Extenders provide support for complex, nontraditional data types. They are packaged separately from the DB2 server code and must be installed on the server and into each database that will use the data type.

For example, the DB2 Image Extender includes:
- The DB2IMAGE UDT
- UDFs to insert/retrieve from a db2image column
- APIs to search based on characteristics of images

DB2 Extenders are implemented using the features of user-defined types and user-defined functions. Each extender comes with one or more UDT, UDFs for operating on the UDT and specific application programming interfaces (APIs), and perhaps other tools.

Before using these data types, you must install the extender support into the database. The installation process for each extender defines the required UDTs and UDFs in the database. Then you can use the UDTs when defining a table and the UDFs when working with the data.
All data is stored in tables in the database. A table consists of one or more columns of various data types. The data is stored in rows or records.

Tables are defined using the CREATE TABLE SQL statement. DB2 provides a GUI tool for creating tables based on information you specify. The tool also generates the CREATE TABLE SQL statement that can be used in a script or application program at a later time.

A database has a set of tables, called the System Catalog Tables, which hold information about all the objects in the database. The catalog table `syscat.tables` contains a row for each table defined in the database. `syscat.columns` contains a row for each column of each table in the database. You can look at the catalog tables just like any other tables in the database using SELECT statements.
Creating a Table

The CREATE TABLE SQL statement is used to define a table in the database.

An example:

```sql
CREATE TABLE books (
  bookid INTEGER,
  bookname VARCHAR(100),
  isbn CHAR(10))
```

An example of creating a table that is like another table or view:

```sql
CREATE TABLE mybooks LIKE books
```

This statement creates a table with the same columns as the original table or view.

There are many options available for the CREATE TABLE statement (they'll be presented in the following pages as new concepts are introduced). The details of the CREATE TABLE SQL statement can be found in the *DB2 SQL Reference*.

Once the table is created, there are several ways to populate it with data. The INSERT statement allows you to insert a row or several rows of data into the table. DB2 also provides utilities to insert large amounts of data from a file. The IMPORT utility inserts rows using INSERT statements. It is designed for loading small amounts of data into the database. The LOAD utility inserts rows directly onto data pages in the database and therefore is much faster than the IMPORT utility. It is intended for loading large volumes of data.
Tables are stored in the database in table spaces.

- Table spaces have physical space allocated to them
- You must create the table space before creating the table

In this CREATE TABLE statement example, the table is placed in the `bookinfo` table space.

```
CREATE TABLE books (
    bookid INTEGER,
    bookname VARCHAR(100),
    isbn CHAR(10))
IN bookinfo
```

When you create a table, you can let DB2 place the table in a default table space, or you can specify in which table space the table should reside.

Although we will not discuss table spaces here in detail, it is important to understand that defining table spaces appropriately has an effect on the performance and maintainability of the database.
**Database Objects**

**Database Objects**

**ALTERING A TABLE**

You can change certain characteristics of a table using the `ALTER TABLE` statement.

Some of the characteristics that can be changed are:

- Add one or more columns
- Add or drop a primary key
- Add or drop one or more unique or referential constraints
- Add or drop one or more check constraints
- Change the length of a VARCHAR column

For example, to add a column `booktype` to the `books` table:

```sql
ALTER TABLE books ADD booktype CHAR(1)
```

**DROP TABLE**

The `DROP TABLE` statement removes a table from the database. The data and the table definition are deleted. If there are indexes or constraints defined on the table, they are also dropped.

This is the `DROP TABLE` statement to delete the `books` table from the database.

```sql
DROP TABLE books
```

*Important!*

When the table is dropped, all related objects (indexes, constraints, triggers, summary tables referencing the table, referential integrity constraints) are also dropped.
RENAME TABLE

If you need to rework a table’s structure, you can use the RENAME TABLE command to rename it, then create a new table with the original name. After copying the data from the renamed table to the new table, and verifying it, you can drop the renamed table.
The columns of a table are specified in the CREATE TABLE statement by a column name and data type. The columns can have additional options specified that restrict the data in the column.

By default, a column allows NULL values. If you do not want to allow NULL values, you can specify the NOT NULL keyword for the column. You can also specify a default value using the WITH DEFAULT keyword and a default value.

DB2 can generate a unique row identifier by using the GENERATED ALWAYS AS IDENTITY clause. You can also use the GENERATED ALWAYS option to have DB2 calculate the value of a column automatically.

The columns of a table are specified in the CREATE TABLE statement by a column name and data type. The columns can have additional options specified that restrict the data in the column.

By default, a column allows NULL values. If you do not want to allow NULL values, you can specify the NOT NULL keyword for the column. You can also specify a default value using the WITH DEFAULT keyword and a default value. The following CREATE TABLE statement creates a table called **books** in which the **bookid** column does not allow NULL values, and the default value for **bookname** is 'TBD'.

```sql
CREATE TABLE books (
    bookid INTEGER NOT NULL,
    bookname VARCHAR(100) WITH DEFAULT 'TBD',
    isbn CHAR(10))
```

In the **books** table, the **bookid** is a unique number assigned to each book. Rather than having the application generate the identifier, we can specify that DB2 is to generate a **bookid** using the GENERATED ALWAYS AS IDENTITY clause.
CREATE TABLE books (  
    bookid INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY  
        (START WITH 1, INCREMENT BY 1),  
    bookname VARCHAR(100) WITH DEFAULT 'TBD',  
    isbn CHAR(10))

GENERATED ALWAYS AS IDENTITY causes a bookid to be generated. The first value generated will be 1 and succeeding values will be generated by incrementing by 1.

You can also use the GENERATED ALWAYS option to have DB2 calculate the value of a column automatically. The following example defines a table called authors, with counts for fiction and nonfiction books. The totalbooks column will be calculated by adding the two counts.

CREATE TABLE authors (  
    authorid INTEGER NOT NULL PRIMARY KEY,  
    lname VARCHAR(100),  
    fname VARCHAR(100),  
    fictionbooks INTEGER,  
    nonfictionbooks INTEGER,  
    totalbooks INTEGER GENERATED ALWAYS AS (fictionbooks + nonfictionbooks))
What are Constraints?

DB2 provides three types of constraints:

- Unique constraints, which are used to ensure that values in a column are unique
- Referential integrity constraints, which are used to define relationships between tables and ensure that these relationships remain valid
- Table check constraints, which are used to verify that column data does not violate rules defined for the column

You can define constraints when you create the table or add them later using the ALTER TABLE statement.

DB2 provides several ways to control what data can be stored in a column. These features are called constraints or rules that the database manager enforces on a data column or set of columns.
Unique constraints can be defined either as the PRIMARY KEY or UNIQUE constraint. These can be defined when a table is created as part of the CREATE TABLE statement or added after the table is created using the ALTER TABLE statement.

When do you define a PRIMARY KEY versus a UNIQUE key? This depends on the nature of the data. In the previous example, the books table has a bookid that is used to uniquely identify a book. This value is also used in other tables that contain information related to this book. In this case, you would define bookid as a primary key. DB2 allows only one primary key to be defined on a table.

The ISBN number column needs to be unique but is not a value that is otherwise referenced in the database. In this case, the isbn column can be defined as unique.

CREATE TABLE books (  
    bookid INTEGER NOT NULL PRIMARY KEY,  
    bookname VARCHAR(100),  
    isbn CHAR(10) NOT NULL CONSTRAINT booksisbn UNIQUE)  

The CONSTRAINT keyword allows you to specify a name for the constraint. In this example, the name of the unique constraint is booksisbn. The name is used, in the ALTER TABLE statement, if you want to drop the specific constraint.
DB2 allows only one primary key to be defined on a table, however, multiple unique constraints may be defined.

Whenever you define a PRIMARY or UNIQUE key on a column, DB2 creates a unique index to enforce uniqueness on the column. DB2 does not allow you to create duplicate unique constraints or duplicate indexes. For example, the following statement against the books table will fail.

```
ALTER TABLE books ADD CONSTRAINT UNIQUE (bookid)
```
Suppose we have one table to hold information about authors, and another table that lists the books the authors have written. There is a relationship between the books table and the authors table — each book has an author and that author must exist in the author table. Each author had a unique identifier stored in the authorid column. The authorid is used in the books table to identify the author of each book. To define the relationship, define the authorid column of the authors table as a primary key and then define a foreign key on the books table to establish the relationship with the authorid column in the authors table.

```
CREATE TABLE authors (  
    authorid INTEGER NOT NULL PRIMARY KEY,  
    lname VARCHAR(100),  
    fname VARCHAR(100))
```

```
CREATE TABLE books (  
    bookid INTEGER NOT NULL PRIMARY KEY,  
    bookname VARCHAR(100),  
    isbn CHAR(10),  
    authorid INTEGER REFERENCES authors)
```
In the relationship described in our example, the **author** table is the parent table and the **books** table is the dependent table. You may define more than one table as a dependent on a parent table.

You can also define relationships between rows of the same table. In this case the parent table and dependent tables are the same table.

When referential constraints are defined on a set of tables, DB2 enforces referential integrity rules on those tables when update operations are performed against those tables.

- DB2 ensures that only valid data is inserted into columns where referential integrity constraints are defined. This means that you must always have a row in the parent table with a key value that is equal to the foreign key value in the row that you are inserting into a dependent table. For example, if a new book is being inserted into the **books** table with an **authorid** of 437, then there must already be a row in the **authors** table where **authorid** is 437.

- DB2 also enforces rules when rows that have dependent rows in a dependent table are deleted from a parent table. The action DB2 takes depends on the delete rule defined on the table. There are four rules that can be specified: **RESTRICT**, **NO ACTION**, **CASCADE** and **SET NULL**.
  - If **RESTRICT** or **NO ACTION** is specified, DB2 does not allow the parent row to be deleted. The rows in dependent tables must be deleted first, followed by the row in the parent table. This is the default so this rule applies to the **authors** and **books** table as they are defined.
  - If **CASCADE** is specified, then deleting a row from the parent table automatically also deletes dependent rows in all dependent tables.
  - If **SET NULL** is specified, then the parent row is deleted from the parent table and the foreign key value in the dependent rows is set to NULL (if nullable).

- When updating key values in the parent table, there are two rules that can be specified: **RESTRICT** and **NO ACTION**. **RESTRICT** will not allow a key value to be updated if there are dependent rows in a dependent table. **NO ACTION** causes the update operation on a parent key value to be rejected if, at the end of the update, there are dependent rows in a dependent table that do not have a parent key in the parent table.
Table check constraints are used to restrict the values in a certain column of a table.

- DB2 ensures that the constraint is not violated during inserts and updates.

Suppose that we add a column to the books table for a book type and the values that are permitted are 'F' (fiction) and 'N' (non-fiction). We can add a column booktype with a check constraint as follows:

```
ALTER TABLE books ADD booktype CHAR(1)
    CHECK (booktype IN ('F', 'N'))
```

You can define check constraints when you create the table or add them later using the ALTER TABLE SQL statement. You can modify check constraints by dropping and then recreating them using the ALTER TABLE SQL statement.
Informational constraints are either referential or check constraints that are not enforced in the server but are exploited in the optimizer.

- Switch constraint enforcement ON/OFF
- Switch constraint exploitation ON/OFF

**Referential integrity constraint example:**

```
ALTER TABLE T ADD CONSTRAINT CHK1 CHECK (C1 < C2)
[[NOT ENFORCED] [(ENABLE|DISABLE) QUERY OPTIMIZATION]]
ALTER TABLE C ADD CONSTRAINT RI1 REFERENCES P
[[NOT ENFORCED] [(ENABLE|DISABLE) QUERY OPTIMIZATION]]
```

**Check constraint example:**

```
ALTER TABLE T ALTER CHECK CHK1
[[NOT ENFORCED] [(ENABLE|DISABLE) QUERY OPTIMIZATION]]
ALTER TABLE C ALTER FOREIGN KEY RI1
[[NOT ENFORCED] [(ENABLE|DISABLE) QUERY OPTIMIZATION]]
```
### What Are Views?

A view:
- Provides a transparent view of the data in underlying tables
  - Contains no data itself
- Makes the data simpler to access
- Can also be used to restrict which rows and columns can be viewed or updated
- Appears just like a table to the user

You can create a view on an existing table (or tables) or on another view or any combination.
- A view defined on another view is called a nested view

Views allow different users or applications to look at the same data in different ways. This not only makes the data simpler to access, but it can also be used to restrict which rows and columns can be viewed or updated.

For example, suppose that a company has a table containing information about its employees. A manager needs to see information about his employees only, while a directory application needs to see all employees and their address and telephone numbers, but not their salaries. A view can be created that shows only the employees in a department. Another view can be created that shows only the name, address and telephone number.

A view appears just like a table to the user. Except for the view definition, a view does not take up space in the database. The data presented in a view is derived from another table. You can create a view on an existing table (or tables) or on another view or any combination. A view defined on another view is called a nested view.

You can define a view with different column names than the corresponding columns of the base table. You can also define views that check that the data inserted or updated stays within the conditions of the view.

The list of views defined in the database is stored in the system catalog table `sysibm.sysviews` which also has a view defined on it called `syscat.views`. The system catalog also has a `syscat.viewdep` that, for each view defined in the database, has a row for each dependent (view or table) of that view. Also, each view has an entry in `sysibm.systables` and entries in `sysibm.syscolumns` since views can be used just like tables.
Creating a View

The CREATE VIEW statement is used to define a view.

- A SELECT statement is used to specify which rows and columns will be presented in the view
- You can define different column names in the view from those that are in the base table by specifying them in the CREATE VIEW statement

The DROP VIEW statement is used to drop a view from the database.

- If a table or another view on which a view is based is dropped, the view remains defined in the database but becomes inoperative

For example, we want to create a view that will show only the nonfiction books in the table.

```
CREATE VIEW nonfictionbooks AS
    SELECT * FROM books WHERE booktype = 'N'
```

Note that after this view is defined, there will be an entry in `syscat.views`, `syscat.viewdep` and `syscat.tables`.

To define different column names in the view from those that are in the base table, you can specify them in the CREATE VIEW statement. This statement creates a `mybookview` view that contains two columns: `title`, which represents the `bookname` column, and `type`, which represents the `booktype` column.

```
CREATE VIEW mybookview (title,type) AS
    SELECT bookname,booktype FROM books
```

The DROP VIEW SQL statement is used to drop a view from the database. If a table or another view on which a view is based is dropped, the view remains defined in the database but becomes inoperative. The `valid` column of `syscat.views` indicates whether a view is valid ('Y') or not ('X'). Even when the base table is re-created, the view must also be re-created.

