DB2 UDB for Linux, UNIX and Windows Programming Using Java

(Course Code CG11)

Instructor Exercises Guide

ERC 3.0

IBM Certified Course Material

Trademarks

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

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

AIX	Approach	CICS
DataJoiner	DB2	DB2 Connect
DB2 Extenders	DB2 Universal Database	Distributed Relational Database Architecture
DRDA	Encina	Lotus
MQSeries	MVS	MVS/ESA
Net.Data	NUMA-Q	OS/2
OS/390	OS/400	QMF
RISC System/6000	SQL/400	VisualAge
1-2-3		

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

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

June 2003 Edition

Contents

Trademarks

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

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

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

AIX®	Approach®	CICS®
DataJoiner®	DB2®	DB2 Connect™
DB2 Extenders™	DB2 Universal Database™	Distributed Relational Database Architecture™
DRDA®	Encina®	Lotus®
MQSeries®	MVS™	MVS/ESA™
Net.Data®	NUMA-Q®	OS/2®
OS/390®	OS/400®	QMF™
RISC System/6000®	SQL/400®	VisualAge®
1-2-3®		

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

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

Instructor Exercises Overview

The objectives of the application programming exercises is to have the students apply the information they should have learned in the lecture.

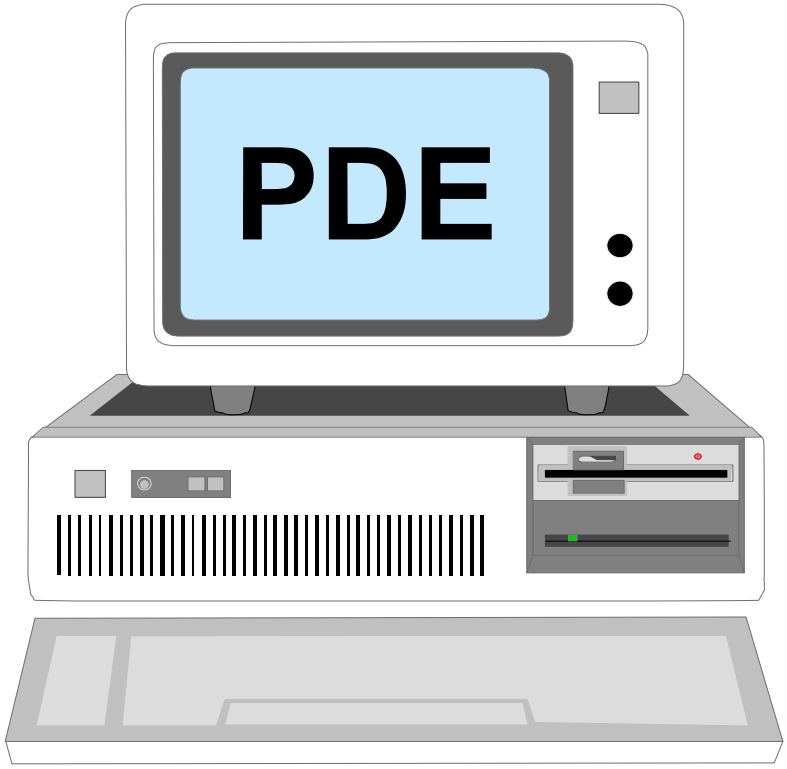
Most of the exercises depend on the previous exercise being completed successfully.

Students may elect to complete the optional distributed unit-of-work program as time permits, at any point after the program preparation topic has been presented.

Exercises Configuration

The lab is setup to be run with each workstation as a stand-alone machine. The workstations will be running DB2 UDB V7.1 Enterprise Edition.

Exercise Setup



Exercises Description

Each exercise in this course is divided into sections as described below. Select the section that best fits your method of performing labs. You may select to use a combination of these sections as appropriate.

Exercise Instructions — This section contains what you are to accomplish. There are no definitive details on how to perform the tasks. You are given the opportunity to work through the exercise given what you learned in the unit presentation, utilizing the unit Student Notebook, your past experience and maybe a little intuition.

Exercise Solutions — This section is an exact duplicate of the Exercise Instructions section except that in addition, answers to questions posed in the lab are documented. You may choose to check your answers with the Exercise Solutions portion of the document, or you may choose to complete the labs totally using the Exercise Solutions.

General Information

This lab guide will provide the information necessary to complete the required and optional labs for the *DB2 Programming Fundamentals* course.

Throughout the guide, the notation "udba" will be used to designate an ID.

The expected results for *ALL* problems presented for the programming lab can be found in the appendices.

Lab members are in D:\CG112.

Lab solutions are in D:\CG112\SOLN.

Tables

Each team will use the SAMPLE database in order to complete the labs. Exercise 7, the Performance and Tuning lab, will use the MUSICDB database. A description of each table will be documented in Exercise 7.

View

A view will be created and used in order to complete the labs. A description of this view follows for reference. (It is based on DEP and EMP, so the attributes of corresponding columns have been included for simplicity.)

Exercise 1. Command Line Processor (CLP) and Command Center

What This Exercise Is About

The Command Line Processor (CLP) can be used to enter DB2 commands as well as interactive SQL.

The Graphical User Interface (GUI) Control Center can be used to manipulate database objects. The Command Center can be use to issue interactive SQL or DB2 commands or run prewritten script files.

The lab is conceived to give you the opportunity to try them both.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Use CLP and Command Center to execute a supplied script
- Alter CLP and Command Center options
- Create the objects that will be used for remaining labs in this class
- Examine the impact of referential integrity using Command Center

Introduction

A database named **EDDB** has already been created for you. You will be learning about the CLP interface and about the Command Center GUI and using one or the other to create the tables and views needed in the rest of the class labs.

Instructor Notes

Introduction — Review the ”Exercises Description” on page xi. Also, give the information on where the lab members are (by default they will be in `D:\CG112`) and where the lab solution is (by default it will be in `D:\CG12\SOLN`).

Review the tables that they will be creating during this lab (following ”Exercises Description” on page xi).

Then have them turn to this lab. They will be using a supplied database named **EDDB. They will be experimenting with CLP to learn what they can do with it. If running the GUI, the students will learn about the Control Center and the Command Center.**

They will be running scripts to build the tables they will be using in class and populating them with data. They should finish this lab before they go on to the following labs.

If a particular student cannot complete this lab without consuming a significant portion of the next lab session, the member **crtabs.mem** can be used to create the objects required. Spending time on the programming labs is more important than completing the CLP lab.

Time for Lab — Allocate 1 hour for this lab.

Things to Review at End of Lab — No formal review is necessary for this lab, other than emphasizing the need to have all objects successfully created for following labs.

Common Mistakes Students May Make

- When they set the DB2OPTIONS environment variable, there must not be blanks on either side of the '=' sign. Also, the environment variable name must be in all caps. To see what the setting is for the DB2OPTIONS environment variable, 'echo %DB2OPTIONS%' on Windows NT.
- The default settings for the options will not ever change. The students can 'LIST COMMAND OPTIONS' to see their current setting.

Exercise Instructions

The lab solutions can be found in ”Exercise Solutions” on page 1-13.

An alternative lab using the Command Center can be found in ”Command Center Exercise Instructions” on page 1-26.

Section 1 - Introduction to Command Line Processor (CLP)

__ 1. If on Windows NT, open a DB2 Command Window (Start -> Programs -> IBM DB2 -> Command Window).

__ 2. Access the command line processor by entering the following command:

db2

Note: Do NOT use the Command Line icon to start a CLP session. Although use of the icon would be acceptable in other situations, this lab is written assuming you are not invoking CLP through the icon.

Read the information presented to you. Using this information, enter a command that will provide you with general help.

__ 3. Enter the command to obtain help on reading the help screens.

What does it mean if parameters are enclosed in brackets [ ]?

_____

_____

_____

Note: You may notice that some information scrolls off of the screen before you can read it. Do not worry about this now; we will learn in a few more steps how to remedy this situation.

__ 4. Look at the available options by entering the following command:

? options

What is the current setting for auto-commit?

What is the current setting for displaying the SQLCA?

_____

_____

_____

__ 5. Assume you would like to display the SQLCA. Read the information provided carefully and get help on the command you would use to change an option setting in an interactive mode. What command did you issue to get help?

_____

_____

_____

__ 6. Execute the command that will cause the SQLCA to be displayed for the current interactive session. Did your command work?

_____

_____

_____

__ 7. Up to this point, you have been using CLP in interactive mode (entered by typing **db2** and usually indicated with the **db2 =>** prompt.) There are two commands that can be used to terminate a CLP session, terminate, and quit. Use the quit command to end your interactive session:

quit

Is the change you made to the command options still active?

_____

_____

_____

__ 8. To use CLP in non-interactive mode, prefix all commands with **db2.** Issue the non-interactive command shown to obtain help on the options:

db2 ? options | more

Is the SQLCA displayed?

What is the apparent duration of changes made via the update command options command? (Read the notes on the displayed help information for a hint.)

_____

_____

_____

__ 9. There are two ways to set an option during a non-interactive session that uses the command line for input. One of these techniques is to specify the option on the command itself. Review the notes in the help information and execute the following command:

db2 -a ? options | more

Was the SQLCA displayed?

According to the notes, how would you explicitly turn an option off? Is this necessary for the -a option considering your current environment?

_____

_____

_____

__ 10. Execute the same command with one change. Do **NOT** use the option to display the SQLCA:

db2 ? options | more

Is the SQLCA displayed?

What is the apparent duration of options specified on a particular command?

_____

_____

_____

__ 11. The second technique to change command options is to use the **DB2OPTIONS** environment variable. Issue the following commands to set your variable to display the SQLCA, verify the setting, and determine the duration of impact on the option:

set DB2OPTIONS=-a (in AIX, **export DB2OPTIONS=-a**)

db2 ? options

db2 ? options

Was the SQLCA displayed?

Is the change to the option effective for only one command or several?

_____

_____

_____

__ 12. Set your DB2OPTIONS variable to cause CLP to stop when a statement error is encountered; to echo the current command, use the semicolon as the statement terminator, and write results to an output file named **clp.out** by issuing the following command:

set DB2OPTIONS=-svtr clp.out

Change directories to your CG112 directory:

cd \CG112

Connect to the database:

db2 connect to eddb user udba using udba

Examine the `sample.mem` file that contains SQL and comments that you will run in the next step:

more < sample.mem

How are comments designated?

What separates one SQL statement from another?

_____

_____

_____

__ 13. Execute the SQL in the file by submitting the following command:

db2 -f sample.mem

What option caused the SQL statements in the file to be displayed?

Why is the output appearing on your screen?

Is the output somewhere else as well?

What happens if an error is encountered?

_____
_____
_____
_____
_____
_____

__ 14. Examine the contents of your report file:

more < clp.out

Does the report reflect everything you previously saw on the output display?

_____
_____
_____

__ 15. Issue the following command:

db2 ? options

Examine the output report file:

more < clp.out

Are the results from your last executed and prior commands reflected in the file?

_____
_____
_____

__ 16. Update the DB2OPTIONS variable so that a report file is no longer used. Keep the other settings active:

set DB2OPTIONS=-svt (in AIX, **export DB2OPTIONS=-svt**)

__ 17. Verify the setting of the DB2OPTIONS environment variable by issuing the following command. (This command can be used to echo the setting of any environment variable. Ensure that the desired variable is bracketed by percent signs.)

echo %DB2OPTIONS% (in AIX, **echo $DB2OPTIONS**)

Note: If you cannot successfully set your DB2OPTIONS environment variable, contact the instructor. Other portions of this lab are dependent on proper setting of this variable.

Section 2 - Creating Lab Objects

__ 1. Issue the following command:

more < dept.mem

View the contents while reading the following information:

- This is the DDL that will CREATE the table called DEP. Since the name of the table is not qualified with an OWNER, your USERID will become the OWNER.
- A list of column definitions follows the create statement. Each item in the list will define a column name, its attribute, and null support characteristic. For example, the first column in the table will be known as DEPTNO. It will be a three-byte column that will contain CHARACTER data. This column will NOT allow NULL data.

  The third column will be known as MGRNO. It is a six-byte column that will contain CHARACTER data. Unlike the DEPTNO column, the MGRNO column will allow NULL data, since no explicit disallowance is specified.

  Other data types include INTEGER, SMALLINT, DECIMAL, DATE, TIME, TIMESTAMP, and VARCHAR. For further discussion of the attributes of these and other data types, consult the *SQL Reference* manual.

  The other specification regarding NULL is NOT NULL WITH DEFAULT. Since a default value was not specified in the command syntax, it is dependent on the type of column defined:

Data Type	Default Value
Numeric	0 (zero)
Fixed Length String	blanks
Variable Length String	String with 0 length
Date	Current Date
Time	Current Time
Timestamp	Current Timestamp

- After the column list, notice the specification of a PRIMARY KEY clause. In this example, the column DEPTNO is named as the PRIMARY KEY. If more than one column comprised the PRIMARY KEY, they would form a list within the parentheses, and each column would be separated by a comma.

  Any column named in the PRIMARY KEY clause must be defined as NOT NULL or NOT NULL WITH DEFAULT. Usually, NOT NULL is specified since the values in a primary key must be unique. An exception may be the use of timestamps. The default value for a timestamp is the CURRENT TIMESTAMP, which is granular to the microsecond.
- In order to enforce unique values in the primary key, DB2 will require a UNIQUE INDEX on the primary key columns. Since such an index cannot currently exist

(the table is being defined in the statement), DB2 will automatically create the necessary index.

__ 2. Execute this file to create the table and verify successful execution:

db2 -f dept.mem

__ 3. Issue the following command:

more < emp.mem

View the contents while reading the following information:

- The first statement will create a table called EMP. Note the column types and null attributes specified.

  Referential integrity characteristics are **NOT** part of the new table definition.

- The CREATE INDEX statement creates a UNIQUE INDEX on what will be named the primary key. If a UNIQUE INDEX exists when a primary key is defined, the database manager will use the index as the primary key index.

- The ALTER statement will add the referential integrity characteristic we desire.

  The PRIMARY KEY clause identifies EMPNO as the only column in the primary key.

  Note the FOREIGN KEY clause. It identifies the foreign key name (RIEMPDEP), the column(s) in the key (WORKDEPT), the parent table referenced (DEP), and the DELETE RULE (SET NULL). Notice that this delete rule is only possible because the foreign key column allows null values.

  If the foreign key is not named explicitly, DB2 will create a name. The name can be 1 to 8 characters in length.

  The foreign key columns must match the specification of the primary key columns in the referenced parent table with respect to type, size, and order.

  No columns contained in the parent table need to be referenced. Since a table can have only one PRIMARY KEY, naming the parent table is sufficient.

  Other delete rules include RESTRICT and CASCADE. If no rule is specified, RESTRICT is assumed.

  Note: The ALTER statement was used for illustrative purposes. Tables with dependencies on other tables can be created using the CREATE statement with FOREIGN KEY clauses.

- The second index created is on the foreign key of the EMP table. Although this index is not required, it is strongly recommended if the dependent table will contain any significant volume of data. Otherwise, referential integrity constraint enforcement checking may be a poor performer.

__ 4. Execute this file to create the table and verify successful execution:

db2 -f emp.mem

__ 5. View the following input

d:\cg112\VIEW.MEM

more < view.mem

__ 6. Note the syntax of the CREATE VIEW statement. We are creating a view which will be a join of the EMP and DEPT tables. The view is called vphone. Execute the script.

__ 7. Execute this file to create the vphone view and verify successful execution. (0 rows are returned from the SELECT statement; the tables are still empty).

db2 -f view.mem

Section 3 - Impacts of Referential Integrity

__ 1. Look at the script file:

more < insert.mem

View the contents while reading the following information:

- Both statements have similar syntax.
- The first inserts all the rows from CF10.EMP into your EMP table.
- The second inserts all the rows from CF10.DEP into your DEP table.

  Note: This technique is useful to populate a sampling of data from a production table to a test table. WHERE clauses could be specified on the SUBSELECT to limit the data selected.

__ 2. Execute this file:

db2 -f insert.mem

Did the file execute successfully? Do you know why or why not?

_____

_____

_____

__ 3. Look at the script file:

more < insert2.mem

View the contents while reading the following information:

- The statements have been reordered so that the rows are loaded into the parent table first.
- The first inserts all the rows from CF10.DEP into your DEP table.
- The second inserts all the rows from CF10.EMP into your EMP table.

__ 4. Execute this file:

db2 -f insert2.mem

Did the file execute successfully? Why?

_____

_____

_____

__ 5. Look at the script file:

more < updatepk.mem

View the contents while reading the following information:

- The company supported by the DEP table is reorganizing and needs to change the department numbers of two departments.
- Department D01 needs to be changed to department C01.
- Department D11 needs to become department D31.

__ 6. Execute the file to meet the needs of the business:

db2 -f updatepk.mem

Did the file execute successfully? Why or why not?

_____

_____

_____

__ 7. It was determined that department D01 actually should be changed to department C10. The original UPDATE statement had a transposed value. Execute the following file, which corrects the transposed value:

Look at the script file D:\cg112\uppk2.mem and execute the script file.

db2 -f uppk2.mem

Why did the first UPDATE statement succeed and the second fail?

What must DB2 check to determine whether or not a row is a "parent" row?

What could assist DB2 in this check?

What is the name of the relationship that caused the second statement to fail?

_____

_____

_____

_____

_____

_____

__ 8. The company wants to change the department number of D11 to D31, but the UPDATE to the DEP table was prevented due to a constraint. Therefore, it has been decided to change the values in the EMP table first.

Look at the file d:\cg112\updatefk.mem.

more < updatefk.mem

Execute the following command which will attempt to update the rows in the EMP table:

db2 -f updatefk.mem

Did the statement succeed? Why or why not?

Can you propose the steps necessary to actually change the value of D11 to D31 in both tables? Contact your instructor or consult the solution for this question if you are not sure of your answer. **DO NOT** actually execute your solution.

_____
_____
_____
_____
_____
_____

__ 9. It has been determined that department D11 should be eliminated from the DEP table. View the contents of a file that will help demonstrate the impact of such an action:

more < D:\cg112\deletepk.mem

View the contents while reading the following information:

- The first SELECT shows the D11 row from the DEP table.
- The second SELECT shows three specific employee rows from the EMP table that work in department D11.
- The DELETE statement eliminates the D11 row from the DEP table.
- The third SELECT shows the same three employee rows displayed by the second SELECT.

__ 10. Execute the file:

db2 -f deletepk.mem

Did the DELETE statement succeed?

Did anything change in the three employee rows that were previously assigned to department D11? Why?

What would have happened if the DELETE RULE was CASCADE?

What would have happened if the DELETE RULE was RESTRICT?

Would the referential constraint defined between the DEP table and the EMP table prevent the deletion of a row from the EMP table? Does your answer depend on the DELETE RULE chosen?

_____
_____
_____
_____
_____
_____

__ 11. The next set of SQL statements to be executed will delete the data from EMP, delete the data from DEP, and insert a fresh set of sample data into both tables. This "refresh" will ensure that the lab tables contain the correct data for subsequent lab exercises. Enter the following command and verify successful execution:

db2 -f refresh.mem

__ 12. Enter the following commands to terminate your CLP session:

db2 terminate

__ 13. Read the following summary regarding the impact of referential integrity on application table usage.

- Primary keys must be unique and cannot allow nulls. DB2 enforces uniqueness by requiring a unique index on the primary key columns.
- Foreign keys need not be unique and may allow nulls. DB2 can check foreign key column values efficiently when indexes are defined on the foreign key columns.
- A non-null foreign key value must have a matching value in the primary key, but a primary key value does not require a matching foreign key value.
- Any INSERT, UPDATE, or DELETE statement that violates any of the above rules will not execute successfully.

__ 14. Reset your db2options:

set DB2OPTIONS=

END OF LAB

Exercise Solutions

Section 1 - Introduction to Command Line Processor (CLP)

__ 1. If on Windows NT, open a DB2 Command Window (Start -> Programs -> IBM DB2 -> Command Window).

__ 2. Access the command line processor by entering the following command:

db2

Note: Do NOT use the Command Line icon to start a CLP session. Although use of the icon would be acceptable in other situations, this lab is written assuming you are not invoking CLP through the icon.

Read the information presented to you. Using this information, enter a command that will provide you with general help.

?

__ 3. Enter the command to obtain help on reading the help screens.

What does it mean if parameters are enclosed in brackets [ ]?

_____

_____

_____

? help

The brackets are used to indicate components of the command that are considered optional. Also, the | is used to designate choices. If an option itself has an optional component, the brackets will be nested.

Note: You may notice that some information scrolls off of the screen before you can read it. Do not worry about this now; we will learn in a few more steps how to remedy this situation.

__ 4. Look at the available options by entering the following command:

? options

What is the current setting for auto-commit?

What is the current setting for displaying the SQLCA?

_____

_____

_____

You cannot tell with this command. Notice that the right column is for the Default Setting not the current setting. You have not yet changed the defaults so: Auto-commit is currently set to ON and the option to display the SQLCA is

currently set to OFF. If you wish to confirm this, enter: LIST COMMAND OPTIONS.

__ 5. Assume you would like to display the SQLCA. Read the information provided carefully and get help on the command you would use to change an option setting in an interactive mode. What command did you issue to get help?

_____

_____

_____

? update command options

__ 6. Execute the command that will cause the SQLCA to be displayed for the current interactive session. Did your command work?

_____

_____

_____

update command options using a on

The command changed the option for displaying the SQLCA from OFF to ON. The evidence of this change was the output from the update command itself.

SQLCA Information

```
sqlcaid : SQLCA     sqlcabc: 136   sqlcode: 0   sqlerrml: 0
sqlerrmc:
sqlerrp :
sqlerrd : (1) 0                (2) 0                (3) 0
          (4) 0                (5) 0                (6) 0
sqlwarn : (1)       (2)       (3)       (4)        (5)        (6)
          (7)       (8)       (9)       (10)       (11)
sqlstate:
```

__ 7. Up to this point, you have been using CLP in interactive mode (entered by typing **db2** and usually indicated with the **db2 =>** prompt.) There are two commands that can be used to terminate a CLP session, terminate and quit. Use the quit command to end your interactive session:

quit

Is the change you made to the command options still active?

_____

_____

_____

The SQLCA was displayed when the quit command was issued. The change made in the prior step is still active during the quit step.

__ 8. To use CLP in non-interactive mode, prefix all commands with **db2.** Issue the non-interactive command shown to obtain help on the options:

db2 ? options | more

Is the SQLCA displayed?

What is the apparent duration of changes made via the update command options command? (Read the notes on the displayed help information for a hint.)

_____

_____

_____

The SQLCA is not displayed. The change made to display the SQLCA was only active during the interactive session in which the change was made. Note also that you can pipe the output of the command to `more` allowing you to see all of the output.

__ 9. There are two ways to set an option during a non-interactive session that uses the command line for input. One of these techniques is to specify the option on the command itself. Review the notes in the help information and execute the following command:

db2 -a ? options | more

Was the SQLCA displayed?

According to the notes, how would you explicitly turn an option off? Is this necessary for the -a option considering your current environment?

_____

_____

_____

The SQLCA is displayed. The notes on the help screen state that "A minus sign (-) immediately following an option letter turns the option off". Therefore, specifying -a- would explicitly turn the option to display the SQLCA off. Since the default setting is not to display the SQLCA, not using a as an option is the same as using -a-.

Note: Using the plus sign (+) will also turn an option off. Therefore, +a is equivalent to -a-

__ 10. Execute the same command with one change. Do **NOT** use the option to display the SQLCA:

db2 ? options | more

Is the SQLCA displayed?

What is the apparent duration of options specified on a particular command?

_____

_____

_____

The SQLCA is not displayed. An option specified on a command impacts that particular command. It does not impact other commands issued later in the session.

__ 11. The second technique to change command options is to use the **DB2OPTIONS** environment variable. Issue the following commands to set your variable to display the SQLCA, verify the setting, and determine the duration of impact on the option:

set DB2OPTIONS=-a (in AIX, **export DB2OPTIONS=-a**)

db2 ? options

db2 ? options

Was the SQLCA displayed?

Is the change to the option effective for only one command or several?

_____

_____

_____

After setting the DB2OPTIONS variable, the SQLCA will be displayed after all subsequent commands until the DB2OPTIONS variable is unset or changed or if the -a- option is explicitly coded on a statement. One can use this technique during a login or logon procedure to effectively customize a CLP processing environment.

__ 12. Set your DB2OPTIONS variable to cause CLP to stop when a statement error is encountered, to echo the current command, use the semicolon as the statement terminator, and write results to an output file named **clp.out** by issuing the following command:

set DB2OPTIONS=-svtr clp.out

Change directories to your CG112 directory:

cd \CG112

Connect to the database:

db2 connect to eddb user udba using udba

Examine the `sample.mem` file that contains SQL and comments that you will run in the next step:

more < sample.mem

How are comments designated?

What separates one SQL statement from another?

_____

_____

Two hyphens identify comments and SQL statements are separated via the semicolon.

__ 13. Execute the SQL in the file by submitting the following command:

db2 -f sample.mem

What option caused the SQL statements in the file to be displayed?

Why is the output appearing on your screen?

Is the output somewhere else as well?

What happens if an error is encountered?

The SQL statements in the file appear in the display because the -v option, which was specified in the variable DB2OPTIONS, requests the commands to be echoed. The output is appearing on the screen because the -o option defaults to ON. We have not disabled this option via a +o or a -o-. Since we have used the -r option to save the output report in a file, the output should appear in that file. If an error is encountered, CLP will provide a message regarding this error. It will stop attempting execution of subsequent statements if the -s option is used. Otherwise, it will attempt to execute the next statement. (Recall that the -svtr options are set in the DB2OPTIONS variable.)

