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

Student Exercises
ERC 8.1

IBM Certified Course Material
Trademarks

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

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

AIX
DB2
Distributed Relational Database Architecture
iSeries
OS/390
RACF
SP
z/OS

Approach
DB2 Connect
DRDA
Lotus
OS/400
RISC System/6000
VisualAge

CICS
DB2 Universal Database
IMS
MVS/ESA
QMF
S/390
WebSphere

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

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

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

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

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

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

June 2005 Edition

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

© Copyright International Business Machines Corporation 1997, 2005. All rights reserved. This document may not be reproduced in whole or in part without the prior written permission of IBM. Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Trademarks ................................................................. vii

Exercises Description .................................................. ix

Exercise 1. DDL - Command Line Processor (CLP) and Command Editor .... 1-1
  Exercise Instructions .................................................. 1-2
    Section 1 - Introduction to Command Line Processor (CLP) ............ 1-2
    Section 2 - Creating Lab Objects .................................. 1-6
    Section 3 - Impacts of Referential Integrity .......................... 1-8
  Exercise Solutions .................................................... 1-13
    Section 1 - Introduction to Command Line Processor (CLP) ............ 1-13
    Section 2 - Creating Lab Objects .................................. 1-18
    Section 3 - Impacts of Referential Integrity .......................... 1-20
  Command Editor Exercise Instructions .................................. 1-26
    Section 1 - Introduction to the Command Editor ....................... 1-26
    Section 2 - Impacts of Referential Integrity .......................... 1-28
  Command Editor Exercise Solutions .................................... 1-32
    Section 1 - Introduction to the Command Editor ....................... 1-32
    Section 2 - Impacts of Referential Integrity .......................... 1-34

Exercise 2. Program Structure I ...................................... 2-1
  Exercise Instructions .................................................. 2-2
    Introduction to the Lab Program .................................... 2-2
    Preliminary Program Editing ....................................... 2-7
    Connecting to the Database ........................................ 2-8
    Embedding Single Row SELECT Statements ............................ 2-8
    Embedding INSERT, UPDATE, and DELETE Statements ................. 2-10
    Preparing and Executing the Program (Optional) ..................... 2-12

Exercise 3. Program Preparation ..................................... 3-1
  Exercise Instructions .................................................. 3-2
    Program Preparation ................................................ 3-2
    Prep Option - nlinemacro .......................................... 3-3
    Investigating the Use of db2dclgn (Optional) ....................... 3-4
    Investigating Other Precompile and Bind Options (Optional) ....... 3-5
  Exercise Solutions .................................................... 3-6
    Program Preparation ................................................ 3-6
    Prep Option - nlinemacro .......................................... 3-7
    Investigating the Use of db2dclgn (Optional) ....................... 3-8
    Investigating Other Precompile and Bind Options (Optional) ....... 3-10

Exercise 4. Program Structure II ..................................... 4-1
  Exercise Instructions .................................................. 4-2
<table>
<thead>
<tr>
<th>Exercise</th>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Restart Program</td>
<td>5-1</td>
</tr>
<tr>
<td></td>
<td>Exercise Instructions</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Setting up the Data</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Coding the Program</td>
<td>5-2</td>
</tr>
<tr>
<td></td>
<td>Running the Program</td>
<td>5-3</td>
</tr>
<tr>
<td>6</td>
<td>Dynamic SQL</td>
<td>6-1</td>
</tr>
<tr>
<td></td>
<td>Exercise Instructions</td>
<td>6-2</td>
</tr>
<tr>
<td></td>
<td>Coding the Dynamic SQL</td>
<td>6-2</td>
</tr>
<tr>
<td>7</td>
<td>Loading Data</td>
<td>7-1</td>
</tr>
<tr>
<td></td>
<td>Exercise Instructions</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Loading Data into EMP Table</td>
<td>7-2</td>
</tr>
<tr>
<td></td>
<td>Exercise Solutions</td>
<td>7-3</td>
</tr>
<tr>
<td></td>
<td>Loading Data into EMP Table</td>
<td>7-3</td>
</tr>
<tr>
<td>8</td>
<td>Application Performance and Explain</td>
<td>8-1</td>
</tr>
<tr>
<td></td>
<td>Exercise Instructions</td>
<td>8-2</td>
</tr>
<tr>
<td></td>
<td>Create Explain Tables for Visual Explain</td>
<td>8-2</td>
</tr>
<tr>
<td></td>
<td>Examine Visual Explain Output</td>
<td>8-2</td>
</tr>
<tr>
<td></td>
<td>Generating Explain Data for Lab Program</td>
<td>8-4</td>
</tr>
<tr>
<td></td>
<td>Investigating Index Usage</td>
<td>8-5</td>
</tr>
<tr>
<td></td>
<td>Exercise Solutions</td>
<td>8-13</td>
</tr>
<tr>
<td></td>
<td>Create Explain Tables for Visual Explain</td>
<td>8-13</td>
</tr>
<tr>
<td></td>
<td>Examine Visual Explain Output</td>
<td>8-13</td>
</tr>
<tr>
<td></td>
<td>Generating Explain Data for Lab Program</td>
<td>8-14</td>
</tr>
<tr>
<td></td>
<td>Investigating Index Usage</td>
<td>8-15</td>
</tr>
<tr>
<td>A</td>
<td>C Language Source Skeleton</td>
<td>A-1</td>
</tr>
<tr>
<td>B</td>
<td>Expected Program Output</td>
<td>B-1</td>
</tr>
<tr>
<td>C</td>
<td>C Language Sample Solution</td>
<td>C-1</td>
</tr>
<tr>
<td>D</td>
<td>Restart Source Skeleton</td>
<td>D-1</td>
</tr>
<tr>
<td>E</td>
<td>Restart: Expected Program Output</td>
<td>E-1</td>
</tr>
<tr>
<td>F</td>
<td>Restart Sample Solution</td>
<td>F-1</td>
</tr>
</tbody>
</table>
Appendix G. Editor Command Summaries

OS/2 EPM Editor Command Summary .................................................. G-1
UNIX vi Editor Command Summary .................................................... G-4
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:

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

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

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

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

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

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

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

Exercise instructions - This section describes what you should 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.

EDDB Database Tables

Each team will use DB2 tables in order to complete the labs. A description of the tables in the EDDB database follows for reference.

Note: In the comments section, **PK** is used to identify a **Primary Key** and **FK** is used to identify a **Foreign Key**.

**EMP Table**

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>TYPE</th>
<th>NULL</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNO</td>
<td>CHAR(6)</td>
<td>NO</td>
<td>Employee serial number</td>
</tr>
<tr>
<td>FIRSTNME</td>
<td>VARCHAR(12)</td>
<td>NO</td>
<td>Employee first name</td>
</tr>
<tr>
<td>MIDINIT</td>
<td>CHAR(1)</td>
<td>NO</td>
<td>Employee middle initial</td>
</tr>
<tr>
<td>LASTNAME</td>
<td>VARCHAR(15)</td>
<td>NO</td>
<td>Employee last name</td>
</tr>
<tr>
<td>WORKDEPT</td>
<td>CHAR(3)</td>
<td>YES</td>
<td>Employee’s dept. number</td>
</tr>
<tr>
<td>PHONENO</td>
<td>CHAR(4)</td>
<td>YES</td>
<td>Employee telephone number</td>
</tr>
<tr>
<td>HIREDATE</td>
<td>DATE</td>
<td>YES</td>
<td>Date hired, YYYY-MM-DD</td>
</tr>
<tr>
<td>JOB</td>
<td>CHAR(8)</td>
<td>YES</td>
<td>Job held by employee</td>
</tr>
<tr>
<td>EDLEVEL</td>
<td>SMALLINT</td>
<td>YES</td>
<td>Years of formal education</td>
</tr>
<tr>
<td>SEX</td>
<td>CHAR(1)</td>
<td>YES</td>
<td>M = male, F = female</td>
</tr>
<tr>
<td>BIRTHDATE</td>
<td>DATE</td>
<td>YES</td>
<td>Date of birth, YYYY-MM-DD</td>
</tr>
<tr>
<td>SALARY</td>
<td>DECIMAL(9,2)</td>
<td>YES</td>
<td>Yearly salary ($999,999.99)</td>
</tr>
<tr>
<td>BONUS</td>
<td>DECIMAL(9,2)</td>
<td>YES</td>
<td>Bonus</td>
</tr>
<tr>
<td>COMM</td>
<td>DECIMAL(9,2)</td>
<td>YES</td>
<td>Commission</td>
</tr>
</tbody>
</table>

**DEP Table**

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>TYPE</th>
<th>NULL</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPTNO</td>
<td>CHAR(3)</td>
<td>NO</td>
<td>Department ID number</td>
</tr>
<tr>
<td>DEPTNAME</td>
<td>VARCHAR(36)</td>
<td>NO</td>
<td>Department name</td>
</tr>
<tr>
<td>MGRNO</td>
<td>CHAR(6)</td>
<td>YES</td>
<td>EMPNO of department’s manager</td>
</tr>
<tr>
<td>ADMRDEPT</td>
<td>CHAR(3)</td>
<td>NO</td>
<td>Admin. dept. for this dept.</td>
</tr>
</tbody>
</table>
**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.)

**VPHONE View**

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>TYPE</th>
<th>NULL</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLASTNAME</td>
<td>VARCHAR(15)</td>
<td>NO</td>
<td>Employee last name</td>
</tr>
<tr>
<td>VFIRSTNME</td>
<td>VARCHAR(12)</td>
<td>NO</td>
<td>Employee first name</td>
</tr>
<tr>
<td>VPHONENO</td>
<td>CHAR(4)</td>
<td>YES</td>
<td>Employee telephone number</td>
</tr>
<tr>
<td>VEMPNO</td>
<td>CHAR(6)</td>
<td>NO</td>
<td>Employee serial number</td>
</tr>
<tr>
<td>VDEPTNAME</td>
<td>VARCHAR(36)</td>
<td>NO</td>
<td>Department name</td>
</tr>
<tr>
<td>VDEPTNO</td>
<td>CHAR(3)</td>
<td>NO</td>
<td>Department number</td>
</tr>
</tbody>
</table>
Exercise 1. DDL - Command Line Processor (CLP) and Command Editor

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

Introduction

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

The lab solutions can be found in “Exercise Solutions”.

An alternative lab using the Command Editor can be found in “Command Editor Exercise Instructions”.

Section 1 - Introduction to Command Line Processor (CLP)

__ 1. If on Windows, open a Command Window (Start -> Programs -> IBM DB2 -> Command Line Tools -> 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 WSPGM directory:

```
cd \WSPGM
```

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 3 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 6 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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>0 (zero)</td>
</tr>
<tr>
<td>Fixed Length String</td>
<td>blanks</td>
</tr>
<tr>
<td>Variable Length String</td>
<td>String with 0 length</td>
</tr>
<tr>
<td>Date</td>
<td>Current Date</td>
</tr>
<tr>
<td>Time</td>
<td>Current Time</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Current Timestamp</td>
</tr>
</tbody>
</table>

   - 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 columns 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.
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 DEPT 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:\wspgm\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:\wspgm\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 < 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 deletpk.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=
Exercise Solutions

An alternative lab using the Command Editor can be found in “Command Editor Exercise Instructions”.

Section 1 - Introduction to Command Line Processor (CLP)

___ 1. If on Windows, open a Command Window (Start -> Programs -> DB2 UDB for Windows -> 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 I 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 from this command. Notice that the right columns 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.

__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.

__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 turn 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?
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 WSPGM directory:

```
cd \WSPGM
```

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 command 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 statement 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 3-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 6-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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>0 (zero)</td>
</tr>
<tr>
<td>Fixed Length String</td>
<td>blanks</td>
</tr>
<tr>
<td>Variable Length String</td>
<td>String with 0 length</td>
</tr>
<tr>
<td>Date</td>
<td>Current Date</td>
</tr>
<tr>
<td>Time</td>
<td>Current Time</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Current Timestamp</td>
</tr>
</tbody>
</table>

- 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 columns 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:

```bash
db2 -f emp.mem
```

__ 5. View the view.mem file:

```bash
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).

```bash
db2 -f view.mem
```

Section 3 - Impacts of Referential Integrity

__ 1. Look at the script file:

```bash
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.

_______________________________________________________________
_______________________________________________________________
_______________________________________________________________

The file executed successfully. Apparently, all 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:\wspgm\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 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.

Look at the file d:\wspgm\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 < 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=
Command Editor Exercise Instructions

The lab solutions can be found in “Command Editor Exercise Solutions”.

Section 1 - Introduction to the Command Editor

__ 1. Access the Control Center.

Open the Command Editor:

- Open the DB2 UDB for Windows folder.
- Open the Control Center, you will be prompted for authentication.
- From the Control Center's tool bar, select the second icon from the tool bar Command Editor.

__ 2. A command that will provide you with general help is '?'. On the Command Editor, request general help.

- Type ? in the top (entry) half of the screen
- Press the green arrow above, or press Ctrl+Enter to run the request

__ 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 Tools -> Options tab on the Command Editor.

What is the current setting for auto-commit?

What is the current setting for displaying the SQLCA?

________________________________________________________________________

________________________________________________________________________

__ 6. Set your Command Editor options to stop when a statement error is encountered.

- Select Tools pull-down menu
- Select Tools Options...menu item
• Select Command Editor tab
• Check the box **Stop execution if errors occur**
• Close the Tools settings window

__ 7. Check that the statement termination character for a command is ;. This will allow for multiple DB2 or SQL commands to be issued from the Command Editor. This can be found at the bottom of the Command Editor screen.

__ 8. Connect to database eddb.
   
   ```
   connect to eddb user **udba** using **udba**
   ```

__ 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.
   
   Why is the output appearing on your screen?
   
   What happens if an error is encountered?
   
   _________________________________________________________________
   
   _________________________________________________________________
   
   _________________________________________________________________
   
   _________________________________________________________________
   
   _________________________________________________________________
   
   _________________________________________________________________
Section 2 - Impacts of Referential Integrity

1. Open the script file:
   d:\wspgm\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:
   insert.mem

   using the Command Editor.

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

3. Open the script file:
   D:\wspgm\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 green arrow from the Command editor, select Selected pull-down menu, then select Execute menu item.

   Did the file execute successfully? Why?

5. Open the script file:
   D:\wspgm\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 Gears icon from the Command Editor 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:\wspgm\uppk2.mem and Execute the script file from the Command Editor.**

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

**Open the file D:\wspgm\updatefk.mem and Execute the file from the Command Editor.**

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:\wspgm\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:\wspgm\deletepk.mem from the Command Editor.
```

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:\wspgm\refresh.mem from the Command Editor.

__ 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 EXERCISE
Command Editor Exercise Solutions

Section 1 - Introduction to the Command Editor

__ 1. Access the Control Center.
   Open the Command Editor:
   • Open the DB2 UDB for Windows folder.
   • Open the Control Center, you will be prompted for authentication.
   • From the Control Center's tool bar, select the second icon from the tool bar Command Editor.

__ 2. A command that will provide you with general help is ‘?’. On the Command Editor, request general help.
   • Type ? in the top (entry) half of the screen
   • Press the green arrow above, or press Ctrl+Enter to run 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 Editor, 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 Script -> Options... (Script tab on the Command Editor).
   What is the current setting for auto-commit?
   What is the current setting for displaying the SQLCA?
• Using the Command Editor, select Script 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 Editor options to stop when a statement error is encountered.
• Select Tools pull-down menu
• Select Options... menu item
• Select Command Editor tab
• Check the box Stop execution if errors occur
• Close the Tools Settings window

7. Check that the statement termination character for a command is ;. This will allow for multiple DB2 or SQL commands to be issued from the Command Editor.
This can be found at the bottom of the Command Editor screen.
• Select Tool Settings icon from the Command Editor tool bar.
• It is the fourth icon from the right. The hover help should assist.
• Check the box Use statement termination character.
• 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

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 Editor:
• Select 'Selected' pull-down menu
• Select 'Open'
• Make sure 'file system' is checked rather than 'Task Center'
• You should get a list of files to select from
• Specify the file D:\WSPGM\SAMPLE.MEM
  - On the Directories panel, select your system name at the top, and the D: path at the bottom right.
  - Select directory WSPGM (double-click)
  - From the Files panel, select the file sample.mem
• Click OK command button
• If you are asked to confirm that you wish to replace the current input, click Yes.

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.

Why is the output appearing on your screen?
What happens if an error is encountered?
On the lower half of your screen, you should see three queries that were executed.

The output will always appear in the lower half of the Command Editor screen.
We could also place the output in a file by selecting via Tools Settings 'Log output to 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 Editor Option.
Otherwise, it will attempt to execute the next statement.

Section 2 - Impacts of Referential Integrity

1. Open the script file:
d:\wspgm\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:\wspgm\insert.mem using the Command Editor.

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:\wspgm\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 green arrow from the Command Editor, select Selected pull-down menu, then 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
5. Open the script file:

D:\wspgm\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 Gears icon from the Command Editor 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:\wspgm\uppk2.mem and Execute the script file from the Command Editor.

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:\wspgm\updatefk.mem` and Execute the file from the Command Editor.

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:\wspgm\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:\wspgm\deletepk.mem from the Command Editor.

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:\wspgm\refresh.mem from the Command Editor.

12. On the Script tab, choose the Interactive radio button, and enter the following commands to terminate your CLP session:
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 EXERCISE
Exercise 2. Program Structure I

What This Exercise Is About

You will be coding an application that uses embedded SQL to
SELECT an employee-by-employee number from the EMP table or by
phone number from the VPHONE view.

What You Should Be Able to Do

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

• Describe the basic flow of the supplied program skeleton
• Code INCLUDE statements for the SQLCA
• Declare a host variable to provide null support
• Code WHENEVER statements to handle SQL error and warning
  checking
• Embed SQL statements in an application that will support selection
  of a single row of information
• Use elemental assignment of DB2 columns to a host variable list
• Use indicator variables for NULL support

Introduction

You will start writing your program.

We have provided a skeleton program so you will be adding your
declares and SQL statements to the provided code.
Exercise Instructions

Introduction to the Lab Program

__1. You have access to three useful Appendices.
   • “C Language Source Skeleton” on page A-1
   • “Expected Program Output” on page B-1
   • “C Language Sample Solution” on page C-1
   We suggest you refer to “C Language Sample Solution” on page C-1 as you complete your code. As you transcribe the code, highlight it in “C Language Sample Solution” on page C-1. This will facilitate your debugging, if you make any mistakes, that is!

__2. The structure of the program is as follows:

   Set up the parms
   Main line
   Read from TRANSIN
   Perform CASE
   Record type 'S'
   Record type 'P'
   ....

__3. Examine the following screens. They represent a fictitious set of transactions that require DB2 interaction.

   The first screen allows several options:
Assume option 'S' was chosen. The transaction will display the following:
The program would use the number provided, obtain the FIRST NAME, MIDDLE INITIAL and LAST NAME from DB2 and display the information on the screen. The transaction would then return to the initial screen:

```
MAIN TRANSACTION

Pick an Option

   _ S - Select Employee Name Information
     P - Display Employee Phone Number
     D - Delete a Department
     I - Insert a Department
     U - Update Employee Phone Number
     A - Add Employee
     F - Display a Department
```

Assume option 'A' was chosen. The transaction will display the following:
ADD A NEW EMPLOYEE TO STAFF

Please ENTER the employee's information below:

Employee Number: _ _ _ _ _ _

Last Name : _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

First Name : _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Initial : _

Phone Number : _ _ _ _

Department : _ _ _

The program would use the information provided and add the new data to DB2.

Your program will NOT actually use screens for input. However, the code you will write in lab could be used in conjunction with screen maps provided you knew how to communicate with a terminal. This lab is intentionally designed so that screen communication expertise is not a prerequisite.

4. Read the following information and relate it to the screens previously examined. If you have any questions, contact your instructor.

An ASCII file will be used to provide input to the program in order to:

• Simplify the external communication of the program
• Eliminate the possibility of input error
• Facilitate comparing program output to expected results

This ASCII file will contain an action code that corresponds to one of the actions shown on the MAIN TRANSACTION screen. For each action, any additional information that is required will be provided. For example, a record from the file with a action code of ‘A’ will also contain the employee number, last name, first name, initial, department and phone number since this information is required. This information reflects what would have been entered on the ADD NEW EMPLOYEE TO STAFF screen if the lab actually used communication to the terminal.

The ASCII DRIVER FILE is contained in
D:\WSPGM\TRANSIN

A description of the information contained in the file and the position of the data is described:

<table>
<thead>
<tr>
<th>Information</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Code</td>
<td>1</td>
</tr>
<tr>
<td>Last Name</td>
<td>2 through 16</td>
</tr>
<tr>
<td>First Name</td>
<td>17 through 28</td>
</tr>
<tr>
<td>Employee Number</td>
<td>29 through 34</td>
</tr>
<tr>
<td>Phone Number</td>
<td>35 through 38</td>
</tr>
<tr>
<td>Department Number</td>
<td>39 through 41</td>
</tr>
<tr>
<td>Middle Initial</td>
<td>42</td>
</tr>
<tr>
<td>Filler</td>
<td>43 through 78</td>
</tr>
<tr>
<td>Last Character</td>
<td>79</td>
</tr>
</tbody>
</table>

The contents of this file are as shown:

```plaintext
| ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8 |
| S 000330 x                      |
| S 111111 x                      |
| P 000210 x                      |
| P 000060 x                      |
| P 777777 x                      |
| D BAD x                         |
| I BAD x                         |
| I D11 x                         |
| UMONE ROE 000180XXXX x          |
| UGENNA 9999998765 x             |
| UBA CHMAN 0002107654 x          |
| AFUZZY BEAR 111111XXXXD01X x    |
| AGOLDIE LOCKS 222222468D01Y x   |
| ADOS NOT MATTER 333333XXXXBADL x|
| F D11 x                         |
| F BAD x                         |
| F D01 x                         |
| *WORKDEPT = 'D11' AND LASTNAME LIKE 'B%' x |
| ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8 |
```

5. The program you will be using for the lab is stored in

D:\WSPGM\LABPGM.SQC
The program used to access an IMS database. However, the data has been migrated to DB2 on the workstation. The IMS programmer has removed all the IMS calls. Your task as the DB2 programmer is to add the DB2 code to support the actions previously supported by IMS.

6. View and examine the skeleton. Do not make changes at this time. Simply note the following:
   - INPUTAREA - data area already coded that will receive the data from the ASCII sequential file.
   - Report headers have been pre-coded.
   - Error messages have been pre-coded.
   - Variables that map to EMP, DEP, and VPHONE have been pre-coded.
   - The main read logic has been coded. Also, code to invoke logic appropriate for given actions is pre-coded.
   - Some output statements have been coded.
   - The error routine labelled DBERROR is provided.

Note: A copy of the C skeleton begins in “C Language Source Skeleton” on page A-1. A sample solution is provided in “C Language Sample Solution” on page C-1.

If you have any questions regarding the existing code, contact your instructor.

Preliminary Program Editing

1. Copy the program skeleton to a program name that you choose (remember that the extension needs to be .sqc), for example:
   ```
   cd wspgm
   copy labpgm.sqc pgmname.sqc
   (We will use pgmname.sqc to refer to the program in the remainder of this lab document.)
   ```

2. EDIT the program copy.

3. Search for the word SQLCA.

4. Code the SQL statements to provide the SQLCA definition to your application program.

5. Search for the words Error handling.

6. Code SQL statements to transfer control to the pre-coded 'DBERROR' routine anytime an SQL ERROR or WARNING is encountered. Also, code the SQL Whenever statement to CONTINUE for a NOT FOUND condition.
Note: In this lab, the use of these statements is prior to any executable code. Therefore, checking the SQLCODE for errors or warnings is not required in subsequent steps. This may or may not reflect your installation's requirements.

7. Search for the word Indicators.

8. Declare an indicator variable that can be used by DB2 to flag NULL values SELECTED and that can be used by the application to set a value to NULL on an INSERT or UPDATE statement.

Note: Ignore the DECLARE CURSOR comments at this time.

Connecting to the Database

1. Search for the words Connect to database.

2. Note that there are strcpy statements to copy a userid, password, and database into the variables userid, password, and database.

   In your normal programming environment, you would probably prompt the user for their userid, password, and possibly even desired database. We have set up the code to simulate an application that would prompt the user for this information and store it in the variables userid, password, and database.

   In our environment, so you will not need to prompt for these items and read them in, you will hard code the userid and password to be your userid and password, and hard code the database name to be “EDDB”.

3. Substitute your team number for the nn in the strcpy for userid and for password.

4. Substitute EDDB for XXXX in the strcpy for the database.

5. Now code the connect statement, using host variables for the database, userid, and password. This connect statement could then be used if, in fact, you were getting these fields by prompting the user or from an input screen.

Embedding Single Row SELECT Statements

Note: It is imperative to examine the existing logic in the skeleton program when completing this and subsequent sections of the programming lab. Such study will enable you to gain a comprehensive knowledge of embedding SQL statements in an application program.

Refer to the table and view description information beginning page viii if necessary.

1. EDIT your program copy.

2. Locate the section of the program that will handle action code 'S'. Use the following information to add code to the program to support action code 'S'.

   __ 7. Search for the word Indicators.
   __ 8. Declare an indicator variable that can be used by DB2 to flag NULL values SELECTED and that can be used by the application to set a value to NULL on an INSERT or UPDATE statement.
   
   Note: Ignore the DECLARE CURSOR comments at this time.

   Connecting to the Database

   __ 1. Search for the words Connect to database.
   __ 2. Note that there are strcpy statements to copy a userid, password, and database into the variables userid, password, and database.

      In your normal programming environment, you would probably prompt the user for their userid, password, and possibly even desired database. We have set up the code to simulate an application that would prompt the user for this information and store it in the variables userid, password, and database.

      In our environment, so you will not need to prompt for these items and read them in, you will hard code the userid and password to be your userid and password, and hard code the database name to be “EDDB”.

   __ 3. Substitute your team number for the nn in the strcpy for userid and for password.
   __ 4. Substitute EDDB for XXXX in the strcpy for the database.
   __ 5. Now code the connect statement, using host variables for the database, userid, and password. This connect statement could then be used if, in fact, you were getting these fields by prompting the user or from an input screen.

   Embedding Single Row SELECT Statements

   Note: It is imperative to examine the existing logic in the skeleton program when completing this and subsequent sections of the programming lab. Such study will enable you to gain a comprehensive knowledge of embedding SQL statements in an application program.

   Refer to the table and view description information beginning page viii if necessary.

   __ 1. EDIT your program copy.
   __ 2. Locate the section of the program that will handle action code 'S'. Use the following information to add code to the program to support action code 'S'.

   Note: In this lab, the use of these statements is prior to any executable code. Therefore, checking the SQLCODE for errors or warnings is not required in subsequent steps. This may or may not reflect your installation's requirements.
In addition to the action code of 'S', the input file will contain the EMPLOYEE NUMBER.

The output required from DB2 is the FIRST NAME, LAST NAME, and MIDDLE INITIAL for the given EMPLOYEE NUMBER. Use the TABLE EMP to obtain the data.

EMPLOYEE NUMBER will be unique, and no data returned can contain nulls so **DO NOT** use indicator logic.

Examine the pre-coded output statements in order to understand the manipulation of the output file.

If the row is not in the table, the program will print error message MSG07. Verify the coding of this function.

__3. Locate the section of the program that will handle action code 'P'. Delete or comment the statement that causes a branch to PGMEND in order to “activate” this portion of the code.

__4. Use the following information to add code to the program to support action code 'P'.

In addition to the action code of 'P', the input file will contain the EMPLOYEE NUMBER.

The output required from DB2 is the FIRST NAME, LAST NAME, and PHONE NUMBER for the given EMPLOYEE NUMBER. Use the view VPHONE to obtain the data.

EMPLOYEE NUMBER will be unique, but the column containing PHONE NUMBER may contain a NULL value. If a NULL is indicated by DB2, place 'NONE' in the output field related to the PHONE NUMBER.

Examine the pre-coded output statements in order to understand the manipulation of the output file.

If the row is not in the table, the program will print error message MSG07. Verify the coding of this function.

You may want to check your work at this point before proceeding with the D, I, and U options. In “Preparing and Executing the Program (Optional)” on page 2-11, you can find the instructions to prepare the program for execution. These steps will be explained in the Program Preparation lecture. If this lecture has not been presented and you have significant time left in the lab period, you may choose to prepare the program for execution following the steps below. The details regarding these steps will be clarified in lecture and subsequent lab exercises.
Embedding INSERT, UPDATE, and DELETE Statements

1. EDIT your program code.

2. Locate the section of the program that will handle action code 'D'. Delete or comment the statement that causes a branch to PGMEND in order to “activate” this portion of the code.

3. Use the following information to add code to the program to support action code 'D'.

   In addition to the action code of 'D', the input file will contain the DEPARTMENT NUMBER.

   The program should DELETE the rows for the given DEPARTMENT from the EMP table. MSG12 will be written, which will indicate the number of employees deleted.

   The program should DELETE the row for the same department from the DEP table.

   If the given DEPARTMENT NUMBER does not exist in the table, MSG11 will be written. If one row was found and deleted, MSG10 is written. Otherwise, MSG14 is written, which contains the number of rows deleted.

4. Locate the section of the program that will handle action code 'I'. Delete or comment the statement that causes a branch to PGMEND in order to “activate” this portion of the code.

5. Use the following information to add code to the program to support action code 'I'.

   In addition to the action code of 'I', the input file will contain the DEPARTMENT NUMBER.

   The program should INSERT the row from CF10.DEP into the DEP table using the given DEPARTMENT NUMBER.

   If the row was found in the CF10.DEP table, the program should INSERT the employees in given DEPARTMENT from the CF10.EMP table to the EMP table.

   If the given DEPARTMENT NUMBER does not exist in CF10.DEP, MSG11 will be written. Otherwise, MSG13 or MSG14 will be written, depending on the number of rows “copied”. MSG09 is also written, which includes a count of the number of employees that were inserted.

6. Locate the section of the program that will handle action code 'U'. Delete or comment the statement that causes a branch to PGMEND in order to “activate” this portion of the code.

7. Use the following information to add code to the program to support action code 'U'.

   In addition to the action code of 'U', the input file will contain the LAST NAME, PHONE NUMBER, and EMPLOYEE NUMBER.

   The program should UPDATE the PHONE NUMBER and LAST NAME in the EMP table for the specified employee.
If the input PHONE NUMBER contains 'XXXX', the program should cause DB2 to UPDATE the PHONE NUMBER in the EMP table to NULL. Any other characters should be used as the new number.

If the given employee does not exist, MSG03 is written.

If any rows were updated, MSG02 or MSG14 is written, depending on the number of rows changed.

The program should then verify that the update was coded properly. The output required from DB2 is the FIRST NAME, LAST NAME, and PHONE NUMBER for the given EMPLOYEE NUMBER. Use the EMP table to obtain the data. If a NULL is indicated by DB2, place 'NONE' in the output field related to the PHONE NUMBER.

__ 8. Locate the section of the program that will handle action code 'A'. Delete or comment the statement that causes a branch to PGMEND in order to "activate" this portion of the code.

__ 9. Use the following information to add code to the program to support action code 'A'.

In addition to the action code of 'A', the input file will contain the EMPLOYEE NUMBER, LAST NAME, FIRST NAME, PHONE NUMBER, and DEPARTMENT NUMBER.

The program should verify that the DEPARTMENT NUMBER is valid by checking the DEP table.

If the given DEPARTMENT NUMBER exists in the DEP table, INSERT the new employee. If the given PHONE NUMBER is 'XXXX', then set the value in the EMP table to NULL. MSG15 will then be written.

If the given DEPARTMENT NUMBER does not exist in the DEP table, MSG11 will be written.

**Note:** You may use prepare, compile, bind and run your program at any time you wish this point forward in the programming lab. You may elect to code several actions before testing or to code one action at a time. Simply ignore any output for actions you have not yet coded. To ensure that your output matches the copy in "Expected Program Output" on page B-1, **complete each action in the order presented to you in this lab guide.**

This is the end of the formal lab. You need not proceed further until the Program Preparation lecture has been presented. If this lecture has not been presented and you have significant time left in the lab period, you may choose to prepare the program for execution following the steps below. The details regarding these steps will be clarified in lecture and subsequent lab exercises.
Preparing and Executing the Program (Optional)

__ 1. Precompile your program, creating a bindfile with the default name, and allowing all other options to default.
   
   `db2 prep pgmname.sqc bindfile`

__ 2. If you have any precompile errors, fix your `.sqc` file and run the precompile again.

__ 3. The precompiler has created a `.c` and a `.bnd` file. We have supplied a `complink.cmd` file that will compile and link your `.c` code. View the `complink.cmd` if you are interested in the compiler options we are invoking.
   
   To compile and link your program, enter:
   ```
   complink pgmname
   ```
   
   If you have any compile errors, correct them in the `.sqc` file, re-run the precompile, then compile and link again.

__ 4. Once you successfully precompile, compile, and link the code, you are ready to bind the package.

__ 5. Bind your bindfile, creating a package. Allow all options to default.
   
   `db2 bind pgmname.bnd`

__ 6. Now you are ready to execute your program. Run your program by typing in the program name, for example,
   ```
   pgmname
   ```

__ 7. Compare the output from your program with the hard copy provided on page B-1. You can ignore any output for transactions that you have not yet coded.
   ```
   more < report
   ```

__ 8. If the program does not execute, examine the messages produced and correct any errors.
   
   Contact your instructor if necessary.

END OF EXERCISE
Exercise 3. Program Preparation

What You Should Be Able to Do

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

- Use CLP to precompile a program
- Use a supplied exec to compile and linkedit to create an executable module
- Use CLP to bind a package
- Execute and test a program
- Identify how the nolinemacro option affects the reporting of a syntax error
- Identify how to prepare your program to be able to access a debugger
- Embed INSERT, UPDATE and DELETE statements in application programs
- Exploit the information in the SQLCA concerning the number of rows affected by an INSERT, UPDATE or DELETE statement
- Highlight the difference between application and system defined referential constraint enforcement

Introduction

You have had the opportunity to embed SQL statements into your program. This lab will direct you to precompile your program, compile and link the C code into an executable module, and bind the package.

You will also have the opportunity to complete more of the SQL code, dealing with inserts, updates, and deletes.
Exercise Instructions

The lab solutions can be found in “Exercise Solutions” on page 3-6.

Program Preparation

__ 1. Access help in CLP for the prep command.

BINDFILE results in the creation of a bind file. If you do not enter a file name, the precompiler will use the name of the program (entered as the filename parameter) and adds the .bnd extension. If this option is specified, a package is not created unless the PACKAGE option is also specified.

ISOLATION LEVEL may be CS (Cursor Stability), RR (Repeatable Read), RS (Read Stability), or UR (Uncommitted Read). This parameter refers to the duration of shared row locks. Cursor stability is the default and is used in most cases.

NOLINEMACRO suppresses the generation of the #line macros in the output .c file. It is useful when the file is used with development tools which require source line information such as cross-reference utilities and debuggers.

__ 2. Precompile your program, creating a bindfile with the default name, and allowing all other options to default.

__ 3. If you have any precompile errors, fix your .sqc file and run the precompile again.

__ 4. The precompiler has created a .c and a .bnd file. We have supplied a complink.cmd file that will compile and link your .c. View the complink.cmd if you are interested in the compiler options we are invoking.

To compile and link your program, enter:

   complink pgmname

If you have any compile errors, correct them in the .sqc file, re-run the precompile, then compile and link again.

__ 5. Once you successfully precompile, compile, and link the code, you are ready to bind the package. Access help in CLP for the bind command.

__ 6. Bind your bindfile, creating a package. Allow all options to default.

__ 7. Now you are ready to execute your program! Run your program by typing in the program name, for example:

   pgmname

__ 8. Compare the output from your program with the hard copy provided in “Exercise Solutions” on page 3. You can ignore any output for transactions that you have not yet coded.

   more < report
9. If the program does not execute, examine the messages produced and correct any errors.
   Contact your instructor if necessary.
   **Note:** You will need a successful PRECOMPILE, BIND, COMPILE, and LINK-EDIT before proceeding to the next item.

10. View the modified source code (your .c file).
   - Find the code that supports transaction code 'S'.
   - Note the original SQL you coded. It should be commented.
   - Note the modified code inserted by the PRECOMPILER on your behalf. You need not understand the details of this code.
   - Find the code that invokes the DB2 Language Interface.
   - Take note of the code inserted to support the WHENEVER statements you have coded.
   - Note the additional #line statements that have been added.
   **Note:** You should NEVER make changes to this file.

**Prep Option - nolinemacro**

1. Edit your .sqc file and create a syntax error (this should be easy!). Suggestion - remove a comma between two of the field names in the fscanf statement.
2. Precompile your program to create the .c file the same way you did in the previous part of this lab.
3. Run complink to compile and link your code. You should get a compile error.
   Which file is referenced?
   Which line number?

4. Now, precompile again, this time using the nolinemacro option.
5. Run complink to compile and link your code. You should get a compile error.
   Which file is referenced?
   Which line number?

Remember, you should NEVER make changes in the .c file. Only the .sqc file should be modified.
6. So, when **should** you use the nolinemacro option?

If you are going to be debugging your code using an online debugger, the nolinemacro option will provide you with a .c file that will be useful with the debugger. Without the nolinemacro option the executable produced will use the .sqc module for source display. Line numbers will not match up due to the fact that each EXEC SQL statement may expand into several C language statements. (Note, for example, that the EXEC SQL WHENEVER statements cause several lines of C code after each EXEC SQL statement.)

The nolinemacro option will cause the debuggable executable to point to the .c file. As you have seen, the .c file has the EXEC SQL statements converted to function calls. These can be seen in the debugger and you can set stop points before and after SQL calls to check values of variables or other debugging operations.

Setting stop points in the .sqc source within the debugger can have unpredictable results.

7. Fix the compile error before you go on to the next section.

**Note:** You may use prepare, compile, bind and run your program at any time you wish from this point forward in the programming lab. You may elect to code several actions before testing or to code one action at a time. Simply ignore any output for actions you have not yet coded. To ensure that your output matches the copy in “Expected Program Output” on page B-1, complete each action in the order presented to you in this lab guide.

## Investigating the Use of db2dclgn (Optional)

In the lecture material, you learned that there is a facility called db2dclgn that will create the host variable declarations for a table. This optional section will allow you to work with it.

1. Obtain help on db2dclgn:

```
db2dclgn
```

What parameters would you specify to identify that you want to create the declarations in C for the EMP table in the EDBDB database? Indicate that you want the declarations to be written into a file named emp.h.

```
________________________________________________________________
________________________________________________________________
```

2. Try running your command. Look at the emp.h file that is created.

```
more < emp.h
```

Are the variables defined in the same way in which you have them defined in your lab program?
3. Did you get any indicator variables defined?

4. Re-run the db2dclgn indicating that you want indicator variables to be created. Document your command:

5. What was the result of this run?

6. If you have time and wish to do so, you can experiment using these declarations in your lab program. It would be a wise idea to make a copy of your working program at this point before doing this experimentation.

Investigating Other Precompile and Bind Options (Optional)

In the lecture material, you learned about some of the options on precompile and bind. This optional section will allow you to examine some more of the options.

1. Look up the precompile command in the online Command Reference:

2. Scroll through the syntax of the precompile command. Then look at the details on the options. Some options that you may want to examine specifically are:

   a. BINDFILE
   b. BLOCKING
   c. COLLECTION
   d. DATETIME
   e. ISOLATION
   f. OWNER
   g. PACKAGE
   h. QUALIFIER

3. Look also at the BIND command. Note that many of the same options can be specified on the BIND. If the options are specified in both commands, the options indicated on the BIND will override the selections made on the PRECOMPILE.
Exercise Solutions

Program Preparation

1. Access help in CLP for the prep command.

   \texttt{db2 \ prep \ | \ more}

   BINDFILE results in the creation of a bind file. If you do not enter a file name, the
   precompiler will use the name of the program (entered as the \textit{filename} parameter)
   and adds the \texttt{.bnd} extension. If this option is specified, a package is not created
   unless the PACKAGE option is also specified.

   ISOLATION LEVEL may be CS (Cursor Stability), RR (Repeatable Read), RS (Read
   Stability), or UR (Uncommitted Read). This parameter refers to the duration of
   shared row locks. Cursor stability is the default and is used in most cases.

   NOLINEMACRO suppresses the generation of the \textit{#line} macros in the output \texttt{.c}
   file. It is useful when the file is used with development tools which require source
   line information such as cross-reference utilities and debuggers.

2. Precompile your program, creating a bindfile with the default name, and allowing all
   other options to default.

   \texttt{db2 prep pgmname.sqc bindfile}

3. If you have any precompile errors, fix your \texttt{.sqc} file and run the precompile again.

4. The precompiler has created a \texttt{.c} and a \texttt{.bnd} file. We have supplied a
   \texttt{complink.cmd} file that will compile and link your \texttt{.c}. View the \texttt{complink.cmd}
   file if you are interested in the compiler options we are invoking.

   To compile and link your program, enter:

   \texttt{complink pgmname}

   If you have any compile errors, correct them in the \texttt{.sqc} file, re-run the precompile,
   then compile and link again.

5. Once you successfully precompile, compile, and link the code, you are ready to bind
   the package. Access help in CLP for the \texttt{bind} command.

6. Bind your bindfile, creating a package. Allow all options to default.
7. Now you are ready to execute your program! Run your program by typing in the program name, for example,

    pgmname

8. Compare the output from your program with the hard copy provided in “Expected Program Output” on page B-1. You can ignore any output for transactions that you have not yet coded.

    more < report

9. If the program does not execute, examine the messages produced and correct any errors.

    Contact your instructor if necessary.

    Note: You will need a successful PRECOMPILE, BIND, COMPILE, and LINK-EDIT before proceeding to the next item.

10. View the modified source code (your .c file).

    • Find the code that supports transaction code 'S'.
    • Note the original SQL you coded. It should be commented.
    • Note the modified code inserted by the PRECOMPLIER on your behalf. You need not understand the details of this code.
    • Find the code that invokes the DB2 Language Interface.
    • Take note of the code inserted to support the WHENEVER statements you have coded.
    • Note the additional #line statements that have been added.

    Note: You should NEVER make changes to this file.

Prep Option - nolinemacro

1. Edit your .sq file and create a syntax error (this should be easy!). Suggestion - remove a comma between two of the field names in the fscanf statement.

2. Precompile your program to create the .c file the same way you did in the previous part of this lab.

3. Run complink to compile and link your code. You should get a compile error.

    Which file is referenced?
    Which line number?
4. Now, precompile again, this time using the nolinemacro option.

5. Run complink to compile and link your code. You should get a compile error.
   Which file is referenced?
   Which line number?

6. So, when should you use the nolinemacro option?

   If you are going to be debugging your code using an online debugger, the nolinemacro option will provide you with a .c file that will be useful with the debugger. Without the nolinemacro option the executable produced will use the .sqc module for source display. Line numbers will not match up due to the fact that each EXEC SQL statement may expand into several C language statements. (Note, for example, that the EXEC SQL WHENEVER statements cause several lines of C code after each EXEC SQL statement.)

   The nolinemacro option will cause the debuggable executable to point to the .c file. As you have seen, the .c file has the EXEC SQL statements converted to function calls. These can be seen in the debugger and you can set stop points before and after SQL calls to check values of variables or other debugging operations.

   Setting stop points in the .sqc source within the debugger can have unpredictable results.

7. Fix the compile error before you go on to the next section.

Investigating the Use of db2dclgn (Optional)

In the lecture material, you learned that there is a facility called db2dclgn that will create the host variable declarations for a table. This optional section will allow you to work with it.

1. Obtain help on db2dclgn:
   
   db2dclgn
What parameters would you specify to identify that you want to create the
declarations in C for the EMP table in the EDDB database? Indicate that you want
the declarations to be written into a file named emp.h.

db2dclgn -d EDDB -t EMP -l C -o emp.h

__ 2. Try running your command. Look at the emp.h file that is created.
   more < emp.h

   Are the variables defined in the same way in which you have them defined in your
   lab program?

   ________________________________________________________________________

   ________________________________________________________________________

   The variables are not defined in exactly the same way. Specifically, for CHAR
datatypes, there is not an additional character for the null character, and for
VARCHAR types, a structure is used rather than a character string.

__ 3. Did you get any indicator variables defined?

   ________________________________________________________________________

   ________________________________________________________________________

   The answer to this question depends on how you issued your db2dclgn, but if
   you used the solution in the lab guide, you would not have gotten indicator
   variables.

__ 4. Re-run the db2dclgn indicating that you want indicator variables to be created. Document your command:
   db2dclgn -d EDDB -t EMP -l C -o emp.h -i true

__ 5. What was the result of this run?

   ________________________________________________________________________

   ________________________________________________________________________

   If you ran the command as indicated in the solutions, then you would now
   have two definitions in your emp.h file. To replace rather than append,
specify:
   db2dclgn -d EDDB -t EMP -a REPLACE -l C -o emp.h -i true

   You should see the indicator variables as an array at the end of the
   declaration.

__ 6. If you have time and wish to do so, you can experiment using these declarations in
   your lab program. It would be a wise idea to make a copy of your working program at
   this point before doing this experimentation.
Investigating Other Precompile and Bind Options (Optional)

In the lecture material, you learned about some of the options on precompile and bind. This optional section will allow you to examine some more of the options.

__1. Look up the precompile command in the online Command Reference:

Start -> Programs -> IBM DB2 -> Information -> Information Center

Select the “Books” tab.

Expand the “References” option.

Double-click the Command Reference.

Scroll down the list on the left side until you get to the Precompile command, and click on it to bring up the text in the right hand pane.

__2. Scroll through the syntax of the precompile command. Then look at the details on the options. Some options that you may want to examine specifically are:

__a. BINDFILE
__b. BLOCKING
__c. COLLECTION
__d. DATETIME
__e. ISOLATION
__f. OWNER
__g. PACKAGE
__h. QUALIFIER

__3. Look also at the BIND command. Note that many of the same options can be specified on the BIND. If the options are specified in both commands, the options indicated on the BIND will override the selections made on the PRECOMPILE.
Exercise 4. Program Structure II

What You Should Be Able to Do

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

• Code DECLARE CURSOR statements in application programs to support retrieval of multirow answer sets
• Code OPEN, FETCH, and CLOSE CURSOR statements
• Compare application enforcement of referential integrity with DB2 enforcement
• Ensure the WHENEVER statements have been coded properly
• Reinforce concepts presented in lecture that have been illustrated in the programming lab

Introduction

This lab will give you the opportunity to complete the coding on the skeleton program you were given. You will be adding the code necessary to process multiple rows being returned from the database.
Exercise Instructions

The lab solutions can be found in “Exercise Solutions” on page 4-5.

Declaring and Using a CURSOR

__ 1. EDIT your program code. Locate the words declare cursor. Use the following information concerning action code 'F' to code a DECLARE CURSOR statement that will support the requirements of this action.

In addition to the action code of 'F', the input file will contain the DEPARTMENT NUMBER.

The program should retrieve all the information from the view VPHONE for all the employees in the given DEPARTMENT NUMBER.

The output should be in alphabetical order on LAST NAME.

__ 2. Locate the section of the program that will handle action code 'F'. Delete or comment the statement that causes a branch to PGMEND in order to "activate" this portion of the code.

__ 3. While completing the code for this action, recall that cursor logic is similar to sequential file processing.

Code the OPEN CURSOR statement.

Code the initial FETCH statement.

If no employees are in the given DEPARTMENT NUMBER, MSG07 will be written.

If the PHONE NUMBER is NULL for a given row, place 'NONE' in the output structure.

Code the FETCH statement that will be executed until no more rows meet the search condition.

The program should handle multiple input records that contain action code 'F'. Code the CLOSE CURSOR statement that will allow reuse of the cursor.

DB2 Constraint Enforcement, Error Checking, and General Questions

__ 1. EDIT your program code. Locate the section of code that supports action code 'A'.

Recall that a referential constraint exists between the EMP and DEP tables.

Comment out the SQL statement that verifies the existence of the given DEPARTMENT NUMBER in DEP table.

__ 2. Prepare and execute your program.
Did the program run to completion? It should not have completed adding all the new employees.

What caused the program to end and display the error message? Why isn’t the code to display MSG11 executed?

Could you code your program to allow DB2 to indicate referential constraint errors, display a message like MSG11, and continue to process the input file? What would be the key changes you would make?

_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________

__ 3. Uncomment the code to check for the existence of the given DEPARTMENT NUMBER in the DEP table. Prepare and execute your program to ensure that it is still working successfully.

__ 4. Examine the sample solution presented in “Exercise Solutions” on page 4-5 to consider the following questions:

- Examine the table definitions found starting on page ix. Also, recall that VPHONE is a view of a JOIN between EMP and DEPT.

  Do you believe the statements you have coded for action code 'P' is the best possible from a performance standpoint? What alternative might you propose? Is your alternative **ALWAYS** going to yield the same result, or is such a consideration data dependent?

  __________________________________________________________
  __________________________________________________________
  __________________________________________________________
  __________________________________________________________

- Examine your code for action 'D'. Is the order of your DELETE statements important, considering the referential integrity that has been defined? Do you understand why the statements supporting action code 'I' are in the reverse order? If you did not DELETE the rows from the EMP table before the DELETE against the DEP table, would anything happen to the rows in the EMP table?

  __________________________________________________________
  __________________________________________________________
  __________________________________________________________
  __________________________________________________________
Student Exercises

- What is the unit of work for this program? Is this realistic in a transaction environment? What are the appropriate statements to end a unit of work for a DB2 program executing in native UNIX, Windows, or OS/2? CICS?

_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________

5. Be prepared to offer your opinions about the above questions during the lab discussion session.

END OF EXERCISE
Exercise Solutions

Declaring and Using a CURSOR

1. EDIT your program code. Locate the words declare cursor. Use the following information concerning action code 'F' to code a DECLARE CURSOR statement that will support the requirements of this action.

   In addition to the action code of 'F', the input file will contain the DEPARTMENT NUMBER.

   The program should retrieve all the information from the view VPHONE for all the employees in the given DEPARTMENT NUMBER.

   The output should be in alphabetical order on LAST NAME.

2. Locate the section of the program that will handle action code 'F'. Delete or comment the statement that causes a branch to PGMEND in order to "activate" this portion of the code.

3. While completing the code for this action, recall that cursor logic is similar to sequential file processing.

   Code the OPEN CURSOR statement.

   Code the initial FETCH statement.

   If no employees are in the given DEPARTMENT NUMBER, MSG07 will be written.

   If the PHONE NUMBER is NULL for a given row, place 'NONE' in the output structure.

   Code the FETCH statement that will be executed until no more rows meet the search condition.

   The program should handle multiple input records that contain action code 'F'. Code the CLOSE CURSOR statement that will allow reuse of the cursor.

DB2 Constraint Enforcement, Error Checking, and General Questions

1. EDIT your program code. Locate the section of code that supports action code 'A'.

   Recall that a referential constraint exists between the EMP and DEP tables.

   Remove or comment the SQL statement that verifies the existence of the given DEPARTMENT NUMBER in DEP table.

2. Prepare and execute your program.

   Did the program run to completion? It should not have completed adding all the new employees.
What caused the program to end and display the error message? Why isn't the code to display MSG11 executed?

Could you code your program to allow DB2 to indicate referential constraint errors, display a message like MSG11, and continue to process the input file? What would be the key changes you would make?

The program should not have executed completely because of the removal of application constraint checking. The code originally bypassed the INSERT into the EMP table if the provided department number was not found in the DEP table. When this check was removed, the program attempted to INSERT an employee with a non-matched department number.

The program displayed an error message and ended because it is using WHENEVER statements to handle errors. The WHENEVER SQLERROR statement transfers control to the label DBERROR. The code in DBERROR calls sqlaintp to format a message, which is then externalized by the program. The evaluation of WHENEVER conditions occurs before control is returned to the program, so the program does not have the opportunity to test the return code explicitly.

The program could be coded to allow DB2 to indicate referential integrity constraint errors and still display a customized message. The key change would be removing the WHENEVER conditions. This would permit testing of the SQLCODE via application logic.

Note: It is normally not appropriate to use "application checking" of primary key values if DB2 has a defined constraint. The application checking requires an additional call to DB2 and does not prevent DB2 from checking the constraints on INSERT of the foreign key. Application checking is redundant in this case.

3. Examine the sample solution presented in “Exercise Solutions” on page 4-5 to consider the following questions:

- Examine the table definitions found starting on page ix. Also, recall that VPHONE is a view of a JOIN between EMP and DEPT.

Do you believe the statements you have coded for action code 'P' is the best possible from a performance standpoint? What alternative might you propose? Is your alternative ALWAYS going to yield the same result, or is such a consideration data dependent?
The requirement for action code 'P' could be satisfied by SELECTing data directly from table EMP. The required columns are present in this single table. The use of the view VPHONE requires the underlying definition of the view to be invoked which causes a join operation between DEP and EMP. This is true even though no data is required from the DEP table to satisfy the SELECT list or the search condition on the WHERE clause.

SELECTing the data directly from table EMP may not ALWAYS yield the same result as SELECTing from the view VPHONE. If a row for an employee in the EMP table contains a WORKDEPT that is not found in the DEPTNO column of the DEP table, the employee would not be found through the view. (The underlying definition of the view would not be satisfied.) However, the employee would be found with a direct search against the EMP table.

Referential integrity could be used to ensure that any value in the column WORKDEPT had a parent in the column DEPTNO. In this case, SELECTing directly against the table would yield the same result as SELECTing against the view.

An application programmer should be aware of the underlying definition of all objects being accessed, including views.

- Examine your code for action 'D'. Is the order of your DELETE statements important, considering the referential integrity that has been defined? Do you understand why the statements supporting action code 'I' are in the reverse order? If you did not DELETE the rows from the EMP table before the DELETE against the DEP table, would anything happen to the rows in the EMP table?

The order of the DELETE statements for action code 'D' is important when one considers the referential integrity defined. The dependent rows are removed first, followed by the parent row. (DELETE the rows from EMP first, then DELETE the row from DEP.) INSERT logic requires the opposite approach. Parent rows must be added before dependents. In the lab problem for action 'D', if the order of the DELETE statements was reversed, the DELETE of the parent (DEP) row would cause the dependent foreign keys (EMP) to be SET NULL. These NULL values would not satisfy the WHERE clause of the second DELETE.
• What is the unit of work for this program? Is this realistic in a transaction environment? What are the appropriate statements to end a unit of work for a DB2 program executing in native UNIX, Windows, or OS/2? CICS?

The unit of work for the program is the entire program. In a transaction environment, a logical unit of work is defined by business requirements and is implemented by appropriate statements in the application. In DB2 under native UNIX, Windows or OS/2, these statements are COMMIT WORK and ROLLBACK WORK. In CICS, the SYNCPOINT should be used.

For more information regarding unit of work and DB2 application recovery, consult the Programmer's Guide.

4. Be prepared to offer your opinions about the above questions during the lab discussion session.

END OF EXERCISE
Exercise 5. Restart Program

What This Exercise Is About

This exercise allows you to examine how to code a program with a restart logic.

What You Should Be Able to Do

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

• Recognize the need for restarting within DB2 programs.
• Create a DB2 table and write code to support restart.

Introduction

This lab will give you the opportunity to understand a possible technique for restart, plus some experience in implementing it.

To give you further experience with different styles of programs, this exercise will use different data, tables, and views from the previous exercises. It is therefore again VERY important to read and understand the code and comments before changing it.
Exercise Instructions

The lab solutions can be found in “Restart Sample Solution” on page F-1.

Setting up the Data

__1. Run the CLP input file restart.ddl to set up the two tables needed for this exercise, VPERS01 and VRESTART.
   
   db2 -tvf restart.ddl
__2. Look at the definitions of the two tables.
   
   db2 describe table vpers01
   db2 describe table vrestart
__3. Run the CLP input file vpers01.ins to set up some personnel numbers in VPERS01 by inserting them from the master table. Check that 77 rows have been inserted.
   
   db2 -tvf vpers01.ins
   
   Note: This file also empties both tables, so it can be used to reset your environment if you need to rerun the program from scratch.
__4. Check that all rows have been inserted successfully.
   
   db2 select count(*) from vpers01
__5. Check that you have a program mupdate in your directory. This is the file containing the tax changes.

Coding the Program

The supplied batch program, restart.sqc (skeleton can be found in “Restart Source Skeleton”) reads in employee data from file MUPDATE, found in the same directory as this program, which includes their monthly and yearly tax exemptions. It updates these details on the VPERS01 table. 20 updates at a time are committed and checkpointed by putting the details of the last row updated in a restart table. Find the words PGM FLOW in the program and read this to understand its logic.

You will run the program twice to check that the program is restarting at the correct position.

__1. Edit, read and understand the program code.
__2. Locate first set of “?????” where the program variables are defined for the tables. Note the definitions which are used.
3. Where the program asks you to **READ RESTART TABLE**, write some SQL to check if the restart table has any rows in it. If it does, then we are restarting the program rather than running it from scratch. Call the appropriate routine.

4. With normal processing, make sure that the number of rows processed is counted and after 20 rows, add the details of this row to the restart table.

5. If it is a restart, read the input file records until you find the one matching the value in the restart table, then process from there in the normal way as above.

6. Do all the usual checking for errors, etc.

**Running the Program**

1. Prepare your program for execution in the usual way.

2. Execute your program. The input file with the employee/tax information is in the file MUPDATE.

3. Examine the output from your program by verifying the following, and comparing it with the sample solutions:
   - The program will update 49 rows. It is coded to stop at this point (simulating a machine or external failure that could occur at any time).
   - Because a checkpoint is taken after every 20 rows, the first 20 updates to VPERS01 (rows 1-20) will be COMMITted, the next 20 updates to VPERS01 (rows 21-40) will be COMMITted, but the last 9 updates will be ROLLBACKed (hence the message in the output).
   - The details of the row at checkpoint time is entered in the VRESTART table to enable a restart. In this case, it should be row 40.

4. Run the program again and verify that it restarts the program at row 41, updates rows 41 to 60, and takes a checkpoint at row 60. It will then continue to update to the end (row 77) but not take another checkpoint.

**END OF EXERCISE**
Exercise 6. Dynamic SQL

What This Exercise Is About

This exercise will give you the opportunity to code a simple fixed-list dynamic SQL statement in the program that you have been using up to this point.

What You Should Be Able to Do

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

- Code a FIXED-LIST dynamic SQL SELECT statement.

Introduction

This lab will add to the program you have been working with for most of the course and use a new option, option ‘*’.

This option will enable you to practice programming a Dynamic SQL statement.

It is the ‘fixed-list’ type of Dynamic statement discussed in the theory session. This means that the SELECT statement will always access a fixed number of columns, but may access any number of rows.

You will therefore find part of the SELECT statement hard-coded in the program already, and you will be taking the WHERE conditions from the input TRANSIN file, appending it to the SELECT, dynamically preparing it for execution and using a cursor to process the rows.
Exercise Instructions

The lab solutions can be found in “C Language Sample Solution”.

Coding the Dynamic SQL

__ 1. Edit the main program that you were using at the beginning of this course.

__ 2. Find the word ‘DYN’. You will see the variables to build the dynamic SQL statement and to receive the result have already been set up.

__ 3. Keep searching for the word ‘DYN’. You will then see that a special part of the program has been reserved for action code ‘*’.

__ 4. Code the following SQL statements:
   • Declare a cursor for the prepared SQL statement
   • A PREPARE statement for the SQL built in the program, partly from TRANSIN, partly hard-coded
   • OPEN, FETCH (within a loop) and CLOSE statements for your cursor.

__ 5. When you have completed coding, prepare the program in the usual way for execution.

__ 6. When the preparation is successful, run the program.

__ 7. Verify the result with the expected solution.

__ 8. Try updating the TRANSIN file yourself to see the range of WHERE conditions you can use. What length limit have we set?

   **88 characters for your WHERE conditions - the width of TRANSIN**

END OF EXERCISE
Exercise 7. Loading Data

What This Exercise Is About

You will be using the import utility to put data into one of your tables.

What You Should Be Able to Do

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

• Refresh data in a test table using the import utility.

Introduction

You have been using the EMP table in labs up to this point. You will refresh the data in the EMP table with data from a sequential dataset.
Exercise Instructions

Exercise solutions can be found in “Exercise Solutions” on page 7-3.

Loading Data into EMP Table

__ 1. Note the current phone number in the EMP table for James Walker. It should be ‘2986’.
__ 2. Delete all rows from EMP.
__ 3. Request help on the import command.
__ 4. Issue the import statement to import data from the input file empin. The file contains delimited data.
__ 5. Verify that the data was loaded correctly. There should be 32 records, and James Walker’s phone number should now be ‘3311’.

END OF EXERCISE
Exercise Solutions

Loading Data into EMP Table

__1. Note the current phone number in the EMP table for James Walker. It should be ‘2986’.

   db2 select firstname, lastname, phoneno
       from emp where lastname = ‘WALKER’

__2. Delete all rows from EMP.

   db2 delete from emp

__3. Request help on the import command.

   db2 ? import

__4. Issue the import statement to import data from the input file empin. The file contains delimited data.

   db2 import from empin of del replace into emp

__5. Verify that the data was loaded correctly. There should be 32 records, and James Walker’s phone number should now be ‘3311’.

   db2 select firstname, lastname, phoneno
       from emp where lastname = ‘WALKER’

END OF EXERCISE
Exercise 8. Application Performance and Explain

What This Exercise Is About

You will generate and examine Visual Explain data. You will also look at some output generated for you of text-based explain output.

What You Should Be Able to Do

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

- Generate explain tables for their own use
- Populate explain tables using bind
- Populate explain tables using the Control Center
- View Visual Explain output using the Control Center and Command Editor
- Determine if indexes are used to satisfy SQL requirements

Introduction

Application programmers that write code to access DB2 data should apply basic tuning knowledge regarding database access.
Exercise Instructions

The lab solutions can be found in “Exercise Solutions” on page 8-13.

Create Explain Tables for Visual Explain

1. Explain tables can be generated in one of two ways:
   • Automatically via the Control Center
   • Manually by running the EXPLAIN.DDL found in sqlib\misc

   Generate your explain tables automatically via the Control Center (Hint: Right-click your database name and choose “Explain SQL”). Use the SQL statement “select lastname, salary from emp where lastname like ‘H%’”.

Examine Visual Explain Output

1. Look at the output generated for the select statement above. Is an index being used in this request?

2. Why isn’t DB2 using an index?

3. At this point, you would normally engage your DBA to work with you to determine the correct indexing given your use of the data, but, since you are researching this now, go ahead and create an index on the LASTNAME column.
4. Run RUNSTATS on the EMP table and its indexes.

5. Run your Visual Explain again and see if it has changed.

6. Has the graph changed?

7. What kind of predicate is being applied? Range-delimiting or Index-Sargable?

8. To see an index sargable predicate, try an SQL statement of “select lastname from emp where lastname like "%H"”. What other characteristic does this query have?
Generating Explain Data for Lab Program

__ 1. Bind your lab program again, using the bind option that will generate explain data that can be viewed via Visual Explain.

__ 2. To view this data, right-click on the database name in the Control Center and select Show Explained Statements History. Find the SQL statement which is selecting FIRSTNME, MIDINIT, and LASTNAME from EMP where EMPNO = a host variable. Show the Access Plan Graph for this statement.

__ 3. Is an index being used in this access?

You can continue to explore these explains as you have interest and time.
Investigating Index Usage

1. The explain data shown below was executed against a table that had an identical definition to the EMP table you have been using in class. The only indexes on the table were on the EMPNO column and the WORKDEPT column.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.

SQL Statement:

```sql
from emp
where edlevel between :H00016 and :H00017
```

Estimated Cost = 28
Estimated Cardinality = 1

Access Table Name = USER11.EMP  ID = 18
| #Columns = 1
| Index Scan: Name = USER11.EMPX2 ID = 2
| #Key Columns = 0
| Index-Only Access
| Index Prefetch: None
| Insert Into Sorted Temp Table  ID = t1
| #Columns = 1
| #Sort Key Columns = 1
| Key 1: (Ascending)
| Sortheap Allocation Parameters:
| | #Rows = 56
| | Row Width = 12
| | Piped
| Isolation Level: Uncommitted Read
| Lock Intents
| | Table: Intent None
| | Row : None
Sorted Temp Table Completion  ID = t1
List Prefetch RID Preparation
Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Fetch Direct
| Lock Intents
| | Table: Intent Share
| | Row : Next Key Share
| Residual Predicate(s)
| | #Predicates = 2
Return Data to Application
| #Columns = 14

What access strategy is being used to satisfy the statement?
Is this a concern? What are some factors that would influence your answer?
What type of locking is being done at the table level? What about at the row level?

2. Assume that the statement being analyzed was being executed repetitively in a high use transaction, and that the EMP table was of significant size. The database administrator for your installation decided to provide a non-unique index on the column EDLEVEL in order to enhance the application's performance. The administrator immediately bound the program and explained the package.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.
Did the presence of the index on EDLEVEL cause the database manager to change the access strategy? Is this necessarily the correct decision?

What additional step should be taken after adding the index so that the optimizer can make an "informed" decision?
3. After peeking at the solutions to the prior lab problem, the database administrator executed RUNSTATS against the EMP table and all indexes. The program was then bound and explained.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.

SQL Statement:

```
select * into :H00008 , :H00009 , :H00010 , :H00011 , :H00012 ,
: H00013 , :H00014 , :H00015 , :H00016 , :H00018 , :H00019 ,
: H00020 , :H00021 , :H00022
from emp
where edlevel between :H00016 and :H00017
```

Estimated Cost = 25
Estimated Cardinality = 1

Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Index Scan: Name = USER11.EDLEVEL ID = 3
| | #Key Columns = 1
| | Data Prefetch: None
| | Index Prefetch: None
| Lock Intents
| | Table: Intent Share
| | Row : Next Key Share
Return Data to Application
| #Columns = 14

End of section

Did the access strategy change? Are you more confident regarding the optimizer's decision?

Could executing RUNSTATS have caused a change to the access strategy?

Is index access always a desirable end goal for application performance?

4. The administrator decided the index on EDLEVEL was justified to support the discussed application as well as many others. The program was promoted to the
production environment. The performance of the application was acceptable in most cases. However, if the users of the application searched for a large range of EDLEVEL values, the response time seemed to degrade. (Assume that EDLEVEL does not actually represent high school or college levels, but is a number with a wider range of possible values.)

Your first response to these "allegations" was that more data being retrieved will take longer. However, the users countered by stating that the same queries run through the Command Line Processor seem to work faster than your application.

You decide to investigate the access strategies used for different ranges of values by using literals in your application code.

Examine the SQL statements and corresponding explain output in order to answer the questions that follow.
Section = 1

SQL Statement:

```
select *
from user11.emp
where edlevel between 12 and 13
```

Estimated Cost        = 25
Estimated Cardinality = 4

Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Index Scan: Name = USER11.EDLEVEL ID = 3
| | #Key Columns = 1
| | Data Prefetch: None
| | Index Prefetch: None
| Lock Intents
| | Table: Intent Share
| | Row : Next Key Share
Return Data to Application
| #Columns = 14

End of section
What access strategy is being used to satisfy the statement that searches for a small range of values? Is this the same as the strategy employed when host variables were referenced? (See the prior problem if necessary.)

Why is the access strategy different when the search criteria uses a large range of values? Can you state why the Command Line Processor may perform better than the application that utilizes host variables when large ranges are supplied?
Create Explain Tables for Visual Explain

__1. Explain tables can be generated in one of two ways:

- Automatically via the Control Center
- Manually by running the EXPLAIN.DDL found in sqllib\misc

Generate your explain tables automatically via the Control Center (Hint: Right-click your database name and choose “Explain SQL”). Use the SQL statement “select lastname, salary from emp where lastname like ‘H%’”.

Open the Control Center (Start -> Programs -> IBM DB2 -> Control Center). Expand the items in the window (Systems, Instances, and so forth) until you see the EDDB database. Right-click the EDDB database and choose “Explain SQL”.

In the Explain SQL Window, type in the SQL statement, “select lastname, salary from emp where lastname like ‘H%’”. Click OK. You should get a message that explain tables were created for you.

Examine Visual Explain Output

__1. Look at the output generated for the select statement above. Is an index being used in this request?

By looking at the Visual Explain graph, you see that DB2 is planning to do a TBSCAN (table scan).

__2. Why isn’t DB2 using an index?

You may be able to guess at this point, but the reason is that there is no usable index. Go back to the Control Center, and expand the EDDB database until you see “Tables”. Left-click Tables, and you will see a list of tables in the right pane. Right-click your EMP table, and select Show Related. In the Show Related window, select the Indexes tab.

You will see two indexes. Double-clicking each of them will bring up the definition of the index. One is on the EMPNO column; the other is on the WORKDEPT column. Neither of those would help in this example.

__3. At this point, you would normally engage your DBA to work with you to determine the correct indexing given your use of the data, but, since you are researching this now, go ahead and create an index on the LASTNAME column.

Right-click on Indexes and choose “Create > Index”. Supply an Index Name of “LAST” and indicate the table name is “EMP”. Select the LASTNAME column.
from the Available columns list, and move it to the Selected columns list by using the “>” button. Click OK to create the index.

4. Run RUNSTATS on the EMP table and its indexes.

   Right-click the EMP table and choose Run Statistics.

5. Run your Visual Explain again and see if it has changed.

   If you still have your Access Plan Graph open, use the Menu Bar option “Statement” and choose “Explain SQL”. If you have closed your Access Plan Graph, right-click the database name, and select “Explain SQL” as you did in the first section of this lab.

6. Has the graph changed?

   You should see that the graph is now using the new index, LAST, that you just created.

7. What kind of predicate is being applied? Range-delimiting or Index-Sargable?

   Right-click the operator “IXSCAN” and select Show Details. On the Operator details window, expand the window, and look in the third scrollable area, labeled “Input arguments”. You should see that it indicates “Start predicates” with a Predicate Text value of “(Q1.LASTNAME >= ‘H  ’)” (scroll to the right to see this) and “Stop predicates” with a Predicate Text value of “(Q1.LASTNAME <= ‘HZZZZZZZZZZZZZZZZZZ’).”

   It is doing a Range-Delimiting search.

8. To see an index sargable predicate, try an SQL statement of “select lastname from emp where lastname like ‘%H’”. What other characteristic does this query have?

   From the Access Plan Graph, select Statement -> Explain SQL. Issue the statement “select lastname from emp where lastname like ‘%H’”. When you right-click IXSCAN this time, you should see that it now indicates that there are “Sargable predicates”, and lists your search criteria. You should also see that it does NOT have Start or Stop predicates.

   The additional special characteristic of this query is that it does not require table access - note that no FETCH is indicated. The search criteria and the data can be derived from the index. This show Index-Only access.

Generating Explain Data for Lab Program

1. Bind your lab program again, using the bind option that will generate explain data that can be viewed via Visual Explain.

   From a DB2 Command Window, in your WSPGM subdirectory, issue:

   `db2 bind labpgm.bnd explsnap all`
2. To view this data, right-click the database name in the Control Center and select Show Explained Statements History. Find the SQL statement which is selecting FIRSTNME, MIDINIT, and LASTNAME from EMP where EMPNO = a host variable. Show the Access Plan Graph for this statement.

Scroll to the right in the Explained Statements History, and see the SQL Text. You can also right-click the statements and choose “Show SQL Text”. When the lab was developed, the statement of interest was the second statement in the list. The statement after DB2’s rewrite looked like this:

```sql
select firstnme, midinit, lastname into :H00011, :H00012, :H00013
from emp
where empno = :H00004
```

Right-click the statement and choose “Show Access Plan”.

3. Is an index being used in this access?

Yes. The index on EMPNO is being used, as a Range-Delimiting predicate.

4. You can continue to explore these explains as you have interest and time.

Investigating Index Usage

1. The explain data shown below was executed against a table that had an identical definition to the EMP table you have been using in class. The only indexes on the table were on the EMPNO column and the WORKDEPT column.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.
SQL Statement:

```sql
select * into :H00008 , :H00009 , :H00010 , :H00011 , :H00012 ,
         :H00013 , :H00014 , :H00015 , :H00016 , :H00018 , :H00019 ,
         :H00020 , :H00021 , :H00022
from emp
where edlevel between :H00016 and :H00017
```

Estimated Cost = 28
Estimated Cardinality = 1

Access Table Name = USER11.EMP  ID = 18
| #Columns = 1
| Index Scan: Name = USER11.EMPX2 ID = 2
| #Key Columns = 0
| Index-Only Access
| Index Prefetch: None
| Insert Into Sorted Temp Table  ID = t1
| #Columns = 1
| #Sort Key Columns = 1
| Key 1: (Ascending)
| Sortheap Allocation Parameters:
| #Rows   = 56
| Row Width = 12
| Piped
| Isolation Level: Uncommitted Read
| Lock Intents
| Table: Intent None
| Row : None

Sorted Temp Table Completion  ID = t1

List Prefetch RID Preparation
Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Fetch Direct
| Lock Intents
| Table: Intent Share
| Row : Next Key Share
| Residual Predicate(s)
| #Predicates = 2

End of section

What access strategy is being used to satisfy the statement?
Is this a concern? What are some factors that would influence your answer?

What type of locking is being done at the table level? What about at the row level?

**DB2 has decided to use a list prefetch using the index on WORKDEPT. List prefetch gathers the row identifiers from an index, sorts the row identifiers into data page order, and then accesses the data directly by knowing exactly which data page it needs.**
In this case, it will be accessing all of the rows of the table via the row identifiers.

In this particular case, the table is so small (56 rows) that there is little difference between one strategy and the next. There would not be a concern in this case, but there might be if there were thousands or millions of rows to process. The size of the table, the percentage of data actually needed by the application, and the frequency of execution for the statement are all factors that would influence any determination regarding the performance of this strategy.

The table is being locked Intent Share (IS). This locking strategy allows a great deal of concurrent access. The rows will be locked using Next Key Share locks (NS).

2. Assume that the statement being analyzed was being executed repetitively in a high use transaction, and that the EMP table was of significant size. The database administrator for your installation decided to provide a non-unique index on the column EDLEVEL in order to enhance the application's performance. The administrator immediately bound the program and explained the package.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.
Did the presence of the index on EDLEVEL cause the database manager to change the access strategy? Is this necessarily the correct decision?

What additional step should be taken after adding the index so that the optimizer can make an "informed" decision?

**The index is being used to satisfy the SQL requirements. The optimizer elected to use the index on EDLEVEL to avoid the requirement to access all rows of the table. Although the optimizer is aware of the presence of the index**
during the subsequent bind process, it is not aware of detailed information regarding the number of unique values in the index, the number of leaf nodes in the index, and other statistics that can influence the optimization process. Therefore, one cannot assume that using the index is actually the best plan.

The administrator should execute the RUNSTATS utility to provide the optimizer with more detailed information regarding the newly created index. This will allow the optimizer to make decisions on statistics based on the installation’s data, rather than on default assumptions.

3. After peeking at the solutions to the prior lab problem, the database administrator executed RUNSTATS against the EMP table and all indexes. The program was then bound and explained.

Examine the SQL statement and corresponding explain output in order to answer the questions that follow.

```
SQL Statement:

from emp
where edlevel between :H00016 and :H00017
```

Estimated Cost        = 25
Estimated Cardinality = 1

Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Index Scan: Name = USER11.EDLEVEL ID = 3
| #Key Columns = 1
| Data Prefetch: None
| Index Prefetch: None
| Lock Intents
| Table: Intent Share
| Row : Next Key Share
Return Data to Application
| #Columns = 14

End of section

Did the access strategy change? Are you more confident regarding the optimizer's decision?

Could executing RUNSTATS have caused a change to the access strategy?

Is index access always a desirable end goal for application performance?

The access strategy remained an index scan against the index created on the EDLEVEL column. However, now the optimizer is aware of how few rows will probably satisfy the predicate, so it is merely doing an index scan to retrieve.
those rows, rather than setting up to do a list prefetch. The statistics provided by the RUNSTATS utility were used as part of the optimization process. Therefore, you can be more confident of the decision of the optimizer.

As seen in this case, executing RUNSTATS can cause a change in the determined access strategy. There may be cases when the assumptions used by the optimizer when no actual statistics are available result in the choice of an access strategy that matches the strategy determined with the true statistics gathered at a later date. In such cases, the access strategy may not change. However, RUNSTATS should be executed in every case to provide the optimizer with accurate statistics.

A major misconception regarding relational database access is that using an index is ALWAYS desirable. Although index usage is many times necessary to achieve adequate performance, there are situations when a relation scan is the better performer. Such situations include access to a small table and access to a large percentage of the data in a table.

4. The administrator decided the index on EDLEVEL was justified to support the discussed application as well as many others. The program was promoted to the production environment. The performance of the application was acceptable in most cases. However, if the users of the application searched for a large range of EDLEVEL values, the response time seemed to degrade. (Assume that EDLEVEL does not actually represent high school or college levels, but is a number with a wider range of possible values.)

Your first response to these "allegations" was that more data being retrieved will take longer. However, the users countered by stating that the same queries run through the Command Line Processor seem to work faster than your application.

You decide to investigate the access strategies used for different ranges of values by using literals in your application code.

Examine the SQL statements and corresponding explain output in order to answer the questions that follow.
Section = 1

SQL Statement:

```sql
select *
from user11.emp
where edlevel between 12 and 13
```

Estimated Cost = 25
Estimated Cardinality = 4

Access Table Name = USER11.EMP  ID = 18
| #Columns = 14
| Index Scan: Name = USER11.EDLEVEL ID = 3
| | #Key Columns = 1
| | Data Prefetch: None
| | Index Prefetch: None
| Lock Intents
| | Table: Intent Share
| | Row : Next Key Share
Return Data to Application
| #Columns = 14

End of section
What access strategy is being used to satisfy the statement that searches for a small range of values? Is this the same as the strategy employed when host variables were referenced? (See the prior problem if necessary.)

Why is the access strategy different when the search criteria uses a large range of values? Can you state why the Command Line Processor may perform better than the application that utilizes host variables when large ranges are supplied?
The index is still used to satisfy the SQL statement that supplies a narrow range of values. This is the same strategy externalized on the prior problem.

When literals are used, the exact search criteria are known. When host variables are used, the optimizer must rely on default filtering based on the predicate type. In this particular case, the large range provided with literals enabled the optimizer to determine that using the index to access the data directly via the index would be inefficient since a large portion of the data would result, and the database manager may have to access the same data page over and over again. In this case, it would be more efficient to perform list prefetch using the index values. Such a determination is not possible when host variables are analyzed.

Dynamic applications prepare statements during execution, and a bind takes place during the prepare. The statement must be complete during the prepare, so the exact search criteria is known. This is why a dynamic application like the Command Line Processor may outperform a static application in certain cases. (The cost of the prepare statement may be offset by the "better" access strategy decision.)

END OF EXERCISE
Appendix A. C Language Source Skeleton

/*****************************************************************************/
/*                                                                    */
/* Sample C program for "DB2 Application Programming Workshop"        */
/*                          ( CG82/CP10 )                             */
/*                                                                    */
/* Last update = 11/16/2001                                           */
/*                                                                    */
/*****************************************************************************/
/* 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.                       */
/*****************************************************************************/

/*****************************************************************************/
/* Include C library definitions                                      */
declare_once
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*****************************************************************************/
/* Input/Output files                                                 */
/*   cardin = input request records                                   */
/*   report = output phone report printed to terminal                 */
declare_once
FILE *cardin;
FILE *report;

/*****************************************************************************/
/* Input transaction request from text file TRANSIN (INPUTAREA)       */
declare_once
EXEC SQL BEGIN DECLARE SECTION;
char function;          /* function code */
char lname[16];         /* last name */
char fname[13];         /* first name */
char eno[7];            /* employee number */
char pno[5];            /* phone number */
char dno[4];            /* dept number */
char mid;               /* middle initial */
char fill[37];          /* filler */
char last_char;         /* end of line character */
EXEC SQL END DECLARE SECTION;

/*****************************************************************************/
/* Output report header structures                                    */
declare_once
static char inpdesc[] = "INPUT DRIVER RECORD FOLLOWS:";
static char outdesc[] = "OUTPUT FROM TRANSACTION FOLLOWS:";
static char pluses[]  = "+++++++++++++++++++++++++++++++++++++++
                      "+++++++++++++++++++++++++++++++++++
                      ";
static char dots[]  =   ".....................................
                      
                      
                      ".....................................
                      ";
static char infoline[] = "REMEMBER TO COMPARE YOUR OUTPUT WITH THE HARD COPY PROVIDED.";
static char intext[]  = "FUNCTION LASTNAME    FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT";
static char indash[] = 
                      "   -     --------------- ------------ ------  ----  ---      -";
static char indtext[] = "FUNCTION TEXT";
static char inddash[] = 
                      "   -     ----------------------------------------";
static char funshead[] = "FIRST NAME    MIDDLE INITIAL  LAST NAME      
                      ";
static char funsdash[] = 
                      "------------        -         ---------------";
static char funhead[] = 
                      "LAST NAME    FIRST NAME  PHONE EMPNO  DEPARTMENT NAME       DEPTNO";
static char funfhead[] = "------------ ------------ ---- ------ ----------------------- ---";
static char funfdash[] = "------------ -----------  ------                                 
                      
                      
                      
                      ";
"MSG05I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED";
static char msg06[] =
"MSG06I - ROLLBACK FAILED, RETURN CODE IS:";
static char msg07[] =
"MSG07I - NO EMPLOYEE FOUND IN TABLE";
static char msg08[] =
"MSG08I - NEGATIVE RETURN CODE FROM SQLAINTP ";
static char msg09[] =
"MSG09I - COUNT OF EMPLOYEES INSERTED =";
static char msg10[] =
"MSG10I - DEPARTMENT DELETED";
static char msg11[] =
"MSG11I - NO DEPARTMENT RECORD FOUND ";
static char msg12[] =
"MSG12I - COUNT OF EMPLOYEES DELETED =";
static char msg13[] =
"MSG13I - DEPARTMENT INSERTED";
static char msg14[] =
"MSG14I - ROWS IMPACTED BY INSERT, UPDATE, OR DELETE =";
static char msg15[] =
"MSG15I - EMPLOYEE INSERTED";

/***************Function prototypes***********************************/
void write_inp(void);
void write_inpd(void);
void sql_err(void);
void Do_req(void);
/*****************************/
EXEC SQL BEGIN DECLARE SECTION;
/***** SQL INCLUDES ******/
/*************************/
char       empno[7];
char       firstnme[13];
char       midinit;
char       lastname[16];
char       workdept[4];
char       phoneno[5];
char       hiredate[11];
char       job[9];
short int edlevel;
char sex;
char birthdate[11];
double salary;
double bonus;
double comm;

/* Variables for VPHONE table */
char vlastname[16];
char vfirstname[13];
char vphonenumber[5];
char vempno[7];
char vdeptname[37];
char vdeptno[4];

/* Variables for connect statement */
char userid[9];
char password[9];
char database[9];

EXEC SQL END DECLARE SECTION;

/********************Error handling*******************************************************************************
/* SQL errors transfer control to DBERROR routine. */
/* SQL warnings transfer control to DBERROR routine. */
*******************************************************************************

/********************Error handling*******************************************************************************
/* SQL error handling  ****/
/* SQL warning handling ****/
*******************************************************************************

EXEC SQL BEGIN DECLARE SECTION;
/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

EXEC SQL BEGIN DECLARE SECTION;
/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

/********************Indicators*******************************************************************************
/* Indicator variable declarations below. */
*******************************************************************************

EXEC SQL END BEGIN DECLARE SECTION;

char hvempno[7];
char hfirstname[13];
char hvlastname[16];
char sql_stmt[121];
char sql_text[50] =
    "SELECT EMPNO, FIRSTNAME, LASTNAME FROM EMP WHERE ";

EXEC SQL END DECLARE SECTION;

/******* declare cursor ***************/
/* SQL cursor for VPHONE that will select all columns */
/* for the specified department number. Rows are */
/* sorted by last name. */
/*****************************/

/***** ???????????????????????*****/
***** SQL declare ****/
***** for cursor ****/
/**************************/

/*****************************/
/* Main routine */
/*
/* Read one record at a time from the input file. */
/* Process the request by looking at the first character in the */
/* record (should be an S, I, D, U, A, F, or P). */
/* Continue until end of input file. */
/*****************************/
int main()
{
    /* Open the input and output files */
    cardin = fopen("TRANSIN", "r");
    report = fopen("REPORT", "w");
    if (cardin == NULL || report == NULL) {
        printf("Either cardin or report could not be opened\n");
        exit(0);
    }

    /**********************
    /* Connect to database */
    ****************************/

    /***** ?????????????????????????????*****/
    /* Set up host variables for */
    /* userid, password, and database */
    /*************************/
    
    strcpy(userid,"usernn");
strcpy(password,"usernn");
strcpy(database,"XXXX");

/********************************************
/*           Code the connect statement    */
/*            using the host variables      */
/***************************************************/

/* Reminder message */
fprintf(report, "%s


", infoline);
fscanf(cardin, fmt3, &function,
    lname,
    fname,
    eno,
    pno,
    dno,
    &mid,
    fill,
    &last_char);

/* While more input, process */
while (!feof(cardin))
{
    Do_req();
    /* Read the next request */
fscanf(cardin, fmt3, &function,
    lname,
    fname,
    eno,
    pno,
    dno,
    &mid,
    fill,
    &last_char);
} /* endwhile */
pgmend();
DBERROR: sql_err();
    return(0);
} /* end main */

/*/ Process the current request */
***************************************************************************/
void Do_req(void)
{

    /* Determine request type */
    switch (function)
    {
        /**  S  **********************************************/
        /*          Given employee number, find first name, middle initial */
        /*          and last name in EMP. (If employee not found an error */


/* msg will be printed. */

/* Write out input file description / data followed by blanks. */
/* This code is not important to understand DB2 coding. It */
/* is a called routine that can be ignored for this and other */
/* transactions. */

write_inp();

/****????????????????????????????*****/
/**** SQL SELECT FROM EMP ****/
/*****************************/

if (sqlca.sqlcode == 100) /* EMPNO not found */
    fprintf(report, "%s\n", msg07);
else {
    /* Print empl data */
    fprintf(report, "%s\n", funshead);
    fprintf(report, "%s\n", funsdash);
    fprintf(report, fmt1, firstnme,
             midinit,
             lastname);
    fprintf(report, "%s\n", funsdash);
}
break; /* end of 'S' request */

/** p *********************************************/
/*
/* Given employee number, find first name, last name, and */
/* phone number in VPHONE. If the phone number is null, */
/* place the string 'NONE' in the output. If no employee */
/* is found, a message indicating such is printed. */
/*****************************/

case 'P':
    /* Delete pgmend statement when completing case 'P' */

    pgmend();

    write_inp();

/****????????????????????????????*****/
/**** SQL SELECT FROM VPHONE ****/
/*****************************/

if (sqlca.sqlcode == 100) /* EMPNO not found */
    fprintf(report, "%s\n", msg07);
else { /* Print eml data */

/***** Code to check indicator****/
/**** and process null values****/
/****************************/

fprintf(report, "%s
", funphead);
fprintf(report, "%s
", funpdash);
fprintf(report, fmt4, vfirstnme, vlastname, vphoneno);
fprintf(report, "%s
", funpdash);
}
break; /* end of 'P' request */

/** D **************************************************/
/*
/* Given dept number, DELETE the employees from that */
/* department from the EMP table and then DELETE the */
/* same department from the DEP table. */
/*
/****************************/
case 'D':

/* Delete pgmend statement when completing case 'D' */
pgmend();

write_inp();

/****************************/
/*** SQL DELETE FROM EMP ***/
/****************************/

fprintf(report, "%s%12d
", msg12, sqlca.sqlerrd[2]);

/****************************/
/*** SQL DELETE FROM DEP ***/
/****************************/

if (sqlca.sqlcode == 100)
    fprintf(report, "%s
", msg11);
else
    if (sqlca.sqlerrd[2] == 1)
        fprintf(report, "%s
", msg10);
    else /* Too many rows deleted */
        fprintf(report, "%s%12d
", msg14, sqlca.sqlerrd[2]);
break; /* end of 'D' request */

/****************************/
/*
/* Given dept number, copy the corresponding row from */
/****************************/
Students:

Exercise 1: DDL and DML Using SQL

Create a table named DEP with columns: depno, dname, location. Insert data into DEP.

Exercise 2: DDL and DML Using SQL

Create a table named EMP with columns: empno, ename, job, salary. Insert data into EMP.

Exercise 3: DDL and DML Using SQL

Create a table named DEPT with columns: deptno, dname, location. Insert data into DEPT.

Exercise 4: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 5: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 6: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 7: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 8: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 9: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 10: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 11: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 12: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 13: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 14: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 15: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 16: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 17: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 18: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 19: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 20: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 21: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 22: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 23: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 24: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 25: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 26: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 27: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 28: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 29: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 30: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 31: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 32: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 33: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 34: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 35: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 36: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 37: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 38: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 39: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 40: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 41: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 42: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 43: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 44: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 45: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 46: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 47: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 48: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 49: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 50: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 51: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 52: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 53: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 54: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 55: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 56: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 57: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 58: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 59: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.

Exercise 60: DDL and DML Using SQL

Create a table named DEPT_emp with columns: department, employee. Insert data into DEPT_emp.
pgmend();
write_inp();

/*** ????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????
/* Given empno, last name, first name, phone, dept and middle initial, */
/* (1) Validate that the dept number exists in DEP ( err msg written if not found ), (2) If dept found, INSERT the new employee row into EMP. If phone number on input record is XXXX then set new employee's phone to null. */
/* ( Completion msg will be printed. ) */
/* **************************************************************************/

case 'A':
    /* Delete pgmend statement when completing case 'A' */
    pgmend();
    write_inp();

    /**************************************************************************/
    if (sqlca.sqlcode == 100)
        fprintf(report, "%s\n", msg11);
    else {
        /**************************************************************************/
            Code "if else" logic to set indicator based on the input pno.  
            SQL INSERT INTO EMP  
            /**************************************************************************/

            fprintf(report, "%s\n", msg15);  
    }
    break; /* end of 'A' request */

    /**************************************************************************/
    /* F  ***********************************************************/
    /* Given department number, list all data from all rows in VPHONE that have that dept number. The list should be in alphabetical order of lastname. If a phone number in */
    /**************************************************************************/
/* VPHONE that you display is null, then print "NONE" on the */
/* output record. If no employees with the given dept number */
/* are found in VPHONE, a message will print. */
/*********************************************************************************/
case 'F':
    /* Delete pgmend statement when completing case 'F' */

pgmend();

write_inp();

/*********************************
*/
/sql OPEN CURSOR  */
/*********************************

/*********************************
*/
/sql FETCH  */
/*********************************

if (sqlca.sqlcode == 100)
    fprintf(report, "%s\n", msg07);
else {
    fprintf(report, "%s\n", funfhead);
    fprintf(report, "%s\n", funfdash);
}
   
while (!(!sqlca.sqlcode)) {
    ********************************
    *** Code to check for null ***
    *** PHONENO and set to ***
    *** "NONE" for printing. ***
    ********************************

    fprintf(report, fmt2, vlastname,
            vfirstname,
            vphonenumber,
            vemployee,
            vdepartment,
            vdepartment);

    ********************************
    *** SQL FETCH NEXT ROW ***
    ********************************

} /* end of while loop */
break; /* end of 'F' request */

/** '*!' **************************************************DYN*************************/
/**
* sql_stmt will contain the dynamic SQL statement read from the
* input file. HVLIST can receive three columns of data
* returned by the dynamic select. You must declare a cursor
* for the dynamic select then the dynamic statement in stnt
* must be prepared. The cursor can then be opened, rows
* fetched (until EOF), and finally, the cursor is closed.
*******************************************************************************

/* ?? DYN ????? DYN ????? */

case '*':
    /* Delete pgmend statement when completing case '*' */

    pgmend();

    write_inpd();

/***********************************************************/
/* Build the SQL statement from the hard coded first part */
/* (contained in sql_text), and the input from TRANSIN */
/***********************************************************/

strcat(sql_stmt,sql_text);
strcat(sql_stmt,lname);
strcat(sql_stmt,fname);
strcat(sql_stmt,eno);
strcat(sql_stmt,pno);
strcat(sql_stmt,dno);

/***********************************************************/
/* SQL Declare Cursor */
/***********************************************************/

/***********************************************************/
/* SQL Prepare statement */
/***********************************************************/

/***********************************************************/
/* SQL Open Cursor */
/***********************************************************/