To drop the `nonfictionbooks` view from the database:

```
DROP VIEW nonfictionbooks
```
You cannot modify a view. To change a view definition, you must drop and re-create the view. The ALTER VIEW statement provided is used only to modify reference types and will not be discussed here.

**Read-Only Views**

When you create a view, it may be defined as a read-only view or as a view that can be updated. The SELECT statement of a view determines whether or not the view is read-only. Generally, if the rows of a view can be mapped to rows of the base table, then the view can be updated. For example, as defined in the previous example, the view `nonfictionbooks` can be updated because each row in the view is a row in the base table.

The rules for creating views that can be updated are complex and depend on the definition of the query. For example, views that use VALUES, DISTINCT, or JOIN features cannot be updated. You can easily determine whether a view can be updated by looking at the `readonly` column of `syscat.views`: 'Y' means it is read-only and 'N' means it is not read-only.

**For More Information**

The detailed rules for creating views that can be updated are documented in the *DB2 SQL Reference*.

---

**V8**

- Completes encapsulation for VIEWS based on UNION ALL
- Provides single interface for user (SELECT, UPDATE, DELETE, INSERT)
- Supports improved database scalability through partitioning
Views WITH CHECK OPTION

Data may be inserted or updated in the underlying tables through a view.

Defining a view using WITH CHECK OPTION tells DB2 to check that statements using the view satisfy the conditions of the view.

The following statement defines a view using WITH CHECK OPTION:

```
CREATE VIEW nonfictionbooks AS
    SELECT * FROM books WHERE booktype = 'N'
    WITH CHECK OPTION
```

The `nonfictionbooks` view defined previously includes only the rows where the `booktype` is 'N'. If you insert a row where the `booktype` is 'F' into the view, DB2 will insert the row into the base table `books`. However, if you then select from the view, the newly inserted row cannot be seen through the view. If you do not want to allow a user to insert rows that are outside the scope of the view, you can define the view with the check option.

This view shown above still restricts the user to seeing only nonfiction books, however, it also restricts inserting rows that do not have a value of 'N' in the `booktype` column, and updating the value of the `booktype` column in existing rows to a value other than 'N'. The following statements will no longer be allowed:

```
INSERT INTO nonfictionbooks VALUES (...,'F')
UPDATE nonfictionbooks SET booktype = 'F' WHERE bookid = 111
```
When defining nested views, the check option can be used to restrict operations. However, there are other options you can specify to define how the restrictions are inherited. The check option can be defined either as CASCADED or LOCAL. CASCADED is the default if the keyword is not specified. To explain the differences between the behavior of CASCADED and LOCAL, we need to look at several possible scenarios.

When a view is created WITH CASCADED CHECK OPTION, all statements executed against the view must satisfy the conditions of the view and all underlying views—even if those views were not defined with the check option. Suppose that the view `nonfictionbooks` is created without the check option and we also create a view `nonfictionbooks1` based on the view `nonfictionbooks` using the CASCADED keyword.

```sql
CREATE VIEW nonfictionbooks AS
    SELECT * FROM books WHERE booktype = 'N'
CREATE VIEW nonfictionbooks1 AS
    SELECT * FROM nonfictionbooks WHERE bookid > 100
    WITH CASCADED CHECK OPTION
```
The following INSERT statements would not be allowed because they do not satisfy the conditions of at least one of the views.

- INSERT INTO nonfictionbooks1 VALUES(10,..,'N')
- INSERT INTO nonfictionbooks1 VALUES(120,..,'F')
- INSERT INTO nonfictionbooks1 VALUES(10,..,'F')

However, the following INSERT statement would be allowed because it satisfies the conditions of both of the views.

- INSERT INTO nonfictionbooks1 VALUES(120,..,'N')
Now suppose we create a view `nonfictionbooks2` based on the view `nonfictionbooks` using WITH LOCAL CHECK OPTION. Now, statements executed against the view need only satisfy conditions of views which have the check option specified.

Example view using the LOCAL CHECK OPTION:

```sql
CREATE VIEW nonfictionbooks2 AS
    SELECT * FROM nonfictionbooks WHERE bookid > 100
    WITH LOCAL CHECK OPTION
```

In this case, the following INSERT statements would not be allowed because they do not satisfy the `bookid` > 100 condition of the `nonfictionbooks2` view.

```sql
INSERT INTO nonfictionbooks2 VALUES(10,..,'N')
INSERT INTO nonfictionbooks2 VALUES(10,..,'F')
```

However, the following INSERT statements would be allowed even though the value 'N' does not satisfy the `booktype = 'N'` condition of the `nonfictionbooks` view.

```sql
INSERT INTO nonfictionbooks2 VALUES(120,..,'N')
INSERT INTO nonfictionbooks2 VALUES(120,..,'F')
```
## What is an Index?

An index is an ordered list of the key values of a column or columns of a table.

There are two reasons why you might create an index:

- To ensure uniqueness of values in a column or columns
- To improve performance of queries against the table
  - The DB2 optimizer chooses to use indexes when performing queries to find the required rows faster or to present results of a query in the order of the index

Two types of indexes:

- Unique
- Non-unique

Indexes can be defined as unique or non-unique. Non-unique indexes allow duplicate key values. Unique indexes allow only one occurrence of a key value in the list. Unique indexes do allow a single NULL to be present. However, a second NULL value would cause a duplicate and, therefore, is not allowed.

Indexes are created using the CREATE INDEX statement. Indexes are also created implicitly in support of a primary key or unique constraint. When a unique index is created, the key data is checked for uniqueness and the operation will fail if duplicates are found.

Indexes are created as ascending, descending or bidirectional. The option you choose depends on how the application accesses the data.

### V8

Version 8 supports type-2 indexes. The primary advantages of type-2 indexes are:

- They improve concurrency because the use of next-key locking is reduced.
- An index can be created on columns that have a length greater than 255 bytes.
- They are required for the new multidimensional clustering facility (beyond the scope of this course).
Example of creating a non-unique, ascending index on the `bookname` column:

```
CREATE INDEX ibookname ON books (bookname)
```

By default, an index is created in ascending order. You can specify different orders for the columns in the index.

```
CREATE INDEX i2bookname ON books (authoid DESC, bookname ASC)
```

Specify the ALLOW REVERSE SCANS for a bidirectional index.

```
CREATE INDEX bibookname ON books (bookname) ALLOW REVERSE SCANS
```

An index can be placed in a separate table space from the table, but only in the CREATE TABLE statement.

In our example, we have a primary key on the `bookid` column. Often, searches are done on the book title so an index on `bookname` would be appropriate. This statement creates a non-unique, ascending index on the `bookname` column.

```
CREATE INDEX ibookname ON books (bookname)
```

By default, an index is created in ascending order. You can specify different orders for the columns in the index. The following statement defines an index on the `authorid` and `bookname` columns. The values of the `authorid` column are sorted in descending order and the values of the `bookname` column are sorted in ascending order within the same `authorid`.

```
CREATE INDEX i2bookname ON books (authoid DESC, bookname ASC)
```

When an index is created in a database, the keys are stored in the specified order. The index helps improve performance of queries requiring the data in the specified order. An ascending index is also used to determine the result of the MIN column function; a descending index is used to determine the result of the MAX column function. If the application needs the data to be ordered in the opposite sequence to the index as well, DB2 allows the creation of a bidirectional index. A bidirectional index eliminates having to create an index in the reverse order, and it eliminates the need for the optimizer to sort the data in the reverse order. It also allows the
efficient retrieval of MIN and MAX functions values. To create a bidirectional index, specify the ALLOW REVERSE SCANS option on the CREATE INDEX statement.

```
CREATE INDEX bibookname ON books (bookname) ALLOW REVERSE SCANS
```

DB2 does not allow indexes with the same definition to be created. This applies even if the index was implicitly created in support of a primary key or unique constraint. So, since the `books` table already has a primary key defined on the `bookid` column, attempting to create an index on `bookid` column will fail.

Creating an index can take a long time. DB2 must read each row to extract the keys, sort the keys, and then write the list to the database. If the table is large, then a temporary table space is used to sort the keys.

The index is stored in a table space. If your table resides in a database managed table space, you have the option of separating the indexes into a separate table space. This must be defined when you create the table using the INDEXES IN clause. The location of indexes is set when the table is created and cannot be changed unless the table is dropped and re-created.

Of course, DB2 also provides the DROP INDEX SQL statement to remove an index from the database. There is no way to modify an index. For example, if you need to change an index to add another column to the key, you must drop and re-create it.
When creating an index, you have the option to include extra column data that are stored with the key but are not actually part of the key itself and are not sorted. The main reason for including additional columns in an index is for performance of certain queries. DB2 does not need to access the data page because the data value is already available on the index page. Included columns can only be defined for unique indexes. However, the included columns are not considered when enforcing uniqueness of the index.

So why not just include all the data in the indexes? First, this would require more physical space in the database because essentially, the data is being duplicated in the index. Second, all the copies of the data need to be updated whenever the data value is updated, and this would be a significant overhead in databases where many updates occur.

Using Include Columns in Indexes

Suppose that often we need to get a list of book names ordered by bookid.

The query would be like this:

```
SELECT bookid, bookname FROM book ORDER BY bookid
```

A possible index that may improve performance would be create by:

```
CREATE UNIQUE INDEX ibookid ON
books (bookid) INCLUDE (bookname)
```

As a result, all the data required for the query result is present in the index, and no data pages need to be retrieved.
## What Indexes Should I Create?

Some considerations when creating indexes:

- Since indexes are a permanent list of the key values, they require space in the database
  - DB2 provides a tool to help you estimate the size of an index
- Indexes are additional copies of the values so they must be updated if the data in the table is updated
  - For frequent updates, this could impact performance
- Indexes will significantly improve performance of queries when defined on the appropriate columns

Use the Index Advisor to help you determine which indexes to define.

These are some considerations when creating indexes.

- Since indexes are a permanent list of the key values, they require space in the database. So creating many indexes will require more storage space in your database. The amount of space required is determined by the length of the key columns. DB2 provides a tool to help you estimate the size of an index.
- Indexes are additional copies of the values so they must be updated if the data in the table is updated. If table data is frequently updated, consider what impact additional indexes will have on update performance.
- Indexes significantly improve performance of queries when defined on the appropriate columns.

DB2 provides a tool called the *Index Advisor* to help you determine which indexes to define. Index Advisor allows you to specify the workload that will be executed against a table and it recommends indexes to create on the table. This tool is accessed through the *Create Index* wizard, or at the command line by executing `db2advis`. 
You should now be able to:

- Demonstrate usage of DB2 data types
- Identify when referential constraints are used
- Identify methods of data validation
- Identify characteristics of a table, view or index
- Explain the concept of extenders
Exercises
Exercise 1

This exercise is designed to reinforce your knowledge of DB2 built-in data types and database objects.

1.1 There are several built-in Numeric data types available in DB2. List them.

______________
______________
______________
______________
______________
______________
______________
______________

1.2 There are several built-in String data types available in DB2. List them.

______________
______________
______________
______________
______________
______________
______________
______________
______________
______________

1.3 There are several built-in Date-time data types available in DB2. List them.

______________
______________
______________

1.4 True or False. You could use a user-defined distinct data type to create a unit-of-measure data type.

______________
1.5 What purpose do DB2 Extenders provide?

1.6 What two methods can you use to create a table in a DB2 database?

1.7 When a table is created, where is its structural information kept in DB2?

1.8 If a table space location is not specified in the CREATE TABLE statement, where is the extent for that table created?

1.9 What is required to remove a column from a table structure?

1.10 How would you drop a table?

1.11 What column option would you use in a CREATE TABLE statement that would require data in a column of every row in the table?

1.12 If you needed to automatically create a unique column value for each row in a table, what type of column option could you use?

1.13 What are the three types of constraints in DB2?

1.14 Must you include the constraints in the CREATE TABLE statement?

1.15 What are the two ways you could use to ensure uniqueness on a table column?

1.16 For referential integrity between two tables, what type of column option would you use for the parent table in that relationship? What type of element would you use on a column of the dependent table?
1.17 In the case where a parent table row had dependent rows in the dependent table and the relationship was set up to use default actions, what would happen if you tried to delete the parent row?

1.18 What would happen in the previous example if, instead of the default actions, you had set the referential integrity with the CASCADE option?

1.19 Assuming the DDL statement shown below, what would happen if, after the alter, you tried to insert data into the books table where the booktype was a value of 'Z'?

   ```db2
   ALTER TABLE books ADD booktype CHAR(1)
   CHECK (booktype IN ('F', 'N'))
   ```

1.20 What is a view? Does a view have a physical structure that can store data?

1.21 Can a view be used to insert data into a table?

1.22 What are the two reasons for using an index on a column or columns of a table?

1.23 What are the two types of indexes?

1.24 What qualifier would you use when creating an index so that the index can be used bi-directionally?

1.25 What does the INCLUDE column qualifier provide for you when creating an index?

1.26 What tool is available to you to help determine the size of an index, and which columns might need an index?
Exercise 2

This exercise will give you practice on creating and altering database objects. You will be asked to use a combination of the GUI tools and the command line.

2.1 Using the **Control Center**, create a table in the **sample** database using the following structure:

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table name</td>
<td>Person</td>
<td></td>
</tr>
<tr>
<td>Column 1</td>
<td>Idnum</td>
<td>INTEGER NOT NULL PRIMARY KEY</td>
</tr>
<tr>
<td>Column 2</td>
<td>FName</td>
<td>CHAR(20) NOT NULL</td>
</tr>
<tr>
<td>Column 3</td>
<td>LName</td>
<td>CHAR(20) NOT NULL</td>
</tr>
<tr>
<td>Column 4</td>
<td>Address</td>
<td>CHAR(30)</td>
</tr>
<tr>
<td>Column 5</td>
<td>City</td>
<td>CHAR(20)</td>
</tr>
<tr>
<td>Column 6</td>
<td>State</td>
<td>CHAR(2)</td>
</tr>
<tr>
<td>Column 7</td>
<td>Phone</td>
<td>CHAR(18)</td>
</tr>
</tbody>
</table>

2.2 In a **Command** window, use the **db2** utility to create the second table, using the following structure:

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table name</td>
<td>Customer</td>
<td></td>
</tr>
<tr>
<td>Column 1</td>
<td>LastOrderDate</td>
<td>DATE</td>
</tr>
<tr>
<td>Column 2</td>
<td>LastOrderAmnt</td>
<td>DECIMAL(6,2)</td>
</tr>
<tr>
<td>Column 3</td>
<td>Age</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Column 4</td>
<td>Gender</td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>Column 5</td>
<td>Idnum</td>
<td>INTEGER NOT NULL</td>
</tr>
</tbody>
</table>

2.3 Now you will add a referential constraint on the **customer** table. This will cause the **person** table to be the parent table, and the **customer** table to be the dependent table. Use the **Command Window** and the **db2** utility to alter the **customer** table and add the foreign key to it.