__ 14. Examine the contents of your report file:

more < clp.out

Does the report reflect everything you previously saw on the output display?

In this example, the output report only contains the results from the select statements. Messages concerning the failure of the last statement are not included.

__ 15. Issue the following command:

db2 ? options

Examine the output report file:

more < clp.out

Are the results from your last executed and prior commands reflected in the file?

_____

_____

_____

If a report file is specified on several CLP commands, the results from the most recent command will be appended to prior results. The user should periodically clean or delete this file.

__ 16. Update the DB2OPTIONS variable so that a report file is no longer used. Keep the other settings active:

set DB2OPTIONS=-svt (in AIX, **export DB2OPTIONS=-svt**)

__ 17. Verify the setting of the DB2OPTIONS environment variable by issuing the following command. (This command can be used to echo the setting of any environment variable. Ensure that the desired variable is bracketed by percent signs.)

echo %DB2OPTIONS% (in AIX, **echo $DB2OPTIONS**)

Note: If you cannot successfully set your DB2OPTIONS environment variable, contact the instructor. Other portions of this lab are dependent on proper setting of this variable.

Section 2 - Creating Lab Objects

__ 1. Issue the following command:

more < dept.mem

View the contents while reading the following information:

- This is the DDL that will CREATE the table called DEP. Since the name of the table is not qualified with an OWNER, your USERID will become the OWNER.
- A list of column definitions follows the create statement. Each item in the list will define a column name, its attribute, and null support characteristic. For example, the first column in the table will be known as DEPTNO. It will be a three-byte column that will contain CHARACTER data. This column will NOT allow NULL data.

  The third column will be known as MGRNO. It is a six-byte column that will contain CHARACTER data. Unlike the DEPTNO column, the MGRNO column will allow NULL data, since no explicit disallowance is specified.

  Other data types include INTEGER, SMALLINT, DECIMAL, DATE, TIME, TIMESTAMP, and VARCHAR. For further discussion of the attributes of these and other data types, consult the *SQL Reference Manual*.

The other specification regarding NULL is NOT NULL WITH DEFAULT. Since a default value was not specified in the command syntax, it is dependent on the type of column defined:

Data Type	Default Value
Numeric	0 (zero)
Fixed Length String	blanks
Variable Length String	String with 0 length
Date	Current Date
Time	Current Time
Timestamp	Current Timestamp

- After the column list, notice the specification of a PRIMARY KEY clause. In this example, the column DEPTNO is named as the PRIMARY KEY. If more than one column comprised the PRIMARY KEY, they would form a list within the parentheses, and each column would be separated by a comma.

  Any column named in the PRIMARY KEY clause must be defined as NOT NULL or NOT NULL WITH DEFAULT. Usually, NOT NULL is specified since the values in a primary key must be unique. An exception may be the use of timestamps. The default value for a timestamp is the CURRENT TIMESTAMP, which is granular to the microsecond.

- In order to enforce unique values in the primary key, DB2 will require a UNIQUE INDEX on the primary key columns. Since such an index cannot currently exist (the table is being defined in the statement), DB2 will automatically create the necessary index.

__ 2. Execute this file to create the table and verify successful execution:

db2 -f dept.mem

__ 3. Issue the following command:

more < emp.mem

View the contents while reading the following information:

- The first statement will create a table called EMP. Note the column types and null attributes specified.

  Referential integrity characteristics are **NOT** part of the new table definition.

- The CREATE INDEX statement creates a UNIQUE INDEX on what will be named the primary key. If a UNIQUE INDEX exists when a primary key is defined, the database manager will use the index as the primary key index.

- The ALTER statement will add the referential integrity characteristic we desire.

  The PRIMARY KEY clause identifies EMPNO as the only column in the primary key.

Note the FOREIGN KEY clause. It identifies the foreign key name (RIEMPDEP), the column(s) in the key (WORKDEPT), the parent table referenced (DEP), and the DELETE RULE (SET NULL). Notice that this delete rule is only possible because the foreign key column allows null values.

If the foreign key is not named explicitly, DB2 will create a name. The name can be 1 to 8 characters in length.

The foreign key columns must match the specification of the primary key columns in the referenced parent table with respect to type, size, and order.

No columns contained in the parent table need to be referenced. Since a table can have only one PRIMARY KEY, naming the parent table is sufficient.

Other delete rules include RESTRICT and CASCADE. If no rule is specified, RESTRICT is assumed.

Note: The ALTER statement was used for illustrative purposes. Tables with dependencies on other tables can be created using the CREATE statement with FOREIGN KEY clauses.

- The second index created is on the foreign key of the EMP table. Although this index is not required, it is strongly recommended if the dependent table will contain any significant volume of data. Otherwise, referential integrity constraint enforcement checking may be a poor performer.

__ 4. Execute this file to create the table and verify successful execution:

db2 -f emp.mem

__ 5. View the following input

d:\cg112\VIEW.MEM

more < view.mem

__ 6. Note the syntax of the CREATE VIEW statement. We are creating a view which will be a join of the EMP and DEPT tables. The view is called vphone. Execute the script.

__ 7. Execute this file to create the vphone view and verify successful execution. (0 rows are returned from the SELECT statement; the tables are still empty).

db2 -f view.mem

Section 3 - Impacts of Referential Integrity

__ 1. Look at the script file:

more < insert.mem

View the contents while reading the following information:

- Both statements have similar syntax.

- The first inserts all the rows from CF10.EMP into your EMP table.
- The second inserts all the rows from CF10.DEP into your DEP table.

  Note: This technique is useful to populate a sampling of data from a production table to a test table. WHERE clauses could be specified on the SUBSELECT to limit the data selected.

__ 2. Execute this file:

db2 -f insert.mem

Did the file execute successfully? Do you know why or why not?

_____

_____

_____

The file did not execute successfully. The first insert statement attempts to add rows into the table EMP, which is defined as a dependent of the table DEP. Since the DEP table is empty at this point, there are no PRIMARY KEY values that match the FOREIGN KEY values.

__ 3. Look at the script file:

more < insert2.mem

View the contents while reading the following information:

- The statements have been reordered so that the rows are loaded into the parent table first.
- The first inserts all the rows from CF10.DEP into your DEP table.
- The second inserts all the rows from CF10.EMP into your EMP table.

__ 4. Execute this file:

db2 -f insert2.mem

Did the file execute successfully? Why?

_____

_____

_____

The file executed successfully. Apparently, all of the FOREIGN KEY values inserted via the second SQL statement are either NULL, or have a matching PRIMARY KEY value. The PRIMARY KEY values were inserted in the first statement.

__ 5. Look at the script file:

more < updatepk.mem

View the contents while reading the following information:

- The company supported by the DEP table is reorganizing and needs to change the department numbers of two departments.
- Department D01 needs to be changed to department C01.
- Department D11 needs to become department D31.

__ 6. Execute the file to meet the needs of the business:

db2 -f updatepk.mem

Did the file execute successfully? Why or why not?

_____
_____
_____

The file did not execute successfully. The first statement failed because a duplicate value for the PRIMARY KEY would result. By definition, PRIMARY KEYS must be unique. This uniqueness is enforced via the required UNIQUE INDEX based on the PRIMARY KEY columns.

__ 7. It was determined that department D01 actually should be changed to department C10. The original UPDATE statement had a transposed value. Execute the following file, which corrects the transposed value:

Look at the script file D:\cg112\uppk2.mem and execute the script file.

db2 -f uppk2.mem

Why did the first UPDATE statement succeed and the second fail?

What must DB2 check to determine whether or not a row is a "parent" row?

What could assist DB2 in this check?

What is the name of the relationship that caused the second statement to fail?

_____
_____
_____
_____
_____
_____

The first UPDATE was successful for two reasons. The new value for the PRIMARY KEY was unique and the old value did not have any matching values in the FOREIGN KEY. The second UPDATE failed because it is not possible to update the key of a parent row. DB2 must check the rows in the dependent table to see if a row is a parent row. If an index is defined on the FOREIGN KEY column(s), DB2 could check the constraint via the index. The name of the relationship that caused the second statement to fail is RIEMPDEP.

__ 8. The company wants to change the department number of D11 to D31, but the UPDATE to the DEP table was prevented due to a constraint. Therefore, it has been decided to change the values in the EMP table first.

Look at the file d:\cg112\updatefk.mem.

more < updatefk.mem

Execute the following command which will attempt to update the rows in the EMP table:

db2 -f updatefk.mem

Did the statement succeed? Why or why not?

Can you propose the steps necessary to actually change the value of D11 to D31 in both tables? Contact your instructor or consult the solution for this question if you are not sure of your answer. **DO NOT** actually execute your solution.

_____

_____

_____

_____

_____

_____

The statement did not succeed because of the referential constraint. The value of D31 does not exist in the parent table. In order to accomplish this change, the company would need to INSERT a row in the parent table (DEP) with the value of D31, UPDATE the rows in the dependent table (EMP) to the value of D31, and then DELETE the row in the parent table (DEP) that contained the "old" value of D11.

__ 9. It has been determined that department D11 should be eliminated from the DEP table. View the contents of a file that will help demonstrate the impact of such an action:

more < D:\cg112\deletepk.mem

View the contents while reading the following information:

- The first SELECT shows the D11 row from the DEP table.
- The second SELECT shows three specific employee rows from the EMP table that work in department D11.
- The DELETE statement eliminates the D11 row from the DEP table.
- The third SELECT shows the same three employee rows displayed by the second SELECT.

__ 10. Execute the file:

db2 -f deletepk.mem

Did the DELETE statement succeed?

Did anything change in the three employee rows that were previously assigned to department D11? Why?

What would have happened if the DELETE RULE was CASCADE?

What would have happened if the DELETE RULE was RESTRICT?

Would the referential constraint defined between the DEP table and the EMP table prevent the deletion of a row from the EMP table? Does your answer depend on the DELETE RULE chosen?

_____
_____
_____
_____
_____
_____

The DELETE statement executed successfully. The value of WORKDEPT in the three rows examined that used to match the D11 PRIMARY KEY value have been set to a NULL. This exemplifies the DELETE RULE of SET NULL that was defined. If the DELETE RULE was CASCADE, all rows in the dependent table with WORKDEPT D11 would have been deleted. If the DELETE RULE was RESTRICT, the presence of dependent rows would have prevented the successful deletion of the parent row in the DEP table.

Referential constraints do not impact a DELETE statement on a dependent table. For example, the constraint between the DEP and EMP tables would not impact a DELETE from the EMP table. The DELETE RULE specified is irrelevant.

__ 11. The next set of SQL statements to be executed will delete the data from EMP, delete the data from DEP, and insert a fresh set of sample data into both tables. This "refresh" will ensure that the lab tables contain the correct data for subsequent lab exercises. Enter the following command and verify successful execution:

db2 -f refresh.mem

__ 12. Enter the following commands to terminate your CLP session:

db2 terminate

__ 13. Read the following summary regarding the impact of referential integrity on application table usage.

- Primary keys must be unique and cannot allow nulls. DB2 enforces uniqueness by requiring a unique index on the primary key columns.
- Foreign keys need not be unique and may allow nulls. DB2 can check foreign key column values efficiently when indexes are defined on the foreign key columns.

- A non-null foreign key value must have a matching value in the primary key, but a primary key value does not require a matching foreign key value.
- Any INSERT, UPDATE, or DELETE statement that violates any of the above rules will not execute successfully.

__ 14. Reset your db2options:

set DB2OPTIONS=

END OF LAB

Command Center Exercise Instructions

The lab solutions can be found in ”Command Center Exercise Solutions” on page 1-32.

Section 4 - Introduction to the Command Center

__ 1. Access the Control Center.

__ 2. From the Command Center, select the Interactive tab and type ? in the entry part of the screen.

__ 3. Look at the output from your ? command.

What is the command to get "help for reading help screens"?

_____

__ 4. Execute the command to get "help for reading help screens".

What does it mean if parameters are enclosed in brackets [ ]?

_____
_____
_____

__ 5. Look at the available options by choosing Command Center -> Options ... (Command Center pull-down menu selection).

What is the current setting for auto-commit?

What is the current setting for displaying the SQLCA?

_____
_____
_____

__ 6. Set your Command Center options to not display the SQLCA, to stop when a statement error is encountered and to echo the current command.

- Select Command Center pull-down menu
- Select Options... menu item
- Select Execution tab
- Check the box **Stop execution if errors occur**
- Select Results tab
- Check the box **Verbose (echo command text to output)**
- Uncheck the box **Display SQLCA data** if it is checked
- Click OK

__ 7. At this time set the termination character for a command to be ";". This will allow for multiple DB2 commands to be issued from the Command Center or a script file.

The statement termination character is set in the Tools Settings.

__ 8. Connect to database eddb.

```
connect to eddb user udba using udba
```

Run four scripts (DEPT.MEM, EMP.MEM, VIEW.MEM, and REFRESH.MEM) to create the dep and emp tables, create the vphone view, and populate the tables and view.

To create the dep table do the following:

- Choose the Script tab.
- From the menu, choose Script..Import
- Select Drive D:
- Under directories, double-click cg112
- Under files, select DEPT.MEM
- Click OK.
- Run the script to create the dep table.

Now, perform the same steps, but import the following three files and run in this order: EMP.MEM, VIEW.MEM, and REFRESH.MEM.

__ 9. We have supplied files that you will be using to work with DB2. The files are in the source subdirectory that you were told about at the beginning of these labs.

Examine the `sample.mem` file that contains SQL and comments that you will run in the next step.

How are comments designated?

What separates one SQL statement from another?

_____

_____

_____

__ 10. Execute the SQL in the file by pressing Ctrl-Enter to execute the statements.

What option caused the SQL statements in the file to be displayed?

Why is the output appearing on your screen?

What happens if an error is encountered?

_____

_____

_____

_____

_____

_____

Section 5 - Impacts of Referential Integrity

__ 1. Open the script file:

d:\cg112\insert.mem

View the contents while reading the following information:

- Both statements have similar syntax.
- The first inserts all the rows from CF10.EMP into your EMP table.
- The second inserts all the rows from CF10.DEP into your DEP table.

Note: This technique is useful to populate a sampling of data from a production table to a test table. WHERE clauses could be specified on the SUBSELECT to limit the data selected.

__ 2. Execute this file:

D:\cg112\insert.mem using the Command Center.

Did the file execute successfully? Do you know why or why not?

_____

_____

_____

__ 3. Open the script file:

D:\cg112\insert2.mem

View the contents while reading the following information:

- The statements have been reordered so that the rows are loaded into the parent table first.
- The first inserts all the rows from CF10.DEP into your DEP table.
- The second inserts all the rows from CF10.EMP into your EMP table.

__ 4. Execute this file:

- **Instead of selecting the three gears from the Command Center tool bar**
- **Select Script pull-down menu**
- **Select Execute menu item.**

Did the file execute successfully? Why?

_____

_____

_____

__ 5. Open the script file:

D:\cg112\updatepk.mem

View the contents while reading the following information:

- The company supported by the DEP table is reorganizing and needs to change the department numbers of two departments.
- Department D01 needs to be changed to department C01.
- Department D11 needs to become department D31.

__ 6. Execute the file to meet the needs of the business:

Select three gears icon from the Command Center tool bar.

Did the file execute successfully? Why or why not?

_____

_____

_____

__ 7. It was determined that department D01 actually should be changed to department C10. The original UPDATE statement had a transposed value. Execute the following file, which corrects the transposed value:

Open the script file D:\cg112\uppk2.mem and Execute the script file from the Command Center.

Why did the first UPDATE statement succeed and the second fail?

What must DB2 check to determine whether a row is a "parent" row?

What could assist DB2 in this check?

What is the name of the relationship that caused the second statement to fail?

_____

_____

_____

_____

_____

_____

__ 8. The company wants to change the department number of D11 to D31, but the UPDATE to the DEP table was prevented due to a constraint. Therefore, it has been decided to change the values in the EMP table first. Execute the following command which will attempt to update the rows in the EMP table:

Open the file D:\cg112\updatefk.mem and Execute the file from the Command Center.

Did the statement succeed? Why or why not?

Can you propose the steps necessary to actually change the value of D11 to D31 in both tables? Contact your instructor or consult the solution for this question if you are not sure of your answer. **DO NOT** actually execute your solution.

_____

_____
_____
_____
_____
_____

__ 9. It has been determined that department D11 should be eliminated from the DEP table. View the contents of a file that will help demonstrate the impact of such an action:

D:\cg112\deletepk.mem

View the contents while reading the following information:

- The first SELECT shows the D11 row from the DEP table.
- The second SELECT shows three specific employee rows from the EMP table that work in department D11.
- The DELETE statement eliminates the D11 row from the DEP table.
- The third SELECT shows the same three employee rows displayed by the second SELECT.

__ 10. Execute the file:

D:\cg112\deletepk.mem from the Command Center.

Did the DELETE statement succeed?

Did anything change in the three employee rows that were previously assigned to department D11? Why?

What would have happened if the DELETE RULE was CASCADE?

What would have happened if the DELETE RULE was RESTRICT?

Would the referential constraint defined between the DEP table and the EMP table prevent the deletion of a row from the EMP table? Does your answer depend on the DELETE RULE chosen?

_____
_____
_____
_____
_____
_____

__ 11. The next set of SQL statements to be executed will delete the data from EMP, delete the data from DEP, and insert a fresh set of sample data into both tables. This "refresh" will ensure that the lab tables contain the correct data for subsequent lab exercises. Enter the following command and verify successful execution:

Execute the file D:\cg112\refresh.mem from the Command Center.

__ 12. On the Script tab, choose the Interactive radio button, and enter the following commands to terminate your CLP session:

terminate

__ 13. Read the following summary regarding the impact of referential integrity on application table usage.

- Primary keys must be unique and cannot allow nulls. DB2 enforces uniqueness by requiring a unique index on the primary key columns.
- Foreign keys need not be unique and may allow nulls. DB2 can check foreign key column values efficiently when indexes are defined on the foreign key columns.
- A non-null foreign key value must have a matching value in the primary key, but a primary key value does not require a matching foreign key value.
- Any INSERT, UPDATE, or DELETE statement that violates any of the above rules will not execute successfully.

END OF LAB

Command Center Exercise Solutions

Section 4 - Introduction to the Command Center

__ 1. Access the Control Center.

Open the Command Center.

- Open the IBM DB2 folder.
- Open the General Administration Tools folder.
- Open the Control Center, you will be prompted for authentication.
- From the Control Center's tool bar, select the fifth icon from the tool bar to open the Command Center. (Optionally, from the Control Center, choose Tools -> Command Center to open the Command Center).

__ 2. From the Command Center, select the Interactive tab and type ? in the Command window.

- Press two gears icon at the top of the screen, or press Ctrl+Enter to execute the request

__ 3. Look at the output from your ? command.

What is the command to get "help for reading help screens"?

_____

? help

__ 4. Execute the command to get "help for reading help screens".

What does it mean if parameters are enclosed in brackets [ ]?

_____

_____

_____

On the Command Center, select Script tab, and enter `? help`

The brackets are used to indicate components of the command that are considered optional. Also, the | is used to designate choices. If an option itself has an optional component, the brackets will be nested.

__ 5. Look at the available options by choosing Command Center -> Options ... (Command Center pull-down menu selection).

What is the current setting for auto-commit?

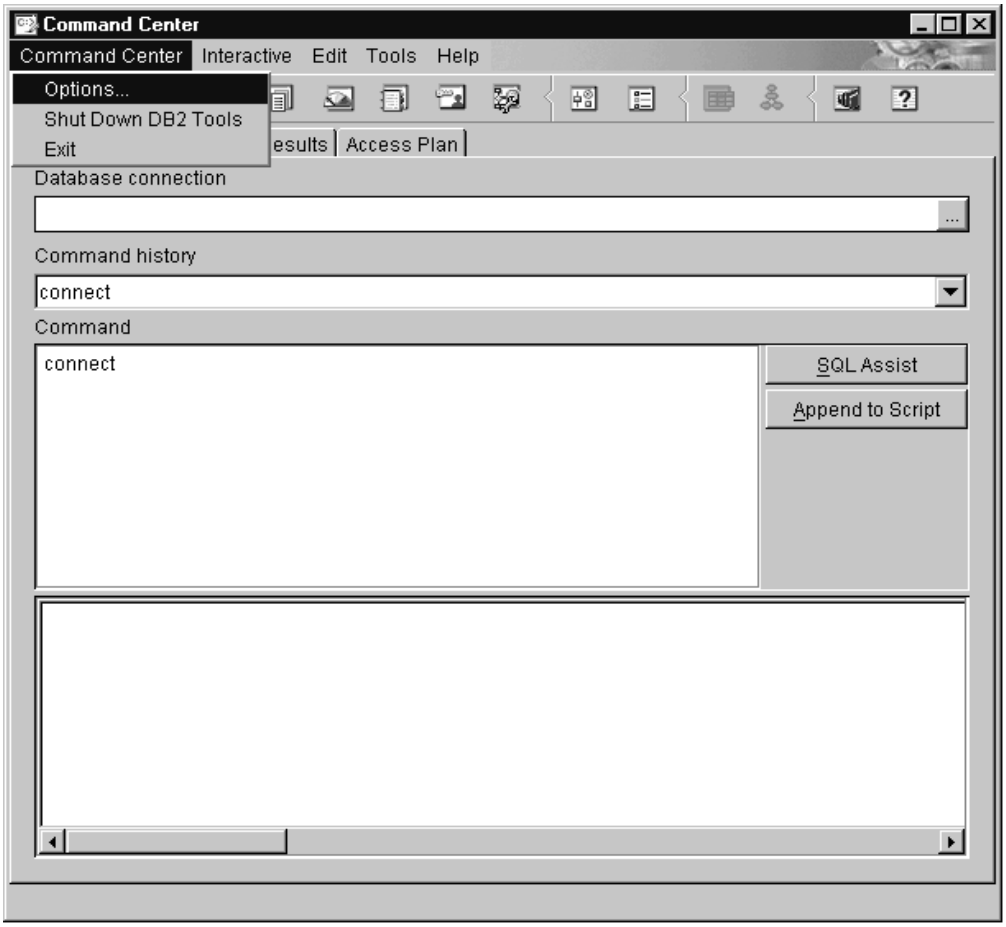
What is the current setting for displaying the SQLCA?

_____

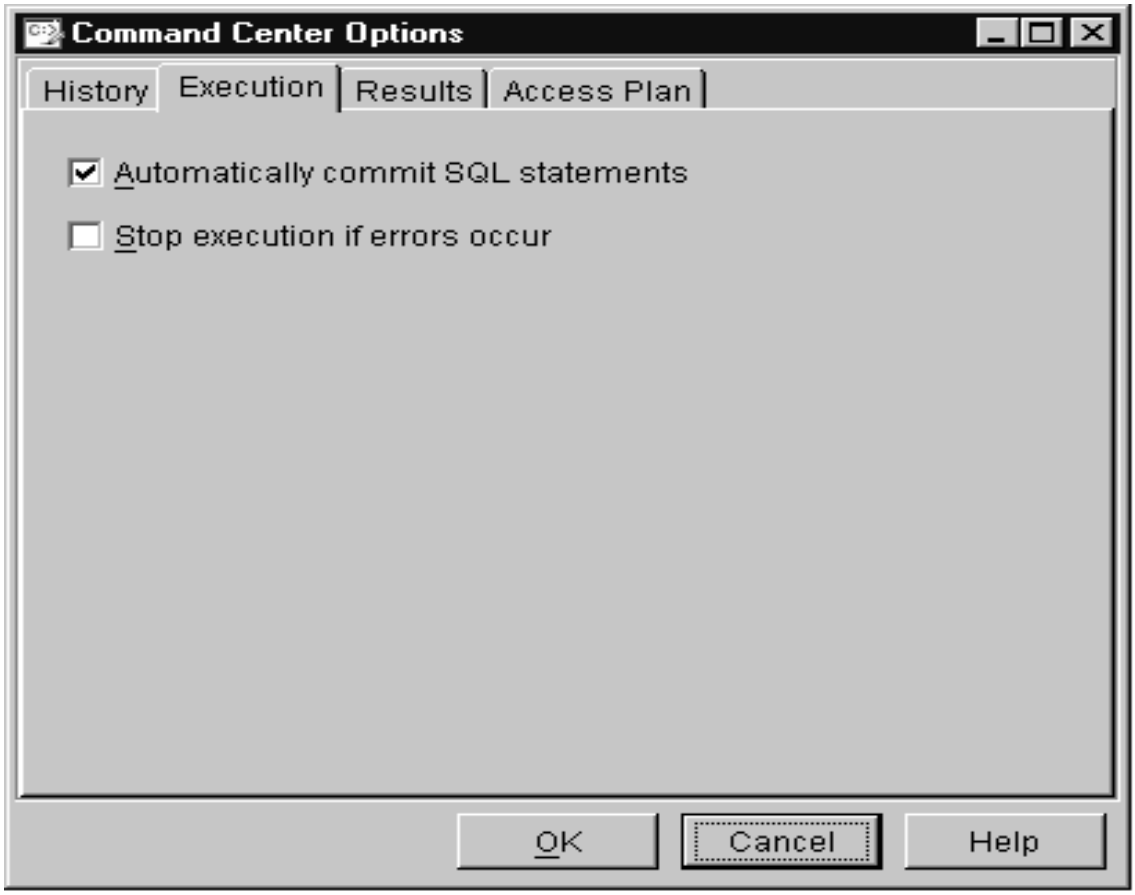
_____

_____

- Using the Command Center, select Command Center pull-down menu.
- Select Options... menu item.
- The Options panel will be displayed.