if (sqlca.sqlcode == 100)
   fprintf(report, "%s\n", msg07);
else {
   fprintf(report, "%s\n", fundhead);
   fprintf(report, "%s\n", funddash);
}

while (!(sqlca.sqlcode)) {
   fprintf(report, fmt7, hvlastname,
           hvfirstnme,
           hvempno);

   if (sqlca.sqlcode != 100)
      fprintf(report, "%s\n", msg07);
   else {
      fprintf(report, "%s\n", fundhead);
      fprintf(report, "%s\n", funddash);
   }
}

fprintf(report, "%s\n", funddash);
break; /* end of '*' request */

/** Invalid function code on input record *******************************************/
/* */
/* If function on input record is unrecognized by program, */
/* a message will print. */
/**************************************************************/

default:
   fprintf(report, "%s\n", msg07);
} /* endswitch */
return;

DBERROR:
   sql_err();
} /* end Do_req */
void write_inp()
{
    fprintf(report, "\n\n\n", pluses);
    fprintf(report, "\n\n\n", inpdesc);
    fprintf(report, "\n\n\n", intext);
    fprintf(report, "\n\n\n", indash);
    fprintf(report, fmt5, function,
              lname,
              fname,
              eno,
              pno,
              dno,
              mid);
    fprintf(report, "\n\n\n", indash);
    fprintf(report, "\n\n\n", dots);
    fprintf(report, "\n\n\n", outdesc);
    return;
}

/*******************************************************
/* Write DYN input header and data. It is not necessary to study */
/* the following code.                                       */
/*******************************************************

void write_inpd()
{
    fprintf(report, "\n\n\n", pluses);
    fprintf(report, "\n\n\n", inpdesc);
    fprintf(report, "\n\n\n", indtext);
    fprintf(report, "\n\n\n", inddash);
    fprintf(report, fmt6, function,
              lname,
              fname,
              eno,
              pno,
              dno,
              mid,
              fill);
    fprintf(report, "\n\n\n", inddash);
    fprintf(report, "\n\n\n", dots);
    fprintf(report, "\n\n\n", outdesc);
    return;
}

/******************************************************
/* SQL error handler                                 */
/******************************************************
void sql_err() {
    #include <sqlenv.h>
    #define DATA_LEN 80
    char error_message_buffer[512];
    short buffer_size = sizeof(error_message_buffer);
    short int rc;
    int i;
fprintf(report, "\%s \%i\n", msg04, sqlca.sqlcode);
/* Format the sqlca */
rc = sqlaintp(error_message_buffer, buffer_size, DATA_LEN, &sqlca);
if (rc == 0)
    fprintf(report, "\%s \%hi\n", msg08, rc);
else
    fprintf(report, error_message_buffer);        /* Print formatted */

/* Attempt to rollback any work already done */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

pgmend();
return;
} /* end of sql_err */

pgmend()
{
    EXEC SQL ROLLBACK;
    if (sqlca.sqlcode != 0)                    /* If rollback fails */
        fprintf(report, "\%s \%i\n", msg06, sqlca.sqlcode);    /* display error msg */

fclose(cardin);
fclose(report);
exit(0);
return(0);
}
Appendix B. Expected Program Output

The following is a hard copy listing of the expected output from the program after it is completed.

REMEMBER TO COMPARE YOUR OUTPUT WITH THE HARD COPY PROVIDED.

INPUT DRIVER RECORD FOLLOWS:

(input driver record)

OUTPUT FROM TRANSACTION FOLLOWS:

(output from transaction)

INPUT DRIVER RECORD FOLLOWS:

(input driver record)

OUTPUT FROM TRANSACTION FOLLOWS:

(MSG07I - NO EMPLOYEE FOUND IN TABLE)

INPUT DRIVER RECORD FOLLOWS:

(input driver record)

OUTPUT FROM TRANSACTION FOLLOWS:
### FIRST NAME    LAST NAME        PHONE
------------  ---------------  ----
WILLIAM      JONES             NONE
------------  ---------------  ----

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td>000060</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

| FIRST NAME    LAST NAME        PHONE |
|---------------|---------------|---------|
| IRVING        | STERN         | 6423    |

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td>777777</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

MSG07I - NO EMPLOYEE FOUND IN TABLE

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>D11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

MSG12I - COUNT OF EMPLOYEES DELETED = 9
MSG10I - DEPARTMENT DELETED
INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

MSG12I - COUNT OF EMPLOYEES DELETED = 0
MSG11I - NO DEPARTMENT RECORD FOUND

++------------------------------------------------------------------------++

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

MSG11I - NO DEPARTMENT RECORD FOUND

++------------------------------------------------------------------------++

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT FROM TRANSACTION FOLLOWS:

MSG13I - DEPARTMENT INSERTED
MSG09I - COUNT OF EMPLOYEES INSERTED = 9

++------------------------------------------------------------------------++

INPUT DRIVER RECORD FOLLOWS:

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>LASTNAME</th>
<th>FIRSTNAME</th>
<th>EMPNO</th>
<th>PHONE</th>
<th>DEPT</th>
<th>MIDINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

______________________________________________________________________
OUTPUT FROM TRANSACTION FOLLOWS:

MSG02I - EMPLOYEE RECORD UPDATED

FIRST NAME    LAST NAME        PHONE
------------  ---------------  ----
MARILYN       MONROE           NONE
------------  ---------------  ----

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION LASTNAME        FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
-     --------------- ------------ ------  ----  ---      -
U     GENNA                        999999  8765
-     --------------- ------------ ------  ----  ---      -

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

OUTPUT FROM TRANSACTION FOLLOWS:

MSG03I - UPDATE FAILED, EMPLOYEE NOT FOUND

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION LASTNAME        FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
-     --------------- ------------ ------  ----  ---      -
U     BACHMAN                      000210  7654
-     --------------- ------------ ------  ----  ---      -

..............................................................................

OUTPUT FROM TRANSACTION FOLLOWS:

MSG02I - EMPLOYEE RECORD UPDATED

FIRST NAME    LAST NAME        PHONE
------------  ---------------  ----
WILLIAM       BACHMAN          7654
------------  ---------------  ----

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION LASTNAME        FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
-     --------------- ------------ ------  ----  ---      -
A     FUZZY           BEAR         111111  XXXX  D01      X
-     --------------- ------------ ------  ----  ---      -

..............................................................................
OUTPUT FROM TRANSACTION FOLLOWS:

MSG15I - EMPLOYEE INSERTED

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION    LASTNAME      FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
              --------------- ------------ ------  ----  ---      -
              -     --------------- ------------ ------  ----  ---      -
A     GOLDIE          LOCKS        222222 2468  D01      Y
              -     --------------- ------------ ------  ----  ---      -

OUTPUT FROM TRANSACTION FOLLOWS:

MSG15I - EMPLOYEE INSERTED

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION    LASTNAME      FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
              --------------- ------------ ------  ----  ---      -
              -     --------------- ------------ ------  ----  ---      -
A     DOESNOT         MATTER       333333  XXXX  BAD      L
              -     --------------- ------------ ------  ----  ---      -

OUTPUT FROM TRANSACTION FOLLOWS:

MSG11I - NO DEPARTMENT RECORD FOUND

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

INPUT DRIVER RECORD FOLLOWS:

FUNCTION    LASTNAME      FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
              --------------- ------------ ------  ----  ---      -
              -     --------------- ------------ ------  ----  ---      -
F                                                D11
              -     --------------- ------------ ------  ----  ---      -

OUTPUT FROM TRANSACTION FOLLOWS:

LAST NAME    FIRST NAME  PHONE EMPNO  DEPARTMENT NAME       DEPTNO
------------ ------------ ---- ------ ----------------------- ---
ADAMSON      BRUCE        NONE 000150 MANUFACTURING SYSTEMS   D11
BACHMAN      WILLIAM      7654 000210 MANUFACTURING SYSTEMS   D11
BROWN        DAVID        4501 000200 MANUFACTURING SYSTEMS   D11
LUTZ         JENNIFER     0672 000220 MANUFACTURING SYSTEMS   D11
MONROE       MARILYN      NONE 000180 MANUFACTURING SYSTEMS   D11
PIANKA       ELIZABETH    3782 000160 MANUFACTURING SYSTEMS   D11
STERN        IRVING       6423 000060 MANUFACTURING SYSTEMS   D11
INPUT DRIVER RECORD FOLLOWS:

FUNCTION LASTNAME    FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
-     --------------- ------------ ------  ----  ---      -
F                                                BAD
-     --------------- ------------ ------  ----  ---      -

OUTPUT FROM TRANSACTION FOLLOWS:

MSG07I - NO EMPLOYEE FOUND IN TABLE

INPUT DRIVER RECORD FOLLOWS:

FUNCTION LASTNAME    FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT
-     --------------- ------------ ------  ----  ---      -
F                                                D01
-     --------------- ------------ ------  ----  ---      -

OUTPUT FROM TRANSACTION FOLLOWS:

LAST NAME    FIRST NAME  PHONE EMPNO  DEPARTMENT NAME       DEPTNO
------------ ----------- ---- ------ ----------------------- ---
FUZZY        BEAR         NONE 111111 DEVELOPMENT CENTER      D01
GOLDIE       LOCKS        2468 222222 DEVELOPMENT CENTER      D01

INPUT DRIVER RECORD FOLLOWS:

FUNCTION TEXT
-     ----------------------------------------
*     WORKDEPT = 'D11' AND LASTNAME LIKE 'B%'
-     ----------------------------------------

OUTPUT FROM TRANSACTION FOLLOWS:

LAST NAME    FIRST NAME    EMPNO
------------ -----------  -----
BROWN        DAVID        000200
BACHMAN      WILLIAM      000210
------------ -----------  -----
Appendix C. C Language Sample Solution

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *cardin;
FILE *report;

EXEC SQL BEGIN DECLARE SECTION;
char function; /* function code */
char lname[16]; /* last name */
char fname[13]; /* first name */
char eno[7]; /* employee number */
char pno[5]; /* phone number */
char dno[4]; /* dept number */
char mid; /* middle initial */
char fill[37]; /* filler */
char last_char; /* end of line character */
EXEC SQL END DECLARE SECTION;

static char inptdesc[] = "INPUT DRIVER RECORD FOLLOWS:";
static char outdesc[] = "OUTPUT FROM TRANSACTION FOLLOWS:";
static char pluses[] = "+---------------------------------------------------+
                 +---------------------------------------------------+
static char dots[] = "+---------------------------------------------+
                 +---------------------------------------------+
static char infoline[] =
  "REMEMBER TO COMPARE YOUR OUTPUT WITH THE HARD COPY PROVIDED.";
static char intext[] =
  "FUNCTION LASTNAME    FIRSTNAME    EMPNO  PHONE  DEPT  MIDINIT";
static char indash[] =
  "   -     --------------- ------------ ------  ----  ---      -";
static char indtext[] =
  "FUNCTION TEXT";
static char inddash[] =
  "   -     ----------------------------------------";
static char funshead[] =
  "FIRST NAME    MIDDLE INITIAL  LAST NAME      ";
static char funsdash[] =
  "------------        -         ---------------";
static char funhead[] =
  "LAST NAME    FIRST NAME  PHONE EMPNO  DEPARTMENT NAME       DEPTNO";
static char fundhead[] =
  "LAST NAME    FIRST NAME   EMPNO                                  ";
static char funfhead[] =
  "LAST NAME    FIRST NAME  PHONE EMPNO  DEPARTMENT NAME       DEPTNO";

/**************************** Formats for scan\print */
/**************************** Formats for scan\print */
static char fmt1[] = "%-12s        %c         %-15s"
static char fmt2[] = "%-12.12s %-12s %-4s %-6s %-23s %-3s"
static char fmt3[] = "%c%15c%12c%6c%4c%3c%1c%36c%1c "
static char fmt4[] = "%-12s  %-15s  %-4s"
static char fmt5[] = "   %c     %-15s %-12s %-6s"
     %-4s %-3s      %c"
static char fmt6[] = "   %c     %-15s%-12s%-6s"%-4s%-3s%c%-36s"
static char fmt7[] = "%-12.12s %-12s %-4s"

/**************************** Messages */
/**************************** Messages */
static char msg01[] =
  "MSG01I - INVALID REQUEST, A-D-F-I-P-S-U VALID"
static char msg02[] =
  "MSG02I - EMPLOYEE RECORD UPDATED"
static char msg03[] =
  "MSG03I - UPDATE FAILED, EMPLOYEE NOT FOUND"
static char msg04[] =
  "MSG04I - SQL ERROR, RETURN CODE IS:";
"MSG05I - ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED";
static char msg06[] =
   "MSG06I - ROLLBACK FAILED, RETURN CODE IS:";
static char msg07[] =
   "MSG07I - NO EMPLOYEE FOUND IN TABLE";
static char msg08[] =
   "MSG08I - NEGATIVE RETURN CODE FROM SQLAINTP ";
static char msg09[] =
   "MSG09I - COUNT OF EMPLOYEES INSERTED =";
static char msg10[] =
   "MSG10I - DEPARTMENT DELETED";
static char msg11[] =
   "MSG11I - NO DEPARTMENT RECORD FOUND ";
static char msg12[] =
   "MSG12I - COUNT OF EMPLOYEES DELETED =";
static char msg13[] =
   "MSG13I - DEPARTMENT INSERTED";
static char msg14[] =
   "MSG14I - ROWS IMPACTED BY INSERT, UPDATE, OR DELETE =";
static char msg15[] =
   "MSG15I - EMPLOYEE INSERTED";

/**********************Function prototypes**********************/
void write_inp(void);
void write_inpd(void);
void sql_err(void);
void Do_req(void);
/**********************Function prototypes**********************/

/* Include for SQLCA.*/
EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
/**********************Variables declared below**********************/
/* Variables for EMP table */
char       empno[7];
char       firstnme[13];
char       midinit;
char       lastname[16];
char       workdept[4];
char       phoneno[5];
char       hiredate[11];
char       job[9];
short int edlevel;
char sex;
char birthdate[11];
double salary;
double bonus;
double comm;

/* Variables for VPHONE table */
char vlastname[16];
char vfirstnme[13];
char vphoneno[5];
char vempno[7];
char vdeptname[37];
char vdeptno[4];

/* Variables for connect statement */
char userid[9];
char password[9];
char database[9];

EXEC SQL END DECLARE SECTION;

/**************************Error handling*************************************/
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER SQLERROR GO TO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

EXEC SQL BEGIN DECLARE SECTION;

/*******************Indicators*****************************/
EXEC SQL END DECLARE SECTION;

short ind;

EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER SQLERROR GO TO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

EXEC SQL BEGIN DECLARE SECTION;

/*******************Indicators*****************************/
EXEC SQL END DECLARE SECTION;

short ind;

EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER SQLERROR GO TO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
char sql_stmt[121];
char sql_text[50] =
   "SELECT EMPNO, FIRSTNME, LASTNAME FROM EMP WHERE ";

EXEC SQL END DECLARE SECTION;

/******************************************
* SQL cursor for VPHONE that will select all columns *
* for the specified department number. Rows are *
* sorted by last name. *
******************************************/

EXEC SQL
   DECLARE K9 CURSOR FOR
      SELECT VLASTNAME,
      VFIRSTNME, 
      VPHONENO, 
      VEMPNO,  
      VDEPTNAME, 
      VDEPTNO FROM VPHONE WHERE VDEPTNO = :dno
   ORDER BY VLASTNAME;

/******************************************/
/* Main routine */
/*
/* Read one record at a time from the input file. */
/* Process the request by looking at the first character in the */
/* record ( should be an S, I, D, U, A, F, or P ). */
/* Continue until end of input file. */
******************************************/
int main()
{
   /* Open the input and output files */
   cardin = fopen("TRANSIN", "r");
   report = fopen("REPORT", "w");
   if (cardin == NULL || report == NULL) {
      printf("Either cardin or report could not be opened\n");
      exit(0);
   }

   /**********************************/
   /* Connect to database */
   /**********************************/

   /**********************************/
   /* SQL Connect to database */
   /**********************************/

   /**********************************/
   /* Set up host variables for */
   /* userid, password, and database */
   /**********************************/
strcpy(userid, "udba");
strcpy(password, "udba");
strcpy(database, "EDDB");

EXEC SQL CONNECT TO :database; /* USER :userid USING :password; */

/* Reminder message */
fprintf(report, "%s


", infoline);
fscanf(cardin, fmt3, &function,
   lname,
   fname,
   eno,
   pno,
   dno,
   &mid,
   fill,
   &last_char);

/* While more input, process */
while (!feof(cardin))
{
   Do_req();
   /* Read the next request */
fscanf(cardin, fmt3, &function,
      lname,
      fname,
      eno,
      pno,
      dno,
      &mid,
      fill,
      &last_char);
}

/pgmend();
DBERROR: sql_err();
return(0);

} /* end main */

/**********************************************************/
/* Process the current request */
/**********************************************************/
void Do_req(void)
{

/* Determine request type */
switch (function)
{

/**********************************************************/
/* S */
/**********************************************************/
/*
 * Given employee number, find first name, middle initial and last name in EMP. (If employee not found an error message will be printed.)

case 'S':

/* Write out input file description / data followed by blanks. */
/* This code is not important to understand DB2 coding. It is a called routine that can be ignored for this and other transactions. */

write_inp();

EXEC SQL SELECT FIRSTNME, LASTNAME, MIDINIT INTO :firstnme, :lastname, :midinit FROM EMP WHERE EMPNO = :eno;

if (sqlca.sqlcode == 100) /* EMPNO not found */
fprintf(report, "%s
", msg07);
else { /* Print empl data */
fprintf(report, "%s
", funshead);
fprintf(report, "%s
", funsdash);
fprintf(report, fmt1, firstnme, midinit, lastname);
fprintf(report, "%s
", funsdash);
}
break; /* end of 'S' request */


/****???????????????????????****/
/**** SQL SELECT FROM VPHONE ****/
*******************************/

/* Given employee number, find first name, last name, and phone number in VPHONE. If the phone number is null, place the string 'NONE' in the output. If no employee is found, a message indicating such is printed. */

case 'P':

/* Delete pgmend statement when completing case 'P' */

/* pgmend(); */

write_inp();

if (sqlca.sqlcode == 100) /* EMPNO not found */
    fprintf(report, "%s\n", msg07);
else { /* Print empl data */

    /*************************************/
    /**** Code to check indicator****/
    /**** and process null values****/
    /******************************/
    if (ind < 0) strcpy (vphoneno, "NONE");

    fprintf(report, "%s\n", funphead);
    fprintf(report, "%s\n", funpdash);
    fprintf(report, fmt4, vfirstnme,
            vlastname,
            vphoneno);
    fprintf(report, "%s\n", funpdash);
}
break; /* end of 'P' request */

/** D **************************************************************************/
/*                                                                */
/*  Given dept number, DELETE the employees from that             */
/*  department from the EMP table and then DELETE the             */
/*  same department from the DEP table.                           */
/*                                                                */
/*******************************************************************/
case 'D':

    /* Delete pgmend statement when completing case 'D' */
    /*    pgmend();       */

    write_inp();

    /*********************************/
    /*** SQL DELETE FROM EMP    */
    /***************************/
    EXEC SQL DELETE FROM EMP WHERE WORKDEPT = :dno;

    fprintf(report, "%s%12d\n", msg12, sqlca.sqlerrd[2]);

    /*********************************/
    /*** SQL DELETE FROM DEP    */
    /***************************/
    EXEC SQL DELETE FROM DEP WHERE DEPTNO = :dno;

    if (sqlca.sqlcode == 100)
    fprintf(report, "%s\n", msg11);
    else
    if (sqlca.sqlerrd[2] == 1)
    fprintf(report, "%s\n", msg10);
    else /* Too many rows deleted */
    fprintf(report, "%s%12d\n", msg14, sqlca.sqlerrd[2]);

    break; /* end of 'D' request */
```c
/*** I ********************************************************************/
/*                                                                */
/*  Given dept number, copy the corresponding row from CF10.DEP into */
/*  your DEP table. If a DEP row is copied, copy the set of rows from  */
/*  CF10.EMP table for the same department number.                  */
/*                                                                         *
/**************************************************************************/
case 'I':
    /* Delete pgmend statement when completing case 'I' */
    /* pgmend(); */