2.4 Check your work by using the Control Center.
2.5 Are there any CHECK constraints on your dependent table?

2.6 What will happen if you try to delete a parent row from the parent table while that row has dependent rows in the dependent table?

2.7 In the above condition, what could you do to remove both the parent row and all of its dependent rows with one DELETE statement?

2.8 Using the Command Window, add a CHECK constraint to the gender column. The viable gender codes are: M and F.

2.9 Using the Control Center, create a view, named lee_sales, on the sales table. Include all of the columns, but use a WHERE clause to select only the rows for sales person LEE.

2.10 Test your view by selecting data from it the same way you would select data from any table.

2.11 Using a Command Window, try to create a unique index on the idnum column in the person table. What error do you receive?
Solutions
This exercise is designed to reinforce your knowledge of DB2 built-in data types and database objects.

1.1 There are several built-in Numeric data types available in DB2. List them.
   - SMALLINT
   - INTEGER
   - BIGINT
   - DECIMAL
   - REAL
   - FLOAT

1.2 There are several built-in String data types available in DB2. List them.
   - CHAR
   - VARCHAR
   - LONG VARCHAR
   - CLOB
   - GRAPHIC
   - VARGRAPHIC
   - LONG VARGRAPHIC
   - DBCLOB
   - BLOB

1.3 There are several built-in Date-time data types available in DB2. List them.
   - DATE
   - TIME
   - TIMESTAMMP

1.4 True or False. You could use a user-defined distinct data type to create a unit-of-measure data type.
   True.
1.5 What purpose do DB2 Extenders provide?

DB2 Extenders provide the ability to create new complex data types within a database. The extender supplies the definition of the new data type (UDT), as well as providing support programs (UDFs) to index and manipulate that data type. Extenders provide the ability to store audio, video, imaging, and text, to name a few.

1.6 What two methods can you use to create a table in a DB2 database?

You can use the CREATE TABLE statement at the command line, using the DB2 command line utility. You can also use the Administration Client part of the GUI tools to create the table. The information supplied through use of the Create Table wizard is compiled into a DDL statement, which is then executed by the DB2 command line utility.

1.7 When a table is created, where is its structural information kept in DB2?

The basic information for each table is registered in the syscat.tables system catalog table for that database. The column information for a table is contained in the syscat.columns system catalog table. To view this information:

```
db2 "SELECT * FROM syscat.tables, syscat.columns WHERE syscat.tables.tabname = syscat.columns.tabname"
```

1.8 If a table space location is not specified in the CREATE TABLE statement, where is the extent for that table created?

The extent for a table (and its index extent) are created in the USERSPACE1 table space by default.

1.9 What is required to remove a column from a table structure?

You cannot remove a column by using the ALTER TABLE statement. You must first export the data to either another table or an external data file, then drop the table, then re-create the table correctly, then reload the table with the data.

1.10 How would you drop a table?

Use the DDL statement:

```
db2 "DROP TABLE tablename"
```

where tablename is the name of the table to be dropped.
1.11 What column option would you use in a CREATE TABLE statement that would require
data in a column of every row in the table?

You would use the NOT NULL column option on a column during the CREATE
TABLE statement.

1.12 If you needed to automatically create a unique column value for each row in a table, what
type of column option could you use?

You would use the GENERATED ALWAYS AS IDENTITY option on the column
definition during the CREATE TABLE statement.

1.13 What are the three types of constraints in DB2?

The unique constraint — ensures values in a column are unique.
The referential integrity constraint — used to define and ensure relationships
between tables.
The table check constraint — used to verify column data values abide by rules set
for that column.

1.14 Must you include the constraints in the CREATE TABLE statement?

No, you could also use the ALTER TABLE statement to impose the constraints
after the table has been created.

1.15 What are the two ways you could use to ensure uniqueness on a table column?

The UNIQUE constraint can be included in the column definition, or you could
impose the PRIMARY KEY constraint on the column. Only one PRIMARY KEY
constraint can be set on a table. You would need to use the UNIQUE constraint on
the other columns in that table requiring uniqueness.

1.16 For referential integrity between two tables, what type of column option would you use
for the parent table in that relationship? What type of element would you use on a
column of the dependent table?

The PRIMARY KEY would be defined on a column in the parent table, while a
foreign key (REFERENCES) would be defined on a column in the dependent table.
1.17 In the case where a parent table row had dependent rows in the dependent table and the relationship was set up to use default actions, what would happen if you tried to delete the parent row?

You would get a referential integrity error and the row would not be deleted because you cannot have dependent rows without parent rows.

1.18 What would happen in the previous example if, instead of the default actions, you had set the referential integrity with the CASCADE option?

This would cause a cascading delete. The dependent rows of the parent, as well as the parent row, would be deleted from the two tables.

1.19 Assuming the DDL statement shown below, what would happen if, after the alter, you tried to insert data into the books table where the booktype was a value of 'Z'?

```sql
db2 "ALTER TABLE books ADD booktype CHAR(1) CHECK (booktype IN ('F', 'N'))"
```

The insert would fail because the check constraint on booktype indicates the 'Z' value is not a valid value for that column.

1.20 What is a view? Does a view have a physical structure that can store data?

A view is a ‘window’ on the data in the tables specified in the view. The view itself has no physical structure, therefore it contains no data. It simply provides a ‘view’ of the data in its underlying tables.

1.21 Can a view be used to insert data into a table?

Yes, in some cases a view can be used to insert data into its underlying table. In order to maintain data integrity, you should use the WITH CHECK OPTION when creating a view you will use for inserts.

1.22 What are the two reasons for using an index on a column or columns of a table?

You would use an index to ensure uniqueness in a column or columns, or you would use an index to improve performance in data queries.

1.23 What are the two types of indexes?

There is a unique index type, and a non-unique index type.
1.24 What qualifier would you use when creating an index so that the index can be used bi-directionally?

   Use the ALLOW REVERSE SCANS qualifier.

1.25 What does the INCLUDE column qualifier provide for you when creating an index?

   You can include other columns to be attached to the index, but they would not be a part of the sorted key—they would be included for convenience. If done properly, the use of include columns in an index may greatly improve query performance, but at the cost of storage space and slower data updates.

1.26 What tool is available to you to help determine the size of an index, and which columns might need an index?

   The Index Advisor, which is part of the Control Center Administration tool GUI Client. Access it as if you were going to create an index using the Create Index wizard.
This exercise will give you practice on creating and altering database objects. You will be asked to use a combination of the GUI tools and the command line.

2.1 Using the Control Center, create a table in the sample database using the following structure:

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table name</td>
<td>Person</td>
<td></td>
</tr>
<tr>
<td>Column 1</td>
<td>Idnum</td>
<td>INTEGER NOT NULL PRIMARY KEY</td>
</tr>
<tr>
<td>Column 2</td>
<td>FName</td>
<td>CHAR(20) NOT NULL</td>
</tr>
<tr>
<td>Column 3</td>
<td>LName</td>
<td>CHAR(20) NOT NULL</td>
</tr>
<tr>
<td>Column 4</td>
<td>Address</td>
<td>CHAR(30)</td>
</tr>
<tr>
<td>Column 5</td>
<td>City</td>
<td>CHAR(20)</td>
</tr>
<tr>
<td>Column 6</td>
<td>State</td>
<td>CHAR(2)</td>
</tr>
<tr>
<td>Column 7</td>
<td>Phone</td>
<td>CHAR(18)</td>
</tr>
</tbody>
</table>

a. On your workstation, select Start > Programs > IBM DB2 > Control Center (for v8, select Start > Programs > IBM DB2 > General Administration Tools > Control Center).

b. Open the objects in the left pane until you reach the SAMPLE database objects.

c. Right-click on Table > Create > Table Using Wizard.

d. Enter the table name — Person — then click Next.

e. On the Change Columns panel, click Add and enter your column characteristics. Do this for all seven columns. Remember to remove the check in the Nullable box for those columns that are NOT NULL. Click Next when done creating the seven columns.

f. The first column, Idnum, needs a PRIMARY KEY added to it. On the Primary Key panel, select the Idnum column from the left pane and click on the > symbol between the panes. Click Next.

g. On the Table Space panel, click Next to use the default table space.
i. **On the Summary panel, click Show SQL to view the DDL statement that will be used to create your new table. Notice that your Schema has been added to the table object. Also note that all columns and the table name are shown in upper case characters. Finally, notice that the PRIMARY KEY clause is shown after all of the columns have been listed. You might see “DATA CAPTURE NONE”. Ignore this; it has reference to replication. Close the Show SQL panel, then click Finish on the Summary panel.**

j. **When finished, the Control Center panel will return. Check on the right pane of the Tables objects to view the table you just created (double-click on your table).**

2.2 In a **Command window,** use the **db2** utility to create the second table, using the following structure:

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table name</td>
<td>Customer</td>
<td></td>
</tr>
<tr>
<td>Column 1</td>
<td>LastOrderDate</td>
<td>DATE</td>
</tr>
<tr>
<td>Column 2</td>
<td>LastOrderAmnt</td>
<td>DECIMAL(6,2)</td>
</tr>
<tr>
<td>Column 3</td>
<td>Age</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Column 4</td>
<td>Gender</td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>Column 5</td>
<td>Idnum</td>
<td>INTEGER NOT NULL</td>
</tr>
</tbody>
</table>

a. **On your workstation, select** Start > Programs > IBM DB2 > Command Window (for v8, select Start > Programs > IBM DB2 > Command Line Tools > Command Window).

b. **Change to the root directory by typing:** cd \n
c. **Create your second table using the following DB2 commands:**
   
   `db2 "connect to sample"
   db2 "create table customer (lastorderdate date, lastorderamnt decimal(6,2), age integer, gender char(1), idnum integer not null)"

   **Note:** enter it all on one line. The above is shown for clarity.
2.3 Now you will add a referential constraint on the **customer** table. This will cause the **person** table to be the parent table, and the **customer** table to be the dependent table. Use the **Command Window** and the db2 utility to alter the **customer** table and add the foreign key to it.

```
db2 "ALTER TABLE customer
    ADD CONSTRAINT fk_idnum FOREIGN KEY(idnum)
    REFERENCES person"
```

2.4 Check your work by using the Control Center.

   a. **With the sample objects shown in the left pane of the Control Center, click on the Tables folder in the left pane.**
   b. **The tables are listed alphabetically in the right pane. Find your customer table and double-click on it.**
   c. **Click on the Columns tab and make sure all of your columns are created, and that they are defined with the correct data type.**
   d. **Click on the Foreign Keys tab. Check that the foreign key is defined on the Idnum column, and references the Person table.**

2.5 Are there any CHECK constraints on your dependent table?

   **No. By viewing the Check Constraint tab on the customer table panel, you find no constraints.**

2.6 What happens if you try to delete a parent row from the parent table while that row has dependent rows in the dependent table?

   **You will receive an error, indicating that you cannot orphan dependent rows by deleting their parent row.**

2.7 In the above condition, what could you do to remove both the parent row and all of its dependent rows with one DELETE statement?

   **You could use the ALTER TABLE statement to drop the foreign key constraint, then re-add it using the ON DELETE CASCADE clause.**
2.8 Using the **Command Window**, add a CHECK constraint to the **gender** column. The viable **gender** codes are: **M** and **F**.

    db2 "alter table customer add check (gender in ('M','F'))"

2.9 Using the Control Center, create a view, named **lee_sales**, on the **sales** table. Include all of the columns, but use a WHERE clause to select only the rows for sales person **LEE**.

   a. **With the sample objects shown in the left pane of the** Control Center, **click on the Views folder in the left pane.**
   b. **Right-click on** Views > Create.
   c. **Enter lee_sales in the View name field.**
   d. **Replace the text in the SQL statement box with:**
       - SELECT * FROM sales
       - WHERE sales_person = 'LEE'
   e. **Click on the Show SQL button:**
       - CREATE VIEW lee_sales AS
       - SELECT * FROM sales
       - WHERE sales_person = 'LEE'
       - Click on the Close button to close the panel.
   f. **Click on the OK button. This will create your view.**

2.10 Test your view by selecting data from it the same way you would select data from any table.

    db2 "SELECT * FROM lee_sales"

2.11 Using a **Command Window**, try to create a unique index on the **idnum** column in the **person** table. What error do you receive?

    db2 "CREATE UNIQUE INDEX idx1 on person(idnum)"

    SQL0605W  The index was not created because the index
    "SYSIBM.SQL020701111301090" already exists with the required
description.
    SQLSTATE=01550
Module 7

Database Concurrency
Objectives

At the end of this module, you will be able to:

- Identify factors that influence locking
- List the objects that locks can be obtained on
- Identify scope of different types of DB2 locks
- Identify factors affecting amount of locks that are used
- Identify factors affecting the type of locks obtained
- Identify the isolation levels that should be used for a given a situation
**Understanding Data Consistency**

DB2 Universal Database uses the following data consistency support mechanisms:
- Transactions
- Locking
- Logging

What is data consistency? The best way to answer this question is by looking at an example. Suppose your company owns a chain of restaurants and you have a database that is designed to keep track of supplies stored at each of those restaurants. To facilitate the supplies purchasing process, your database contains an inventory table for each restaurant in the chain. Whenever supplies are received or used by an individual restaurant, the corresponding inventory table for that restaurant is modified to reflect the changes. Now, suppose some bottles of ketchup are physically moved from one restaurant to another. The ketchup bottle count value stored in the donating restaurant's table needs to be lowered and the ketchup bottle count value stored in the receiving restaurant's table needs to be raised in order to accurately represent this inventory move. If a user lowers the ketchup bottle count in the donating restaurant's inventory table but fails to raise the ketchup bottle count in the receiving restaurant's inventory table, the data will become inconsistent. Now the total ketchup bottle count for the chain of restaurants is no longer accurate.