- Select Execution tab to determine if Auto-commit is set.
- Select Results tab to determine current setting for SQLCA.



Notice that Auto-commit is currently set to ON and the option to display the SQLCA is currently set to OFF. These settings are fine for the remainder of our labs.

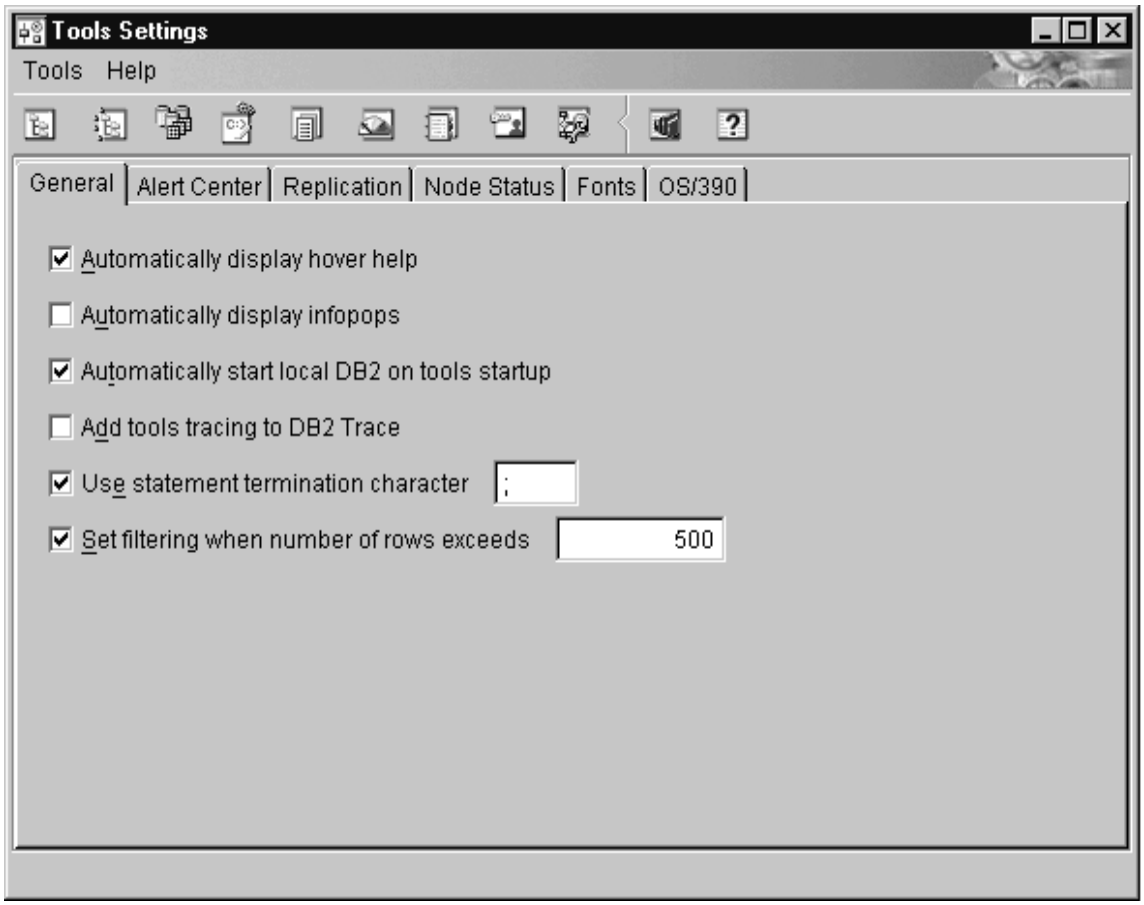
__ 6. Set your Command Center options to not display the SQLCA, to stop when a statement error is encountered and to echo the current command.

- Select Command Center pull-down menu.
- Select Options... menu item.
- Select Execution tab.
- Check the box **Stop execution if errors occur**.
- Select Results tab.
- Check the box **Verbose (echo command text to output)**.
- Uncheck the box **Display SQLCA data** if it is checked.
- Click OK.

__ 7. At this time, set the termination character for a command to be ";". This will allow for multiple DB2 commands to be issued from the Command Center or a script file.

The statement termination character is set in the Tools Settings.

- Select Tool Settings icon from the Command Center tool bar.
- It is the thirteenth icon from the left. The hover help should assist.



- Check the box Use statement termination character. (It should already be set.)
- The semicolon should be in the box to the right.

Close Tools Settings window.

__ 8. Connect to database eddb.

```
connect to eddb user udba using udba
```

Run four scripts (DEPT.MEM, EMP.MEM, VIEW.MEM, and REFRESH.MEM) to create the dep and emp tables, create the vphone view, and populate the tables and view.

To create the dep table do the following:

- Choose the Script tab.
- From the menu, choose Script..Import.
- Select Drive D:

- Under directories, double-click cg112
- Under files, select DEPT.MEM
- Click OK.
- Run the script to create the dep table.

Now, perform the same steps, but import the following three files and run in this order: EMP.MEM, VIEW.MEM, and REFRESH.MEM.

__ 9. We have supplied files that you will be using to work with DB2. The files are in the source subdirectory that you were told about at the beginning of these labs.

Examine the `sample.mem` file that contains SQL and comments that you will run in the next step.

How are comments designated?

What separates one SQL statement from another?

_____

_____

_____

From the Command Center,

- Select Script tab
- We are going to Open the script file sample.mem
- Select Script pull-down menu.
- Select Import... menu item
- You may get message DBA2172 - Do you want to discard your changes...? If so, click Yes.
- Specify the file D:\cg112\SAMPLE.MEM
  - On the Directories panel, select your system name at the top, and D: path at the bottom right.
  - Select directory CG112 (double-click).
  - From the Files panel, select the file sample.mem.
- Click OK command button.

Two hyphens identify comments.

SQL statements are separated via the semicolon.

__ 10. Execute the SQL in the file by pressing Ctrl-Enter to execute the statements.

What option caused the SQL statements in the file to be displayed?

Why is the output appearing on your screen?

What happens if an error is encountered?

_____

_____

_____

_____

_____

_____

On the Results panel you should see three queries that were executed.

The SQL statements in the file appear in the display because the verbose option, which was specified in the Options Panel, requests the commands to be echoed.

The output will always appear in the Results panel. We could place the output to a file by selecting the Options... in the Results panel Pipe output to a file.

If an error is encountered, DB2 will provide a message regarding this error. It will stop attempting execution of subsequent statements, if the option Stop execution if errors occur had been selected as a Command Center Option. Otherwise, it will attempt to execute the next statement.

Section 5 - Impacts of Referential Integrity

__ 1. Open the script file:

d:\cg112\insert.mem

View the contents while reading the following information:

- Both statements have similar syntax.
- The first inserts all the rows from CF10.EMP into your EMP table.
- The second inserts all the rows from CF10.DEP into your DEP table.

  Note: This technique is useful to populate a sampling of data from a production table to a test table. WHERE clauses could be specified on the SUBSELECT to limit the data selected.

__ 2. Execute this file:

D:\cg112\insert.mem using the Command Center.

Did the file execute successfully? Do you know why or why not?

_____

_____

_____

The file did not execute successfully. The first insert statement attempts to add rows into the table EMP, which is defined as a dependent of the table DEP. Since the DEP table is empty at this point, there are no PRIMARY KEY values that match the FOREIGN KEY values.

__ 3. Open the script file:

D:\cg112\insert2.mem

View the contents while reading the following information:

- The statements have been reordered so that the rows are loaded into the parent table first.
- The first inserts all the rows from CF10.DEP into your DEP table.
- The second inserts all the rows from CF10.EMP into your EMP table.

__ 4. Execute this file:

- **Instead of selecting the three gears from the Command Center tool bar, Select Script pull-down menu.**
- **Select Execute menu item.**

Did the file execute successfully? Why?

_____

_____

_____

The file executed successfully. Apparently, all of the FOREIGN KEY values inserted via the second SQL statement are either NULL, or have a matching PRIMARY KEY value. The PRIMARY KEY values were inserted in the first statement.

__ 5. Open the script file:

D:\cg112\updatepk.mem

View the contents while reading the following information:

- The company supported by the DEP table is reorganizing and needs to change the department numbers of two departments.
- Department D01 needs to be changed to department C01.
- Department D11 needs to become department D31.

__ 6. Execute the file to meet the needs of the business:

Select three gears icon from the Command Center tool bar.

Did the file execute successfully? Why or why not?

_____

_____

_____

The file did not execute successfully. The first statement failed because a duplicate value for the PRIMARY KEY would result. By definition, PRIMARY KEYS must be unique. This uniqueness is enforced via the required UNIQUE INDEX based on the PRIMARY KEY columns.

__ 7. It was determined that department D01 actually should be changed to department C10. The original UPDATE statement had a transposed value. Execute the following file, which corrects the transposed value:

Open the script file D:\cg112\uppk2.mem and Execute the script file from the Command Center.

Why did the first UPDATE statement succeed and the second fail?

What must DB2 check to determine whether or not a row is a "parent" row?

What could assist DB2 in this check?

What is the name of the relationship that caused the second statement to fail?

_____

_____

_____

_____

_____

_____

The first UPDATE was successful for two reasons. The new value for the PRIMARY KEY was unique and the old value did not have any matching values in the FOREIGN KEY. The second UPDATE failed because it is not possible to update the key of a parent row. DB2 must check the rows in the dependent table to see if a row is a parent row. If an index is defined on the FOREIGN KEY columns, DB2 could check the constraint via the index. The name of the relationship that caused the second statement to fail is RIEMPDEP.

__ 8. The company wants to change the department number of D11 to D31, but the UPDATE to the DEP table was prevented due to a constraint. Therefore, it has been decided to change the values in the EMP table first. Execute the following command which will attempt to update the rows in the EMP table:

Open the file D:\cg112\updatefk.mem and Execute the file from the Command Center.

Did the statement succeed? Why or why not?

Can you propose the steps necessary to actually change the value of D11 to D31 in both tables? Contact your instructor or consult the solution for this question if you are not sure of your answer. **DO NOT** actually execute your solution.

_____

_____

_____

_____

_____

_____

The statement did not succeed because of the referential constraint. The value of D31 does not exist in the parent table. In order to accomplish this change, the company would need to INSERT a row in the parent table (DEP) with the value of D31, UPDATE the rows in the dependent table (EMP) to the value of D31, and then DELETE the row in the parent table (DEP) that contained the "old" value of D11.

__ 9. It has been determined that department D11 should be eliminated from the DEP table. View the contents of a file that will help demonstrate the impact of such an action:

D:\cg112\deletepk.mem

View the contents while reading the following information:

- The first SELECT shows the D11 row from the DEP table.
- The second SELECT shows three specific employee rows from the EMP table that work in department D11.
- The DELETE statement eliminates the D11 row from the DEP table.
- The third SELECT shows the same three employee rows displayed by the second SELECT.

__ 10. Execute the file:

D:\cg112\deletepk.mem from the Command Center.

Did the DELETE statement succeed?

Did anything change in the three employee rows that were previously assigned to department D11? Why?

What would have happened if the DELETE RULE was CASCADE?

What would have happened if the DELETE RULE was RESTRICT?

Would the referential constraint defined between the DEP table and the EMP table prevent the deletion of a row from the EMP table? Does your answer depend on the DELETE RULE chosen?

_____

_____

_____

_____

_____

_____

The DELETE statement executed successfully. The value of WORKDEPT in the three rows examined that used to match the D11 PRIMARY KEY value have been set to a NULL. This exemplifies the DELETE RULE of SET NULL that was defined. If the DELETE RULE was CASCADE, all rows in the dependent table with WORKDEPT D11 would have been deleted. If the DELETE RULE was

RESTRICT, the presence of dependent rows would have prevented the successful deletion of the parent row in the DEP table.

Referential constraints do not impact a DELETE statement on a dependent table. For example, the constraint between the DEP and EMP tables would not impact a DELETE from the EMP table. The DELETE RULE specified is irrelevant.

__ 11. The next set of SQL statements to be executed will delete the data from EMP, delete the data from DEP, and insert a fresh set of sample data into both tables. This "refresh" will ensure that the lab tables contain the correct data for subsequent lab exercises. Enter the following command and verify successful execution:

Execute the file D:\cg112\refresh.mem from the Command Center.

__ 12. On the Script tab, choose the Interactive radio button, and enter the following commands to terminate your CLP session:

terminate

__ 13. Read the following summary regarding the impact of referential integrity on application table usage.

- Primary keys must be unique and cannot allow nulls. DB2 enforces uniqueness by requiring a unique index on the primary key columns.
- Foreign keys need not be unique and may allow nulls. DB2 can check foreign key column values efficiently when indexes are defined on the foreign key columns.
- A non-null foreign key value must have a matching value in the primary key, but a primary key value does not require a matching foreign key value.
- Any INSERT, UPDATE, or DELETE statement that violates any of the above rules will not execute successfully.

END OF LAB

Exercise 2. JDBC Programming I

What This Exercise Is About

You will be coding an application that uses dynamic SQL in Java (JDBC) capabilities to read data from the STAFF table.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- How to create the SAMPLE database
- Describe basic flow of an application to retrieve data
- Code the statement to load the DB2 driver
- Code statement to connect to the Sample database

Introduction

Modify a program to retrieve data from the database SAMPLE.

Instructor Notes

Introduction — The introduction to this first JDBC lab will give students an easy program to build confidence. The students will create the database SAMPLE. Use the command from the DB2 command line processor.

db2sampl -k

The SAMPLE database is created with several tables that will contain the schema of the logged on user in this case UDBA.

The first program **labstaff.java** will have the students create a ResultSet that will contain one row from the STAFF table. You may want to review how to define a Connection object. Connection sample = DriverManager.getConnection("jdbc:db2:sample");.

Have the students look at the Java source code.

The second part of the exercise will have the student code the Java program, **labupdate.java**, to update all the salaries for department. The students need to define a PreparedStatement object. You may want to review parameter markers. The students must determine the number of rows, that were updated by the SQL statement.

Since the lab will update all the employees of a particular department, have the students
Issue: SELECT * FROM STAFF WHERE DEPT = 84
Execute the program labupdate and supply the dept of 84.
Issue: SELECT * FROM STAFF WHERE DEPT = 84

Ask the students did the SALARY change for any user? The answer is no, because no commit work was issued.

Time for Lab — 30 minutes.

Things to Review at End of Lab — Many times you will have students that do not know how to program in Java. It would be prudent to cover the try … catch … exception block. Review how to handle a ResultSet and the next method that is used to traverse the ResultSet in a forward direction.

Exercise 2-1 Instructions - JDBC Usage

Modify a program to build a result set.
The program **labstaff.java** will issue a SELECT statement retrieving the columns NAME, JOB, SALARY from the STAFF table.

Section 0 - Introduction to the Lab Program

__ 1. Create the SAMPLE database.

_____

Section 1 - Modify the program labstaff.java

__ 1. Code the Connect statement. Connect to the SAMPLE database. Connect to the database using the userid udba and password udba.

_____

__ 2. Instantiate the statement object stmt using the createStatement methods.

_____

__ 3. Define the ResultSet object. Have the ResultSet object use the statement object named stmt. The ResultSet object will execute the query SELECT NAME, JOB, SALARY FROM STAFF WHERE ID = 10.

_____

__ 4. Code ResultSet method to move to the next row.

_____

__ 5. Code the ResultSet method that will move to the next row.

_____

__ 6. Compile the java program labstaff.java.

_____

__ 7. Execute the program.

_____

Exercise 2-1 Solutions - JDBC Usage

Modify a program to build a result set.
The program **labstaff.java** will issue a SELECT statement retrieving the columns NAME, JOB, SALARY from the STAFF table.

Section 0

__ 1. Create the SAMPLE database.

db2sampl -k

Section 1 - Modify the program labstaff.java

__ 1. Code the Connect statement. Connect to the SAMPLE database. Connect to the database using the userid udba and password udba.

Connection sample =
DriverManager.getConnection("jdbc:db2:sample","udba","udba");

__ 2. Instantiate the statement object stmt using the createStatement methods.

Statement stmt = sample.createStatement();

__ 3. Define the ResultSet object. Have the ResultSet object use the statement object named stmt. The ResultSet object will execute the query SELECT NAME, JOB, SALARY FROM STAFF WHERE ID = 10.

ResultSet rs = stmt.executeQuery("select NAME, JOB, SALARY from staff Where ID = 10");

__ 4. Code ResultSet method to move to the next row.

boolean more = rs.next();

__ 5. Code the ResultSet method that will move to the next row.

more = rs.next();

__ 6. Compile the java program labstaff.java.

javac labstaff.java

__ 7. Execute the program.

java labstaff

```
C:\cg112\java labstaff
Connect statement follows:
Connect completed

NAME      JOB   SALARY
-------   ---   --------
Sanders   Mgr   18357.50
```


Exercise 2-2 Instructions - JDBC Update Statement

This program, **labupdate.java**, will update all the rows of a particular department in the STAFF table. The salary of all the employees of that department will be increase by 5 percent. The program will return the number of rows updated by the program.

Section 1

Update the program **labupdate.java** to update all the rows in the department number 10 to receive a 5 percent update in salary.

__ 1. Code the necessary import statements to run a JDBC program.

_____

__ 2. Code the statement to load the DB2 Driver.

_____

__ 3. Code the statement that will connect to DB2 database SAMPLE. Define the connection object named **sample**.

_____

__ 4. Set the program to turn Automatic commit off. This way the updates will not automatically be applied to the SAMPLE database.

_____

__ 5. Create a PreparedStatement object name **pstmt**. This object will use the sqlstmt string variable that has been define with the value of:

```
"UPDATE STAFF SET SALARY = SALARY * 1.05 WHERE DEPT = ?"
```

You should use the prepareStatement( ) method to instantiate the object pstmt.

_____

__ 6. Set parameter marker variable to be valued received from the console. The variable name containing the console value is **mydeptno**.

_____

__ 7. At this time add the instruction that will execute the update instruction. Save the number of rows that were updated in the variable named **updateCount**.

_____

__ 8. Code the statements to retrieve:

The SQLCODE into the variable **SQLCode**
The SQLSTATE into the variable **SQLState**
The SQLERRMC into the variable **Message**.

_____

_____

_____

__ 9. Compile the program labupdate.java.

_____

__ 10. Execute the program.

_____

END OF LAB

Exercise 2-2 Solutions - JDBC Update Statement

This program, **labupdate.java**, will update all the rows of a particular department in the STAFF table. The salary of all the employees of that department will be increase by 5 percent. The program will return the number of rows updated by the program.

Section 1

Update the program **labupdate.java** to update all the rows in the department number 10 to receive a 5 percent update in salary.

__ 1. Code the necessary import statements to run a JDBC program.

import java.sql.*;

__ 2. Code the statement to load the DB2 Driver.

Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");

__ 3. Code the statement that will connect to DB2 database SAMPLE. Define the connection object named **sample**.

Connection sample = DriverManager.getConnection("jdbc:db2:sample");

__ 4. Set the program to turn Automatic commit off. This way the updates will not automatically be applied to the SAMPLE database.

sample.setAutoCommit(false);

__ 5. Create a PreparedStatement object name **pstmt**. This object will use the sqlstmt string variable that has been define with the value of:

```
"UPDATE STAFF SET SALARY = SALARY * 1.05 WHERE DEPT = ?"
```

You should use the prepareStatement( ) method to instantiate the object pstmt.

PreparedStatement pstmt = sample.prepareStatement(sqlstmt);

__ 6. Set parameter marker variable to be valued received from the console. The variable name containing the console value is **mydeptno**.

pstmt.setInt(1, mydeptno)

__ 7. At this time add the instruction that will execute the update instruction. Save the number of rows that were updated in the variable named **updateCount**.

int updateCount = pstmt.executeUpdate( );

__ 8. Code the statements to retrieve:

The SQLCODE into the variable **SQLCode**
The SQLSTATE into the variable **SQLState**
The SQLERRMC into the variable **Message**.

```
SQLCode = x.getErrorCode();
SQLState = x.getSQLState();
String Message = x.getMessage();
```

__ 9. Compile the program labupdate.java.

```
javac labupdate.java
```

__ 10. Execute the program.

```
java labupdate
```

END OF LAB

Exercise 3. JDBC Programming II

What This Exercise Is About

You will be coding an application that uses dynamic SQL in Java (JDBC) capabilities. Use an SQL statement to code a parameter marker.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Describe the basic JDBC statements to retrieve data in a ResultSt
- Code the appropriate import statements to include SQL classes and packages
- Use db2dclgn to provide definitions for the STAFF table
- Assign input variable values to parameter markers
- Retrieve DB2 column data

Introduction

Modify the program to code the appropriate JDBC statements.

Instructor Notes

Introduction — This lab will have the 4 sections.

Section 1 will have the students create and populate a result set. The lab will require the usage of SQL warnings, the getWarnings() method and parameter markers.

Section 2 will demonstrate to the students how to position in a result set.

Section 3 will use the addbatch to processing multiple SQL commands at the same time eliminating network traffic.

Section 4 will test the students ability to use metadata to determine information in a database.

Time for Lab — 1 hours and 30 minutes.

Things to Review at End of Lab — How to set AutoCommit on would be something that should be review along with handling of errors. The positioning in a result set is quite important. Metadata is normally only required for complex operations. We only looked at the tables and views in this program but the metadata methods can be used to determine the number of parameters required for a stored procedure.

3.1 Exercise Instructions

The objective of this exercise to show how to use a parameter marker in an SQL statement.

Section 1

Modify the program **labstaff1.java** to issue a
SELECT NAME, JOB, SALARY FROM STAFF WHERE DEPT = ?

__ 1. Define the DB2 driver.

_____

__ 2. Code a variable SQLWarn that will be used to hold SQLWarnings.

_____

__ 3. Connect to the database SAMPLE. Provide the userid udba and the password udba.

_____

__ 4. Instantiate the preparedStatement object stmt. Use the SQL Statement SELECT NAME, JOB, SALARY FROM STAFF WHERE DEPT = ?

_____

__ 5. Set the parameter marker to be department number. The variable mydeptno will contain the integer value of the department.

_____

__ 6. Instantiate the ResultSet Object named rs. Use the previously created preparedStatement object named stmt. Use the executeQuery( ) method.

_____

__ 7. If a warning occurs, display the warning.

_____

__ 8. Use the next( ) method to find the first row of the result set. Verify if there is row using the Boolean function.

_____

__ 9. Move to the next row of the ResultSet.

_____

__ 10. Compile the program.

_____

__ 11. Execute the java program.

_____

3.1 Exercise Solutions

Section 1

Modify the program **labstaff1.java** to issue a
SELECT NAME, JOB, SALARY FROM STAFF WHERE DEPT = ?

__ 1. Define the DB2 driver.

Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver")

__ 2. Code a variable SQLWarn that will be used to hold SQLWarnings.

SQLWarning SQLWarn = null;

__ 3. Connect to the database SAMPLE. Provide the userid udba and the password udba.

Connection sample = DriverManager.getConnection("jdbc:db2:sample", "udba","udba");

__ 4. Instantiate the preparedStatement object stmt. Use the SQL Statement SELECT NAME, JOB, SALARY FROM STAFF WHERE DEPT = ?

```
PreparedStatement stmt = sample.prepareStatement(
  "select id, name,salary from staff where Dept = ?");
```

__ 5. Set the parameter marker to be department number. The variable mydeptno will contain the integer value of the department.

stmt.setInt(1, mydeptno);

__ 6. Instantiate the ResultSet Object named rs. Use the previously created preparedStatement object named stmt. Use the executeQuery( ) method.

ResultSet rs = stmt.executeQuery();

__ 7. If a warning occurs, display the warning.

if ((SQLWarn = stmt.getWarnings()) != null)

__ 8. Use the next( ) method to find the first row of the result set. Verify if there is row using the Boolean function.

boolean more = rs.next();

__ 9. Move to the next row of the ResultSet.

more = rs.next();

__ 10. Compile the program.

javac labstaff1.java

__ 11. Execute the java program.

```
C:\metlife>java labstaff1
Set AutoCOmmit off
Autocommit off
Enter the Department number
10
mydeptno = 10 deptno = 010
PreparedStatement follows
JOB   NAME          SALARY
---   -----------   --------
160   Molinare      22959.20
210   Lu            20010.00
240   Daniels       19260.25
260   Jones         21234.00
```


Exercise 3.2 Using JDBC 2.0 Features

What This Exercise Is About

You will be coding an application that uses positioning methods in JDBC 2.0. Code JDBC statements that will allow you to scroll through a resultset.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Code the connection statement
- Code the appropriate JDBC statements that will allow you scroll through a ResultSet
- Code an application to use the addbatch method
- Code an application to use metadata methods

Introduction

Modify the program to code the appropriate JDBC statements.

3.2 Exercise Instructions

Section 2 - Modify the Program labposition.java

Connect to SAMPLE database.
Move to the first row of the table and print out that value.
Move to the last row of the ResultSet and print that value.
Move to the previous row and print that value.
Move to the first row of the result set and print the value.

__ 1. Code a Java statement that will connect to the SAMPLE database using the userid udba and the password udba.

_____

__ 2. Define a ResultSet name rs and set its initial value to be null.

_____

__ 3. Define a string named sql. This string will contain a select statement that will contain all the row for the table STAFF and the columns: NAME, JOB and SALARY.

_____

__ 4. Define a PreparedStatement object named stmt and instantiate the statement to be a scrollable cursor.

_____

__ 5. Code a ResultSet method to move to the next row in the ResultSet.

_____

__ 6. Code a ResultSet method to move to the last row of the ResultSet.

_____

__ 7. Code a ResultSet method to move to the previous row of the ResultSet.

_____

__ 8. Code a ResultSet method to move to the first row of the ResultSet.

_____

__ 9. Compile the Java program.

_____

__ 10. Execute the program.

_____

3.2 Exercise Solutions

Section 2 - Modify the Program labposition.java

Connect to SAMPLE database.
Move to the first row of the table and print out that value.
Move to the last row of the ResultSet and print that value.
Move to the previous row and print that value.
Move to the first row of the result set and print the value.

__ 1. Code a Java statement that will connect to the SAMPLE database using the userid udba and the password udba.

Connection sample =
DriverManager.getConnection("jdbc:db2:sample","udba","udba");

__ 2. Define a ResultSet name rs and set its initial value to be null.

ResultSet rs = null;

__ 3. Define a string named sql. This string will contain a select statement that will contain all the row for the table STAFF and the columns: NAME, JOB and SALARY.

String sql = "select NAME, JOB, SALARY from staff ";

__ 4. Define a PreparedStatement object named stmt and instantiate the statement to be a scrollable cursor.

PreparedStatement stmt = sample.prepareStatement(sql,
rs.TYPE_SCROLL_INSENSITIVE, rs.CONCUR_READ_ONLY);

__ 5. Code a ResultSet method to move to the next row in the ResultSet.

rs.next();

__ 6. Code a ResultSet method to move to the last row of the ResultSet.

rs.last();

__ 7. Code a ResultSet method to move to the previous row of the ResultSet.

rs.previous();

__ 8. Code a ResultSet method to move to the first row of the ResultSet.

rs.first();

__ 9. Compile the Java program.

javac labposition.java

__ 10. Execute the program.

java labposition

```
Connect statement follows:
```

```
Connect completed
Statement stmt follows
NAME      JOB     SALARY
Sanders   Mgr     18357.50
Gafney    Clerk   13030.50
Edwards   Sales   17844.00
Sanders   Mgr     18357.50
```

Exercise 3.3 How to Code Add Batch Method

3.3 Exercise Instructions

Introduction

The addbatch( ) method is new to JDBC 2.0. The DB2 system must be updated to use the new JDBC 2.0. This was done on the previous step using usejdbc2.

Modify the program labaddbatch.java. This program will add 5 rows to table DEPARTMENT using the executeBatch command.

Section 1 - Program Editing

__ 1. Code the connect statement to connect to the SAMPLE database.

_____

__ 2. Turn off the AutoCommit feature for this application.

_____

__ 3. Instantiate the Statement object stmt.

_____

__ 4. Add a an INSERT statement to the list of commands associated with the Statement object stmt.

_____

__ 5. Repeat the step above four times.

_____

__ 6. Execute the batch command. Save row count into an array UpdateCounts.

_____

__ 7. Commit the logical unit of work.

_____

__ 8. Compile the program.

_____

__ 9. Execute the program.

_____

__ 10. Verify the rows have been added to the department table.

_____

_____

3.3 Exercise Solutions

Section 1 - Program Editing

__ 1. Code the connect statement to connect to the SAMPLE database.

Connection sample =
DriverManager.getConnection("jdbc:db2:sample","udba","udba");

__ 2. Turn off the AutoCommit feature for this application.

sample.setAutoCommit(false);

__ 3. Instantiate the Statement object stmt.

Statement stmt = sample.createStatement();

__ 4. Add a an INSERT statement to the list of commands associated with the Statement object stmt.

stmt.addBatch("INSERT INTO UDBA.DEPARTMENT " +

"VALUES ('BT6','BATCH6 NEWYORK','BBBBB1','BTT','NEW YORK CITY6')");

__ 5. Repeat the step above four times.

```
stmt.addBatch("INSERT INTO UDBA.DEPARTMENT " +
"VALUES ('BT7','BATCH7 NEWYORK','BBBBB2','BT2','NEW YORK CITY7')");
stmt.addBatch("INSERT INTO UDBA.DEPARTMENT " +
"VALUES ('BT8','BATCH8 NEWYORK','BBBBB3','BT3','NEW YORK CITY8')");
stmt.addBatch("INSERT INTO UDBA.DEPARTMENT " +
"VALUES ('BT9','BATCH9 NEWYORK','BBBBB4','BT4','NEW YORK CITY9')");
stmt.addBatch("INSERT INTO UDBA.DEPARTMENT " +
"VALUES ('BTA','BATCH10 NEWYORK','BBBBB5','BT5','NEW YORK CITY10')");
```

__ 6. Execute the batch command. Save row count into an array UpdateCounts.

int [] updateCounts = stmt.executeBatch();

__ 7. Commit the logical unit of work.

sample.commit();

__ 8. Compile the program.

javac labaddbatch.java

__ 9. Execute the program.

java labaddbatch

__ 10. Verify the rows have been added to the department table.

```
db2 connect to sample user udba using udba
db2 select * from department
```

Exercise 3.4 Using MetaData() Methods with Java

3.4 Exercise Instructions

This exercise will demonstrate how to handle metadata. Determine all the tables for the user udba. Modify the program labTables.java to print all the tables that are created with the schema UDBA.

Create the DatabaseMetaData object to retrieve all the tables for a particular schema. Use the DatabaseMetaData method getTables.

ResultSet **getTables** (String catalog, String SchemaPattern, String TablePattern,
String types[ ])

Parameters:

catalog	a String object representing a catalog name. DB2 UDB will always use a null as a parameter
schemaPattern	a String object representing a schema pattern.
tableNamePattern	a String object representing a table name pattern

Returns:
A ResultSet object.

The following columns are returned.

1. TABLE_CAT with DB2 UDB this column value will be null.
2. TABLE_SCHEM String object giving the table schema.
3. TABLE_NAME String object giving the table name.
4. TABLE_TYPE String object giving the table type. Typical types are "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS:, "SYSNONYM"

Example:

```
String [ ] tableTypes = { "TABLE", "VIEW" }
ResultSet rs = dbmd.getTables( null, "SCHEMA", "TABLES%", tableTypes)
```

The ResultSet rs will contain one row for each column.

Section 1

Modify the program labTables.java to find all the tables for the schema UDBA.

__ 1. Code the import statement to include the required SQL packages and interfaces.

_____

__ 2. Code the statement which will load the DB2Driver.

_____

__ 3. Code a java statement that will instantiate the connection object sample. Connect to the database named SAMPLE.

_____

__ 4. Create the DatabaseMetaData object named dbmd. Use the method getMetaData( ) to instantiate the object dbmd.

_____

__ 5. Define a String that is an array named tableType. The array should be created so that when used with the method getTables( ) will return the tables and views for a particular schema.

_____

__ 6. Instantiate the ResultSet object named rs. The ResultSet rs will be used to return all the tables who have a schema name of UDBA.

_____

__ 7. Compile the program.

_____

__ 8. Execute the program.

_____

END OF LAB

3.4 Exercise Solutions

Section 1

Modify the program labTables.java to find all the tables for the schema UDBA.

__ 1. Code the import statement to include the required SQL packages and interfaces.

import java.sql.*;

__ 2. Code the statement which will load the DB2Driver.

Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");

__ 3. Code a java statement that will instantiate the connection object sample. Connect to the database named SAMPLE.

Connection sample = DriverManager.getConnection("jdbc:db2:sample");

__ 4. Create the DatabaseMetaData object named dbmd. Use the method getMetaData( ) to instantiate the object dbmd.

DatabaseMetaData dbmd = sample.getMetaData();

__ 5. Define a String that is an array named tableType. The array should be created so that when used with the method getTables( ) will return the tables and views for a particular schema.

String [] tableTypes = {"TABLE", "VIEW"};

__ 6. Instantiate the ResultSet object named rs. The ResultSet rs will be used to return all the tables who have a schema name of UDBA.

ResultSet rs = dbmd.getTables(null, "UDBA", "%",tableTypes);

__ 7. Compile the program.

javac labTables.java

__ 8. Execute the program.

java labTables

END OF LAB

Exercise 4. SQLJ Programming

What This Exercise Is About

You will be coding an application that uses embedded SQL to SELECT an employee from the STAFF table.

What You Should Be Able to Do

At the end of the lab, you should be able to:

- Code Embedded SQL statements in an SQLJ program
- Describe basic flow of an application to retrieve data using SQLJ statements
- Code a statement to load the DB2 driver
- Code statement to connect to the SAMPLE database
- Precompile an SQLJ program
- Bind and create package for an SQLJ program

Introduction

Modify a program to retrieve data from the database SAMPLE.

Instructor Exercise Overview

Introduction — The lab will have two parts.

The first lab will take the program **labstatic.sqlj**. Students must know the import commands to run SQLJ programs. This program is primarily to show students how to pre-compile a program using SQLJ and creating packages using db2sqljcustomize. The program is an example of a single row select. If more than one row is returned SQLCODE = -811 is returned. When no package option is used 4 packages will be created with the schema name of "NULLID". A package is created for each of the isolation levels.

In Section 2, the student will create an execution context to read through a result set using SQLJ. The student will be shown how to set options using the bindoption parameter.

Note: bindoption="package using package_name" is not supported in Version 8. Explain to students package name should not be greater than 7 characters.

Time for Lab — 1 hour.

Things to Review at End of Lab — It would be imperative to look at the packages created for each db2sqljcustomize executed. Notice the four packages created when no bindoption package was used.

Exercise Instructions - Section 1

Introduction

This exercise will attempt to demonstrate a simple SQLJ program accessing the sample database. Modify the program labstatic.sqlj. The program will issue a SELECT statement using the SQLJ construct. This SELECT statement will retrieve a single row from the table staff.

Program Editing

__ 1. Code the java statement to import the packages required to run SQLJ application.

_____

__ 2. Code the connect statement. Connect to the SAMPLE database using the userid udba and the password udba.

_____

__ 3. Define the default context named ctx.

_____

__ 4. Have the default context for the application to be ctx.

_____

__ 5. Code a single row select statement that will retrieve the columns NAME, YEARS, SALARY from the STAFF table and place the retrieved column values in the variables name, years and salary. Retrieve the rows for the ID = 10.

_____

__ 6. Retrieve the warnings from the executed SQL statement. Place the warnings in the SQLWarn variable. It was previously defined SQLWarning SQLWarn .

_____

__ 7. Precompile the program labstatic.sqlj.

_____

__ 8. Compile the program.

_____

__ 9. Bind the program labstatic into the database SAMPLE.

_____

__ 10. Verify the package exists using the list packages command.

_____

__ 11. Execute the program.

_____

END OF LAB

Exercise Solutions - Section 1

Introduction

This exercise will attempt to demonstrate a simple SQLJ program accessing the sample database. Modify the program labstatic.sqlj. The program will issue a SELECT statement using the SQLJ construct. This SELECT statement will retrieve a single row from the table staff.

Program Editing

__ 1. Code the java statement to import the packages required to run SQLJ application.

import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

__ 2. Code the connect statement. Connect to the SAMPLE database using the userid udba and the password udba.

Connection sample =
DriverManager.getConnection("jdbc:db2:sample","udba","udba");

__ 3. Define the default context named ctx.

DefaultContext ctx = new DefaultContext(sample);

__ 4. Have the default context for the application to be ctx.

DefaultContext.setDefaultContext(ctx);

__ 5. Code a single row select statement that will retrieve the columns NAME, YEARS, SALARY from the STAFF table and place the retrieved column values in the variables name, years and salary. Retrieve the rows for the ID = 10.

```
#sql { SELECT NAME , YEARS ,SALARY
         INTO :name, :years, :salary
       FROM STAFF
       WHERE ID = 10 };
```

__ 6. Retrieve the warnings from the executed SQL statement. Place the warnings in the SQLWarn variable. It was previously defined SQLWarning SQLWarn .

SQLWarn = sample.getWarnings();

__ 7. Precompile the program labstatic.sqlj.

sqlj -url=jdbc:db2:sample -compile=false labstatic.sqlj

__ 8. Compile the program.

javac labstatic.java

__ 9. Bind the program labstatic into the database SAMPLE.

db2sqljcustomize -url jdbc:db2://localhost:50000/sample
-user udba -password udba labstatic_SJProfile0.ser

Note: localhost is the TCP/IP address of this machine 127.0.0.1.
50000 is the TCP/IP services port of the instance named DB2.

__ 10. Verify the package exists using the list packages command.

db2 connect to sample
db2 list packages for schema nullid

__ 11. Execute the program.

java labstatic

END OF LAB

Exercise Instructions - Section 2

Introduction

This program labresultset.java will retrieve rows using SQLJ. Define an iterator that will handle the ResultSet.

Program Editing

__ 1. Define a positional iterator named Staff1. The iterator will handle three values. The first parameter will be a String variable. The second parameter will be an integer variable. The third parameter will be a float variable.

_____

__ 2. Load the DB2 driver.

_____

__ 3. Define the Execution Context named execCtx.

_____

__ 4. Create the Connection object named con.

_____

__ 5. Code the connect statement.

_____

__ 6. Execute the SELECT statement. This SELECT statement define the ResultSet and assigns the iterator results. Use the SELECT statement:
SELECT NAME, YEARS, SALARY FROM STAFF WHERE DEPT = :indeptno

_____

__ 7. Code a Fetch statement using the Execution Context execCtx. Fetch the 3 columns into variables name, years, and salary.

_____

__ 8. Precompile the program labresultset.sqlj using the SQLJ command.

_____

__ 9. Compile the program labresultset.java.

_____

__ 10. Create the package in the database SAMPLE.

_____

__ 11. Execute the program.

_____

END OF LAB

Exercise Solutions - Section 2

Introduction

This program labresultset.java will retrieve rows using SQLJ. Define an iterator that will handle the ResultSet.

Program Editing

__ 1. Define a positional iterator named Staff1. The iterator will handle three values. The first parameter will be a String variable. The second parameter will be an integer variable. The third parameter will be a float variable.

#sql iterator Staff1(String, int, float);

__ 2. Load the DB2 driver.

Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");

__ 3. Define the Execution Context named execCtx.

ExecutionContext execCtx = new ExecutionContext();

__ 4. Create the Connection object named con.

Connection con = null;

__ 5. Code the connect statement.

con = DriverManager.getConnection("jdbc:db2:sample","udba","udba");

__ 6. Execute the SELECT statement. This SELECT statement define the ResultSet and assigns the iterator results. Use the SELECT statement:
SELECT NAME, YEARS, SALARY FROM STAFF WHERE DEPT = :indeptno

```
#sql results =  { SELECT NAME , YEARS ,SALARY
        FROM UDBA.STAFF
        WHERE DEPT = :indeptno };
```

__ 7. Code a Fetch statement using the Execution Context execCtx. Fetch the three columns into variables name, years, and salary.

#sql [execCtx] {Fetch :results INTO :name, :years, :salary };

__ 8. Precompile the program labresultset.sqlj using the SQLJ command.

sqlj -url=jdbc:db2:sample -compile=false labresultset.sqlj

__ 9. Compile the program labresultset.java.

javac labresultset.java

__ 10. Create the package in the database SAMPLE.

db2sqljcustomize -url=jdbc:db2:localhost:5000/sample
-user udba -password udba labresultset_SJProfile0.ser

__ 11. Execute the program.

C:\CG112>**java labresultset**

```
NAME            YEARS          SALARY
--------------- ----------     -------------

Lu              10             20010.00
Daniels         5              19260.25
Jones           12             21234.00
Molinare        7              22959.20
```

END OF LAB

Exercise 5. JDBC and Stored Procedures

What This Exercise Is About

This exercise will teach you how to use Java and stored procedures.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Generate a stored procedure using the Development Center
- Write a Java application to retrieve a result set from the stored procedure

Instructor Exercise Overview

Introduction — The first section of the lab will have the students write a stored procedure using the Development Center. The stored procedure will issue a SELECT statement that will return a result set. The students will modify the program labcall1.java to call this stored procedure. It may be useful to remind students about ResultSetMetaData methods to determine the number of columns in a result set.

The second section will have a stored procedure created using a SQLJ program. Lab will require student to create a jar file for the stored procedure. The lab will require stored procedure to be defined to the database. A script file named MultRes.ddl will contain the information to complete this task.

Time for Lab — 2 hours.

Thing to Review at the End of Lab — When a stored procedure is created using the Development Center, information for the stored procedure in the SYSIBM.SYSROUTINES table. This is a new system catalog tables for DB2 UDB Version 8. Debugging a Stored Procedure will help students understand a powerful feature of the Development Center.

Exercise Instructions

Section 0 - Calling a Stored Procedure Built with the Development Center

Introduction

Create a stored procedure name STAFF1. This stored procedure will issue a:

```
SELECT NAME, JOB, SALARY, DEPTNAME
  FROM STAFF, ORG
  WHERE DEPTNUMB = DEPTNO
  AND DEPTNO ?
```

Modify the application labcall1.java to call the stored procedure STAFF1 and retrieve the result set. The application will have to pass the department number to the stored procedure and determine the number of columns in the result set.

Section 1 - Create the Stored Procedure STAFF1

__ 1. Open the Development Center.

_____

__ 2. Create a new project named UDBA_Project1.

_____

__ 3. Connect to the SAMPLE database.

_____

__ 4. Create a new stored procedure using the SQL Stored Procedure Wizard.

_____

__ 5. Name the Stored Procedure UDBA.STAFF1.

_____

__ 6. The stored procedure will run a single SQL statement and return a result set.

_____

__ 7. Define the SQL statement.

```
SELECT NAME, JOB, SALARY, DEPTNAME
FROM STAFF, ORG
WHERE DEPTNUMB = DEPTNO
AND DEPTNO = ?
```

_____

__ 8. Define the parameter to be a small integer.

_____

__ 9. Generate and Build the stored procedure.

_____

__ 10. Test the stored procedure.

_____

Section 2 - Modify the Program labcall1.java to Call the Stored Procedure STAFF1

__ 1. Define the string that will contain the calling SQL string. This SQL statement, **callSQL**, will call the stored procedure STAFF1 and pass 1 parameter to the stored procedure.

_____

__ 2. Create a CallableStatement object named **callstmt**. This CallableStatement object will use the calling SQL string object defined in Step 1.

_____

__ 3. Set the parameter in the CallableStatement object to contain the value that is held in the variable named **mydeptno**.

_____

__ 4. Code the Java statement that will cause the CallableStatement object created in Step 3 to execute.

_____

__ 5. Assign the result set generated by the stored procedure labstaff1 to the ResultSet object named **rs**.

_____

__ 6. Determine the number of columns in the result set that is returned by the stored procedure STAFF1. You should define a ResultSetMetaData object named **rsmd**. The getMetaData( ) method should be used. Once the ResultSetMetaData object is instantiated, use the getColumnCount( ) method to generate the number of columns. The variable **numcols** will used to hold the number of columns.

_____

__ 7. Compile the program

_____

__ 8. Execute the client program labcall1.

_____

END OF LAB

Exercise Solutions

Section 0 - Calling a Stored Procedure Built with Development Center

Introduction

Create a stored procedure name STAFF1. This stored procedure will issue a:

```
SELECT NAME, JOB, SALARY, DEPTNAME
  FROM STAFF, ORG
  WHERE DEPTNUMB = DEPTNO
  AND DEPTNO ?
```

Modify the application labcall1.java to call the stored procedure STAFF1 and retrieve the result set. The application will have to pass the department number to the stored procedure and determine the number of columns in the result set.

Section 1 - Create the Stored Procedure STAFF1

__ 1. Open the Development Center.

Start -> Programs -> IBM DB2 -> Development Tools -> Development Center

__ 2. Create a new project named UDBA_Project1.

Select **1. Create Project.**
Project name: **UDBA_Project1**
OK (command button).

__ 3. Connect to the SAMPLE database.

- __ a. Add Connection
- __ b. Connect Type Online (radio button)
- __ c. Next (command button)
- __ d. Database Alias: SAMPLE
- __ e. UserID: udba
- __ f. Password: udba
- __ g. Finish (command button)

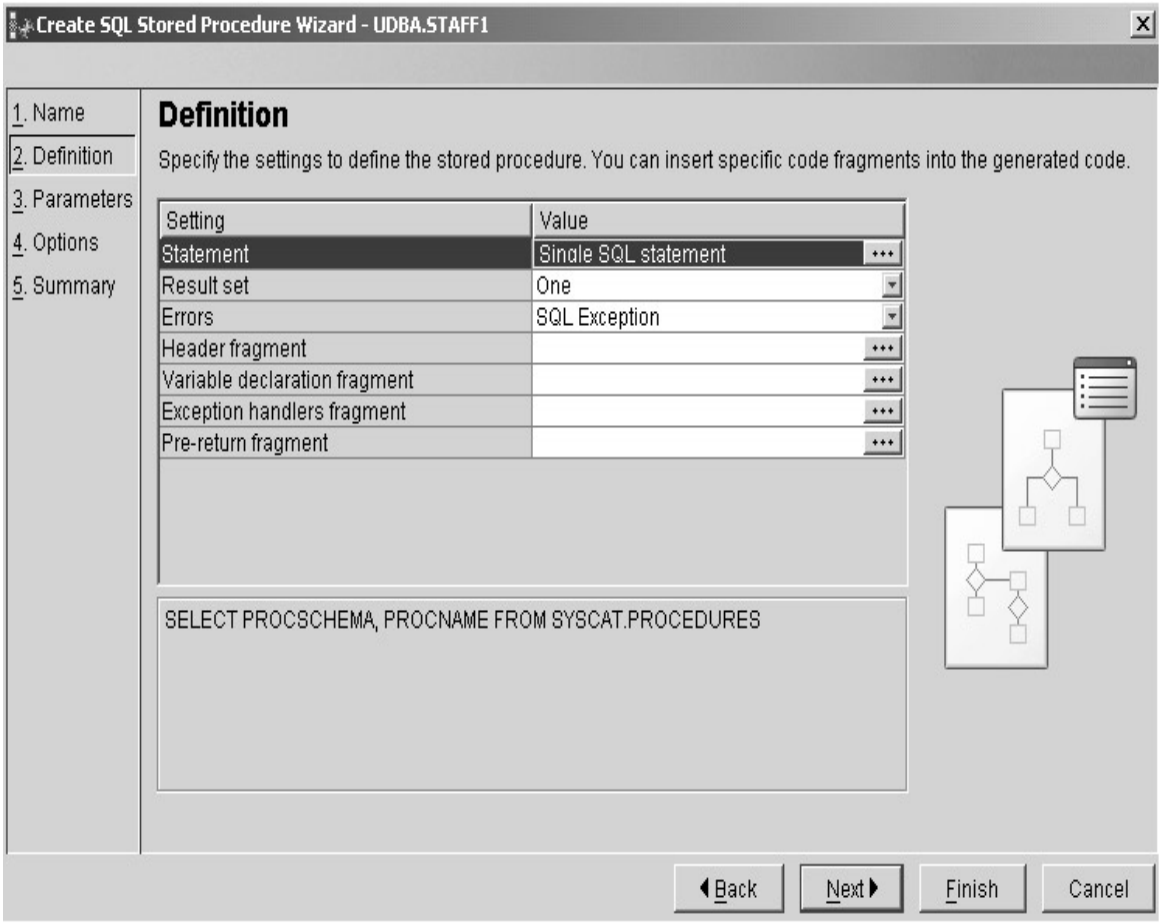
__ 4. Create a new stored procedure using the SQL Stored Procedure Wizard.

- __ a. Create Object (command button)
- __ b. Select **Stored Procedure** and **SQL**
- __ c. OK (command button)

__ 5. Name the Stored Procedure UDBA.STAFF1.

- __ a. Name: **UDBA STAFF1**

__ b. **Next** (command button)



__ 6. The stored procedure will run a single SQL statement and return a result set.

Statement Single SQL Statement … **(Select the 3 dots)**
Generate one SQL statement **(radio button)**

__ 7. Define the SQL statement.

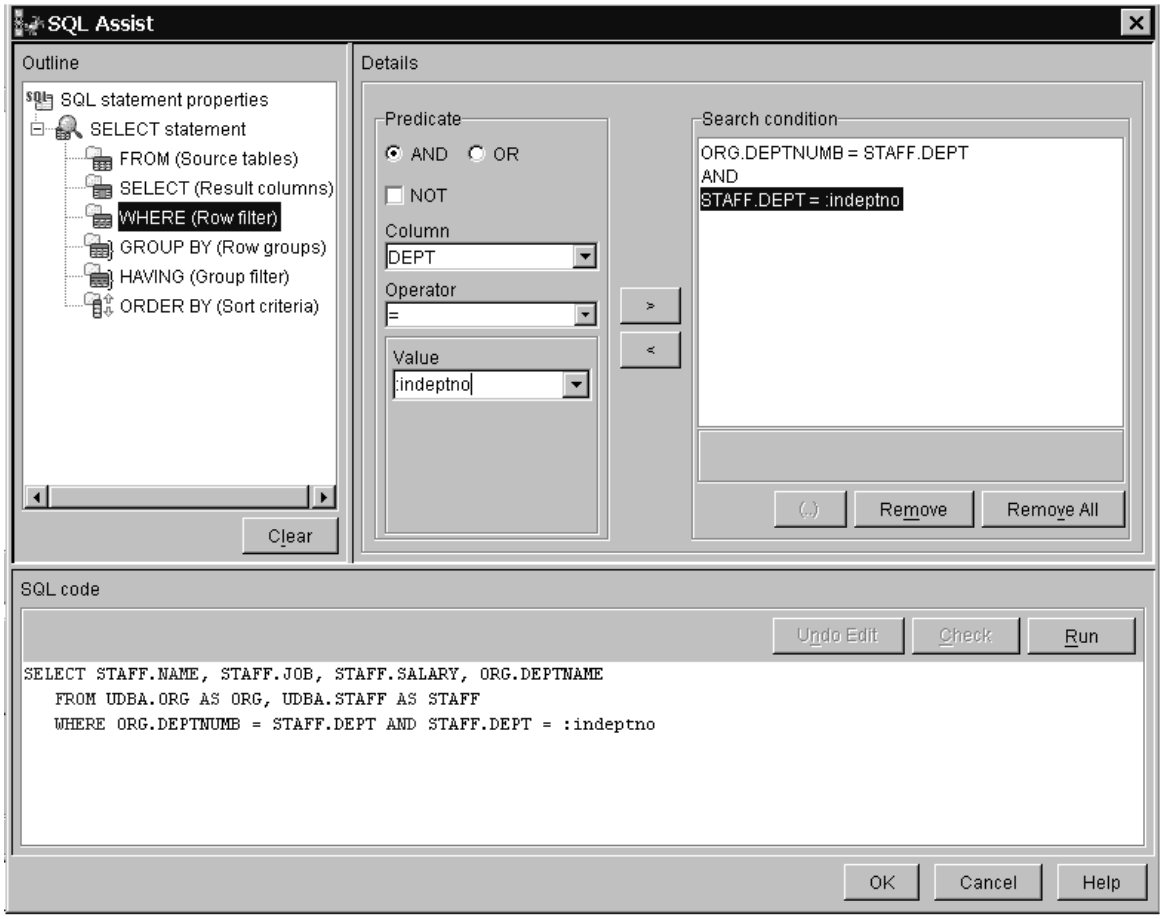
```
SELECT NAME, JOB, SALARY, DEPTNAME
FROM STAFF, ORG
WHERE DEPTNUMB = DEPTNO
AND DEPTNO = ?
```

Select the **SQL Assist …** (command button).