Data in a database can become inconsistent if a user forgets to make all necessary changes (as in the previous example), if the system crashes while the user is in the middle of making changes, or if a database application, for some reason, stops prematurely. Inconsistency can also occur when several users are accessing the same database tables at the same time.
Transactions and Transaction Boundaries

A transaction, (otherwise known as a unit of work) is a recoverable sequence of one or more SQL operations grouped together as a single unit, usually within an application process.

- The initiation and termination of a transaction defines the points of database consistency
- Either:
  - The effects of all SQL operations performed within a transaction are applied to the database (COMMIT)
  or
  - The effects of all SQL operations performed are completely "undone" and thrown away (ROLLBACK)

With embedded SQL applications and scripts that are run from the Command Center, the Script Center, or the Command Line Processor, transactions are automatically initiated the first time an executable SQL statement is executed, either after a connection to a database has been established or after an existing transaction has been terminated. Once initiated, a transaction must be explicitly terminated by the user or application that initiated it. In most cases, transactions are terminated by executing either the COMMIT or the ROLLBACK statement. When the COMMIT statement is executed, all changes made to the database since the transaction was initiated are made permanent (committed). When the ROLLBACK statement is executed, all changes made to the database since the transaction was initiated are backed out and the database is returned (rolled back) to the state it was in before the transaction began. In either case, the database is guaranteed to be returned to a consistent state at the completion of the transaction.

Note

It is important to note that, while transactions provide generic database consistency by ensuring that changes to data only become permanent after a transaction has been successfully committed, it is up to the user/application to ensure that the sequence of SQL operations within each transaction always results in a consistent database.
A commit or rollback operation only affects changes that are made within the transaction that ends the COMMIT or ROLLBACK operation.

- As long as data changes remain uncommitted, other users and applications are usually unable to see them
  - There are exceptions which we will look at later
- They can be backed out with a ROLLBACK operation

Once data changes are committed, they become accessible to other users and applications and can no longer be removed by a ROLLBACK operation.

It was mentioned earlier that in most cases, transactions are terminated by executing either the COMMIT or the ROLLBACK statement. When the COMMIT statement is executed, all changes made to the database since the transaction was initiated are made permanent. When the ROLLBACK statement is executed, all changes made to the database since the transaction was initiated are backed out and the database is returned to the state it was in before the transaction began. To understand how each of these statements work, it helps to look at an example:

```sql
CONNECT TO my_db
CREATE TABLE department (  
department_id INTEGER NOT NULL, department_name VARCHAR(20))

INSERT INTO department VALUES(100, 'PAYROLL')
INSERT INTO department VALUES(200, 'ACCOUNTING')  
COMMIT

INSERT INTO department VALUES(300, 'SALES')
ROLLBACK

INSERT INTO department VALUES(500, 'MARKETING')
COMMIT
```
If the above SQL statements are executed in the order shown, a table named `department` is created that looks something like this:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>PAYROLL</td>
</tr>
<tr>
<td>200</td>
<td>ACCOUNTING</td>
</tr>
<tr>
<td>500</td>
<td>MARKETING</td>
</tr>
</tbody>
</table>

That's because when the first COMMIT statement is executed, the creation of the table named `department`, along with the insertion of two records into the `department` table are made permanent. On the other hand, when the ROLLBACK statement is executed, the third record inserted into the `department` table is removed and the table is returned to the state it was in before the insert operation was performed. Finally, when the second COMMIT statement is executed, the insertion of the fourth record into the `department` is made permanent and the database is again returned to a consistent state.
We have just seen what happens when a transaction is terminated by a COMMIT or a ROLLBACK statement. But what happens if a system failure occurs before a transaction can be completed? In these situations the DB2 Database Manager backs out all uncommitted changes in order to restore the database consistency it assumes existed when the transaction was initiated. This is done through the use of log files, which contain information about each SQL statement executed by a transaction, along with information about whether or not that transaction was successfully committed or rolled back.

The slide shows the effects of a transaction that fails before it can be successfully terminated.
The above illustration shows the effects of a successful transaction.
Multiple Users Accessing a Database

When transactions are not isolated from each other in multi-user environments, four types of events (or phenomenon) can occur:

- Lost updates
- Dirty reads
- Non-repeatable reads
- Phantoms

DB2 UDB uses the following isolation levels to enforce concurrency:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read

With single-user databases, each transaction is executed serially and does not have to contend with interference from other transactions. In a multi-user database environment, however, transactions can be executed simultaneously and each has the potential to interfere with any other transaction that is running. When transactions are not isolated from each other in multi-user environments, four types of events (or phenomena) can occur:

- **Lost updates**: This event occurs when two transactions read the same data, attempt to update that data, and one of the updates is lost. For example: Transaction 1 and Transaction 2 read the same row of data and both calculate new values for that row based upon the data read. If Transaction 1 updates the row with its new value and Transaction 2 updates the same row, the update operation performed by Transaction 1 is lost.

- **Dirty reads**: This event occurs when a transaction reads data that has not yet been committed. For example: Transaction 1 changes a row of data and Transaction 2 reads the changed row before Transaction 1 has committed the change. If Transaction 1 rolls back the change, Transaction 2 will have read data that is considered never to have existed.
- Non-repeatable reads: This event occurs when a transaction reads the same row of data twice, but gets different data values each time. For example: Transaction 1 reads a row of data and Transaction 2 changes or deletes that row and commits the change. When Transaction 1 attempts to reread the row, it retrieves different data values (if the row was updated) or discover that the row no longer exists (if the row was deleted).

- Phantoms: This event occurs when a row of data that matches search criteria is seen in a later read operation, but not initially. For example: Transaction 1 reads a set of rows that satisfy some search criteria and Transaction 2 inserts a new row that matches Transaction 1's search criteria. If Transaction 1 re-executes the query that produced the original set of rows, a different set of rows are retrieved.

Maintaining database consistency and data integrity, while allowing more than one application to access the same data at the same time, is known as *concurrency*. One of the ways DB2 Universal Database attempts to enforce concurrency is through the use of *isolation levels*, which determines how data used in one transaction is locked or isolated from other transactions while it is being accessed.
When the Repeatable Read isolation level is used, all rows referenced by a single transaction are locked for the duration of that transaction. With this isolation level, any SELECT statement that is issued more than once within the same transaction always yields the same results; lost updates, dirty reads, non-repeatable reads, and phantoms cannot occur.

Each row referenced by the isolating transaction is locked—not just the rows that are actually retrieved and/or modified.

When the Repeatable Read isolation level is used, all rows referenced by a single transaction are locked for the duration of that transaction. With this isolation level, any SELECT statement that is issued more than once within the same transaction always yields the same results; lost updates, dirty reads, non-repeatable reads, and phantoms cannot occur.

Transactions using the Repeatable Read isolation level can retrieve the same set of rows multiple times and perform any number of operations on them until terminated (either by a COMMIT or a ROLLBACK operation); other transactions are not allowed to perform INSERT, UPDATE, or DELETE operations that affect the set of rows being used as long the isolating transaction exists. To guarantee that the data being accessed by a transaction running under the Repeatable Read isolation level is not adversely affected by other transactions, each row referenced by the isolating transaction is locked—not just the rows that are actually retrieved and/or modified. Thus, if a transaction scans 1000 rows but only retrieves 10, locks are acquired and held on all 1000 rows scanned—not just on the 10 rows retrieved.
The Read Stability Isolation Level

For Read Stability, all rows that are retrieved by a single transaction are locked for the duration of that transaction.

- When this isolation level is used, no row read by the isolating transaction can be changed by other transactions until the isolating transaction terminates
- SELECT statements that are issued more than once within the same transaction may not always yield the same results
  - Lost updates, dirty reads, and non-repeatable reads cannot occur
  - Phantoms, on the other hand, can and may be seen

Only the rows that are actually retrieved and/or modified by the isolating transaction are locked.

When the Read Stability isolation level is used, all rows that are retrieved by a single transaction are locked for the duration of that transaction. When this isolation level is used, no row read by the isolating transaction can be changed by other transactions until the isolating transaction terminates. In addition, changes made to other rows by other transactions are not seen by a transaction running under the Read Stability isolation level until they have been committed. Thus, when the Read Stability isolation level is used, SELECT statements that are issued more than once within the same transaction may not always yield the same results. (Lost updates, dirty reads, and non-repeatable reads cannot occur; phantoms, on the other hand, can and may be seen.)

Unlike the Repeatable Read isolation level where each row that is referenced by the isolating transaction is locked, when the Read Stability isolation level is used, only the rows that are actually retrieved and/or modified by the isolating transaction are locked. Thus, if a transaction scans 1000 rows but only retrieves 10, locks are only acquired and held on the 10 rows retrieved—not on all 1000 rows scanned.
The Cursor Stability Isolation Level

For Cursor Stability, each row referenced by a cursor that is being used by the isolating transaction is locked as long as the cursor is positioned on that row.

- The lock acquired remains in effect until either the cursor is repositioned or until the isolating transaction terminates
- SELECT statements that are issued more than once within the same transaction may not always yield the same results
- Lost updates and dirty reads cannot occur
  - Non-repeatable reads and phantoms, however, can and may be seen

By default, Cursor Stability is the isolation level used by most transactions.

When the Cursor Stability isolation level is used, each row referenced by a cursor that is being used by the isolating transaction is locked, as long as the cursor is positioned on that row. The lock acquired remains in effect until either the cursor is repositioned (usually by calling the FETCH statement) or until the isolating transaction terminates. Thus, when this isolation level is used, SELECT statements that are issued more than once within the same transaction may not always yield the same results. (Lost updates and dirty reads cannot occur; non-repeatable reads and phantoms, however, can and may be seen.)

When a transaction using the Cursor Stability isolation level retrieves a row from a table via an updatable cursor, no other transaction can update or delete that row while the cursor is positioned on it. However, other transactions can add new rows to the table and perform update and/or delete operations on rows positioned on either side of the locked row, provided the locked row itself was not accessed using an index. Furthermore, if the isolating transaction modifies any row it retrieves, no other transaction can update or delete that row until the isolating transaction is terminated, even after the cursor is no longer positioned on the modified row.

Transactions using the Cursor Stability isolation level will not see changes made to other rows by other transactions until those changes have been committed. By default, Cursor Stability is the isolation level used by most transactions.
When the Uncommitted Read isolation level is used, rows that are retrieved by a single transaction are only locked for the duration of that transaction. Because rows often remain unlocked when this isolation level is used, lost updates, dirty reads, non-repeatable reads, and phantoms can occur. Such transactions can neither see nor access tables, views, or indexes that have been created by other transactions until they are committed.

This isolation level is commonly used for transactions that access read-only tables and/or transactions that execute SELECT statements for which uncommitted data from other transactions have no adverse affect.

When the Uncommitted Read isolation level is used, rows that are retrieved by a single transaction are only locked for the duration of that transaction. Because rows often remain unlocked when this isolation level is used, lost updates, dirty reads, non-repeatable reads, and phantoms can occur.

In most cases, transactions using the Uncommitted Read isolation level can see changes made to rows by other transactions before those changes are committed or rolled back. However, such transactions can neither see nor access tables, views, or indexes that have been created by other transactions until those transactions have been committed. Likewise, transactions using the Uncommitted Read isolation level only learn that an existing table, view, or index has been dropped by another transaction when that transaction terminates. There is one exception to this behavior: when a transaction running under the Uncommitted Read isolation level uses a cursor for update, that transaction behaves as if it is running under the Cursor Stability isolation level, and the constraints of the Cursor Stability isolation level are applied.

The Uncommitted Read isolation level is commonly used for transactions that access read-only tables and/or transactions that execute SELECT statements of which uncommitted data from other transactions will have no adverse affect.
Specifying the Isolation Level

Isolation levels are specified at the application level.

- For embedded SQL applications, the isolation level to be used is specified at precompile time or when the application is bound to a database.

When no isolation level is specified, the Cursor Stability isolation level is used by default.

Isolation level used by commands and scripts run from the Command Line Processor can also be set by executing the CHANGE ISOLATION command.

Although isolation levels control how resources are locked for transactions, they are actually specified at the application level. For embedded SQL applications, the isolation level to be used is specified at precompile time or when the application is bound to a database. In most cases, the isolation level for applications written in a supported compiled language (such as C and C++) is set via the ISOLATION option of the PRECOMPILE PROGRAM and BIND commands/APIs. For Call Level Interface (CLI) applications, the isolation level to be used is set at application run time by calling the SQLSetConnectAttr() function with the SQL_ATTR_TXN_ISOLATION connection attribute specified. Isolation levels for CLI applications can also be set by assigning a value to the TXNISOLATION keyword, which can be found in the db2cli.ini configuration file. With JDBC and SQLJ applications, the isolation level is set at application run time by calling the setTransactionIsolation() method that resides within the java.sql connection interface.

When no isolation level is specified, the Cursor Stability isolation level is used by default. This is true for commands and scripts that are executed from the Command Line Processor as well as for embedded SQL, CLI, JDBC, and SQLJ applications. Thus, the isolation level used by commands and scripts run from the Command Line Processor can also be specified; in this case, the isolation level to be used is set by executing the CHANGE ISOLATION command.
Choosing the Proper Isolation Level

When more restrictive isolation levels are used, less concurrency support is provided and overall performance may be decreased because more resources are required.

- **Uncommitted Read**: Read-only transactions needed—high data stability not required
- **Cursor Stability**: Read/write transactions needed—high data stability not required
- **Read Stability**: Read-only or read/write transactions needed—high data stability required
- **Repeatable Read**: Read-only transactions needed—extremely high data stability required

Choosing the appropriate isolation level to use for a transaction is very important. The isolation level not only influences how well the database supports concurrency, but it also affects the overall performance of the application containing the transaction. That's because the resources needed to acquire and free locks vary with each isolation level.