From the Outline frame, select **FROM (Source Tables)**.
Select tables ORG and STAFF from the UDBA schema.
Deselect the table Procedures.
Select the **Join Tables** (command button).
Select **OK** (No foreign key relationships found.)
Highlight Tables ORG and JOIN.

Join (command button)
First Column **DEPTNUMB** and Second Column **DEPT**
Add (command button).
OK (command button).

Select the **SELECT (Result Columns)** from the Outline frame.
Select the columns NAME, DEPT, JOB from the Staff table.
Select the column DEPTNAME from the ORG table.
The four columns should be displayed in the Selected Columns list box.
Select the **WHERE (Row filter)** from the Outline frame.
Select the DEPT column from the UDBA.STAFF table.
Select the Operator =.
Value -> Variable…
Variable Name: indeptno
Select **OK** (command button).
Select '>' (command button)



OK (command button)
OK (command button)

__ 8. Define the parameter to be a small integer.

Next (command button).

__ 9. Generate and Build the stored procedure.

Next (command button) -> **Finish** (command button) -> **OK** (command button) -> Close the Development Center Launchpad.

__ 10. Test the stored procedure.

UDBA.STAFF1 (rmb) -> **Run**

Section 2 - Modify the Program labcall1.java to Call the Stored Procedure STAFF1

__ 1. Define the string that will contain the calling SQL string. This SQL statement, **callSQL**, will call the stored procedure STAFF1 and pass 1 parameter to the stored procedure.

String callSql = "Call staff1 (?)";

__ 2. Create a CallableStatement object named **callstmt**. This CallableStatement object will use the calling SQL string object defined in Step 1.

CallableStatement callStmt = con.prepareCall(callSql);

__ 3. Set the parameter in the CallableStatement object to contain the value that is held in the variable named **mydeptno**.

callStmt.setInt(1,mydeptno);

__ 4. Code the Java statement that will cause the CallableStatement object created in Step 3 to execute.

callStmt.execute( );

__ 5. Assign the result set generated by the stored procedure labstaff1 to the ResultSet object named **rs**.

rs = callStmt.getResultSet( );

__ 6. Determine the number of columns in the result set that is returned by the stored procedure STAFF1. You should define a ResultSetMetaData object named **rsmd**. The getMetaData( ) method should be used. Once the ResultSetMetaData object is instantiated, use the getColumnCount( ) method to generate the number of columns. The variable **numcols** will used to hold the number of columns.

ResultSetMetaData rsmd = rs.getMetaData( );
int numcols = rsmd.getColumnCount( );

__ 7. Compile the program.

javac labcall1.java

__ 8. Execute the client program labcall1.

java labcall1

END OF LAB

Exercise Instructions - Section 3

The program solutions can be found starting in Appendix B, “JDBC and Stored Procedures Solutions” on page B-1.

Processing Result Sets

You will be writing a program to invoke a stored procedure that return result sets.

Client code

__ 1. The client code will set up a callable statement, call the stored procedure, then process result sets returned. The client program that is partially coded can be found in `JDBCcli.java`. A copy of this program can be found in “JDBCcli.java” on page 5-15.

__ 2. Edit the program, finding the ?'s that indicate where code changes need to be made.

__ 3. Create an executable for this program using javac.

Server code

__ 1. The server code opens one to many result sets and leave them open. The program that is partially coded can be found in `MultRes.sqlj`. A copy of this program can be found in “MultRes.sqlj” on page 5-19.

__ 2. Make a subdirectory for this server code:

```
mkdir multres
cd multres
copy ..\MultRes.sqlj
```

__ 3. Edit the program, finding the ?'s that indicate where code changes need to be made.

Create a named iterator, MultRes_Cursor1, that will return the salary using the following SELECT statement:
`SELECT salary FROM EMP ORDER BY salary`

Create a positioned iterator, MultRes_ByPos, that will return the four columns in the DEP table as String variables using the following SELECT statement:
`SELECT * FROM DEP`.

Create a named iterator, MultRes_ByName, that will return the employee number and lastname as String variables from the EMP table using the following SELECT statement: `SELECT EMPNO, LASTNAME FROM EMP`.

__ 4. Create an executable for the stored procedure using SQLJ, db2sqljcustomize, and javac.

sqlj -url=jdbc:db2:eddb MultRes.sqlj

javac MultRes.java

db2sqljcustomize -url jdbc:db2://localhost:50000/sample -user udba -password udba MultRes_SJProfile0.ser

__ 5. Create a jar file for the stored procedure:

```
Jar cvf MultRes.jar MultRes*.class MultRes*.ser
```

__ 6. Install the jar file in the database:

```
db2 connect to eddb
db2 call sqlj.install_jar
('file:\cg112\multres\MultRes.jar','multresjar')
```

__ 7. Create the procedure definition in the database. Create a file named MultRes.ddl and code the CREATE PROCEDURE statement. Run the file by issuing:

```
db2 -tvf MultRes.ddl
```

Use the file D:\cg112\soln\MultRes.mem as a template to code the CREATE PROCEDURE statement.

__ 8. Test the programs (JDBCcli and MultRes) by running the Client (or calling) program:

```
cd ..
java JDBCcli
```

Note: If you must recompile the server program, to recreate the jar file, issue:

```
jar uvf MultRes.jar MultRes*.class MultRes*.ser
```

and to replace the jar file issue:

```
db2 connect to eddb
db2 call sqlj.replace_jar
('file:\cg112\multres\MultRes.jar','multresjar')
```

The lab solutions can be found in “JDBCcli.java Solution” on page B-1, “MultRes.sqlj Solution” on page B-5, and “MultRes.mem Solution” on page B-7.

END OF LAB

JDBCcli.java

The lab solution can be found in “JDBCcli.java Solution” on page B-1.

```
/*********************************************************************/
/*                                                                   */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"          */
/*                          ( CG11 )                                 */
/*                                                                   */
/*                                                                   */
/* Last update = 01/31/2000                                          */
/*                                                                   */
/*********************************************************************/
/*  Notes:                                                           */
/*                                                                   */
/*  This program is intended to be completed with the lab guide      */
/*  as a reference.  The lab guide is the set of instructions that   */
/*  should be followed.  The comments in this program are intended   */
/*  to clarify statements made in the lab guide.                     */
/*********************************************************************/

/*********************************************************************/
/* Import Java Classes                                               */
/*********************************************************************/
import java.lang.*;
import java.util.*;
import java.io.*;
import java.sql.*;              // JDBC classes
import COM.ibm.db2.jdbc.app.*;  // DB2 UDB JDBC classes

class JDBCcli
{ static
  { try
    { Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (Exception e)
    { System.out.println ("\n  Error loading DB2 Driver...\n");
      System.out.println (e);
      System.exit(1);
    }
  }

  // main application: .connect to the database
  //                   .call the stored procedure
  public static void main (String argv[])
  { Connection con = null;
    try
```

```
{ System.out.println ("Java Multiple Resultsets Stored Procedure");
  // Connect to EDDB database

  // URL is jdbc:db2:dbname
  String url = "jdbc:db2:eddb";

  if (argv.length == 0)
  { // connect with default id/password
    con = DriverManager.getConnection(url);
  }
  else if (argv.length == 2)
  { String userid = argv[0];
    String passwd = argv[1];

    // connect with user-provided username and password
   con = DriverManager.getConnection(url, userid, passwd);
  }
  else
  {   throw new Exception("\nUsage: java JDBCcli [username password]\n");
  }

  // Set AutoCommit
  con.setAutoCommit(true);
  String callName = "MultRes";

  /****************** ?????????????????????? *********************/
  /* Create callable statement to call the Stored Procedure       */
  /* Print the statement out in the following line                */
  /****************************************************************/
  String callSql = /* ?? */;
  System.out.println("Creating CallableStatement = " + callSql);

  /****************** ?????????????????????? *********************/
  /* Prepare the callable statement                               */
  /****************************************************************/


  ResultSet rs = null;
  System.out.println("\nExecuting the Java Stored Procedure now...\n");
  /****************** ?????????????????????? *********************/
  /* Execute the callable statement                               */
  /****************************************************************/

  int rsCount = 0;

  while( true )
  {
```

```
      /************************ ???????????????????? ******************/
      /* Assign the results to a the variable named rs (defined above) */
      /*****************************************************************/

      if( rs != null )
      { rsCount++;
        System.out.println("Fetching all the rows from the result set #"
                           + rsCount);
        fetchAll(rs);
        /**************** ???????????????? ****************************/
        /* Move to the next result set                                 */
        /*****************************************************************/

        System.out.println("");
        continue;
      }

      break;
    }


    // close off everything before we leave
    System.out.println("Closing statements and connection.");
    callStmt.close ();
    con.close ();
  }
  catch (Exception e)
  { try
    { if( con != null )
      { con.close();
      }
    }
    catch (Exception x)
    { //ignore this exception
    }
    System.out.println (e);
  }
}

// ================================================
// Method: fetchAll
// ================================================
static public void fetchAll( ResultSet rs )
{ try
  {         System.out.println("================================="
                               +"=================================");
    ResultSetMetaData stmtInfo = rs.getMetaData();
```

```
    int numOfColumns = stmtInfo.getColumnCount();
    int r = 0;
    System.out.println("number of columns is " + numOfColumns);
    while( rs.next() )
    { r++;
      System.out.print("Row: " + r + ": ");
      for( int i=1; i <= numOfColumns; i++ )
      { System.out.print(rs.getString(i));
        if( i != numOfColumns ) System.out.print(" , ");
      }
      System.out.println("");
      }
    }
    catch (SQLException e)
    { System.out.println("Error: fetchALL: exception");
      System.out.println (e);
    }
  }
}
```

MultRes.sqlj

The lab solution can be found in “MultRes.sqlj Solution” on page B-5.

```
/*********************************************************************/
/*                                                                   */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"          */
/*                          ( CG11 )                                 */
/*                                                                   */
/*                                                                   */
/* Last update = 01/31/2000                                          */
/*                                                                   */
/*********************************************************************/
/*  Notes:                                                           */
/*                                                                   */
/*  This program is intended to be completed with the lab guide      */
/*  as a reference.  The lab guide is the set of instructions that   */
/*  should be followed.  The comments in this program are intended   */
/*  to clarify statements made in the lab guide.                     */
/*********************************************************************/

/*********************************************************************/
/* Import Java Classes                                               */
/*********************************************************************/
import java.sql.*;              // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

/*************************??????????????????????????? ***************/
/* Define iterators for 3 select statements                          */
/*********************************************************************/


///////
// Java stored procedure is in this class
///////
public class MultRes
{   // (1) stored procedure body
  public static void  SPBody ( ) throws Exception
  {
     /****************** ???????????????????? ******************/
     /* Define iterator variables for local use                   */
     /*************************************************************/

     try
     {
        /****************** ???????????????????? *****************/
        /*  Issue Select statments to open cursors and leave open. */
        /*************************************************************/

     }
     catch (Exception e)
     {   throw e;
     }
  }
}
```

Exercise 6. Object-Relational Capabilities - LOBs, UDTs, and UDFs

What This Exercise Is About

This exercise will allow you to define and code a user-defined function. It will also allow you to code a program that manipulates LOB data.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Define a User-Defined Function to DB2
- Code a User-Defined Function
- Invoke a User-Defined Function
- Define a User-Defined Type
- Manipulate LOB data using JDBC 2.0 methods

Introduction

You will code a user-defined function. You will need to define your function to DB2. You will invoke your function through using CLP.

The first UDF will be created using the SQL Procedure Language. The following UDF will not have SQL statements in it, so it requires no PREP or BIND. The programs that we will change will not be `.sqlj` but rather `.java` and contain only Java code.

You will also be able to create a new user-defined distinct type and use it in a table. You will explore what is required to use your UDF in conjunction with your UDT.

The third section will allow you to write a table UDF.

The fourth section will require you to complete a program skeleton that will access a LOB column.

Note: During any lab, if you need to refresh your tables, you can run the **crtabs.mem** member to drop and recreate your tables.

The fifth section is required if you will be doing any other labs using these databases following this lab. It simply involves resetting the lab environment and ***MUST*** be completed before you proceed to other lab exercises.

Instructor Notes

Introduction — This lab will introduce students to UDTs, UDFs, and manipulating large objects with JDBC 2.0 methods.

Time for Lab — 1 hour, 45 minutes.

Things to Review at End of Lab — Review the JDBC 2.0 methods used to manipulate CLOB data.

Exercise Instructions

The program solutions can be found starting in Appendix C, “UDT/UDF/LOB Program Solutions” on page C-1.

Create a User-Defined Function that will take the column EDLEVEL from the table EMP and convert it to a character string.
If the EDLEVEL (Education Level) is 20 or greater, display the value DOCTORATE.
If the EDLEVEL is 18 or 19, display the value MASTERS.
If the EDLEVEL is 16 or 17, display the value COLLEGE DEGREE.
If the EDLEVEL is 13, 14, 15, display the value SOME COLLEGE.
If the EDLEVEL is 12, display the value HIGH SCHOOL GRADUATE.
Otherwise, display the value GED.

The function should be named DEGREE.

Section 1 - Creating the function DEGREE

__ 1. Using an editor create a script file named **degree.mem**. This script file will contain the procedure language code to create the procedure named DEGREE.

CONNECT TO EDDB@

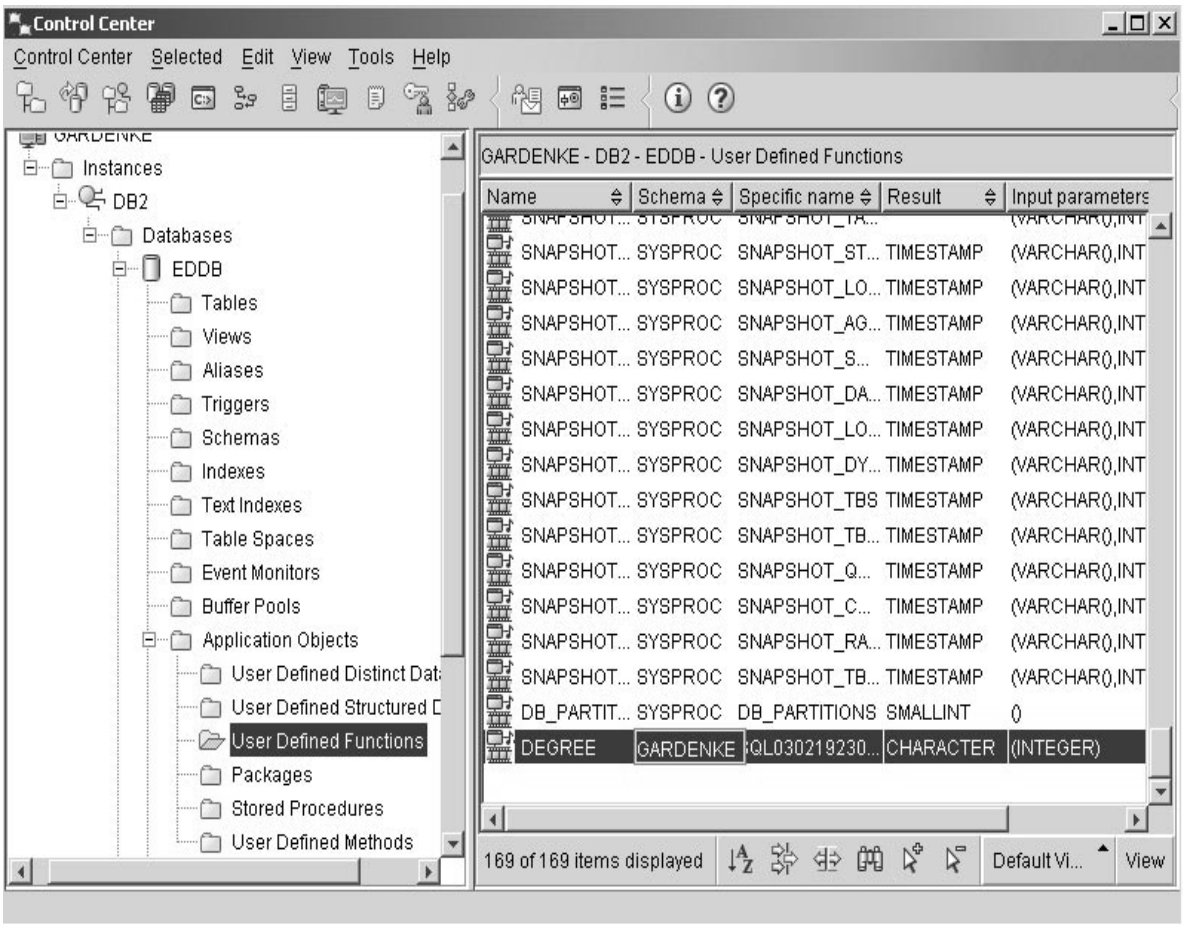
```
CREATE FUNCTION DEGREE (in_value integer)
 RETURNS char(20)
 LANGUAGE SQL
 CONTAINS SQL
 RETURN
  CASE
  WHEN in_value > 19 Then 'DOCTORATE'
  WHEN in_value > 17 Then 'MASTERS'
  WHEN in_value > 15 Then 'COLLEGE DEGREE'
  WHEN in_value > 12 Then 'SOME COLLEGE'
  WHEN in_value > 11 Then 'High School Graduate'
  ELSE 'GED'

END @
```

__ 2. Execute the script file.

---

db2 -td@ -vf degree.mem

__ 3. Using the Control Center verify the user-defined function exists.



__ 4. Issue a SELECT statement that will display the columns LASTNAME, EDLEVEL and the EDLEVEL column using the DEGREE function.

SELECT LASTNAME, EDLEVEL, DEGREE(EDLEVEL) FROM EMP.

Section 2 - Money Conversion UDF

Money Conversion Function - Coding

You will be writing a conversion routine to convert from US dollars to foreign currency. The exchange rate will be stored in an external file. The user will take advantage of the function in statements such as:

```
SELECT EMPNO, SALARY, CAN(SALARY) FROM EMP
            (to retrieve the salary in Canadian dollars)
SELECT EMPNO, SALARY, DMK(SALARY) FROM EMP
          (to retrieve the salary in Deutschmarks)
```

__ 1. The user-defined function will take in US dollars and return the converted money amount. The function should determine from the manner in which it is called what conversion should be done. An input file, `convfile.txt`, will be read to determine the current conversion rate. This file contains the name of the function that has been called and the conversion rate to use for that calling of the function. This file could be updated at any time. A copy of this file can be found in “convfile.txt” on page 6-17. You will need to update this file with your userid as the schema name of the function reference.

__ 2. The function code has been partially coded and can be found in `convert.java`. A copy of this program can be found in “convert.java” on page 6-18.

__ 3. Make a subdirectory for your work on this function:

```
mkdir convert
cd convert
copy ..\convert.java
copy ..\convfile.txt
```

__ 4. Edit the program, finding the ?'s that indicate where code changes need to be made.

__ 5. Create an executable for the UDF using javac. Create the jar file as described previously.

In summary, you will code one program, `convert.java`, and create three functions, `CAN`, `DMK`, and `YEN`. Each one of these functions will call the `convert` program which will read the `convfile.txt` file to determine the exchange rate.

Note: The information in the `convfile.txt` file is case sensitive. Make sure to use upper case characters.

The lab solution can be found in “convert.java Solution” on page C-4.

Money Conversion Function - Definition

You've coded your UDF, but before it can be used, you must tell DB2 about it.

__ 1. Define your UDF to DB2. You may want to create a CLP member for doing the definition so you can modify it for future definitions. Define it so you can convert the salary column of the EMP table to Canadian Dollars.

Things to consider:

- What name are you going to use to invoke your function?
- What type is your program taking in?
- What type will it return? Is that how the result should be displayed? (Hint: We want the Canadian salary to display in the same format as the SALARY column)
- What jar file should the UDF invoke?
- Could anything outside the program execution cause the result to be different even for the same input values?
- Should your function be invoked if the input amount is null?
- What parameters are required?

Note: You may get a warning message SQL0477W indicated that run-time data truncation could occur. This is due to a conversion from FLOAT (which your function returns) to DECIMAL (which may be how you want to display your data). FLOAT data can hold more precision than DECIMAL.

The lab solution can be found in “convert DDL” on page C-7.

Money Conversion Function - Testing

__ 1. Test the program from the client machine using CLP.

Some test cases you might want to try:

- Convert the SALARY column in the EMP table for Christine Haas (empno = '000010') to Canadian Dollars (at a conversion rate of 1 US Dollar = 1.34 Canadian Dollars, her US salary of 52750.00 should be 70685.00 in Canadian Dollars).
- Convert the average of the SALARY column of the EMP table to Canadian Dollars. (At an exchange rate of 1.34, the average in Canadian Dollars should be 36586.81.)

Test by executing this SQL statement:

```
db2 select empno, salary, can(salary) canadian_sal from emp where
empno = '00010'
```

__ 2. Now, it doesn't make sense, but could you convert the value of an integer or smallint column (for example, EDLEVEL in the EMP table) to Canadian Dollars? What happens? Did you expect this? Why?

_____

_____

_____

You can use your existing "can" function to process against the EDLEVEL column since integer can be promoted to the input argument type of FLOAT.

__ 3. Employee number (EMPNO) has numeric values in a character field. This still doesn't make business sense, but can you convert the employee number to a "Canadian" value? What happens? Did you expect this? Why?

_____

_____

EMPNO is defined as a character field. There is not an allowed promotion of CHAR to FLOAT in UDF arguments.

__ 4. Due to the shipment of the DB2 Universal Database product from the IBM lab in Toronto, Canada, there has been a tremendous improvement in the value of the Canadian Dollar. The exchange rate has changed from 1.34 to 1.20. Does your program have to change to accommodate this business change?

_____

_____

Only the conversion file has to change.

Note: You can either change the file on the server machine or ftp it up from your client machine.

__ 5. Test your program with the new exchange rate of 1.20. Employee '000010's Canadian salary should now be 63300.00.

Money Conversion Function - Further Definition

__ 1. The conversion file has values for a function named 'YEN' and 'DMK' defined by your userid. Can you invoke these functions now? Determine the value of employee '000010's salary in Deutschmarks.

_____

_____

The conversion function hasn't been defined to DB2 yet.

__ 2. Define the DMK and YEN conversion functions to DB2. Do you need to make any changes to your program?

_____

_____

No changes are needed to the program.

Note: You may get warning SQL0477W due to possible run-time data truncation.

__ 3. Define the DMK and YEN conversion functions to DB2.

__ 4. Test your program with the Deutschmark exchange rate. Employee '000010's DMK salary should be 72795.00.

The lab solution can be found in “DM and YEN DDL” on page C-8.

User-defined Distinct Type - Definition

Now we want to try some work with UDTs. We are going to define two very similar UDTs so we can see some considerations that you should consider when using them.

You may want to create a CLP member for doing the definition so you can modify it for future definitions.

__ 1. Create a user-defined distinct type, based on a decimal type, named US_DOLLAR, with precision 9 and scale 2.

__ 2. Create a user-defined distinct type, based on a DOUBLE type, named FLT_DOLLAR.

The lab solution can be found in “UDT DDL” on page C-9.

User-defined Distinct Type - Usage

__ 1. We've decided that whatever the lowest salary is in each job category as of today, that will be our minimum salary for that job category in the future; and whatever is the highest salary in each job category as of today, that will be our maximum salary for that job category in the future. So, we want to create a table with the job names and the minimum salary and maximum salaries for that job. We want to use our new UDTs for the types of the minimum and maximum salary columns. (We'll make one of each type just for purposes of experimentation in our lab work.)

Define a table named SALARY_MIN_MAX with the three columns as described:

- JOB - character 8
- MAX_SALARY - FLT_DOLLAR
- MIN_SALARY - US_DOLLAR

```
create table salary_min_max (job char(8), max_salary flt_dollar,
   min_salary us_dollar)
```

You may use the sample code found in `D:\CG112\SOLN\min_max.mem`.

__ 2. Populate the salary_min_max table with data from the EMP table. Execute the following from the DB2 Command Center to do the insert:

```
Insert into salary_min_max (job, min_salary, max_salary) select job,
min(salary), max(salary) from emp group by job
```

This may also be found in D:\CG112\SOLN\avg.mem.

The lab solution can be found in “Create Table DDL” on page C-10.

User-defined Distinct Type with DB2 Defined Function

__ 1. Can you use the AVG built-in function to determine the average MAX_SALARY in the SALARY_MIN_MAX table?

Can you use the AVG built-in function to determine the average MIN_SALARY in the SALARY_MIN_MAX table?

_____

_____

No, the AVG built-in function has not been defined to be able to take FLT_DOLLAR or US_DOLLAR as input.

__ 2. Define an AVG function for your two UDTs.

Note: You may get SQL0477W - the warning about possible run-time data truncation from DECIMAL to US_DOLLAR.

You may use the sample code found in `d:\cg112\soln\avg.mem`.

__ 3. Test your AVG functions.

```
SELECT AVG(MAX_SALARY), AVG(MIN_SALARY) FROM SALARY_MIN_MAX

1                       2
----------------------- -----------
  +3.49537500000000E+004    27501.25
```

The lab solution can be found in “Average Function DDL” on page C-11.

User-defined Distinct Type with User-Defined Function

__ 1. Use your CAN user-defined function to convert the MAX_SALARY column from the SALARY_MIN_MAX table to the maximum salary in Canadian Dollars. What happens? Did you expect this? Why?

_____

_____

The conversion function won't work for the user-defined distinct type of FLT_DOLLARS, even though it is sourced on the type (FLOAT) that is defined as a valid input for the current UDF.

__ 2. Use your CAN user-defined function to convert the minimum salary from the SALARY_MIN_MAX table to the minimum salary in Canadian Dollars. What happens? Did you expect this? Why?

_____

_____

The conversion function won't work for the user-defined distinct type of US_DOLLAR, even though it is sourced on a type (DECIMAL) that will work with the currently defined UDF.

__ 3. **Without defining a new UDF**, can you get the values to be converted?

Hint: Can you cast to an appropriate type?

_____

_____

`SELECT CAN(DECIMAL(MIN_SALARY)) FROM SALARY_MIN_MAX` **will convert the US_DOLLAR column to a DECIMAL column prior to calling the CAN function. Then DB2 will promote the DECIMAL type to a FLOAT which will be accepted by the CAN function. You can do a similar statement to convert the FLT_DOLLAR column to a FLOAT prior to invoking the UDF.**

The casting functions were automatically created when you created the UDT (by specifying the 'WITH COMPARISONS' clause).

__ 4. You don't want your users to always need to do the casting to use the CAN function. Try to define a user-defined function that will work with the FLT_DOLLAR type. Were you successful?

_____

_____

Yes, a user-defined function could be defined that takes FLT_DOLLAR as input and uses your existing program as the conversion routine.

Note: You may see SQL0477W warning message about data truncation.

__ 5. Try to define a user-defined function that will work with the US_DOLLAR type. Were you successful?

_____

_____

No, you cannot define a user-defined function that takes a decimal type or any UDT that is defined on a decimal type as input. (This is the reason we wanted you to define two UDTs - so you could see this limitation when a UDT is defined on a DECIMAL type - UDFs cannot be defined to work with a UDT that is defined on a DECIMAL type - you must always convert the UDT to a type acceptable to C language prior to invoking a UDF to work on it.)

The lab solution can be found in “UDT with UDF DDL” on page C-12.

Section 3 - Table UDF

Table UDF - Coding

The folks in your corporation who appraise employees have come to you for assistance.

They have a flat file with sample text that they want to be able to use when they are appraising employees (it is stored in a file named `perform.txt` - a sample of the contents of this file can be found in “perform.txt” on page 6-24). They would like to select an employee from a DB2 table, supply the appraisal level to a function invoked via an SQL statement, and have the result returned with the employee's name and sample appraisal text.

For example, they would like to enter:

```
SELECT E.LASTNAME, TF.APPRAISAL
    FROM EMP E, TABLE(PERFORM()) as TF
    WHERE E.EMPNO = '000010' AND TF.RATING= 1
```

to have employee '000010's lastname and appraisal text at a level of '1' be returned. (The appraisal levels that are currently in the file range from 1 to 3 with appraisal level 1 being the best.) and have a character string representation of the EDLEVEL be returned.

__ 1. The table function will take in a number (integer) representing the appraisal level, and return multiple 'rows' from the `perform.txt` file representing sample phrases that can be used for this appraisal level. The function code has been partially coded and can be found in `perform.java`. A copy of this program can be found in “perform.java” on page 6-21.

   Note: Make sure you include logic that returns an SQLSTATE that indicates that all rows have been returned ('02000').

__ 2. Make a subdirectory for your work on this function:

```
mkdir perform
cd perform
copy ..\perform.java
copy ..\perform.txt
```

__ 3. Edit the program, finding the ?'s that indicate where code changes need to be made.

__ 4. Create a jar file for the UDF as described before.

The lab solution can be found in “perform.java Solution” on page C-13.

Table Function - Definition

You've coded your UDF, but before it can be used, you must tell DB2 about it.

__ 1. Define your UDF to DB2. You may want to create a CLP member for doing the definition so you can modify it for future definitions. Define it so you can call your UDF as a table function, passing in an integer, and returning a variable character string of up to 80 characters.

   Things to consider:

   - What name are you going to use to invoke your function?
   - What type is your program taking in?

- What type will it return?
- What code should the UDF invoke?
- What language convention should be used in calling your function?
- Could anything outside the program execution cause the result to be different even for the same input values?
- Should your function be invoked if the input value is null?
- Are you dependent upon having a scratchpad area supplied on each call?
- Do you need DB2 to call you a final time so you can release storage?
- What parameters are required?

The lab solution can be found in “perform DDL” on page C-16.

Table Function - Testing

__ 1. Test the program from the client machine using CLP.

Test the program by executing the following statement from the CLP.

```
db2 select e.lastname, tf.appraisal from emp e, table (perform()) as
tf where e.empno = '000010' and tf.rating = 1
```

You can now contact the individuals who need to write appraisals, and let them know that their new function is now available.

Section 4 - Manipulating LOB data

You will be completing a JDBC program skeleton that utilizes JDBC 2.0 methods to extract a portion of a CLOB column data from DB2 table.

A copy of this program, javalob.java, is included later in this exercise.

A sample solution for the program can be found beginning in “javalob Program Solution” on page C-17.

Note: If the EMP_RESUME table in the SAMPLE database exists under the "admin" schema and not the "udba" schema, then in your Java program either connect to the database as userid "admin", or reference table `admin.EMP_RESUME` where appropriate in your SQL statements.

__ 1. We will be accessing the EMP_RESUME table of the SAMPLE database. If the SAMPLE database has not been created, you can create it from the DB2 Command Window by running the db2sampl program.

Within the EMP_RESUME table there is a CLOB column named RESUME. In this exercise, we will retrieve information from the RESUME column for a given employee. We will exclude 'personal' information.

__ 2. In a DB2 Command window, copy javalob_skel.java to javalob.java and edit the javalob.java file using the 'editpad' utility. Examine the components that have already been coded and the general flow of the program.

__ 3. You will be required to code multiple SQL statements which will locate the position of the "Personal" information and the position of the "Department" information within the RESUME column. You will return the data from the column for a specific employee, excluding the "Personal" information.

Hint: Use the POSSTR function to locate the position of "Personal" in one SELECT statement, and the location of "Department" in a second SELECT statement. The position value will be saved to different variables in each SELECT. Next you will find the position of one character before the word "Personal" by subtracting 1 from the variable containing the position of "Personal". Store this position in a new variable. Now, create a SELECT statement that selects EMPNO, RESUME_FORMAT, and a concatenation of RESUME column data - excluding "Personal" data. Use the SUBSTR function to determine the data to concatenate. Finally, use the JDBC 2.0 methods getClob() and getSubString() to return the data in String format.

__ 4. After coding is complete, compile the program: 'javac javalob.java'.

__ 5. Execute the program, and when prompted, provide a value for EMPNO. If you do not obtain the desired results, make appropriate coding changes, do a javac, and test the program again.

Section 5 - Resetting the Lab Environment

Complete the following to ensure that your table data is set back to its original contents if you will be doing any other labs following this one.

__ 1. **db2 connect to eddb user udba using udba**

__ 2. **db2 -vtf "crtabs.mem"**

degree.java

The lab solution can be found in “degree.java Solution” on page C-1.

```
/*****************************************************************/
/*                                                               */
/* Sample Java program for "DB2 UDB Advanced Programming"        */
/*                          ( CF11 )                             */
/*                                                               */
/*                                                               */
/* Last update = 01/20/2000                                      */
/*                                                               */
/*****************************************************************/
/* This function will provide for conversions of EDLEVEL to a    */
/* character string representation of the education level.       */
/*    20 or above - DOCTORATE                                    */
/*    18, 19      - MASTERS                                      */
/*    16, 17, 18  - COLLEGE DEGREE                               */
/*    13, 14, 15  - SOME COLLEGE                                 */
/*    12          - HIGH SCHOOL GRADUATE                         */
/*    1 - 11      - GED                                          */
/*****************************************************************/
/*  Notes:                                                       */
/*                                                               */
/*  This program is intended to be completed with the lab guide  */
/*  as a reference.  The lab guide is the set of instructions that*/
/*  should be followed.  The comments in this program are intended*/
/*  to clarify statements made in the lab guide.                 */
/*****************************************************************/

/*****************************************************************/
/* Import Java Classes                                           */
/*****************************************************************/
import java.sql.*;
import COM.ibm.db2.app.*;        // UDF and associated classes
import java.io.*;
import java.util.*;

class Degree extends UDF
{

/*****************************************************************************/
/*  EDLEVEL to education level conversion                                    */
/*****************************************************************************/
/*********??????????????????????????????????????????????????????************/
/* Code the entry point information for your function                        */
/*   - indicate the return value for the function                            */
/*   - indicate the entry point name for the function                        */
/*   - indicate input parameter(s) for the function                          */
/*****************************************************************************/
```

```
  {
    String degree_char = null;

  /******??????????????????????????????????????????????????????????******/
  /* Determine the value to be returned.                                */
  /**********************************************************************/

   if ;
              /* check the input value, if it is 20 or greater */
              /* put the value of "DOCTORATE" into the return value   */

   switch ( /* ??? */  )     /* switch on the input value...*/
   {
      case 19:
      case 18:  degree_char = "MASTERS";
                break;
      case 17:
      case 16:  degree_char = "COLLEGE DEGREE";
                break;
      case 15:
      case 14:
      case 13:  degree_char = "SOME COLLEGE";
                break;
      case 12:  degree_char = "HIGH SCHOOL GRADUATE";
                break;
      default:  degree_char = "GED";
                break;
} // end switch

/****** ?????????????????????????????????????????????????? ******/
/*  Identify the value to return                                      */
/**********************************************************************/


   } // end method degree

}  // end class Degree
```

convfile.txt

```
UDBA.CAN01.55
UDBA.DMK01.38
UDBA.YEN89.67
```

convert.java

The lab solution can be found in “convert.java Solution” on page C-4.

```
/************************************************************************/
/*                                                                      */
/* Sample Java program for "DB2 UDB Programming Using Java"             */
/*                          ( CG11 )                                    */
/*                                                                      */
/*                                                                      */
/* Last update = 01/31/2000                                             */
/*                                                                      */
/************************************************************************/
/*  Notes:                                                              */
/*                                                                      */
/*  This program is intended to be completed with the lab guide         */
/*  as a reference.  The lab guide is the set of instructions that      */
/*  should be followed.  The comments in this program are intended      */
/*  to clarify statements made in the lab guide.                        */
/************************************************************************/

/************************************************************************/
/* Import Java Classes                                                  */
/************************************************************************/
import java.lang.*;
import COM.ibm.db2.app.*;
import java.sql.*;
import java.io.*;

/*********************** ?????????????????? **********************/
/* Code the method definition.                                          */
/************************************************************************/

{
  BufferedReader in;
//  BufferedWriter outfile;  /* debugging file */
  String s;
  String FuncName;
  String func_from_file;
  String double_from_file;
  double convrate = 0.0;

/*********************** ??????????????????? **********************/
/* Code the entry point for this function.                              */
/************************************************************************/

  {
```



```
    try
    {
      /******************** ???????????????????? *********************/
      /* Check the file name in the next statement.                  */
      /*************************************************************/
      in = new BufferedReader(new FileReader("d:\\cg11\\convfile.txt"));
//      outfile = new BufferedWriter(new FileWriter("d:\\cg11\\outfile.txt"));
//      outfile.write("\n This is the way to write to a file\n");
    }
    catch ( IOException x)
    {
      throw x;
    }
    try
    {
      /***************** ????????????????????? ***********************/
      /* Assign the function name that was used to call this function   */
      /* to FuncName.                                                   */
      /******************************************************************/

      while ( null != (s = in.readLine() ) )
      {
        func_from_file = s.substring(0,8);
        double_from_file = s.substring(9);
        /******************* ???????????????????? ***********************/
        /* Look for a matching function name in the file                */
        /************************************************************/
        if ( /* ?? */ )
        {
          convrate = Double.valueOf(double_from_file).doubleValue();
          break;
         }
      }
      if ( convrate != 0.0 )
      {
        /************************ ???????????????????? ******************/
        /* Set the return value to be returned to DB2.                */
        /**************************************************************/

      }
      else
      {
        otherCurrency = 0.0;
      }
//      outfile.close();
      return;
    }
```

```
    catch ( Exception x)
    { // outfile.close();
       throw x;
    }
  } // end usToOth
}
```

perform.java

The lab solution can be found in “perform.java Solution” on page C-13.

```
/**************************************************************************/
/*                                                                        */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"               */
/*                          ( CG11 )                                      */
/*                                                                        */
/*                                                                        */
/* Last update = 01/31/2000                                               */
/*                                                                        */
/**************************************************************************/
/*  Notes:                                                                */
/*                                                                        */
/*  This program is intended to be completed with the lab guide           */
/*  as a reference.  The lab guide is the set of instructions that        */
/*  should be followed.  The comments in this program are intended        */
/*  to clarify statements made in the lab guide.                          */
/**************************************************************************/

/**************************************************************************/
/* Import Java Classes                                                    */
/**************************************************************************/

import java.lang.*;
import COM.ibm.db2.app.*;
import java.sql.*;
import java.io.*;

/****************************** ??????????? ************************/
/* Define the method                                                      */
/**************************************************************************/

{
  BufferedReader filein;
  String s;
  int file_position = 1;

  /************************ ???????????????????? ********************/
  /* Define function with entry parameters                               */
  /***********************************************************************/

    {
      double double_from_string = 0;
      int local_rating = 0;
      switch (getCallType())
        {
```

```
case SQLUDF_TF_FIRST:
  // do initialization which is independent
  //   of input parameters
  //  (nothing to do for this function)
  break;

case SQLUDF_TF_OPEN:
  // do initialization valid for this return table
  // open scan for web data
  /****************** ???????????????? **********************/
  /* Check the file open in the following statement, ensure     */
  /* the file referenced is correct.                            */
  /**************************************************************/
  try
  { filein =
  new BufferedReader(new FileReader("d:\\cg11\\perform_list.txt")); }
  catch (IOException x){ throw x; }
  break;

case SQLUDF_TF_FETCH:
  // return "column" data for one "row"
  // when finished, indicate complete to DB2 via SQLSTATE
  int i = 0;
  try
  { // position in file to place where previous invocation had ended
    // (file_position keeps that information from run to run)
    while ( i < file_position )
    {
      s = filein.readLine();
      i++;
    }
    if ( s == null )          //  ???? indicate end of rows to DB2
      { setSQLstate("02000"); }
    else
      {
        double_from_string =
        Double.valueOf(s.substring(0,1)).doubleValue();
        local_rating = (int) double_from_string;
        /*********************** ??????????? ************************/
        /* Assign values for the first and second parameters           */
        /***************************************************************/


        file_position = i;
      }
  } catch ( IOException x )
    { throw x; }
```

```
          break;

        case SQLUDF_TF_CLOSE:
          // may reposition to top of scan, or close scan
          // state can be saved in scratchpad
          /*********************** ?????????????? ***********************/
          /* Close the file                                             */
          /**************************************************************/

          break;

        case SQLUDF_TF_FINAL:
          // disconnect from web server
          //  (nothing to do in this case)
          break;
      } // end switch
  } // end perform
}
```

perform.txt

```
1is a great employee.
1is the epitome of excellence.
1works and plays well with others.
1delivers all projects on time.
1produces high quality executables.
2is a good employee.
2defines 'good worker'.
2works and plays ok with others.
2delivers most projects on time.
2produces good quality executables.
3is an ok employee.
3defines 'ok worker'.
3works and plays ok with others some of the time.
3delivers some projects on time.
3produces mediocre quality executables.
```

javalob.java

The lab solution can be found in “javalob Program Solution” on page C-17.

```
/*************************************************************************/
/*                                                                       */
/*        Java program for "DB2 UDB PROGRAMMING USING JAVA"              */
/*        JDBC program to process a large object (LOB) column            */
/*                          ( CG112 )                                    */
/*                                                                       */
/*                                                                       */
/* Last update = 05/17/2001                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/
import java.sql.*;
import java.io.*;
import java.util.*;

/*************************************************************************/
/* Class definition                                                      */
/*************************************************************************/
public class javalob
{
/*************************************************************************/
/* Register the class with the db2 Driver                                */
/*************************************************************************/
 static
    {   try
        {   Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (Exception e)
        {   System.out.println ("\n  Error loading DB2 Driver...\n");
            System.out.println (e);
            System.exit(1);
        }
```

```
    }

/*********************************************************************/
/* Main routine                                                      */
/*********************************************************************/
public static void main( String args[]) throws Exception
{
    // establish connection to sample database
    Connection samplecon = DriverManager.getConnection("jdbc:db2:samp
    samplecon.setAutoCommit(false);

    // Provide information about what this program does
    String empnum = " ";
    System.out.println("This program will retrieve all information but the");
    System.out.println("Personal information for a given employee.");
    System.out.println("(Resumes exist for employees 000130, 000140, 000150, 000190)");
    System.out.println("Please enter an employee number:  ");

    // Read in the employee number that is to be processed
    DataInputStream dis = null;
    try
    { dis = new DataInputStream(System.in);
    } catch (Exception e) {e.printStackTrace(); System.exit(0);}
    try
    {
     String s;
     s = dis.readLine();
     empnum = s.substring(0,6);
    }
    catch (Exception e) {e.printStackTrace(); System.exit(0);}

    try
    {
      /******************** ??????????????????????? ***********************/
      /* Define ResultSets which will be used to process the employee info */
      /* The result set names are: rslob1, rslob2, and rslob3.             */
      /* Initialize the resutl set instances to null.                      */
      /*********************************************************************/

      /*********************************************************************/
      /*  Define variables for LOB processing                              */
      /*********************************************************************/
      String            resume = null;
      int               persindex = 0, persindex1 = 0, deptindex = 0;
      String            sql1 = null, sql2 = null, sql3 = null;
      PreparedStatement stmt1 = null, stmt2 = null, stmt3 = null;
      String            empno = null, resumefmt = null;
      /************************ ???????????????????? ************************/
      /*  Define a Clob variable, called resumelob, for LOB processing.    */
      /*  Initialize resumelob to null.                                    */
      /*********************************************************************/
```

```
/************************ ????????????????????? **********************/
/* Determine the starting position of the string 'Personal' and place */
/* it into 'persindex'.                                               */
/* Hint: Use the POSSTR function.                                     */
/**********************************************************************/
 sql1 = "        ";

 stmt1 = samplecon.prepareStatement ( sql1 );
 System.out.println ("Value of stmt1 is: \n");
 System.out.println (stmt1);
 stmt1.setString( 1,empnum );
 System.out.println ("\n Value of empnum is: \n");
 System.out.println (empnum);
 rslob1 = stmt1.executeQuery();

 while (rslob1.next()) {
/************************ ????????????????????? **********************/
/* Get integer value for the current row and assign to persindex      */
/**********************************************************************/

   persindex = ????????? ;
   System.out.println ("\n Value of persindex is: \n");
   System.out.println (persindex);

 } // end while

/************************ ????????????????????? **********************/
/* Determine the starting position of the string 'Department' and     */
/* place it into 'deptindex'.                                         */
/* Hint: Use the POSSTR function.                                     */
/**********************************************************************/

sql2 = "               ";


 stmt2 = samplecon.prepareStatement ( sql2 );
 stmt2.setString( 1,empnum );
 System.out.println ("\n Value of empnum is: \n");
 System.out.println (empnum);
 rslob2 = stmt2.executeQuery();

 while (rslob2.next()) {

/************************ ????????????????????? **********************/
/* Get integer value for the current row and assign to deptindex      */
/**********************************************************************/

   deptindex = ??????? ;
   System.out.println ("\n Value of deptindex is: \n");
   System.out.println (deptindex);

 } // end while
```

```
/************************ ????????????????????? ***********************/
/* Retrieve the employee number, resume format, and the part of the   */
/* resume not containing Personal information for the employee number */
/* supplied.                                                          */
/* Hint: Use the SUBSTR function and concatenation operator to        */
/* retrieve that portion of the RESUME column needed.                 */
/**********************************************************************/

/**********************************************************************/
/*    First calculate value for persindex1 as persindex - 1 to define */
/*    offset to one (1) position before personal data                 */
/**********************************************************************/

persindex1 = persindex - 1;
System.out.println ("\n Value of persindex1 is: \n");
System.out.println (persindex1);

sql3 = "            ";

 stmt3 = samplecon.prepareStatement ( sql3 );

/*********************?????????????????????????????????????????????*/
/*    Set the values of the parameter markers in the SELECT statement */
/*    Hint: Use setXXX methods to set the values of parameter         */
/*          markers to perindex1, deptindex, and empnum values.       */
/**********************************************************************/

 stmt3.????????;
 stmt3.????????;
 stmt3.????????;

 System.out.println ("\n Value of empnum is: \n");
 System.out.println (empnum);
 rslob3 = stmt3.executeQuery();

 /******************* ??????????????????????????? ***********************/
 /* For each row, print out the employee number, the resume format,    */
 /* and the resultant data after the Personal information has been     */
 /* excluded.                                                          */
 /* Hint: Use getXXX methods to retrieve column values into variables. */
 /**********************************************************************/
 while ( rslob3.next() )
 {
   empno = rslob3.????????;
   resumefmt = rslob3.????????;
   resumelob = rslob3.????????;
```

```
        //  Determine the length of the clob data
        long len = resumelob.length();
        //  Assign the length to an int variable for use with getSubString
        //  Note that loss of precision may occur
        int len1 = (int)len;
        /******************??????????????????*********************************/
        /* Materialize clob data for displaying using the getSubString method */
        /*********************************************************************/
        String resumeout = resumelob.???????? ;
        System.out.println("Empno " + empno + " has resume format of " + resumefmt);
        System.out.println("Resume contents are: " + resumeout );
      }
   }  // end try
   catch ( SQLException x )
   {
     System.out.println("Error on fetch of rows" + x.getErrorCode() );
     x.printStackTrace();
   }
   System.exit(0);

} // end main

}  // end of javalob class
```

Exercise 7. Performance and Tuning

What This Exercise Is About

This exercise will allow you test the students knowledge of performance parameters with DB2 UDB V8.1 as it relates to performance and tuning.

What You Should Be Able to Do

At the end of the lab, students should be able to:

- Determine the query running
- How to update statistics on table
- How to run Visual Explain
- Decipher the output of the character explain

Introduction

You will invoke the program perform.java. Determine the SQL statement is running. The query will run against the table: TITLES.

The database being used is MUSICDB. The program perform2.java will execute a query against the tables: ARTISTS and TITLES. The program perform3.java and perform3_static.sqlj will execute a query against the tables: ARTISTS, TITLES and WAREHOUSE.

Instructor Notes

Introduction — The programs will test the ability of the students to tune SQL queries.

First program perform1.java will access the TITLES table.

```
SELECT TITLE, COMPANY, YEAR, ARTNO FROM TITLES
  WHERE TITLE LIKE 'Chri%' ORDER BY TITLE
```

Second program perform2.java will access the tables ARTISTS and TITLES.

```
SELECT  NAME, TITLES, TITLES.COMPANY, YEAR
  FROM ARTISTS, TITLES
  WHERE ARTISTS.ARTNO = TITLES.ARTNO
  AND NAME LIKE 'Ra%'
    ORDER BY NAME
```

The program perform3.java will access the database MUSICDB. The query:

```
SELECT NAME, TITLE, TITLES.COMPANY, QTY, WAREHOUSENAME
  FROM ARTISTS, TITLES, WAREHOUSE
  WHERE ARTISTS.ARTNO = TITLES.ARTNO
    AND  TITLES.ITEMNO = WAREHOUSE.ITEMNO"
    AND ARTISTS.NAME = 'Temptations'
  ORDER BY TITLES.COMPANY
```

The program perform3_static will access the database MUSICDB. The query:

```
SELECT NAME, TITLE, TITLES.COMPANY, QTY, WAREHOUSENAME
  FROM ARTISTS, TITLES, WAREHOUSE
  WHERE ARTISTS.ARTNO = TITLES.ARTNO
    AND  TITLES.ITEMNO = WAREHOUSE.ITEMNO"
    AND ARTISTS.NAME = 'Temptations'
  ORDER BY TITLES.COMPANY
```

The program will be an SQLJ program. Request that the student compare the execution time of each program perform3.java, which will be running JDBC, and the perform3_static.sqlj, which will be running SQLJ.