Generally, when more restrictive isolation levels are used, less concurrency support is provided and overall performance may be decreased because more resources are required. However, when deciding on the best isolation level to use, which phenomena are acceptable and which phenomena are not should be the deciding factor. The following heuristic can be used to help you decide which isolation level to use for a particular situation:

- If you are executing queries on read-only databases or if you are executing queries and do not care if uncommitted data values are returned, use the Uncommitted Read isolation level. (Read-only transactions needed—high data stability not required.)
- If you want maximum concurrency without seeing uncommitted data values, use the Cursor Stability isolation level. (Read/write transactions needed—high data stability not required.)
- If you want concurrency and you want qualified rows to remain stable for the duration of an individual transaction, use the Read Stability isolation level. (Read-only or read/write transactions needed—high data stability required.)
If you are executing queries and do not want to see changes made to the result data sets produced, use the Repeatable Read isolation level. (Read-only transactions needed—extremely high data stability required.)
In the pages on Concurrency and isolation levels, we saw that DB2 Universal Database isolates transactions from each other through the use of locks. A lock is a mechanism that is used to associate a data resource with a single transaction, with the purpose of controlling how other transactions interact with that resource while it is associated with the owning transaction. (The transaction that a locked resource is associated with is said to "hold" or "own" the lock.) The DB2 Database Manager uses locks to prohibit transactions from accessing uncommitted data written by other transactions (unless the Uncommitted Read isolation level is used) and to prohibit the updating of rows by other transactions when the owning transaction is using a restrictive isolation level. Once a lock is acquired, it is held until the owning transaction is terminated—at that point, the lock is released and the data resource is made available to other transactions.

If one transaction attempts to access a data resource in a way that is incompatible with the lock being held by another transaction (we'll look at lock compatibility shortly), that transaction must wait until the owning transaction has ended. This is known as a lock wait. When a lock wait event occurs, the transaction attempting to access the data resource simply stops execution until the owning transaction has terminated and the incompatible lock is released.
## Lock Attributes

All locks have the following basic attributes:

- **Object** — identifies the data resource that is being locked
- **Size** — specifies the physical size of the portion of the data resource that is being locked
- **Duration** — specifies the length of time a lock is held
- **Mode** — specifies the type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked data resource

---

All locks have the following basic attributes:

- **Object**: The object attribute identifies the data resource that is being locked. The DB2 Database Manager acquires locks on data resources, such as table spaces, tables, and rows, whenever they are needed.

- **Size**: The size attribute specifies the physical size of the portion of the data resource that is being locked. A lock does not always have to control an entire data resource. For example, rather than giving an application exclusive control over an entire table, the DB2 Database Manager can give an application exclusive control over a specific row in a table.

- **Duration**: The duration attribute specifies the length of time a lock is held. A transaction's isolation level usually controls the duration of a lock.

- **Mode**: The mode attribute specifies the type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked data resource. This attribute is commonly referred to as the *lock state*.
The state of a lock determines the type of access:
- Allowed for the lock owner
- Permitted for concurrent users of a locked data resource

States (mode) of locks:

<table>
<thead>
<tr>
<th>Lock State (Mode)</th>
<th>Applicable Objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent None (IN)</td>
<td>Table spaces and tables</td>
<td>The lock owner can read data in the locked table, including uncommitted data, but cannot change this data. In this mode, the lock owner does not acquire row-level locks; therefore, other concurrent applications can read and change data in the table.</td>
</tr>
<tr>
<td>Intent Share (IS)</td>
<td>Table spaces and tables</td>
<td>The lock owner can read data in the locked table, but cannot change this data. Again, because the lock owner does not acquire row-level locks, other concurrent applications can both read and change data in the table. (When a transaction owns an Intent Share lock on a table, it acquires a Share lock on each row it reads.) This lock is acquired when a transaction does not convey the intent to update rows in the table.</td>
</tr>
</tbody>
</table>
Lock State (Mode): Next Key Share (NS)
Applicable Objects: Rows
Description: The lock owner and all concurrent transactions can read, but cannot change data in the locked row. This lock is acquired in place of a Share lock on data that is read using the Read Stability or Cursor Stability transaction isolation level.

Lock State (Mode): Share (S)
Applicable Objects: Tables and rows
Description: The lock owner and any other concurrent transactions can read, but cannot change data in the locked table or row. As long as a table is not Share locked, individual rows in that table can be Share locked. If, however, a table is Share locked, row-level Share locks in that table cannot be acquired by the lock owner. If either a table or a row is Share locked, other concurrent transactions can read the data, but they cannot change it.

Lock State (Mode): Intent Exclusive (IX)
Applicable Objects: Table spaces and tables
Description: The lock owner and any other concurrent applications can read and change data in the locked table. When the lock owner reads data from the table, it acquires a Share lock on each row it reads, and it acquires both an Update and an Exclusive lock on each row it updates. Other concurrent applications can both read and update the locked table. This lock is acquired when a transaction conveys the intent to update rows in the table. (The SELECT FOR UPDATE, UPDATE ... WHERE, and INSERT statements convey the intent to update.)

Lock State (Mode): Share With Intent Exclusive (SIX)
Applicable Objects: Tables
Description: The lock owner can both read and change data in the locked table. The lock owner acquires Exclusive locks on the rows it updates but does not acquire locks on rows that it reads; therefore, other concurrent applications can read but cannot update the data in the locked table.

Lock State (Mode): Update (U)
Applicable Objects: Tables and rows
Description: The lock owner can update data in the locked table and the lock owner automatically acquires Exclusive locks on any rows it updates. Other concurrent applications can read but cannot update the data in the locked table.

Lock State (Mode): Next Key Exclusive (NX)
Applicable Objects: Rows
Description: The lock owner can read but cannot change the locked row. This lock is acquired on the next row in a table when a row is deleted from or inserted into the index for that table.
Lock State (Mode): Next Key Weak Exclusive (NW)
Applicable Objects: Rows
Description: The lock owner can read but cannot change the locked row. This lock is acquired on the next row in a table when a row is inserted into the index of a non-catalog table.

Lock State (Mode): Exclusive (X)
Applicable Objects: Tables and rows
Description: The lock owner can both read and change data in the locked table or row. If an Exclusive lock is acquired, only applications using the Uncommitted Read isolation levels are allowed to access the locked table or row(s). Exclusive locks are acquired for data resources that are going to be manipulated with the INSERT, UPDATE, and/or DELETE statements.

Lock State (Mode): Weak Exclusive (WE)
Applicable Objects: Rows
Description: The lock owner can read and change the locked row. This lock is acquired on a row when it is inserted into a non-catalog table.

Lock State (Mode): Super Exclusive (Z)
Applicable Objects: Table spaces and tables
Description: The lock owner can alter a table, drop a table, create an index, or drop an index. This lock is automatically acquired on a table, whenever a transaction attempts to perform any one of these operations. No other concurrent transactions are allowed to read or update the table until this lock is removed.
## Lock Compatibility

Locks are compatible if the state of one lock placed on a data resource enables another lock to be placed on the same resource.

- Application A holds a lock on a table that application B also wants to access
- Application B requests a lock of some particular mode

If the mode of the lock held by A permits the lock requested by B, the two locks (or modes) are said to be compatible.

If the lock mode requested for application B is not compatible with the lock held by application A, application B cannot continue.

- It must wait until application A releases its lock, and all other existing incompatible locks are released

If the state of one lock placed on a data resource enables another lock to be placed on the same resource, the two locks (or states) are said to be compatible. Whenever one transaction holds a lock on a data resource and a second transaction requests a lock on the same resource, the DB2 Database Manager examines the two lock states to determine whether or not they are compatible. If the locks are compatible, the lock is granted to the second transaction (provided no other transaction is waiting for the data resource). If however, the locks are incompatible, the second transaction must wait until the first transaction releases its lock (in fact, the second transaction must wait until all existing incompatible locks are released) before it can gain access to the resource and continue processing. Refer to the *IBM DB2 Universal Database Administration Guide: Performance* documentation for specific information on which locks are compatible and which are not.
## State of Held Resource

<table>
<thead>
<tr>
<th>State Being Requested</th>
<th>none</th>
<th>IN</th>
<th>IS</th>
<th>NS</th>
<th>S</th>
<th>IX</th>
<th>SIX</th>
<th>U</th>
<th>NX</th>
<th>X</th>
<th>Z</th>
<th>NW</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>IN</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>IS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>NS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>S</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>IX</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>SIX</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>U</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>NX</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>X</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Z</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>NW</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>W</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Where:

- I Intent
- N None
- NS Next Key Share
- S Share
- NX Next Key Exclusive
- X Exclusive
- U Update
- Z Super Exclusive
- NW Next Key Weak Exclusive
- W Weak Exclusive

- **yes** - **grant** lock requested immediately
- **no** - **wait** for held lock to be released or timeout to occur

---

7-24 Database Concurrency
### Lock Conversion

**Lock conversion:**
- Is the operation of changing the state of a lock already held to a more restrictive state
- Occurs because a transaction can hold only one lock on a data resource at a time
- Is mostly performed for row-level locks
- Only occurs if a held lock can increase its restriction

Once a lock's state has been converted, the lock stays at the highest state obtained until the transaction holding the lock is terminated.

When a transaction attempts to access a data resource it already holds a lock on, and the mode of access needed requires a more restrictive lock than the one already held, the state of the lock held is changed to the more restrictive state. The operation of changing the state of a lock already held to a more restrictive state is known as lock conversion. Lock conversion occurs because a transaction can hold only one lock on a data resource at a time.

In most cases, lock conversion is performed for row-level locks and the conversion process is pretty straightforward. For example, if a Share (S) or an Update (U) row-level lock is held and an Exclusive (X) lock is needed, the held lock is converted to an Exclusive (X) lock. Intent Exclusive (IX) locks and Share (S) locks are special cases, however, since neither is considered to be more restrictive than the other. Thus, if one of these row-level locks is held and the other is requested, the held lock is converted to a Share with Intent Exclusive (SIX) lock. Similar conversions result in the requested lock state becoming the new lock state of the held lock, provided the requested lock state is more restrictive. (Lock conversion only occurs if a held lock can increase its restriction.) Once a lock's state has been converted, the lock stays at the highest state obtained until the transaction holding the lock is terminated.
Lock Escalation

Lock escalation is performed automatically whenever too many locks (of any type) have been acquired.

- Lock escalation is the conversion of several individual row-level locks within the same table to a single table-level lock

When a lock is requested and the lock storage space is full:

- One of the tables associated with the transaction is selected
- A table-level lock is acquired
- All row-level locks for that table are released
- The table-level lock is added to the lock list

If lock space is still unavailable, the transaction is asked to either COMMIT or ROLLBACK and the transaction is terminated.

All locks require space for storage and, because the space available is not infinite, the DB2 Database Manager must limit the amount of space that can be used for locks (this is done through the MAXLOCKS database configuration parameter). In order to prevent a specific database agent from exceeding the lock space limitations established, a process known as lock escalation is performed automatically whenever too many locks (of any type) have been acquired. Lock escalation is the conversion of several individual row-level locks within the same table to a single table-level lock. Since lock escalation is handled internally, the only externally detectable result might be a reduction in concurrent access on one or more tables.

Here's how lock escalation works: When a transaction requests a lock and the lock storage space is full, one of the tables associated with the transaction is selected, a table-level lock is acquired, all row-level locks for that table are released (to create space in the lock list data structure), and the table-level lock is added to the lock list. If this process does not free up enough space, another table is selected and the process is repeated until enough free space is available. At that point, the requested lock is acquired and the transaction resumes execution. However, if the necessary lock space is still unavailable after all the transaction's row-level locks have been escalated, the transaction is asked to either COMMIT or ROLLBACK all changes that have been made since it's initiation (that is, an SQL error code is generated) and the transaction is terminated.
A deadlock can occur when there is contention for locks by two or more transactions. 

Contention for locks by two or more transactions can sometimes result in a situation known as a deadlock. The best way to illustrate how a deadlock can occur is by example: Suppose Transaction 1 acquires an Exclusive (X) lock on Table A and Transaction 2 acquires an Exclusive (X) lock on Table B. Now, suppose Transaction 1 attempts to acquire an Exclusive (X) lock on Table B and Transaction 2 attempts to acquire an Exclusive (X) lock on Table A. Processing by both transactions are suspended until their second lock request is granted. However, because neither lock request can be granted until one of the transactions releases the lock it currently holds (by performing a COMMIT or ROLLBACK operation), and because neither transaction can release the lock it currently holds (because both are suspended and waiting on locks), a deadlock situation has occurred.

When a deadlock situation occurs, all transactions involved will wait indefinitely for a lock to be released, unless some outside agent takes action. DB2 Universal Database's approach to handling deadlocks is an asynchronous system background process, known as the deadlock detector. The sole responsibility of the deadlock detector is to locate and resolve any deadlocks found in the locking subsystem. The deadlock detector stays "asleep" most of the time, but "wakes up" at preset intervals to determine whether or not a deadlock situation exists. If the deadlock detector discovers a deadlock in the locking subsystem, it selects, terminates, and rolls back one of the transactions involved. (The transaction that is terminated and rolled back receives an SQL error code and all locks it had acquired are released.) Usually, the remaining transaction(s) can then proceed.
Any time a transaction holds a lock on a particular data resource (for example a table or row), other transactions may be denied access to that resource until the owning transaction terminates and frees all locks it has acquired. Without some sort of lock timeout detection mechanism in place, a transaction might wait indefinitely for a lock to be released. Such a situation might occur, for example, when a transaction is waiting for a lock that is held by another user's application to be released, and the other user has left their workstation without performing some interaction that would allow their application to terminate the owning transaction. Obviously, these types of situations can cause poor application performance. To avoid stalling other applications when these types of situations occur, a lock timeout value can be specified in a database's configuration file (via the LOCKTIMEOUT database configuration parameter). When used, this value controls the amount of time any transaction waits to obtain a requested lock. If the desired lock is not acquired before the time interval specified elapses, the waiting application receives an error and the transaction requesting the lock is rolled back. By using lock timeouts, global deadlock situations can be avoided, especially in distributed transaction application environments.
The above illustration shows the logic that is used to determine which type of lock to acquire for a referenced object.

In most cases, the DB2 Database Manager implicitly acquires locks as they are needed, and these locks remain under the DB2 Database Manager's control. Except in situations where the Uncommitted Read isolation level is used, it is never necessary for a transaction to explicitly request a lock. In fact, the only database object that can be explicitly locked by a transaction is a table object.

The DB2 Database Manager always attempts to acquire row-level locks. However, this behavior can be modified by executing a special form of the ALTER TABLE statement. The syntax for this form of the ALTER TABLE statement is:

```
ALTER TABLE table_name LOCKSIZE TABLE
```

where:

table_name — Identifies the name of an existing table that all transactions are to acquire table-level locks for when accessing it.
The DB2 Database Manager can also be forced to acquire a table-level lock on a table for a specific transaction by executing the LOCK TABLE statement. The syntax for this statement is:

```sql
LOCK TABLE table_name IN SHARE MODE
```

or

```sql
LOCK TABLE table_name IN EXCLUSIVE MODE
```

where:

- `table_name` — Identifies the name of an existing table that a table-level lock is to be acquired for (provided no other transaction has an incompatible lock on this table).