Time for Lab — 1.5 hours.

Things to Review at End of Lab — The DB2 UDB cost-based optimizer requires current statistics to determine the best access path. Indexes created on the proper column of TITLES.TITLE will improve the performance of the program perform.java. An index on the column TITLES.NAME and multiple column index on the table ARTISTS using the columns ARTNO and NAME will improve the performance of the program perform2.java.

The SQLJ program should run faster than the JDBC program because the SQL statement has been pre-compiled. If the student runs the program perform3.java multiple times, the execution times between JDBC and the SQLJ programs will be fairly equal. Using the bindoptions = 'BLOCKING ALL' should improve the performance of the SQLJ program.

You should try and keep the database active by issuing a **db2 activate db musicdb**. This way the start up time of the database is not a factor in the execution performance of the SQL statement.

Use the **db2 flush package cache dynamic** command to clear the SQL statement from package cache, after each execution of the program.

Exercise Instructions

Section 0 - Introduction

Three tables have been created on the database MUSICDB.

ARTISTS

Artno	Name	Company	BIO	Picture
Smallint	Char(40)	Char(20)	CLOB(100K)	BLOB(500K)

TITLES

Artno	Title	Type	Year	Company	Distrib	Cost	Retail	Itemno	
Smallint	Char(20)	Char(1)	Char(4)	Char(20)	Char(20)	Dec(5,2)	Dec(5,2)	Smallint	

WAREHOUSE

Itemno	Warehousename	QTY
Smallint	Char(20)	Integer

The tables have no indexes. There are no referential constraints on any tables.

Introduction

This exercise will attempt to demonstrate how to determine the access path of SQL statement. Determine the SQL statement that was issued. Determine what objects may be created to improve the performance.

Section 1

__ 1. Compile the java program perform1.java.

_____

__ 2. Execute the program and record the execution time of the SQL statement.

_____

__ 3. Run Visual Explain on the query. Did the query use any indexes? What was the timeron cost? Did the table have statistics updated.

_____

__ 4. Run statistics against each of the tables.

_____

__ 5. Execute the program again.

_____

__ 6. Add index(es), that will improve performance on the query.

_____

__ 7. Run Visual Explain on the query. Did the query use any indexes? What was the timeron cost?

_____

__ 8. Execute the program again. Did the elapsed time of the query decrease?

_____

Exercise Solutions

Section 1

__ 1. Compile the java program perform1.java.

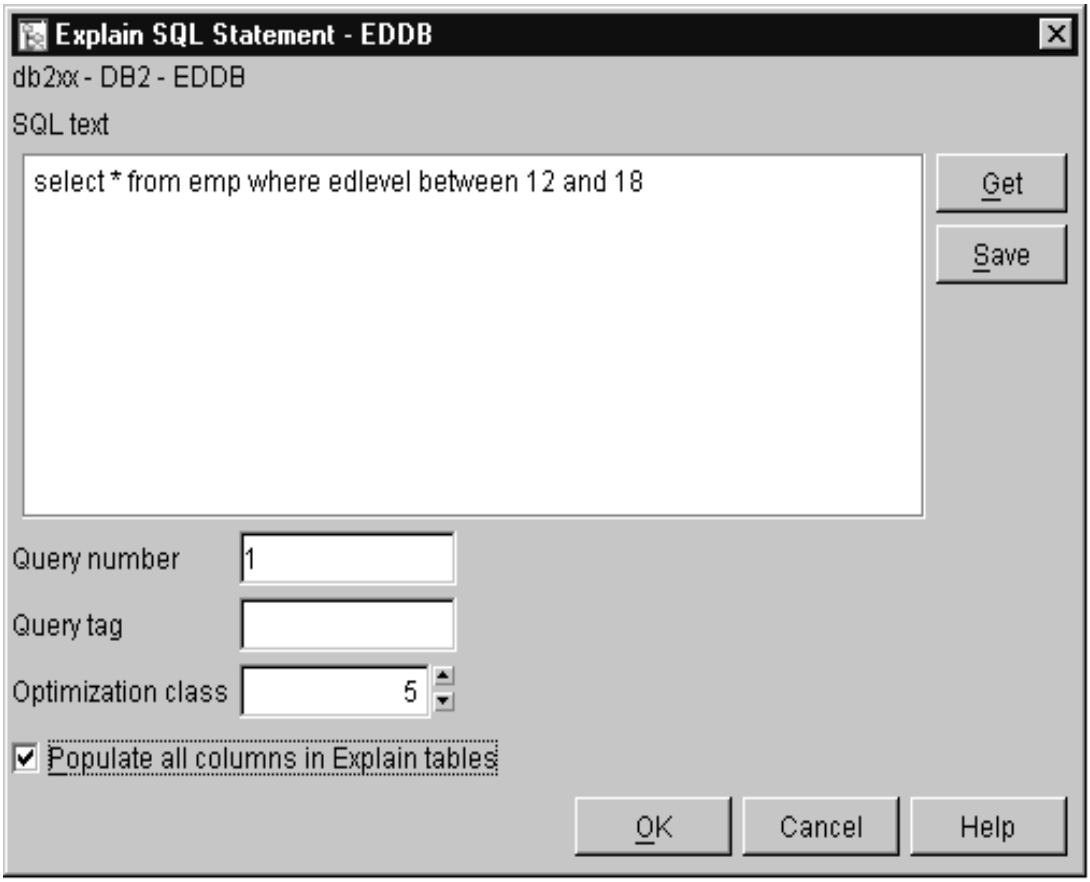
javac perform1.java

__ 2. Execute the program and record the execution time of the SQL Statement.

```
SELECT TITLE, COMPANY, YEAR, ARTNO
    FROM TITLES
WHERE TITLE LIKE 'Chr%'
ORDER BY TITLE
```

__ 3. Run Visual Explain on the query. Did the query use any indexes? What was the timeron cost?

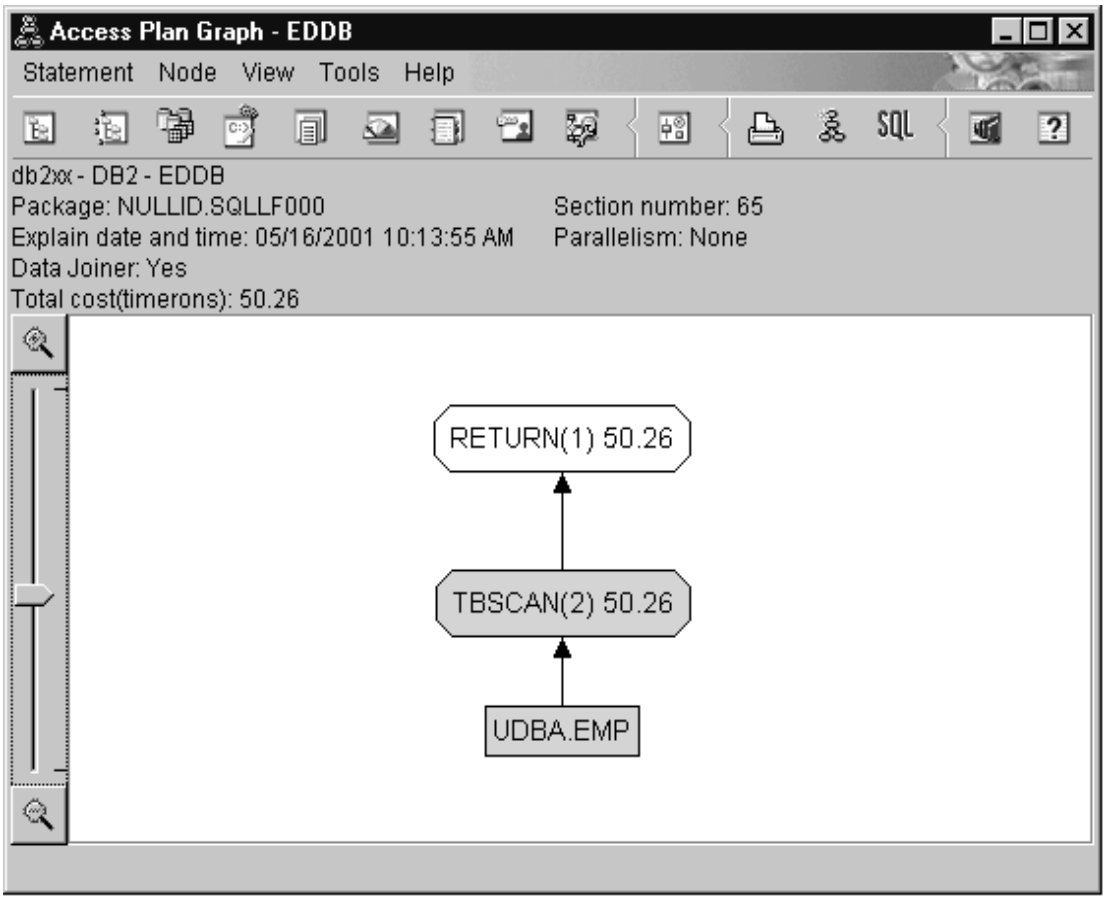
Open the Control Center
Select the database **MUSICDB** with right mouse button
Select the menu item **Explain SQL**.
Select **Get** command button.
File Name: **C:\CG112\perform.sql**. Select **Open** command button.
Place the SQL statement in the SQL text box.
Select **OK** command button.



The query did not use any indexes.

To determine if statistics had been updated against the table TITLES,

Double-click the TITLES box at the bottom of the access path.



Notice the STATS_TIME has not been updated. Statistics have not been run against table TITLES.

__ 4. Run statistics the table TITLES.

RUNSTATS ON TABLE UDBA.TITLES ON ALL COLUMNS WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL ALLOW WRITE ACCESS;

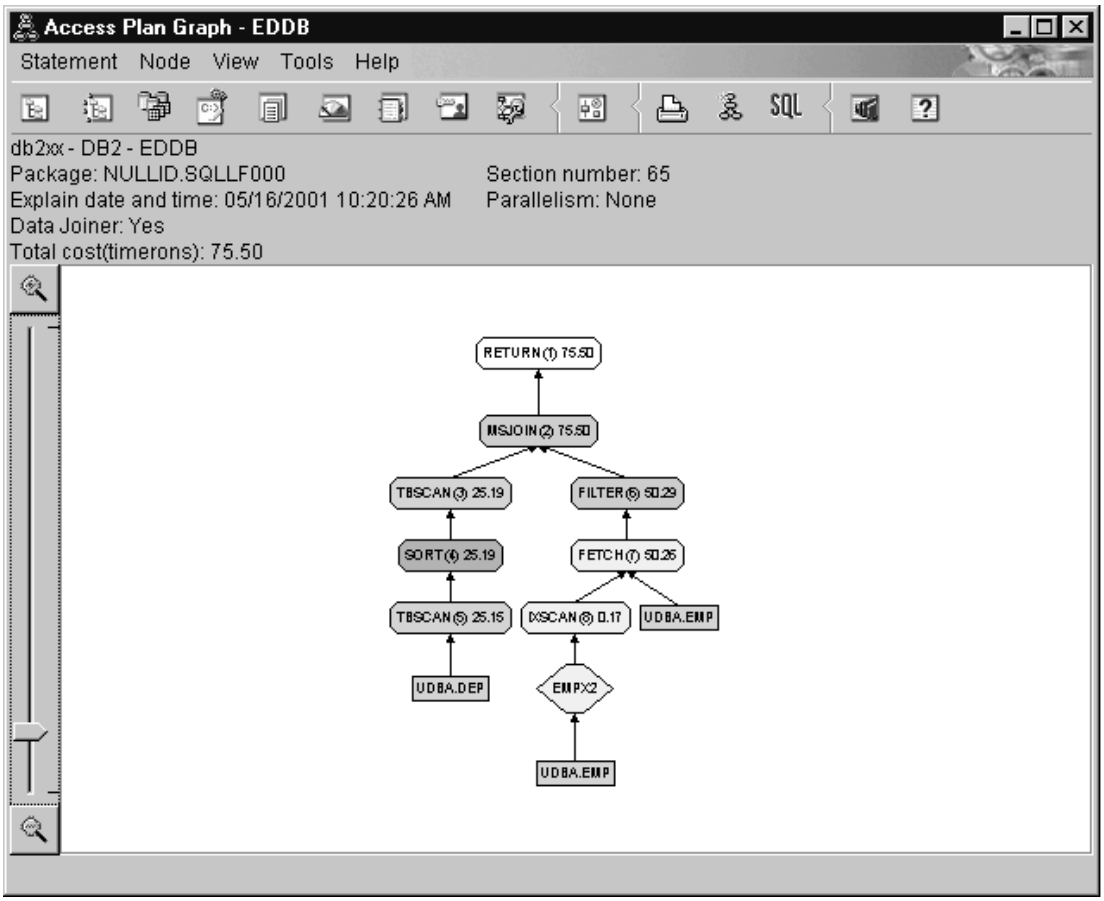
__ 5. Execute the program again.

java perform

__ 6. Add the index, that will improve performance on the query.

CREATE INDEX TITLES_TITLE_IDX on TITLES (TITLE)

__ 7. Run Visual Explain on the query. Did the query use any indexes? What was the timeron cost?



You may also want to run dynexpln to see a character explain output.

dynexpln -d musicdb -z @ -f perform.sql -o perform.out

__ 8. Execute the program again. Did the elapsed time of the query decrease?

java perform

Yes

Exercise Instructions

The program **perform2.java** is going to retrieve all the albums for each artists. The tables to be accessed will be ARTISTS and TITLES.

Section 2

__ 1. Execute the program perform2.java. Determine the SQL statement that is executed.

_____

__ 2. Execute Visual Explain on the table. Determine if statistics have been run against the two tables ARTISTS and TITLES.

_____

__ 3. Try and improve the performance of the query by running statistics and creating indexes.

_____

__ 4. Run Visual Explain and see if the timeron cost decreases.

_____

Exercise Solutions

The program **perform2.java** is going to retrieve all the albums for each artists. The tables to be accessed will be ARTISTS and TITLES.

Section 2

__ 1. Execute the program perform2.java. Determine the SQL statement that is executed.

javac perform2.java
java perform

db2 get snapshot for dynamic sql for musicdb

```
SELECT NAME, TITLE, TITLES.COMPANY, YEAR
    FROM ARTISTS, TITLES
WHERE ARTISTS.ARTNO = TITLES.ARTNO
    AND NAME = 'Ra%'
    ORDER BY NAME
```

__ 2. Execute Visual Explain on the table. Determine if statistics have been run against the two tables ARTISTS and TITLES.

Open the Control Center.
Select database MUSICDB with the right mouse button.
Select the Explain SQL menu item.
Select **Get** command button.
File Name: **C:\CG112\perform2.sql**. Select **Open** command button.
Place the SQL statement in the SQL text box.
Select **OK** command button.

__ 3. Try and improve the performance of the query by running statistics and creating indexes.

CREATE INDEX ARTISTS_NAME ON ARTISTS (NAME)

RUNSTATS ON TABLE UDBA.ARTISTS ON ALL COLUMNS WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL

RUNSTATS ON TABLE UDBA.TITLES ON ALL COLUMNS WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL

__ 4. Run Visual Explain and see if the timeron cost decreases.

Exercise Instructions

Programs perform3.java and perform3_static.sqlj issue the same query. Compare the execution time between the two programs. Add indexes that you feel might assist the query in executing in shorter time frame.

Note: You will have to precompile the program perform3_static.sqlj and create a new package for the changes to affect the execution of the program. The query is:

```
SELECT NAME, TITLE, TITLES.COMPANY, QTY, WAREHOUSENAME
  FROM ARTISTS, TITLES, WAREHOUSE
  WHERE ARTISTS.ARTNO = TITLES.ARTNO
    AND TITLES.ITEMNO = WAREHOUSE.ITEMNO
    AND ARTISTS.NAME = 'Temptations'
ORDER BY TITLES.COMPANY
```

Section 3

__ 1. Compile and execute the program **perform3.java**.

_____

__ 2. Use Visual Explain to verify proper indexes and statistics have been completed.

_____

__ 3. Precompile, compile and run the db2sqljcustomize to create a package for the program **perform3_static.sqlj**.

_____

__ 4. Execute the program **perform3_static.sqlj**.

_____

__ 5. Compare the timings.

_____

__ 6. If you have time try to improve the performance of the programs. Suggestions could be indexes, statistics, sort heap size, buffer pools, and so forth.

_____

Exercise Solutions

Programs perform3.java and perform3_static.sqlj issue the same query. Compare the execution time between the two programs. Add indexes that you feel might assist the query in executing in shorter time frame.

Note: You will have to precompile the program perform3_static.sqlj and create a new packages for the changes to affect the execution of the program. The query is:

```
SELECT NAME, TITLE, TITLES.COMPANY, QTY, WAREHOUSENAME
  FROM ARTISTS, TITLES, WAREHOUSE
  WHERE ARTISTS.ARTNO = TITLES.ARTNO
    AND TITLES.ITEMNO = WAREHOUSE.ITEMNO
    AND ARTISTS.NAME = 'Temptations'
ORDER BY TITLES.COMPANY
```

Section 3

__ 1. Compile and execute the program **perform3.java**.

javac perform3.java
java perform3

__ 2. Use Visual Explain to verify proper indexes and statistics have been completed.

Open the Control Center.
Select database MUSICDB with the right mouse button.
Select the Explain SQL menu item.
Select **Get** command button.
File Name: **C:\CG112\perform3.sql**. Select **Open** command button.
Place the SQL statement in the SQL text box.
Select **OK** command button.

__ 3. Precompile, compile and run the db2sqljcustomize to create a package for the program **perform3_static.sqlj**.

sqlj -url=jdbc:db2:musicdb -compile=false perform3_static.sqlj

javac perform3_static.java

db2sqljcustomize -url jdbc:db2://localhost:50000/musicdb -user udba -password udba perform3_static_SJProfile0.ser

__ 4. Execute the program perform3_static.sqlj.

java perform3_static

__ 5. Compare the timings.

__ 6. If you have time try to improve the performance of the programs. Suggestions could be indexes, statistics, sort heap size, buffer pools, and so forth.

CREATE INDEX TITLES_ARTNO_IDX ON TITLES (ARTNO)
CREATE INDEX WARE_ITEMNO_IDX ON WAREHOUSE (ITEMNO)

RUNSTATS ON TABLE UDBA.WAREHOUSE ON ALL COLUMNS
WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL

RUNSTATS ON TABLE UDBA.TITLES ON ALL COLUMNS
WITH DISTRIBUTION ON ALL COLUMNS AND INDEXES ALL

Attempt to turn on blocking.

db2sqljcustomize -url jdbc:db2://localhost:50000 -user udba -password udba -bindoptions "BLOCKING ALL" perform3_static_SJProfile0.ser

END OF LAB

Appendix A. Application Alternatives - Paper Lab

What This Exercise Is About

There are several alternative ways to approach any programming problem. While this course will address the coding of static SQL, you should have the opportunity to identify other alternatives to specific business needs so you can determine the usefulness of these other solutions in your business environment.

What You Should Be Able to Do

At the end of the lab, students should be able to determine an appropriate solution type for given application scenarios.

Introduction

In this paper lab, you will evaluate specific application requirements and determine the appropriate way to answer the requirements based on various application alternatives.

Instructor Notes

Introduction — This is a paper lab to give the students an opportunity to evaluate business needs and what kind of application program would be needed to support that need. At this point, remind the students that the solutions are available following the lab questions that provide the answers to all the problems posed in the lab guide.

A student may not provide an identical solution to those presented in the solutions guide. This is not necessarily incorrect. The key is that the student applied knowledge of the various application alternatives to justify any answer given. In other words, there may be more than one solution appropriate for a given requirement - depending on the interpretation of that requirement.

Time for Lab — 10 minutes.

Things to Review at End of Lab — Go quickly over each answer and get their answers.

Exercise Instructions

The lab solutions can be found in “Exercise Solutions” on page A-4.

__ 1. Chris is a DBA who wants to automate the creation of backups on her DB2 system. She comes to you to develop a program to do this.

You decide to use:

_____

_____

__ 2. Ron needs to access employee names and department numbers from the EMPLOYEE table. We do not want to give him direct access to the EMPLOYEE table, so you have been asked to provide an application program that will give Ron a nice interface for accessing this information.

You decide to use:

_____

_____

__ 3. You have been hired by a software house that provides an application to perform ad hoc queries against a number of different database products.

In order to prevent having to provide specific code for each database product supported, you decide to use:

_____

_____

__ 4. The software application described in the prior step is going to be implemented in Microsoft Windows.

You decide to use:

_____

_____

__ 5. Harvey has asked you to help him retrieve test data from the production databases. While programmers are not allowed access to the entire production database, they will be allowed to sample data through your application. Your application should accept a table name from any number of clients and return 5% of the rows to the calling program.

You decide to implement this as:

_____

_____

__ 6. Lynn has requested an application that will be used by all of the marketing reps (100,000) to query and update their work plans and work-in-progress on a daily basis. The requests they need to enter are well known, and the database statistics are not expected to change dramatically over time.

You decide to use:

_____

_____

__ 7. You are writing an application against a DB2 database. Some of the columns desired and the predicate is not totally known until execution time.

You decide to use:

_____

_____

END OF LAB

Exercise Solutions

__ 1. Chris is a DBA who wants to automate the creation of backups on her DB2 system. She comes to you to develop a program to do this.

You decide to use:

_____

_____

APIs (utility cannot be run through SQL calls)

Stored procedures (must be on the server to run the backup utility)

__ 2. Ron needs to access employee names and department numbers from the EMPLOYEE table. We do not want to give him direct access to the EMPLOYEE table, so you have been asked to provide an application program that will give Ron a nice interface for accessing this information.

You decide to use:

_____

_____

Static SQL (probably with a view)

__ 3. You have been hired by a software house that provides an application to perform ad hoc queries against a number of different database products.

In order to prevent having to provide specific code for each database product supported, you decide to use:

_____

_____

CLI or JDBC

__ 4. The software application described in the prior step is going to be implemented in Microsoft Windows.

You decide to use:

_____

_____

ODBC

__ 5. Harvey has asked you to help him retrieve test data from the production databases. While programmers are not allowed access to the entire production database, they will be allowed to sample data through your application. Your application should accept a table name from any number of clients and return 5% of the rows to the calling program.

You decide to implement this as:

_____

_____

Stored procedures using dynamic SQL

__ 6. Lynn has requested an application that will be used by all of the marketing reps (100,000) to query and update their work plans and work-in-progress on a daily basis. The requests they need to enter are well known, and the database statistics are not expected to change dramatically over time.

You decide to use:

_____

_____

Static SQL

__ 7. You are writing an application against a DB2 database. Some of the columns desired and the predicate is not totally known until execution time.