If this statement is executed with the SHARE mode specified, a table-level lock that allows other transactions to read (but not change the data stored in it) is acquired; if executed with the EXCLUSIVE mode specified, a table-level lock that does not allow other transactions to read or modify data stored in the table is acquired.

To find the locking granularity for tables, execute this SQL statement:

```sql
db2 "SELECT tabname, locksize
    FROM syscat.tables
    WHERE tabschema NOT LIKE 'SYS&'
"
```

Values for LOCKSIZE are R for row, T for table, or blank if not applicable.
It was mentioned earlier that any time a transaction holds a lock on a particular data resource, other transactions may be denied access to that resource until the owning transaction terminates. Therefore, to optimize for maximum concurrency, row-level locks are usually better than table-level locks because they limit access to a much smaller resource.

- A single table-level lock requires less overhead than several individual row-level locks
- Row-level locks are acquired by default

Global approach:
- Use the ALTER TABLE ... LOCKSIZE TABLE statement to acquire table-level locks for a particular table

Individual transaction approach:
- Use the LOCK TABLE statement to allow table-level locks to be acquired at an individual transaction level

The granularity of locks (that is, whether row-level locks or table-level locks are acquired) can be controlled through the use of the ALTER TABLE ... LOCKSIZE TABLE, ALTER TABLE ... LOCKSIZE ROW, and LOCK TABLE statements. The ALTER TABLE ... LOCKSIZE TABLE statement provides a global approach to granularity that results in table-level locks being acquired by all transactions that access rows within a particular table. The LOCK TABLE statement on the other hand, allows table-level locks to be acquired at an individual transaction level. When either of these statements are used, a single Share (S) or Exclusive (X) table-level lock is acquired whenever a lock is needed. As a result, overall performance is usually improved since only one table-level lock must be acquired and released instead of several different row-level locks.
**Transaction Processing**

For locking, all transactions typically fall under one of the following categories:

- **Read-Only** — use Intent Share (IS) and/or Share (S) locks
- **Intent-To-Change** — use Update (U), Intent Exclusive (IX) and Exclusive (X) locks for tables; Share (S), Update (U), and Exclusive (X) locks for rows
- **Change** — use Intent Exclusive (IX) and/or Exclusive (X) locks
- **Cursor-Controlled** — use Intent Exclusive (IX) and/or Exclusive (X) locks

From a locking standpoint, all transactions typically fall under one of the following categories:

- **Read-Only** — This refers to transactions that contain SELECT statements (which are intrinsically read-only), SELECT statements that have the FOR READ ONLY clause specified, or SQL statements that are ambiguous, but are presumed to be read-only because of the BLOCKING option specified as part of the precompile and/or bind process.
- **Intent-To-Change** — This refers to transactions that contain SELECT statements that have the FOR UPDATE clause specified, or SQL statements that are ambiguous, but are presumed to be intended for change because of the way they are interpreted by the SQL precompiler.
- **Change** — This refers to transactions that contain INSERT, UPDATE, and/or DELETE statements but not UPDATE ... WHERE CURRENT OF ... or DELETE ... WHERE CURRENT OF ... statements.
- **Cursor-Controlled** — This refers to transactions that contain UPDATE ... WHERE CURRENT OF ... and DELETE ... WHERE CURRENT OF ... statements.

Read-Only transactions typically use Intent Share (IS) and/or Share (S) locks. Intent-To-Change transactions, on the other hand, use Update (U), Intent Exclusive (IX) and Exclusive (X) locks for tables; Share (S), Update (U), and Exclusive (X) locks for rows. Change transactions tend to use Intent Exclusive (IX) and/or Exclusive (X) locks while Cursor Controlled transactions often use Intent Exclusive (IX) and/or Exclusive (X) locks.
When an SQL statement is precompiled, the DB2 Optimizer explores various ways to satisfy that statement's request and estimates the execution cost involved for each approach. Based on this evaluation, the DB2 Optimizer then selects what it believes to be the optimal access plan to use. (The access plan specifies the operations required and the order in which those operations are to be performed, to resolve an SQL request.) An access plan can use one of two ways to access data in a table: by directly reading the table or by reading an index on that table and then retrieving the row in the table referred by the index.

The access path chosen by the DB2 Optimizer can have a significant effect on the amount of locks acquired and the lock states used.

When an SQL statement is precompiled, the DB2 Optimizer explores various ways to satisfy that statement's request and estimates the execution cost involved for each approach. Based on this evaluation, the DB2 Optimizer then selects what it believes to be the optimal access plan to use. (The access plan specifies the operations required and the order in which those operations are to be performed, to resolve an SQL request.) An access plan can use one of two ways to access data in a table: by directly reading the table (which is known as performing a table or a relation scan), or by reading an index on that table (which is known as performing an index scan) and then retrieving the row in the table that a particular index entry refers to.

The access path chosen by the DB2 Optimizer, which is often determined by the database's design, can have a significant effect on the amount of locks acquired and the lock states used. For example, when an index scan is used to locate a specific row, the DB2 Database Manager will most likely acquire one or more Intent Share (IS) row-level locks. However, if a table scan is used, because the entire table must be scanned, in sequence, to locate a specific row, the DB2 Database Manager may opt to acquire a single Share (S) table-level lock.
You should now be able to:

- Identify factors that influence locking
- List the objects that locks can be obtained on
- Identify scope of different types of DB2 locks
- Identify factors affecting amount of locks that are used
- Identify factors affecting the type of locks obtained
- Identify the isolation levels that should be used for a given a situation
Exercises
Exercise 1

This exercise is designed to test your knowledge of DB2 database concurrency in a multi-user environment.

1.1 What are the three support mechanisms used to enforce data consistency in a database?

__________
__________
__________

1.2 Given that an SQL statement starts a unit of work (transaction), what SQL statement can be used to permanently store the data changes and terminate the transaction?

__________

1.3 Given that an SQL statement starts a unit of work (transaction), what SQL statement can be used to return the data to its previously stored values and terminate the transaction?

__________

1.4 Which of the following Isolation Level ignores the locking condition on data objects?
   A. Repeatable Read
   B. Read Stability
   C. Cursor Stability
   D. Uncommitted Read

1.5 Which of the following Isolation Level acquires the most locks while querying data?
   A. Repeatable Read
   B. Read Stability
   C. Cursor Stability
   D. Uncommitted Read
1.6 Which of the following Isolation Level is the default?
   A. Repeatable Read
   B. Read Stability
   C. Cursor Stability
   D. Uncommitted Read

1.7 Which of the following Isolation Levels ensures that phantom rows cannot occur?
   A. Repeatable Read
   B. Read Stability
   C. Cursor Stability
   D. Uncommitted Read

1.8 If Isolation Levels are used to provide data concurrency during a data read operation, what is used to ensure data consistency during a write operation?
   ________

1.9 Of the four Lock attributes shown below, which one is used to specify the type of access allowed by the lock owner and others?
   A. Object
   B. Size
   C. Duration
   D. Mode

1.10 Of the twelve lock states provided, which three are used most?
   ________
   ________
   ________

1.11 What objects will the lock states answered in question 1.10 apply to?
   ________
   ________
1.12 What is the mechanism in DB2 used to protect the lock resource usage?

1.13 What is a deadlock?

1.14 What is used to break the deadlock?

1.15 What optional mechanism, based on a time factor, can be used to free a lock on a resource?

1.16 What is the default locking granularity? Can the lock granularity be changed?

1.17 What two methods could the DB2 Optimizer use to query data?

1.18 What might the consequences be for a full-table scan, and an isolation level of Repeatable Read?
In this exercise, you will explore data locking. You will need to open three Command windows.

2.1 Open a Command window and title it App1.
   Select Start > Programs > IBM DB2 > Command Window. Change to the C:\ directory. Title the window App1.

If you are using v8: Select Start > Programs > IBM DB2 > Command Line Tools > Command Window. Same for the next two actions.

2.2 Open a Command window and title it App2.
   Select Start > Programs > IBM DB2 > Command Window. Change to the C:\ directory. Title the window App2.

2.3 Open a Command window and title it Monitor.
   Select Start > Programs > IBM DB2 > Command Window. Change to the C:\ directory. Title the window Monitor.

2.4 In the Monitor window, use the DB2 command to "FORCE APPLICATIONS ALL". Then connect to the sample database.

2.5 In the Monitor window, examine the current setting of the database monitor switches.
   db2 "GET MONITOR SWITCHES"

What is the current setting of every switch?

2.6 From the App1 window, connect to the sample database and update a single row in the staff table, but DO NOT commit the change! Use the +c command option to prevent automatic commit. Type this command exactly as shown:
   db2 "CONNECT TO sample"
   db2 +c "UPDATE staff SET comm = comm + 500.0 WHERE id = 100"
2.7 From the Monitor window, list the agents associated with the three applications you currently have connected/attached to this instance. Record the results below. Your Handle numbers may be different.

Monitor Application Handle: ______
App1 Application Handle: ______

2.8 From your Monitor window, examine the database monitor elements related to locking by issuing the following command:

```
db2 "GET SNAPSHOT FOR LOCKS ON sample" > lock1out.txt
```

View the lock1out.txt file, found in the C:\ directory.

2.9 Are any locks currently held?

2.10 How many applications are currently connected?

2.11 What agents are currently holding locks?

2.12 Is there ever a case where a row lock is possessed without a corresponding lock on the table which contains the row?

2.13 What lock modes are shown for tables under which a row is locked?

2.14 How many different tables are being locked by the agent?

2.15 In the App2 window, connect to the sample database. Then issue a select statement against the row being updated in the App1 window.

2.16 What happened? Execute the GET SNAPSHOT command you issued earlier, in the Monitor window. What type of lock is the agent in the App2 window holding?

2.17 In the App1 window, issue a ROLLBACK statement. Did the application in the App2 window continue?
Solutions
This exercise is designed to test your knowledge of DB2 database concurrency in a multi-user environment.

1.1 What are the three support mechanisms used to enforce data consistency in a database?
   - Transactions
   - Locking
   - Logging

1.2 Given that an SQL statement starts a unit of work (transaction), what SQL statement can be used to permanently store the data changes and terminate the transaction?
   - COMMIT

1.3 Given that an SQL statement starts a unit of work (transaction), what SQL statement can be used to return the data to its previously stored values and terminate the transaction?
   - ROLLBACK

1.4 Which of the following Isolation Level ignores the locking condition on data objects?
   - A. Repeatable Read
   - B. Read Stability
   - C. Cursor Stability
   - D. Uncommitted Read

1.5 Which of the following Isolation Level acquires the most locks while querying data?
   - A. Repeatable Read
   - B. Read Stability
   - C. Cursor Stability
   - D. Uncommitted Read
1.6 Which of the following Isolation Level is the default?
   A. Repeatable Read
   B. Read Stability
   C. **Cursor Stability**
   D. Uncommitted Read

1.7 Which of the following Isolation Levels ensures that phantom rows cannot occur?
   A. **Repeatable Read**
   B. Read Stability
   C. Cursor Stability
   D. Uncommitted Read

1.8 If Isolation Levels are used to provide data concurrency during a data read operation, what is used to ensure data consistency during a write operation?
   **Locks**

1.9 Of the four Lock attributes shown below, which one is used to specify the type of access allowed by the lock owner and others?
   A. Object
   B. Size
   C. Duration
   D. **Mode**

1.10 Of the twelve lock states provided, which three are used most?
   **Share (S)**
   **Update (U)**
   **Exclusive (X)**

1.11 What objects will the lock states answered in question 1.10 apply to?
   **Tables**
   **Rows**
1.12 What is the mechanism in DB2 used to protect the lock resource usage?
   Lock escalation

1.13 What is a deadlock?
   A deadlock can occur if two transactions are trying to access the same object, and neither transaction can terminate normally because of lock contention.

1.14 What is used to break the deadlock?
   The DB2 Deadlock Detector checks for deadlocks periodically, and resolves the deadlock by terminating and rolling back one of the transactions.

1.15 What optional mechanism, based on a time factor, can be used to free a lock on a resource?
   The LOCKTIMEOUT database configuration parameter can be set to terminate and ROLLBACK a transaction that has had a resource locked for too long.

1.16 What is the default locking granularity? Can the lock granularity be changed?
   The default locking granularity is at the row level.
   Yes. Use the ALTER TABLE ... LOCKSIZE TABLE to acquire locks at the table level for all transactions using that table. Use the LOCK TABLE SQL statement to provide table-level locking for just the current transaction.

1.17 What two methods could the DB2 Optimizer use to query data?
   The Optimizer can create an access plan that either:
   directly reads data from the table in a full-table scan
   reads the associated index entries first, then reads just the data rows indicated by the index.

1.18 What might the consequences be for a full-table scan, and an isolation level of Repeatable Read?
   A share (S) lock would be acquired and retained for each row read in the scan. This could potentially exhaust the lock list resource.
In this exercise, you will explore data locking. You will need to open three Command windows.

2.1 Open a Command window and title it **App1**.
   Select **Start > Programs > IBM DB2 > Command Window**. Change to the C:\ directory. Title the window **App1**.
   ```
   cd \title App1
   ```

2.2 Open a Command window and title it **App2**.
   Select **Start > Programs > IBM DB2 > Command Window**. Change to the C:\ directory. Title the window **App2**.
   ```
   cd \title App2
   ```

2.3 Open a Command window and title it **Monitor**.
   Select **Start > Programs > IBM DB2 > Command Window**. Change to the C:\ directory. Title the window **Monitor**.
   ```
   cd \title Monitor
   ```

2.4 In the **Monitor** window, use the DB2 command to "FORCE APPLICATIONS ALL". Then connect to the **sample** database.
   ```
   db2 "FORCE APPLICATIONS ALL"
   db2 "CONNECT TO sample"
   ```

2.5 In the **Monitor** window, examine the current setting of the database monitor switches.
   ```
   db2 "GET MONITOR SWITCHES"
   ```
   What is the current setting of every switch?
   **The monitor switches should all be off.**
2.6 From the **App1** window, connect to the **sample** database and update a single row in the **staff** table, but DO NOT commit the change! Use the +c command option to prevent automatic commit. Type this command exactly as shown:

```
  db2 "CONNECT TO sample"
  db2 +c "UPDATE staff SET comm = comm + 500.00 WHERE id = 100"
```

2.7 From the **Monitor** window, list the agents associated with the three applications you currently have connected/attached to this instance. Record the results below. Your Handle numbers may be different.

```
  db2 "LIST APPLICATIONS SHOW DETAIL"
  Monitor Application Handle: 1
  App1 Application Handle: 2
```

2.8 From your **Monitor** window, examine the database monitor elements related to locking by issuing the following command:

```
  db2 "GET SNAPSHOT FOR LOCKS ON sample" > lock1out.txt
```

View the **lock1out.txt** file, found in the C:\ directory.