You decide to use:

_____

_____

Dynamic SQL

(CLI or JDBC would also be options)

END OF LAB

Appendix B. JDBC and Stored Procedures Solutions

JDBCcli.java Solution

```
/*********************************************************************/
/*                                                                   */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"          */
/*                          ( CG11 )                                 */
/*                                                                   */
/*                                                                   */
/* Last update = 01/31/2000                                          */
/*                                                                   */
/*********************************************************************/
/*  Notes:                                                           */
/*                                                                   */
/*  This program is intended to be completed with the lab guide      */
/*  as a reference.  The lab guide is the set of instructions that   */
/*  should be followed.  The comments in this program are intended   */
/*  to clarify statements made in the lab guide.                     */
/*********************************************************************/

/*********************************************************************/
/* Import Java Classes                                               */
/*********************************************************************/
import java.lang.*;
import java.util.*;
import java.io.*;
import java.sql.*;              // JDBC classes
import COM.ibm.db2.jdbc.app.*;  // DB2 UDB JDBC classes

class JDBCcli
{ static
  { try
    { Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (Exception e)
    { System.out.println ("\n  Error loading DB2 Driver...\n");
      System.out.println (e);
      System.exit(1);
    }
  }
```

```
// main application: .connect to the database
//                   .call the stored procedure
public static void main (String argv[])
{ Connection con = null;
  try
  { System.out.println ("Java Multiple Resultsets Stored Procedure");
    // Connect to EDDB database

    // URL is jdbc:db2:dbname
    String url = "jdbc:db2:eddb";

    if (argv.length == 0)
    { // connect with default id/password
      con = DriverManager.getConnection(url);
    }
    else if (argv.length == 2)
    { String userid = argv&lbrk.0&rbrk.;
      String passwd = argv&lbrk.1&rbrk.;

      // connect with user-provided username and password
      con = DriverManager.getConnection(url, userid, passwd);
    }
    else
    {   throw new Exception("\nUsage: java JDBCcli [username password]\n");
    }

    // Set AutoCommit
    con.setAutoCommit(true);
    String callName = "MultRes";

    /****************** ?????????????????????? *********************/
    /* Create callable statement to call the Stored Procedure       */
    /* Print the statement out in the following line                */
    /****************************************************************/
    String callSql = "Call " + callName + "()";
    System.out.println("Creating CallableStatement = " + callSql);

    /****************** ?????????????????????? *********************/
    /* Prepare the callable statement                               */
    /****************************************************************/
    CallableStatement callStmt = con.prepareCall(callSql);

    ResultSet rs = null;
    System.out.println("\nExecuting the Java Stored Procedure now...\n");
    /****************** ?????????????????????? *********************/
    /* Execute the callable statement                               */
    /****************************************************************/
```

```
      callStmt.execute();
      int rsCount = 0;

      while( true )
      {
        /************************ ?????????????????? ******************/
        /* Assign the results to a the variable named rs (defined above) */
        /*************************************************************/
        rs = callStmt.getResultSet();
        if( rs != null )
        { rsCount++;
          System.out.println("Fetching all the rows from the result set #" + rsCount);
          fetchAll(rs);
          /**************** ??????????????? ***************************/
          /* Move to the next result set                              */
          /*************************************************************/
          callStmt.getMoreResults();
          System.out.println("");
          continue;
        }

        break;
      }


      // close off everything before we leave
      System.out.println("Closing statements and connection.");
      callStmt.close ();
      con.close ();
    }
    catch (Exception e)
    { try
      { if( con != null )
        { con.close();
        }
      }
     catch (Exception x)
     { //ignore this exception
     }
     System.out.println (e);
   }
}

// ================================================
// Method: fetchAll
// ================================================
static public void fetchAll( ResultSet rs )
```

```
 { try
   {         System.out.println("================================="
                                + "=================================");
      ResultSetMetaData stmtInfo = rs.getMetaData();
      int numOfColumns = stmtInfo.getColumnCount();
      int r = 0;
      System.out.println("number of columns is " + numOfColumns);
      while( rs.next() )
      { r++;
        System.out.print("Row: " + r + ": ");
        for( int i=1; i <= numOfColumns; i++ )
        { System.out.print(rs.getString(i));
          if( i != numOfColumns ) System.out.print(" , ");
        }
        System.out.println("");
      }
    }
    catch (SQLException e)
    { System.out.println("Error: fetchALL: exception");
      System.out.println (e);
    }
  }
}
```


MultRes.sqlj Solution

```
/*************************************************************************/
/*                                                                       */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"              */
/*                          ( CG11 )                                     */
/*                                                                       */
/*                                                                       */
/* Last update = 01/31/2000                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/
import java.sql.*;              // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

/************************??????????????????????????? ********************/
/* Define iterators for 3 select statements                              */
/*************************************************************************/
#sql iterator MultRes_Cursor1 (double salary) ;
#sql iterator MulTRes_ByPos (String, String,String,String);
#sql iterator MultRes_ByName (String EmpNo, String LastName);

///////
// Java stored procedure is in this class
///////
public class MultRes
{   // (1) stored procedure body
  public static void  SPBody ( ) throws Exception
  {
     /******************* ????????????????????? *******************/
     /* Define iterator variables for local use                   */
     /*************************************************************/
     MultRes_ByPos results;
     MultRes_ByName  nameiter;
     MultRes_Cursor1  cursor1;
     try
```

```
      {
         /****************** ???????????????????? *****************/
         /*  Issue Select statments to open cursors and leave open. */
         /*********************************************************/
         #sql cursor1 = { SELECT salary FROM EMP ORDER BY salary };
         #sql results = { SELECT * FROM DEP };
         #sql nameiter = { SELECT EMPNO, LASTNAME FROM EMP };
      }
      catch (Exception e)
      {   throw e;
      }
   }
}
```

MultiRes.mem Solution

```
drop procedure MultRes;
create procedure MultRes()

  language java
  parameter style java
  fenced
  external name 'MultResjar:MultRes.SPBody'
  reads sql data;
```

JDBCcli1.java Solution

```
/*************************************************************************/
/*                                                                       */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"              */
/*                          ( CG11 )                                     */
/*                                                                       */
/*                                                                       */
/* Last update = 01/31/2000                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/
import java.lang.*;
import java.util.*;
import java.io.*;
import java.sql.*;              // JDBC classes
import COM.ibm.db2.jdbc.app.*;  // DB2 UDB JDBC classes

class JDBCcli2
{ static
  { try
    { Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (Exception e)
    { System.out.println ("\n  Error loading DB2 Driver...\n");
      System.out.println (e);
      System.exit(1);
    }
  }

  // main application: .connect to the database
  //                   .call the stored procedure
  public static void main (String argv&lbrk.&rbrk.)
  { Connection con = null;
    try
    { System.out.println ("Java Multiple Resultsets Stored Procedure");
      // Connect to EDDB database
```



```
// URL is jdbc:db2:dbname
String url = "jdbc:db2:eddb";

if (argv.length == 0)
{ // connect with default id/password
  con = DriverManager.getConnection(url);
}
else if (argv.length == 2)
{ String userid = argv&lbrk.0&rbrk.;
  String passwd = argv&lbrk.1&rbrk.;

  // connect with user-provided username and password
  con = DriverManager.getConnection(url, userid, passwd);
}
else
{   throw new Exception("\nUsage: java JDBCcli2 [username password]\n");
}

// Set AutoCommit
con.setAutoCommit(true);
/******************** ????????????????????? **********************/
/* Assign the correct procedure name to be called                 */
/******************************************************************/
String callName = "RESSET1";

String callSql = "Call " + callName + "()";
System.out.println("Creating CallableStatement = " + callSql);

CallableStatement callStmt = con.prepareCall(callSql);

ResultSet rs = null;
System.out.println("\nExecuting the Java Stored Procedure now...\n");
callStmt.execute();
int rsCount = 0;

while( true )
{
  rs = callStmt.getResultSet();
  if( rs != null )
  { rsCount++;
    System.out.println("Fetching all the rows from the result set #" + rsCount);
    fetchAll(rs);
    callStmt.getMoreResults();
    System.out.println("");
    continue;
  }

  break;
}
```

```
      // close off everything before we leave
      System.out.println("Closing statements and connection.");
      callStmt.close ();
      con.close ();
    }
    catch (Exception e)
    { try
      { if( con != null )
        { con.close();
        }
      }
     catch (Exception x)
     { //ignore this exception
     }
     System.out.println (e);
   }
 }

 // ================================================
 // Method: fetchAll
 // ================================================
 static public void fetchAll( ResultSet rs )
 { try
   {         System.out.println("================================="
                                + "=================================");
      ResultSetMetaData stmtInfo = rs.getMetaData();
      int numOfColumns = stmtInfo.getColumnCount();
      int r = 0;
      System.out.println("number of columns is " + numOfColumns);
      while( rs.next() )
      { r++;
        System.out.print("Row: " + r + ": ");
        for( int i=1; i <= numOfColumns; i++ )
        { System.out.print(rs.getString(i));
          if( i != numOfColumns ) System.out.print(" , ");
        }
        System.out.println("");
      }
    }
    catch (SQLException e)
    { System.out.println("Error: fetchALL: exception");
      System.out.println (e);
    }
  }
}
```

Appendix C. UDT/UDF/LOB Program Solutions

degree.java Solution

```
/*********************************************************************/
/*                                                                   */
/* Sample Java program for "DB2 UDB Advanced Programming"            */
/*                          ( CF11 )                                 */
/*                                                                   */
/*                                                                   */
/* Last update = 01/20/2000                                          */
/*                                                                   */
/*********************************************************************/
/* This function will provide for conversions of EDLEVEL to a        */
/* character string representation of the education level.           */
/*    20 or above - DOCTORATE                                        */
/*    18, 19      - MASTERS                                          */
/*    16, 17, 18  - COLLEGE DEGREE                                   */
/*    13, 14, 15  - SOME COLLEGE                                     */
/*    12          - HIGH SCHOOL GRADUATE                             */
/*    1 - 11      - GED                                              */
/*********************************************************************/
/*  Notes:                                                           */
/*                                                                   */
/*  This program is intended to be completed with the lab guide      */
/*  as a reference.  The lab guide is the set of instructions that   */
/*  should be followed.  The comments in this program are intended   */
/*  to clarify statements made in the lab guide.                     */
/*********************************************************************/

/*********************************************************************/
/* Import Java Classes                                               */
/*********************************************************************/
import java.sql.*;
import COM.ibm.db2.app.*;        // UDF and associated classes
import java.io.*;
import java.util.*;

/********************** ???????????????????? ************************/
/* Define the method                                                 */
/*********************************************************************/
class Degree extends UDF
{
```

```
/*****************************************************************************/
/*  EDLEVEL to education level conversion                                    */
/*****************************************************************************/
/*********??????????????????????????????????????????????????????************/
/* Code the entry point information for your function                        */
/*   - indicate the return value for the function                            */
/*   - indicate the entry point name for the function                        */
/*   - indicate input parameter(s) for the function                          */
/*****************************************************************************/
  public static String degree( short edlevel ) throws SQLException
  {
    String degree_char = null;

   /******??????????????????????????????????????????????????????******/
   /* Determine the value to be returned.                                   */
   /*************************************************************************/

    if ( edlevel >= 20)  return "DOCTORATE";
               /* check the input value, if it is 20 or greater */
               /* put the value of "DOCTORATE" into the return value   */

    switch (edlevel)     /* switch on the input value...*/
    {
       case 19:
       case 18:  degree_char = "MASTERS";
                 break;
       case 17:
       case 16:  degree_char = "COLLEGE DEGREE";
                 break;
       case 15:
       case 14:
       case 13:  degree_char = "SOME COLLEGE";
                 break;
       case 12:  degree_char = "HIGH SCHOOL GRADUATE";
                 break;
       default:  degree_char = "GED";
                 break;
    }  // end switch

    /****** ?????????????????????????????????????????????????? ******/
    /*  Identify the value to return                                         */
    /*************************************************************************/
    return (degree_char);

  } // end method degree

}  // end class Degree
```

degree DDL

```
drop function degree;
create function degree ( smallint )
   returns char(20)
   external name 'degreejar:Degree.degree'
   language java
   parameter style java
deterministic
fenced
not null call
   no sql
no external action
no scratchpad
no final call
allow parallel
no dbinfo;
```

convert.java Solution

```
/*************************************************************************/
/*                                                                       */
/* Sample Java program for "DB2 UDB Programming Using Java"              */
/*                          ( CG11 )                                     */
/*                                                                       */
/*                                                                       */
/* Last update = 01/31/2000                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/
import java.lang.*;
import COM.ibm.db2.app.*;
import java.sql.*;
import java.io.*;

/************************ ?????????????????? ***********************/
/* Code the method definition.                                           */
/*************************************************************************/
class convert extends UDF
{
  BufferedReader in;
//  BufferedWriter outfile;  /* debugging file */
  String s;
  String FuncName;
  String func_from_file;
  String double_from_file;
  double convrate = 0.0;

/************************ ?????????????????? ***********************/
/* Code the entry point for this function.                               */
/*************************************************************************/
  public void us_to_oth (double usCurrency,
                         double otherCurrency) throws Exception
  {
    try
```



```
    {
      /******************** ????????????????? *********************/
      /* Check the file name in the next statement.                */
      /*************************************************************/
      in = new BufferedReader(new FileReader("c:\\cg11\\convfile.txt"));
//      outfile = new BufferedWriter(new FileWriter("c:\\cg11\\outfile.txt"));
//      outfile.write("\n This is the way to write to a file\n");
    }
    catch ( IOException x)
    {
      throw x;
    }
    try
    {
      /***************** ??????????????????? ***********************/
      /* Assign the function name that was used to call this function   */
      /* to FuncName.                                                   */
      /******************************************************************/
      FuncName = getFunctionName();
      while ( null != (s = in.readLine() ) )
      {
        func_from_file = s.substring(0,8);
        double_from_file = s.substring(9);
        /******************** ????????????????? ***********************/
        /* Look for a matching function name in the file                */
        /************************************************************/
        if ( FuncName.equals(func_from_file) )
        {
           convrate = Double.valueOf(double_from_file).doubleValue();
           break;
        }
      }
      if ( convrate != 0.0 )
      {
         /************************ ????????????????? *****************/
         /* Set the return value to be returned to DB2.                */
         /**************************************************************/
         set(2,usCurrency * convrate);
      }
      else
      {
         otherCurrency = 0.0;
      }
//      outfile.close();
      return;
    }
    catch ( Exception x)
    { // outfile.close();
      throw x;
```

```
    }
  } // end usToOth
}
```

convert DDL

```
drop function can;
create function can ( double )
   returns dec(9,2) cast from double
   external name 'moneyconvertjar:convert.us_to_oth'
   language java
   parameter style db2general
deterministic
fenced
not null call
   no sql
no external action
no scratchpad
no final call
allow parallel
no dbinfo;
```

DM and YEN DDL

```
drop function dmk;
create function dmk ( double )
   returns dec(9,2) cast from double
   external name 'moneyconvertjar:moneyconvert.us_to_oth'
   language java
   parameter style db2general
deterministic
fenced
not null call
   no sql
no external action
no scratchpad
no final call
allow parallel
no dbinfo;

drop function yen;
create function yen ( double )
   returns dec(9,2) cast from double
   external name 'moneyconvertjar:moneyconvert.us_to_oth'
   language java
   parameter style db2general
deterministic
fenced
not null call
   no sql
no external action
no scratchpad
no final call
allow parallel
no dbinfo;
```

UDT DDL

```
-- udt.mem

create distinct type us_dollar
   as decimal(9,2)
   with comparisons;

create distinct type flt_dollar
   as float
   with comparisons;
```

Create Table DDL

```
-- min_max.mem

create table salary_min_max
      ( job           char(8),
        min_salary    us_dollar,
        max_salary    flt_dollar);
```

Average Function DDL

```
-- avg.mem

create function
  AVG(us_dollar)
  returns us_dollar
  source avg(decimal);

create function
  AVG(flt_dollar)
  returns flt_dollar
  source avg(float);
```

UDT with UDF DDL

```
-- fltdudf.mem

create function can(flt_dollar)
   returns dec(9,2) cast from double
   external name 'moneyconvertjar:convert.us_to_oth'
   language java
   parameter style db2general
   deterministic
   fenced
   not null call
   no sql
   no external action
   no scratchpad
   no final call
   allow parallel
   no dbinfo;
```

perform.java Solution

```
/*************************************************************************/
/*                                                                       */
/* Sample Java program for "DB2 UDB PROGRAMMING USING JAVA"              */
/*                          ( CG11 )                                     */
/*                                                                       */
/*                                                                       */
/* Last update = 01/31/2000                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/

import java.lang.*;
import COM.ibm.db2.app.*;
import java.sql.*;
import java.io.*;

/****************************** ???????????? ************************/
/* Define the method                                                     */
/*************************************************************************/
class perform extends UDF
{
  BufferedReader filein;
  String s;
  int file_position = 1;

  /************************ ????????????????????? ********************/
  /* Define function with entry parameters                              */
  /*********************************************************************/
  public void perform ( int rating, String appraisal ) throws Exception
    {
      double double_from_string = 0;
      int local_rating = 0;
      switch (getCallType())
      {
        case SQLUDF_TF_FIRST:
          // do initialization which is independent
          //   of input parameters
          //  (nothing to do for this function)
          break;
```

```
case SQLUDF_TF_OPEN:
  // do initialization valid for this return table
  // open scan for web data
  /******************* ???????????????? **********************/
  /* Check the file open in the following statement, ensure     */
  /* the file referenced is correct.                            */
  /************************************************************/
  try
  { filein = new BufferedReader(new FileReader("d:\\cg11\\perform_list.txt")); }
  catch (IOException x){ throw x; }
  break;

case SQLUDF_TF_FETCH:
  // return "column" data for one "row"
  // when finished, indicate complete to DB2 via SQLSTATE
  int i = 0;
  try
  { // position in file to place where previous invocation had ended
    // (file_position keeps that information from run to run)
    while ( i < file_position )
    {
      s = filein.readLine();
      i++;
    }
    if ( s == null )          //  ???? indicate end of rows to DB2
      { setSQLstate("02000"); }
    else
      {
        double_from_string = Double.valueOf(s.substring(0,1)).doubleValue();
        local_rating = (int) double_from_string;
        /************************ ???????????? ************************/
        /* Assign values for the first and second parameters           */
        /***********************************************************/
        set(1,local_rating);
        set(2,s.substring(1));
        file_position = i;
      }
  } catch ( IOException x )
    { throw x; }
  break;

case SQLUDF_TF_CLOSE:
  // may reposition to top of scan, or close scan
  // state can be saved in scratchpad
  /************************ ???????????????? ***********************/
  /* Close the file                                                 */
  /*****************************************************************/
  filein.close();
  break;
```

```
        case SQLUDF_TF_FINAL:
          // disconnect from web server
          //  (nothing to do in this case)
          break;
      } // end switch
  } // end perform
}
```

perform DDL

```
-- perform.mem
drop function perform;
create function perform()
   returns table (rating integer, appraisal varchar(80) )
   specific perform10
   external name 'performjar:perform.perform'
   language java
   parameter style DB2general
   external action
   deterministic
   fenced
   no sql
   scratchpad
   final call
   disallow parallel
   no dbinfo
   cardinality 10
;
```

javalob Program Solution

```
/*************************************************************************/
/*                                                                       */
/*        Java program for "DB2 UDB PROGRAMMING USING JAVA"              */
/*        JDBC program to process a large object (LOB) column            */
/*                          ( CG112 )                                    */
/*                                                                       */
/*                                                                       */
/* Last update = 05/17/2001                                              */
/*                                                                       */
/*************************************************************************/
/*  Notes:                                                               */
/*                                                                       */
/*  This program is intended to be completed with the lab guide          */
/*  as a reference.  The lab guide is the set of instructions that       */
/*  should be followed.  The comments in this program are intended       */
/*  to clarify statements made in the lab guide.                         */
/*************************************************************************/

/*************************************************************************/
/* Import Java Classes                                                   */
/*************************************************************************/
import java.sql.*;
import java.io.*;
import java.util.*;

/*************************************************************************/
/* Class definition                                                      */
/*************************************************************************/
public class javalob
{
/*************************************************************************/
/* Register the class with the db2 Driver                                */
/*************************************************************************/
 static
    {   try
        {   Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (Exception e)
        {   System.out.println ("\n  Error loading DB2 Driver...\n");
            System.out.println (e);
            System.exit(1);
        }
    }
```

```
/*********************************************************************/
/* Main routine                                                      */
/*********************************************************************/
public static void main( String args[]) throws Exception
{
   // establish connection to sample database
   Connection samplecon = DriverManager.getConnection("jdbc:db2:sample");
   samplecon.setAutoCommit(false);

   // Provide information about what this program does
   String empnum = " ";
   System.out.println("This program will retrieve all information but the");
   System.out.println("Personal information for a given employee.");
   System.out.println("(Resumes exist for employees 000130, 000140, 000150, 000190)");
   System.out.println("Please enter an employee number:  ");

   // Read in the employee number that is to be processed
   DataInputStream dis = null;
   try
   { dis = new DataInputStream(System.in);
   } catch (Exception e) {e.printStackTrace(); System.exit(0);}
   try
   {
     String s;
     s = dis.readLine();
     empnum = s.substring(0,6);
   }
   catch (Exception e) {e.printStackTrace(); System.exit(0);}

   try
   {
     /******************** ?????????????????????? **********************/
     /* Define ResultSets which will be used to process the employee info */
     /*********************************************************************/
     ResultSet rslob1 = null;
     ResultSet rslob2 = null;
     ResultSet rslob3 = null;

     /************************ ???????????????????? ************************/
     /*  Define variables for LOB processing                              */
     /*********************************************************************/
     String            resume = null;
     int               persindex = 0, persindex1 = 0, deptindex = 0;
     String            sql1 = null, sql2 = null, sql3 = null;
     PreparedStatement stmt1 = null, stmt2 = null, stmt3 = null;
     String            empno = null, resumefmt = null;
     Clob              resumelob = null;
```

```
/************************ ?????????????????? ***********************/
/* Determine the starting position of the string 'Personal' and place */
/* it into 'persindex'.                                               */
/**********************************************************************/
 sql1 = "SELECT POSSTR(RESUME,'Personal') "
       + "FROM EMP_RESUME "
       + "WHERE EMPNO = ? AND RESUME_FORMAT = 'ascii' ";

 stmt1 = samplecon.prepareStatement ( sql1 );
 System.out.println ("Value of stmt1 is: \n");
 System.out.println (stmt1);
 stmt1.setString( 1,empnum );
 System.out.println ("\n Value of empnum is: \n");
 System.out.println (empnum);
 rslob1 = stmt1.executeQuery();

 while (rslob1.next()) {
    //  Get integer value for the current row
    persindex = rslob1.getInt(1);
    System.out.println ("\n Value of persindex is: \n");
    System.out.println (persindex);

 } // end while

/************************ ?????????????????? ***********************/
/* Determine the starting position of the string 'Department' and     */
/* place it into 'deptindex'.                                         */
/**********************************************************************/

sql2 = "SELECT POSSTR(RESUME,'Department') "
       + "FROM EMP_RESUME "
       + "WHERE EMPNO = ? AND RESUME_FORMAT = 'ascii' ";


 stmt2 = samplecon.prepareStatement ( sql2 );
 stmt2.setString( 1,empnum );
 System.out.println ("\n Value of empnum is: \n");
 System.out.println (empnum);
 rslob2 = stmt2.executeQuery();

 while (rslob2.next()) {
    //  Get integer value for the current row
    deptindex = rslob2.getInt(1);
    System.out.println ("\n Value of deptindex is: \n");
    System.out.println (deptindex);

 } // end while

/************************ ?????????????????? ***********************/
/* Retrieve the employee number, resume format, and the part of the   */
/* resume not containing Personal information for the employee number */
/* supplied.                                                          */
/**********************************************************************/
```

```
    /*********************************************************************/
    /*    First calculate value for persindex1 as persindex - 1 to define */
    /*    offset to one (1) position before personal data                 */
    /*********************************************************************/

    persindex1 = persindex - 1;
    System.out.println ("\n Value of persindex1 is: \n");
    System.out.println (persindex1);

    sql3 = "SELECT EMPNO, RESUME_FORMAT, "
         + "SUBSTR(RESUME,1,?)||SUBSTR(RESUME,?) AS RESUME "
         + "FROM EMP_RESUME "
         + "WHERE EMPNO = ? AND RESUME_FORMAT = 'ascii'";

     stmt3 = samplecon.prepareStatement ( sql3 );
     stmt3.setInt( 1,persindex1 );
     stmt3.setInt ( 2,deptindex );
     stmt3.setString ( 3,empnum );
     System.out.println ("\n Value of empnum is: \n");
     System.out.println (empnum);
     rslob3 = stmt3.executeQuery();

    /****************** ????????????????????????? **********************/
    /* For each row, print out the employee number, the resume format,    */
    /* and the resultant data after the Personal information has been     */
    /* excluded.                                                          */
    /*********************************************************************/
    while ( rslob3.next() )
    {
      empno = rslob3.getString(1);
      resumefmt = rslob3.getString(2);
      resumelob = rslob3.getClob(3);
      //  Determine the length of the clob data
      long len = resumelob.length();
      //  Assign the length to an int variable for use with getSubString
      //  Note that loss of precision may occur
      int len1 = (int)len;
      //  Materialize clob data for displaying using the getSubString method
      String resumeout = resumelob.getSubString(1, len1);
      System.out.println("Empno " + empno + " has resume format of " + resumefmt);
      System.out.println("Resume contents are: " + resumeout );
    }
  }  // end try
  catch ( SQLException x )
  {
    System.out.println("Error on fetch of rows" + x.getErrorCode() );
    x.printStackTrace();
  }
  System.exit(0);

} // end main

}  // end of javalob class
```

Appendix D. .bat Files for Compilation

compjava - SQLJ, d2profc, javac

```
@echo off
if "%1" == "" goto error
if "%2" == "" goto error
@echo on

echo Compiling %1.sqlj against database %2
del %1.class
del %1.java
sqlj -url=jdbc:db2:%2 %1.sqlj
db2 connect to %2
db2profc -url=jdbc:db2:%2 -prepoptions="bindfile using %1.bnd
package using %1" %1_SJProfile0.ser
javac %1.java
goto exit

:error
echo Usage:  compjava prog_name dbname

:exit
@echo on
```

compcli - SQLJ, javac

```
@echo off
if "%1" == "" goto error
if "%2" == "" goto error
@echo on
echo Compiling %1.sqlj against database %2
del %1.class
del %1.java
sqlj -url=jdbc:db2:%2 %1.sqlj
javac %1.java
goto exit

:error
echo Usage:  compjava prog_name dbname

:exit
@echo on
```

1to_jar - SQLJ, db2profc, javac, jar, install_jar

```
@echo off
if "%1" == "" goto error
if "%2" == "" goto error
@echo on

echo Compiling %1.sqlj against database %2
del %1.class
del %1.java
sqlj -url=jdbc:db2:%2 %1.sqlj
db2 connect to %2
db2profc -url=jdbc:db2:%2 -prepoptions="bindfile using %1.bnd
package using %1" %1_SJProfile0.ser
javac %1.java
jar cvf \%1.jar %1*.class
db2 call sqlj.install_jar('file:\%1.jar','%1jar')
goto exit

:error
echo Usage:  compjava prog_name dbname

:exit
@echo on
```

comp2jar - SQLJ, db2profc, javac, jar, replace_jar

```
@echo off
if "%1" == "" goto error
if "%2" == "" goto error
@echo on

echo Compiling %1.sqlj against database %2
del %1.class
del %1.java
sqlj -url=jdbc:db2:%2 %1.sqlj
db2 connect to %2
db2profc -url=jdbc:db2:%2 -prepoptions="bindfile using %1.bnd
package using %1" %1_SJProfile0.ser
javac %1.java
jar uvf \%1.jar %1*.class
db2 call sqlj.replace_jar('file:\%1.jar','%1jar')
goto exit

:error
echo Usage:  compjava prog_name dbname

:exit
@echo on
```