2.9 Are any locks currently held?

**Yes. 3 locks are held.**

The first part of the lock snapshot provides the total number of locks held. The number should be greater than zero since **App1** has an outstanding unit of work in progress. The number of locks held will be variable, but general statements concerning lock volume can be made using this parameter.

**Important!**

You may get a different number on the number of locks. The discrepancy may depend on how you got to this exercise and whether other processes are open against the database. Occasionally you may find differences if you are using different release or fixpak levels. As usual, you should consider the solutions in this book as representative rather than definitive.

2.10 How many applications are currently connected?

**There should be two applications connected. One is the Monitor, and has no locks held. The other is App1, which has locks held based on the update SQL statement. Remember that agent association will remain until the DB2 TERMINATE command is issued in that window.**
2.11 What agents are currently holding locks?

The only agent holding locks is the one associated with App1. (Use the Application Handle from the LIST APPLICATIONS command to help determine association to a window). The lock snapshot consists of a general information block and detailed information blocks for each connected application. Only the detailed information block for the App1 application shows held locks.

2.12 Is there ever a case where a row lock is possessed without a corresponding lock on the table which contains the row?

No. Row locks are never obtained unless a supporting table lock is possessed first.

2.13 What lock modes are shown for tables under which a row is locked?

The lock modes of IX and IS are used to support the underlying row locks obtained. In other situations, a SIX lock may be used as well.

2.14 How many different tables are being locked by the agent?

The staff table is being locked.

Note

You may also see an Internal P lock. This is a package lock. Any execution of SQL, whether static or dynamic, must be done while holding a package lock. This ensures that no one can drop the package that is being executed.

2.15 In the App2 window, connect to the sample database. Then issue a select statement against the row being updated in the App1 window.

```
db2 "CONNECT TO sample"
db2 "SELECT * FROM staff WHERE id = 100"
```

2.16 What happened? Execute the GET SNAPSHOT command you issued earlier, in the Monitor window. What type of lock is the agent in the App2 window holding?

The App2 application cannot query the row because it is in a lock-wait status, waiting for the X lock (held by the agent in the App1 window) on that row to be released.
2.17 In the App1 window, issue a ROLLBACK statement. Did the application in the App2 window continue?

The App2 application continued and retrieved the requested row in the staff table. The ROLLBACK statement released the exclusive lock caused by the lock-wait condition.

Note

It is possible that the operation in the App1 window already rolled back due to a deadlock detection or timeout occurring. If this occurs, re-run the statement in the App1 window, and immediately execute the SELECT statement in the App2 window, then immediately issue a ROLLBACK in the App1 window.
Module 8

Course Summary
This section provides you with a summary of knowledge gained in this course and resources you can use to further your education on DB2 UDB subjects. Included are document sources and other courses you can attend.

**Course Summary**

You should now have a basic understanding of:

- Planning and implementing an installation
- Implementing instances and their security
- Creating and Accessing DB2 databases
- Using SQL statements
- Various database objects
- Database concurrency
Where to Go From Here

Two possible choices for your next step:

- *Fast Path to DB2 UDB for Experienced Relational DBAs*
- *DB2 Universal Database Administration Workshop*

There is an excellent tutorial available for Exam #700 (Family Fundamentals, v8.1) at:

www7b.software.ibm.com/dmdd/library/tutorials/db2cert/db2cert_V8_tut.html

A CBT self-study course, *Fast Path to DB2 UDB for Experienced Relational DBAs* (CT28), contains a superb explanation of privileges and is available for download, free of charge, at:

www.ibm.com/software/data/db2/selfstudy

You will be required to register for this free download copy.

A classroom course, *DB2 Universal Database Administration Workshop*, is available for the following operating systems:

- Linux (CF20)
- UNIX (CF21)
- Windows NT (CF23)
- Solaris (CF27)

There are also a variety of advanced courses described in the next few pages.
The following courses are available to you. Most of these are classroom courses, but there are several CBT courses included in the list.

**Fast Path to DB2 UDB for Experienced Relational DBAs**

What you are taught:

- List and describe the components of a DB2 UDB
- Implement DB2 UDB security
- Perform basic administration of a DB2 UDB database system using commands, or the graphical user interface (GUI)
- Perform the tasks necessary to support a basic recovery strategy
DB2 UDB Fundamentals

What you are taught:

- List and describe the components of DB2 UDB
- Create a DB2 database
- Create objects

DB2 SQL Workshop

What you are taught:

- Code simple and complex SELECT statements
- Code INSERT, DELETE, and UPDATE statements

DB2 UDB Administration Workshop

What you are taught:

- Administer a DB2 UDB database system using commands and Graphical User Interface (GUI) tools
- Implement DB2 UDB security
- Manage System Managed Storage (SMS) and Database Managed Storage (DMS) table spaces within a database and apply data placement principles
- Define a DB2 UDB recovery strategy and perform the tasks necessary to support the strategy
- Physically implement a given logical database design using DB2 UDB support integrity and concurrency requirements
- List and describe the components of DB2 UDB
- Describe the application development process with respect to DB2 UDB considerations

DB2 UDB for UNIX, Windows, and OS/2 Database Admin Certification Preparation

What you are taught:

- Learn the knowledge necessary to enhance the probability of passing the certification tests
These courses are considered advanced and should be taken only after mastering the basic courses outlined on the previous pages.

**DB2 Advanced SQL Workshop**

What you are taught:

- Use advanced SQL constructs, such as recursive SQL, case expressions, check constraints, and triggers
- Discuss basic relational database concepts, such as referential integrity, tables, and indexes
- Create tables and indexes
- Use outer joins
- Use complex subqueries
- Use the major scalar functions
- Use views

Also available as CBT on CD-ROM

**Description of DB2 UDB Advanced Courses**

- DB2 Advanced SQL Workshop
  - Also available as CBT on CD-ROM
- DB2 UDB Performance Tuning and Monitoring Workshop
- DB2 UDB Advanced Administration Workshop
- DB2 UDB Advanced Recovery and High Availability Workshop
- DB2 Stored Procedures Programming Workshop
  - Also available as CBT on CD-ROM
DB2 UDB Performance Tuning and Monitoring Workshop

What you are taught:

- Define the impact of database design (tables, indexes, and data placement) on database performance
- Describe database application programming considerations and how they affect performance
- Identify and describe the parameters (database and non-database) that affect performance
- Tune parameters to achieve optimum performance
- Identify and use the tools that assist in monitoring and tuning of a database

DB2 UDB Advanced Administration Workshop

What you are taught:

- Effectively apply advanced techniques to administer a DB2 UDB using the control center
- Explore parallelism and SMP enablement
- Explore multiple bufferpool and extended storage support
- Explore client administration
- Perform a command line interface (CLI) trace
- Explore problem reporting and management
- Explore the stored procedure builder
- Configure the DB2 Governor to enforce time and central processing unit (CPU) restrictions
- Access data stored in a non-relational format using table functions
- Manage a distributed data environment
- Administer DB2 UDB from a remote client
- Explore federated databases
DB2 UDB Advanced Recovery and High Availability Workshop

What you are taught:

- Explore the DB2 UDB recovery facilities and database configuration options
- Plan the implementation of a user exit for archival of database logs
- Recover a DB2 table following a DROP TABLE command issued in error
- Plan and execute the recovery of table spaces to a selected point in time
- Effectively utilize incremental backup and restore to reduce the size and duration of DB2 database backups
- Gain a better understanding of DB2 UDB crash recovery facilities
- Utilize the redirected restore option to recover DB2 data to alternate disk configurations
- Execute recovery scenarios, including loss of DB2 log data or access to the DB2 catalog information
- Utilize the information in the DB2 recovery history file to plan and execute various DB2 utilities
- Explore the options for operation of DB2 databases in high availability environments including the use of split mirrors of the database
- Utilize the DB2DART utility to examine a DB2 database for problem determination
- Gain a better understanding of the unique recovery planning requirements for DB2 UDB Enterprise Extended Edition (EEE) databases

DB2 Stored Procedures Programming Workshop

What you are taught:

- Describe a stored procedure and justify its use in an application
- Understand the Stored Procedure Builder and its capabilities
- Describe the DB2 SQL Procedure Language (SQL PL) statements and how to use them in an application
- Describe the basic structure of a Java application using stored procedures
- Create a DB2 stored procedure using the SQL DDL statement CREATE PROCEDURE
- Describe troubleshooting approaches for stored procedures
<table>
<thead>
<tr>
<th>To Enroll in Courses</th>
</tr>
</thead>
</table>

There are two ways to enroll in courses:

- Call phone number **1-800-IBM-TEACH** (1-800-426-8322)
- Go to the following Web site:

<table>
<thead>
<tr>
<th>DB2 UDB Technical Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 UDB Administration Guide: Planning</td>
</tr>
<tr>
<td>DB2 UDB Administration Guide: Implementation</td>
</tr>
<tr>
<td>DB2 UDB Administration Guide: Performance</td>
</tr>
<tr>
<td>DB2 UDB Command Reference</td>
</tr>
<tr>
<td>DB2 UDB SQL Reference</td>
</tr>
</tbody>
</table>

The list of technical documents shown above are the basic references needed to properly maintain and administer the DB2 UDB database. These documents are provided on CD-ROM with the product, but hardcopy can be ordered through your IBM sales representative.
Additional Technical Documents

For general overview:

- *DB2 UDB Quick Beginnings*

For certification:

- *DB2 UDB v7.1 Database Administration Certification Guide*

For advanced study:

- *DB2 UDB System Monitor Guide and Reference*
- *DB2 UDB Data Movement Utilities Guide and Reference*
- *DB2 UDB Troubleshooting Guide*
Evaluation Sheet

- Kindly provide us with your feedback.
- Please include written comments, which are better than checked boxes.

Thank You!
Appendixes
Appendix A

Additional Reference Information
Additional documentation for your study includes the documents listed below. Also, there are many other documents (white papers, books, web pages) that are available from IBM and other third-party vendors.

For More Information


IBM offers many courses for your information needs. Check this web site for more information on these in-depth courses:

and
http://www.ibm.com/software/data/db2/selfstudy

Related Classes

CBT self study course, Fast Path to DB2 UDB for Experienced Relational DBAs (CT28)

DB2 Universal Database Administration Workshop for UNIX (CF211)

DB2 UDB Advanced Admin Workshop (CF45)
Appendix B

Example Configuration Parameters
The following configuration parameters are from a Linux system. Although a Windows system has the same types of parameters, the parameter names on a Windows system may differ slightly.

Admin Server Configuration

The following is a list of the administration configuration parameters (DBM) for a Linux installation (v8).

```
Admin Server Configuration

Authentication Type DAS (AUTHENTICATION) = SERVER_ENCRYPT

DAS Administration Authority Group Name (DASADM_GROUP) = db2das

DAS Discovery Mode (DISCOVER) = SEARCH
Name of the DB2 Server System (DB2SYSTEM) = FLORIDA

Java Development Kit Installation Path DAS (JDK_PATH) = /opt/IBMJava2-131
Java Development Kit Installation Path DAS (JDK_64_PATH) =

DAS Code Page (DAS_CODEPAGE) = 0
DAS Territory (DAS_TERRITORY) = 0

Location of Contact List (CONTACT_HOST) =
Execute Expired Tasks (EXEC_EXP_TASK) = NO
Scheduler Mode (SCHED_ENABLE) = ON
SMTP Server (SMTP_SERVER) = florida
Tools Catalog Database (TOOLSCAT_DB) = TOOLSDB
Tools Catalog Database Instance (TOOLSCAT_INST) = db2101
Tools Catalog Database Schema (TOOLSCAT_SCHEMA) = SYSTOOLS
Scheduler User ID =
```

Instance Configuration Parameters

The following is a list of the instance configuration parameters (DBM) for a Linux installation (v8).

```
Database Manager Configuration

Node type = Enterprise Server Edition with local and remote clients

Database manager configuration release level = 0x0a00

CPU speed (millisec/instruction) (CPUSPEED) = 2.408956e-06
Communications bandwidth (MB/sec) (COMM_BANDWIDTH) = 1.000000e+00

Max number of concurrently active databases (NUMDB) = 8
Data Links support (DATALINKS) = NO
Federated Database System Support (FEDERATED) = NO
Transaction processor monitor name (TP_MON_NAME) =
```
Default charge-back account          (DFT_ACCOUNT_STR) =

Java Development Kit installation path     (JDK_PATH) = /opt/IBMJava2-131

Diagnostic error capture level          (DIAGLEVEL) = 3
Notify Level                             (NOTIFYLEVEL) = 3
Diagnostic data directory path           (DIAGPATH) = /home/insta8/sqlib/db2dump

Default database monitor switches
  Buffer pool          (DFT_MON_BUFPOOL) = OFF
  Lock                 (DFT_MON_LOCK) = OFF
  Sort                 (DFT_MON_SORT) = OFF
  Statement            (DFT_MON_STMT) = OFF
  Table                (DFT_MON_TABLE) = OFF
  Timestamp            (DFT_MON_TIMESTAMP) = ON
  Unit of work         (DFT_MON_UOW) = OFF

Monitor health of instance and databases    (HEALTH_MON) = OFF

SYSADM group name                      (SYSADM_GROUP) = INSTA8
SYSCTRL group name                     (SYSCTRL_GROUP) =
SYSMAINT group name                    (SYSMAINT_GROUP) =

Database manager authentication          (AUTHENTICATION) = SERVER
Cataloging allowed without authority    (CATALOG_NOAUTH) = NO
Trust all clients                       (TRUST_ALLCLNTS) = YES
Trusted client authentication           (TRUST_CLNTAUTH) = CLIENT
Use SNA authentication                   (USE_SNA_AUTH) = NO
Bypass federated authentication          (FED_NOAUTH) = NO

Default database path                   (DFTDBPATH) = /home/insta8

Database monitor heap size (4KB)         (MON_HEAP_SZ) = 90
Java Virtual Machine heap size (4KB)    (JAVA_HEAP_SZ) = 2048
Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0
Size of instance shared memory (4KB)     (INSTANCE_MEMORY) = AUTOMATIC
Backup buffer default size (4KB)         (BACKBUFSZ) = 1024
Restore buffer default size (4KB)        (RESTBUFSZ) = 1024

Sort heap threshold (4KB)                (SHEAPTHRES) = 20000

Directory cache support                  (DIR_CACHE) = YES

Application support layer heap size (4KB) (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)     (RQRIOBLK) = 32767
Query heap size (4KB)                    (QUERY_HEAP_SZ) = 1000
DRDA services heap size (4KB)            (DRDA_HEAP_SZ) = 128

Priority of agents                      (AGENTPRI) = SYSTEM
Max number of existing agents            (MAXAGENTS) = 400
Agent pool size                         (NUM_POOLAGENTS) = 200(calculated)
Initial number of agents in pool         (NUM_INITAGENTS) = 0
Max number of coordinating agents        (MAX_COORDAGENTS) = (MAXAGENTS - NUM_INITAGENTS)
Max no. of concurrent coordinating agents (MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections         (MAX_CONNECTIONS) = MAX_COORDAGENTS
Keep fenced process (KEEPFENCED) = YES
Number of pooled fenced processes (FENCED_POOL) = MAX_COORDAGENTS
Initialize fenced process with JVM (INITFENCED_JVM) = NO
Initial number of fenced processes (NUM_INITFENCED) = 0

Index re-creation time (INDEXREC) = RESTART

Transaction manager database name (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec) (RESYNC_INTERVAL) = 180

SPM name (SPM_NAME) =
SPM log size (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit (SPM_MAX_RESYNC) = 20
SPM log path (SPM_LOG_PATH) =

TCP/IP Service name (SVCENAME) = db2c_insta8
Discovery mode (DISCOVER) = SEARCH
Discovery communication protocols (DISCOVER_COMM) = TCPIP
Discover server instance (DISCOVER_INST) = ENABLE

Maximum query degree of parallelism (MAX_QUERYDEGREE) = ANY
Enable intra-partition parallelism (INTRA_PARALLEL) = NO

No. of int. communication buffers (FCM_NUM_BUFFERS) = 4096
Node connection elapse time (sec) (CONN_ELAPSE) = 10
Max number of node connection retries (MAX_CONNRETRIES) = 5
Max time difference between nodes (min) (MAX_TIME_DIFF) = 60

db2start/db2stop timeout (min) (START_STOP_TIME) = 10

Database Configuration Parameters

The following is a list of the database configuration parameters for the storesdb database (v8).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database configuration release level</td>
<td>0x0a00</td>
</tr>
<tr>
<td>Database release level</td>
<td>0x0a00</td>
</tr>
<tr>
<td>Database territory</td>
<td>US</td>
</tr>
<tr>
<td>Database code page</td>
<td>1208</td>
</tr>
<tr>
<td>Database code set</td>
<td>UTF-8</td>
</tr>
<tr>
<td>Database country/region code</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic SQL Query management</td>
<td>DISABLE</td>
</tr>
<tr>
<td>Discovery support for this database</td>
<td>ENABLE</td>
</tr>
<tr>
<td>Default query optimization class</td>
<td>5</td>
</tr>
<tr>
<td>Degree of parallelism</td>
<td>1</td>
</tr>
<tr>
<td>Continue upon arithmetic exceptions</td>
<td>NO</td>
</tr>
<tr>
<td>Default refresh age</td>
<td>0</td>
</tr>
<tr>
<td>Number of frequent values retained</td>
<td>10</td>
</tr>
<tr>
<td>Number of quantiles retained</td>
<td>20</td>
</tr>
</tbody>
</table>

B-4  Example Configuration Parameters
Backup pending = NO
Database is consistent = YES
Rollforward pending = NO
Restore pending = NO

Multi-page file allocation enabled = NO
Log retain for recovery status = NO
User exit for logging status = NO

Data Links Token Expiry Interval (sec) (DL_EXPINT) = 60
Data Links Write Token Init Expiry Intvl (DL_WT_IEXPINT) = 60
Data Links Number of Copies (DL_NUM_COPIES) = 1
Data Links Time after Drop (days) (DL_TIME_DROP) = 1
Data Links Token in Uppercase (DL_UPPER) = NO
Data Links Token Algorithm (DL_TOKEN) = MAC0

Database heap (4KB) (DBHEAP) = 1200
Size of database shared memory (4KB) (DATABASE_MEMORY) = AUTOMATIC
Catalog cache size (4KB) (CATALOGCACHE_SZ) = (MAXAPPLS*4)
Log buffer size (4KB) (LOGBUFSZ) = 8
Utilities heap size (4KB) (UTIL_HEAP_SZ) = 5000
Buffer pool size (pages) (BUFFPAGE) = 1000
Extended storage segments size (4KB) (ESTORE_SEG_SZ) = 16000
Number of extended storage segments (NUMESTORE_SEGS) = 0
Max storage for lock list (4KB) (LOCKLIST) = 100

Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000
Percent of mem for appl. group heap (GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB) (APP_CTL_HEAP_SZ) = 128

Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB) (SORTHEAP) = 256
SQL statement heap (4KB) (STMTH HEAP) = 2048
Default application heap (4KB) (APPLHEAPSZ) = 256
Package cache size (4KB) (PCKCACHESZ) = (MAXAPPLS*8)
Statistics heap size (4KB) (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms) (DLCHKTIME) = 10000
Percent. of lock lists per application (MAXLOCKS) = 10
Lock timeout (sec) (LOCKTIMEOUT) = -1

Changed pages threshold (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners (NUM_IOCLEANERS) = 1
Number of I/O servers (NUM_IOSERVERS) = 3
Index sort flag (INDEXSORT) = YES
Sequential detect flag (SEQDETECT) = YES
Default prefetch size (pages) (DFT_PREFETCH_SZ) = 32

Track modified pages (TRACKMOD) = OFF
Default number of containers = 1
Default tablespace extentsize (pages) (DFT_EXTENT_SZ) = 32
Max number of active applications          (MAXAPPLS) = AUTOMATIC
Average number of active applications      (AVG_APPLS) = 1
Max DB files open per application          (MAXFILOP) = 64

Log file size (4KB)                        (LOGFILSIZ) = 1000
Number of primary log files               (LOGPRIMARY) = 3
Number of secondary log files             (LOGSECOND) = 2
Changed path to log files                 (NEWLOGPATH) =
Path to log files                          = /home/insta8/insta8/

NODE0000/SQL00002/SQLOGDIR/
Overflow log path                          (OVERFLOWLOGPATH) =
Mirror log path                            (MIRRORLOGPATH) =
First active log file                      =
Block log on disk full                     (BLK_LOG_DSK_FUL) = NO
Percent of max active log space by transaction (MAX_LOG) = 0
Num. of active log files for 1 active UOW (NUM_LOG_SPAN) = 0

Group commit count                         (MINCOMMIT) = 1
Percent log file reclaimed before soft chkpt (SOFTMAX) = 100
Log retain for recovery enabled            (LOGRETAIN) = OFF
User exit for logging enabled              (USEREXIT) = OFF

Auto restart enabled                       (AUTORESTART) = ON
Index re-creation time                     (INDEXREC) = SYSTEM (RESTART)
Default number of loadrec sessions         (DFT_LOADREC_SES) = 1
Number of database backups to retain       (NUM_DB_BACKUPS) = 12
Recovery history retention (days)           (REC_HIS_RETENTN) = 366

TSM management class                      (TSM_MGMTCLASS) =
TSM node name                              (TSM_NODENAME) =
TSM owner                                  (TSM_OWNER) =
TSM password                               (TSM_PASSWORD) =
Index
## Index

### A
- Administration Client 10
- Administration Server configuration file 6
- Alert Center 21
- Alter Table 17

### C
- CATALOG 21, 27
  - Client 24
- Character Search Operators 20
- Check constraints 7
- Client Configuration Assistant 27, 26, 28
- Command Center 15
  - Access Plan tab 19
  - Interactive tab 16
  - Query Results tab 18
  - Script tab 17
  - toolbar 16
- Command line
  - db2cc 13
  - db2cca 27
  - db2cctor 15
  - db2icrt 41, 42
  - db2idrop 45
  - db2ilist 45
  - db2start 44
  - db2stop 44
- Command Line Processor 9, 31
- Communication Protocol 31
- Connect Server 9
- Constraints 21
  - Unique 21, 22
- Control Center
  - toolbar 14
- Cost-based 53
- CREATE DATABASE 10, 12
  - Example 19
  - SYSCATSPACE 10, 11
  - TEMPSPACE1 11
  - USERSPACE1 11
- Create Table 15

### D
- DAS 5
  - Database Administration Server 3, 4

### Data concurrency 10
- Isolation level 9
  - CHANGE ISOLATION command 15
- Choosing the isolation level 16
- Cursor Stability 13
- Read Stability 12
- Repeatable read 11
- Uncommitted Read 14

### Locking 18
- COMMIT 26
- Deadlock 27
- Deadlock detector 27
- Lock acquisition 29
- Lock compatibility 23
- Lock conversion 25
- Lock escalation 26
- Lock states (modes) 20
- LOCK TABLE statement 30
- Lock timeout 28
- Lock-Exclusive (X) 22, 25
- Lock-Share (S) 21, 25
- LOCKSIZE option 29
- LOCKTIMEOUT parameter 28
- Lock-Update (U) 21, 25
- MAXLOCKS parameter 26
- ROLLBACK 26

### Data consistency 3, 10

### Data types 3
- Binary String 12
- Character String 9
- DATALINK 10
- Date-time 13, 9
  - DATE 9
  - TIME 9
  - TIMESTAMP 9
- Graphic String 11
- Numeric 7, 5
  - BIGINT 5
  - DECIMAL 5
  - DOUBLE 6
  - Floating point 6
  - INTEGER 5
  - REAL 6
  - SMALLINT 5
- Strings 7
  - CHAR 7
  - GRAPHIC 7
  - VARCHAR 7
VARGRAPHIC 8
Strings-long
BLOB 8
CLOB 8
DBCLOB 8
LONG VARCHAR 8
LONG VARGRAPHIC 8
UDTs
  Distinct type 11
  Extenders 13
  Reference type 12
  Structured type 12
Data Warehouse Manager
  Data Warehouse Center 33
  Information Catalog 34
  Query Management Facility 36
  Query Patroller 34
Database components 3
  Container 4
  IMPLICIT_SCHEMA 23
  Index 6
  Long field 7
  SCHEMA 23
  System catalog 17
  Table 6
  Table space 4, 16
  DMS 8
  SMS 8
Database Manager configuration file 6
Database objects 27
Database parameter
  CODESET 15
  DFT_PREFETCH_SZ 45
  DFTDBPATH 14
  EXTENTSIZe 16, 45
  PREFETCHSIZE 16, 45
  TERRITORY 15
Database-managed space 42
Discovery 25
Drop Table 17

E
Environment variable
  DB2INSTANCE 43, 35
  DB2SYSTEM 35

F
Fenced user 26
First Steps 30
Fixpack 43
Foreign key 9, 10

Functions
  Column 31
  Scalar 32

G
Gateway 14
Generated column 19

H
Help
  Information Center 26

I
IF-THEN-ELSE logic 36
Information Center 26
Informational constraints 27
Installation
  First Steps 30

J
Journal 20, 22

L
License Center 23

M
Multi-user
  Dirty reads 9
  Lost updates 9
  Non-repeatable reads 10
  Phantoms 10

O
OLAP 37
Optimizer
  Data access path 33

P
Performance Monitor 21, 29
Predicate 50, 51
Primary key 9
Privilege
  22
R
Relational Operators
   does not equal 19
equals 19
greater than 19
greater than or equal to 19
less than 19
less than or equal to 19
Run-Time Client 9

S
Script Center 20
Security
   Authentication 8, 10, 14, 18
      CLIENT 12
      Client 16
      Client to Host 17
      Client to Server 16
      DCE 12
      DCE_SERVER_ENCRYPT 12
      DCS 12
      DCS_ENCRYPT 12
      KERBEROS 12
      KRB_SERVER_ENCRYPT 12
      SERVER 12
      SERVER_ENCRYPT 12
   Authority 22
      DBADM 22
      LOAD 22
      SYSADM 21
      SYSCTRL 21
      SYSMAIN 22
   Authorization 8
   Obtaining DBADM Authority 29
   Obtaining LOAD Authority 31
   Obtaining SYSADM Authority 25
   Obtaining SYSCTRL Authority 27
   Obtaining SYSMAIN Authority 28
   Privilege 8, 34
      CONTROL 22
      DELETE 22
      Explicit 36
      GRANT 22, 25
      Implicit 38
      Indirect 40
      INSERT 22
      PUBLIC 25
      REVOKE 22, 26
      SELECT 22
      UPDATE 22
      WITH GRANT OPTION 22, 25

SQL statement
   ALTER TABLE 29
   Character 5
   COMMIT 4
   Constant 6
   CREATE TABLE 27
   CUBE 30
   Data type 5
   DCL 3, 19
   DDL 3
   DELETE 35
   DML 3
   DROP 30
   EXCEPT 28
   Expression 6
   Function 6
   GROUP BY 29
   GROUPING SETS 30
   HAVING 30
   Identifier 5
   INNER JOIN 25
   INSERT 33
   INTERSECT 28
   JOIN 24
   ORDER BY 23
   OUTER JOIN 26
   Predicate 6
   ROLLBACK 4
   ROLLUP 30
   SELECT 17
   Special register 16
   Token 5
   UNION 28
   UPDATE 34
   WHERE clause 21
   Statement termination character 17
   Stored Procedure Builder 24
   System-managed space 42

T
Table space 42
   database-managed space 42
   system-managed space 42
Temporary tables 28
Toolbar 14, 16
Tools Settings notebook 17, 28
Transaction
   COMMIT 4
   Data Concurrency and Lock Granularity 31
   Effects 7, 8
   Processing
      Change 32
Cursor-controlled 32
Intent-to-change 32
Lock-Exclusive (X) 32
Lock-Share (S) 32
Lock-Update (U) 32
Read-only 32
ROLLBACK 4
Unit of work 4

U
Unique key 9
Untrusted Clients 18

V
Visual Explain 19