    write_inp();

    /**?...................................................................*/
    /** SQL INSERT INTO DEP *** */
    /**************************************************************************/

    EXEC SQL INSERT INTO DEP
    SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT FROM CF10.DEP
    WHERE DEPTNO = :dno;

    if (sqlca.sqlcode == 100)                                    /* dept not found */
        fprintf(report, "%s\n", msg11);
    else {
        if (sqlca.sqlerrd[2] == 1)
            fprintf(report, "%s\n", msg13);
        else /* Too many rows deleted */
            fprintf(report, "%s%d\n", msg14, sqlca.sqlerrd[2]);
    }

    /**?...................................................................*/
    /** SQL INSERT INTO EMP *** */
    /**************************************************************************/

    EXEC SQL INSERT INTO EMP
    SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
           WORKDEPT, PHONENO, HIREDATE, JOB,
           EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS,
           COMM
    FROM CF10.EMP
    WHERE WORKDEPT = :dno;

    fprintf(report, "%s%12d\n", msg09, sqlca.sqlerrd[2]);
}
break; /* end of 'I' request */

/** U ********************************************************************/
/*                                                                */
/*  Given the last name, phone number, and employee number: */
/*  Update the lastname and phone of the matching employee in EMP. */
/*  If the phone number in the input record is XXXX, */
/*  then set the employee's phone to null in EMP. */
/*  If employee number not found in EMP, an error msg is */
/*  printed. If employee updated, confirmation msg printed. */
/*  The program should then select the row just updated to */
```
/* verify the update coding logic. */
/*******************
case 'U':
    /* Delete pgmend statement when completing case 'U' */
*/

/* pgmend(); */
write_inp();

/***???????????????????????????????????????????????*/
/*** Code "if else" logic to set ***/
/*** indicator based on the input ***/
/*** pno. ***
ительн*/
*****************************************************************************/

if (!(strcmp(pno,"XXXX"))) ind = -1;
else ind = 0;

/**????????????????????????*/
/*** SQL UPDATE EMP ***/
*****************************************************************************/

EXEC SQL UPDATE EMP
    SET LASTNAME = :lname, PHONENO = :pno:ind
    WHERE EMPNO = :eno;

if (sqlca.sqlcode == 100)
    fprintf(report, "%s\n", msg03);
else {
    if (sqlca.sqlerrd[2] == 1)
        fprintf(report, "%s\n\n", msg02);
    else  /* Too many rows changed */
        fprintf(report, "%s%d\n\n", msg14, sqlca.sqlerrd[2]);

/**???????????????????????????????*/
/*** SQL SELECT FROM ***/
/*** EMP TABLE. ***/
*****************************************************************************/

EXEC SQL SELECT FIRSTNME, LASTNAME, PHONENO
    INTO :vfirstnme, :vlastname, :vphoneno:ind
FROM EMP
    WHERE EMPNO = :eno;

/**??????????????????????????????????????????????????????????????????????*/
**** C TO CHECK INDICATOR ****
**** AND PROCESS NULL VALUES****
*****************************************************************************/

if (ind < 0) strcpy(vphoneno,"NONE");

fprintf(report, "%s\n", funphead);
fprintf(report, "%s\n", funpdash);
fprintf(report, fmt4, vfirstnme,
vlastname,
vphoneno);
fprintf(report, "%s\n", funpdash);
break; /* end of 'U' request */

/** A **********************************************************/
/**
/* Given empno, last name, first name, phone, dept and middle initial, */
/* (1) Validate that the dept number exists in DEP ( err msg written if not found ), (2) If dept found, INSERT the */
/* new employee row into EMP. If phone number on input */
/* record is XXXX then set new employee's phone to null. */
/* ( Completion msg will be printed. ) */
/*****************************/
case 'A':

    /**********??????????????????????**********/
    /******** SQL to verify that **********/
    /******** given department is valid **********/
    /******************************************/

    EXEC SQL SELECT DEPTNO INTO :workdept
        FROM DEP
        WHERE DEPTNO = :dno;

    if (sqlca.sqlcode == 100)
        fprintf(report, "%s\n", msg11);
    else {

        /**********??????????????????????**********/
        /******** Code "if else" logic to set indicator based on the input **********/
        /******** pno. **********/
        /**********************************/

        if (!(strcmp(pno,"XXXX"))) ind = -1;
        else ind = 0;

        /**********??????????????????????**********/
        /******** SQL INSERT INTO EMP **********/
        /**********************************/

        EXEC SQL INSERT INTO EMP
            (EMPNO, LASTNAME, FIRSTNAME, PHONENO, WORKDEPT, MIDINIT)
            VALUES (:eno, :lname, :fname, :pno, :ind, :dno, :mid);

        fprintf(report, "%s\n", msg15);
    }

break; /* end of 'A' request */
/**  F  **************************************************/  
/*                                                            */
/* Given department number, list all data from all rows in     */
/* VPHONE that have that dept number. The list should be in    */
/* alphabetical order of lastname. If a phone number in        */
/* VPHONE that you display is null, then print "NONE" on the    */
/* output record. If no employees with the given dept number   */
/* are found in VPHONE, a message will print.                 */
/*************************************************************************/
case 'F':
    /* Delete pgmend statement when completing case 'F' */
    /* pgmend(); */
    write_inp();

    EXEC SQL OPEN K9;
    EXEC SQL FETCH K9
        INTO :vlastname, :vfirstname, :vphonenumber, :ind,
            :vempno, :vdeptname, :vdeptno;

    if (sqlca.sqlcode == 100)
        fprintf(report, "%s\n", msg07);
    else {
        fprintf(report, "%s\n", funfhead);
        fprintf(report, "%s\n", funfdash);
    }
    while (!(sqlca.sqlcode)) {
        /* Code to check for null */
        /* PHONENO and set to "NONE" */
        /* for printing. */
        if (ind < 0) strcpy(vphonenumber, "NONE");
        fprintf(report, fmt2, vlastname, 
            vfirstname, vphonenumber, vempno, 
            vdeptname, vdeptno);

    }

    EXEC SQL FETCH NEXT ROW
EXEC SQL FETCH K9
    INTO :vlastname, :vfirstname, :vphonenumber :ind,
        :vempno, :vdeptname, :vdeptno;
}

EXEC SQL CLOSE K9;
break; /* end of 'F' request */

/* *' */
EXEC SQL DECLARE C1 CURSOR FOR S1;

EXEC SQL DECLARE CURSOR K9;
EXEC SQL CLOSE CURSOR K9;

EXEC SQL CLOSE CURSOR K9;
break; /* end of 'F' request */

/* ?? DYN ????? DYN ????? */
case '*':
    /* Delete pgmend statement when completing case '*' */
    /* pgmend(); */
    write_inpd();

    /* Build the SQL statement from the hard coded first part */
    /* (contained in sql_text), and the input from TRANSIN */
    strcat(sql_stmt,sql_text);
    strcat(sql_stmt,lname);
    strcat(sql_stmt,fname);
    strcat(sql_stmt,eno);
    strcat(sql_stmt,pno);
    strcat(sql_stmt,dno);

    /* SQL Declare Cursor */
EXEC SQL DECLARE C1 CURSOR FOR S1;

    /* SQL Prepare statement */
EXEC SQL PREPARE S1 FROM :sql_stmt;

/***??????????????????????????????????****
/*** SQL Open Cursor *********/
****************************************************************************/

EXEC SQL OPEN C1;

/***??????????????????????????????????****
/*** SQL FETCH *********/
****************************************************************************/
EXEC SQL FETCH C1 INTO :hvempno, :hvfirstnme, :hvlname;
if (sqlca.sqlcode == 100)
    fprintf(report, "%s
", msg07);
else {
    fprintf(report, "%s
", fundhead);
    fprintf(report, "%s
", funddash);
}
while (!(sqlca.sqlcode)) {
    fprintf(report, fmt7, hvlastname,
            hvfirstnme,
            hvempno);

    /****?????????????????????????????????****
    /**** SQL FETCH NEXT ROW ***********/
    /******************************************/
    EXEC SQL FETCH C1 INTO :hvempno, :hvfirstnme, :hvlname;
}

/****?????????????????????????????????****
/**** SQL CLOSE CURSOR ***********/
******************************************/

EXEC SQL CLOSE C1;
fprintf(report, "%s
", funddash);
broadcast; /* end of '*' request */

/** Invalid function code on input record ***********************/
/*****
/* If function on input record is unrecognized by program, */
/* a message will print. */
*******************************************************************************/

default:
    fprintf(report, "%s
", msg07);
} /* endswitch */
return;

DBERROR:
    sql_err();
} /* end Do_req */
/********************************************************************/
/* Write input header and data. It is not necessary to study */
/* the following code. */
/**************************************************************************/

void write_inp()
{
    fprintf(report, "\n%s\n", pluses);
    fprintf(report, "\n%s\n", inpdesc);
    fprintf(report, "\n%s\n", intext);
    fprintf(report, "\n%s\n", indash);
    fprintf(report, fmt5, function,
                     lname,
                     fname,
                     eno,
                     pno,
                     dno,
                     mid);
    fprintf(report, "\n%s\n", indash);
    fprintf(report, "\n%s\n", dots);
    fprintf(report, "\n%s\n", outdesc);
    return;
}

/**************************************************************************/
/* Write DYN input header and data. It is not necessary to study */
/* the following code. */
/**************************************************************************/

void write_inpd()
{
    fprintf(report, "\n%s\n", pluses);
    fprintf(report, "\n%s\n", inpdesc);
    fprintf(report, "\n%s\n", indtext);
    fprintf(report, "\n%s\n", inddash);
    fprintf(report, fmt6, function,
                     lname,
                     fname,
                     eno,
                     pno,
                     dno,
                     mid,
                     fill);
    fprintf(report, "\n%s\n", inddash);
    fprintf(report, "\n%s\n", dots);
    fprintf(report, "\n%s\n", outdesc);
    return;
}

/**************************************************************************/
/* SQL error handler */
/**************************************************************************/

void sql_err() {
#include <sqlenv.h>
#define DATA_LEN 80

}
char error_message_buffer[512];
short buffer_size = sizeof(error_message_buffer);
short int rc;
int i;

fprintf(report, "%s %i\n", msg04, sqlca.sqlcode);
/* Format the sqlca */
rc = sqlaintp(error_message_buffer, buffer_size, DATA_LEN, &sqlca);
if (rc == 0)
    fprintf(report, "%s %hi\n", msg08, rc);
else
    fprintf(report, error_message_buffer); /* Print formatted */

/* Attempt to rollback any work already done */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
pgmend();
return;
} /* end of sql_err */

gpmend()
{
    EXEC SQL ROLLBACK;
    if (sqlca.sqlcode != 0) /* If rollback fails */
        fprintf(report, "%s %i\n", msg06, sqlca.sqlcode); /* display error msg */

fclose(cardin);
fclose(report);
exit(0);
return(0);
}
Appendix D. Restart Source Skeleton

/* Restart Program */
/*********************************************************************/
/* THIS BATCH PROGRAM READS IN EMPLOYEE DATA CONCERNING THEIR TAX */
/* CLASS WITH MONTHLY AND YEARLY TAX EXEMPTIONS AND INSERTS THE */
/* DETAILS ON TO A DB2 TABLE. EVERY 20 INSERTS ARE COMMITTED AND */
/* CHECKPOINTED BY PUTTING THE DETAILS ON TO A RESTART TABLE. */
/*********************************************************************/
/* changed May 2005 */
/* in_variable lengths */
/* compare only first 16 bytes for restarting */
/* */
/* PGM FLOW */
/*********************************************************************/
/* ! READ RESTART-TABLE (SQL) */
/* ! */
/* ! IS THERE A ROW IN THE RESTART TABLE? */
/* ! */
/* ! YES */
/* ! ! */
/* ! ! **** 1) (RESTART_ROUTINE) ! NORMAL_ROUTINE */
/* ! ! */
/* ! ! ! */
/* ! ! **** 2) (NORMAL_ROUTINE) */
/* ! */
/* ! */
/* ! ! DELETE THE ROW IN THE RESTART TABLE (SQL) */
/* ! */
/* ! */
/* ! ! DELETE OK */
/* ! */
/* ! ! YES */
/* ! ! ! */
/* ! END OF PGM */
/*********************************************************************/
/*********************************************************************/
/* 1) SUBROUTINE RESTART_ROUTINE *********************************** */
/* +---------------------------------------------------------------+ */
/* ! ! READ Input-FILE */
/* ! ! EOF Input-FILE? */
/* ! ! NO */
/* ! ! YES */
/* ! ! ! ! */
/* ! ! ! ! ERROR ROUTINE */
/* ! ! ! */
/* ! ! ! A */
/* ! ! ! ! ABNORMAL PROGRAM END */
/* ! ! ! */
/* ! ! DO UNTIL (EOF OR THE DATA FROM THE RESTART TABLE */
/* ! ! ! = THE DATA FROM THE Input-FILE) */
/* ! ! */
/* ! ! ! END OF ROUTINE */
/* ! ! */
/*********************************************************************/
/*********************************************************************/
/* 2) SUBROUTINE NORMAL_ROUTINE ************************************ */
/* +---------------------------------------------------------------+ */
/* ! READ Input-FILE (FIRST RECORD IF NORMAL OR ! */
/* ! NEXT RECORD IF A RESTART) ! */
/* +---------------------------------------------------------------------+ */
/* ! DO WHILE (Input-FILE NOT EOF) ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! UPDATE EMPLOYEE TABLE (SQL) ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! EMPLOYEE FOUND? ! */
/* ! ! YES ! NO !NEITHER! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! IS NUMBER OF RECORDS READ > CHECKPOINT LIMIT? ! */
/* ! ! YES ! NO ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! **** 3) (CHECKPOINT_ROUTINE) ! % ! */
/* ! ! +---------------------------------------------------------------------+ */
/* ! ! READ NEXT RECORD FROM Input-FILE ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! INCREMENT NUMBER OF RECORDS READ ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! **** 3) SUBROUTINE CHECKPOINT_ROUTINE ******************************* */
/* ! FILL UP RESTART STRUCTURE WITH DETAILS OF THE RECORD ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! INSERT/UPDATE TO RESTART TABLE (SQL) ! */
/* ! +---------------------------------------------------------------------+ */
/* ! ! INSERT/UPDATE OK ! yes ! no ! */
/* ! ! db2 commit ! yes ! no ! */
/* ! ! COMMIT OK ! yes ! no ! */
/* ! ! END OF ROUTINE ! yes ! no ! */
/* ! +---------------------------------------------------------------------+ */
/* 4) SUBROUTINE DBERRORP ************************************************** */
/* ! ERRORTEXT = ????????? ERROR MESSAGE ! */
/* ! CALL DBERRORP ERROR-ROUTINE ! */
/* ! CALL PLIRETC(12) SHOW RETURN-CODE ! */
/* ! RETURN ABNORMAL PROGRAM END ! */
/* ! +---------------------------------------------------------------------+ */
/* ********************************************************************** */
/* ********************************************************************** */
/* TABELLEN / VIEWS THAT ARE USED */
/* ********************************************************************** */
/* * TABLE / MEMBER NAME STRUCTURE NAME FIELD NAMES */
/* * VIEW */
/* ------------------------------------------------------------- */
/*    VRESTART     MRESTART     SRESTART     PGID      */
/*    TIMESTAMP   */
/*    RTEXT      */
/*    */
/*    VPERS01    MPERS01    SPERS01    EMPNO     */
/*    TAXCLASS  */
/*    TAXFREEM  */
/*    TAXFREEY  */
/*    */
/*-------------------------------------------------------------*/
/*---*/
/*-------------------------------------------------------------*/
/* *** PSEUDOCODE *** */
/* READ RESTART TABLE (SQL) */
/* SELECT */
/* IF THERE IS A ROW IN THE RESTART TABLE */
/* REPEAT UNTIL (END OF FILE (Input)) OR */
/* (DATA FROM Input FILE = DATA FROM RESTART TABLE) */
/* READ Input FILE */
/* IF END OF FILE (Input) */
/* ERROR ROUTINE */
/* END PGM ABNORMALLY */
/* CALL RESTART_ROUTINE */
/* IF THERE IS NO ROW IN THE RESTART TABLE */
/* CALL NORMAL_ROUTINE */
/* OTHERWISE CALL DBERRORP */
/* END PGM ABNORMALLY */
/* READ Input-FILE */
/* REPEAT UNTIL (END OF FILE (Input)) */
/* UPDATE VPERS01 TABLE (SQL) */
/* SELECT */
/* IF NO ROWS FOUND IN VPERS01 TABLE */
/* PUT OUT MESSAGE */
/* IF ROW(S) FOUND IN VPERS01 TABLE */
/* NO ACTION */
/* OTHERWISE CALL DBERRORP */
/* END PGM ABNORMALLY */
/* IF Input_COUNTER > CHECKPOINT-LIMIT */
/* INSERT/UPDATE ROW IN RESTART TABLE (SQL) */
/* IF INSERT/UPDATE NOT OK */
/* CALL DBERRORP */
/* END PGM ABNORMALLY */
/* CALL CHECKPOINT_ROUTINE */
/* READ Input FILE */
/* DELETE ROW IN RESTART TABLE (SQL) */
/* IF DELETE NOT OK */
/* CALL DBERRORP */
/* END PGM ABNORMALLY */
/* END OF PGM */
/*-------------------------------------------------------------*/
/*---*/
/*-------------------------------------------------------------*/
/* Include C library definitions */
/*-------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/******************************************************************************/
/* Input/Output files
/*   cardin = input request records
/*   report = output phone report printed to terminal
/******************************************************************************/
FILE *cardin;
FILE *report;

/******************************************************************************/
/* Input transaction request from text file MUPDATE
/* DESCRIPTION OF DATA ELEMENTS
/*   TYPE         TYPE 'MUPDATE'
/*   in_empno     PERSONNEL NUMBER (EMPNO)
/*   in_taxclass  TAX CLASS (TAX CLASS)
/*   in_taxfreem  MONTHLY TAX EXEMPTION (TAXFREEM)
/*   in_taxfreey  YEARLY TAX EXEMPTION (TAXFREEY)
/*   REST         FILLER TO MAKE IT 80 BYTES
/******************************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
    char type[7];
    char in_empno[9];      /*May 2005*/
    char in_taxclass[2];   /*May 2005*/
    char in_taxfreem[7];   /*May 2005*/
    char in_taxfreey[7];   /*May 2005*/
    char rest[49];        /*May 2005*/
EXEC SQL END DECLARE SECTION;

short restart_switch;
short Input_counter;
short inpz;
char restext[81];
short x;

static char fmt1[] = "%7c%9c%2c%7c%7c%49c"
static char fmt2[] = "%-4s%-26s%-80s\n";

static char msg01[] =
    "CHECKPOINT : PGMID = ";
static char msg02[] =
    " TIMESTAMP = ";
static char msg03[] =
    " RTEXT = ";
static char msg04[] =
    "MSG04I - ROLLING BACK, RETURN CODE IS:"
static char msg05[] =
    "MSG05I - EITHER INPUT OR OUTPUT FILE CANNOT BE OPENED"
static char msg06[] =
    "MSG06I - ROLLBACK FAILED, RETURN CODE IS:"
static char msg07[] =
    "MSG07I - NORMAL START"
static char msg08[] =
    "MSG08I - NEGATIVE RETURN CODE FROM SQLAINTP ";
static char msg09[] =
"MSG09I - ERROR DELETING FROM VRESTART";
static char msg10[] =
"MSG10I - RESTARTING";
static char msg11[] =
"MSG11I - ERROR RESTARTING";
static char msg12[] =
"MSG12I - PERS NUM NOT FOUND FOR UPDATE";
static char msg13[] =
"MSG13I - ERROR FORCED - RESTART....";
static char msg14[] =
"MSG14I - COMMIT SUCCESSFUL";
static char msg15[] =
"MSG15I - END OF PROGRAM";
/********************Function prototypes*******************************/
void restart_routine(void);
void normal_routine(void);
void checkpoint_routine(void);
void sql_err(void);
/********************Function prototypes*******************************/
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
/*******??????????????????????????????????????????????????????????????????/ *
/ * Variables that map to the VPERS01 and RESTART tables are declared */
/ * below. Note the definitions.                                      */
/ *                                                                      */
/ ****************?????????????????????????????????????????????????????????????*/
/* Variables for VPERS01 table */
char empno[9];
char taxclass[2];
float taxfreem; /*TAXFREEM          DECIMAL(7, 2),*/
float taxfreey; /*TAXFREEY          DECIMAL(7, 2) */
/* Variables for RESTART table */
char pgmid[4];
char timestamp[26];
char rtext[81];
/* Indicator Variables */
short pgmt_i;
short pgyr_i;
/* Variables for connect statement */
char userid[9];
char password[9];
char database[9];

EXEC SQL END DECLARE SECTION;

/******************************/
/** SQL error handling ****/
/** SQL warning handling ****/
******************************/
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER SQLERROR GO TO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

/******************************/
/** Main routine           */
/******************************/
int main()
{
  float dummy=2;
  /* Open the input and output files */
  cardin = fopen("MUPDATE", "r");
  report = fopen("REPORT", "w");
  if (cardin == NULL || report == NULL) {
    fprintf(report, "%s\n", msg05);
    exit(0);
  }

  /******************************/
  /* Connect to database */
  /******************************/

  /******************************/
  /* Set up host variables for */
  /* userid, password, and database */
  /******************************/

  strcpy(userid,"udba");
  strcpy(password,"udba");
  strcpy(database,"EDDB");

  /******************************/
  /* Code the connect statement */
  /******************************/
  /* using the host variables */
V1.0

/*********************************************************************/
/* UPDATE VPERS01 DATA                                               */
/*********************************************************************/
    restart_switch = 0;
    x = 0;
    taxfreem = 0;
    taxfreey = 0;
    strcpy(in_taxfreem,""); /*May 2005*/
    strcpy(in_taxfreey,""); /*May 2005*/

/*********************************************************************/
/* SET UP WHERE STATEMENT FOR RESTART TABLE                          */
/* READ RESTART TABLE - DETERMINE IF THERE ARE ANY ROWS              */
/* FOR THE pgmid.                                                    */
/*********************************************************************/
    strcpy(pgmid,"TP01");

EXEC SQL

/*********************************************************************/
/* IF THERE IS A ROW IN RESTART                                    */
/* DO THE RESTART PROCESSING                                     */
/*********************************************************************/
    if (sqlca.sqlcode == 0)

/*********************************************************************/
/* If no row in restart table                                      */
/*********************************************************************/
    if (sqlca.sqlcode == 100)
        fprintf(report, "%s\n\n", msg07);

/*********************************************************************/
/* IF NO ROW IN RESTART                                            */
/* DO NORMAL PROCESSING                                            */
/* NORMAL PROCESSING VIA SUBROUTINE                                  */
/*********************************************************************/

EXEC SQL

/*********************************************************************/
/* DELETE ROW FROM RESTART TABLE AND CHECK FOR ERRORS               */
/*********************************************************************/
    if (sqlca.sqlcode < 0)
        fprintf(report, "%s\n\n", msg09);
void restart_routine()
{

  /*********************************************************************/
  /* READ Input FILE UNTIL                                          */
  /*   DATA FROM Input-FILE = DATA FROM RESTART TABLE                 */
  /*********************************************************************/
  fscanf(cardin, fmt1, type,
         in_empno,
         in_taxclass,
         in_taxfreem,
         in_taxfreey,
         rest);
  strcpy(restext,"");
  strcat(restext,type);
  strcat(restext,in_empno);
  strcat(restext,in_taxclass);
  strcat(restext,in_taxfreem);
  strcat(restext,in_taxfreey);
  strcat(restext,rest);

  /* compare only 16 bytes because restext read from vrestart table */
  /* has garbage at the end; the first 16 bytes are unique (May 2005) */

  while (!feof(cardin))
  {
    if (!strncmp(rtext,restext,16))
    {
      /*********************************************************************/
      /* RESTART PROCESSING - MESSAGE                                    */
      /*********************************************************************/
      fprintf(report, "%s\n\n", msg10);
      fprintf(report, fmt2, pgmid,
               timestamp,
               rtext);
      restart_switch = 1;
    }
  }
} /* end main */


```c
return;

}  
else  
{
    fscanf(cardin, fmt1, type,
    in_empno,
    in_taxclass,
    in_taxfreem,
    in_taxfreey,
    rest);
    strcpy(restext,"");
    strcat(restext,type);
    strcat(restext,in_empno);
    strcat(restext,in_taxclass);
    strcat(restext,in_taxfreem);
    strcat(restext,in_taxfreey);
    strcat(restext,rest);
}
} /* end while */

fprintf(report, "%s\n\n", msg11);

/*********************************************************************/
/* SET RESTART SWITCH TO '1' (-> UPDATE IN RESTART-TABLE)          */
/*********************************************************************/
restart_switch = 1;

return;

} /* END RESTART_ROUTINE */
/*********************************************************************/
/*--*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* SUBROUTINE NORMAL_ROUTINE                                          */
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/

void normal_routine()
{

/*********************************************************************/
/* READ Input-FILE (FIRST ROW IF NORMAL OR NEXT                     */
/*   ROW IF RESTART)                                                 */
/*  INCREMENT Input_counter (COUNTER) BY 1                           */
/*********************************************************************/
fscanf(cardin, fmt1, type,
    in_empno,
    in_taxclass,
    in_taxfreem,
    in_taxfreey,
    rest);

Input_counter = 1;
/*********************************************************************/
/* READ Input-FILE WHILE (Input-FILE NOT EOF)                       */
```
/*********************** while (!feof(cardin))  
{

/******************** ASSIGNED DATA FROM THE Input-FILE INTO THE SQL HOST VARIABLES */
/*********************************************************************************/
sscanf(in_taxfreem, " %f ", &taxfreem);
sscanf(in_taxfreey, " %f ", &taxfreey);

if (!strcmp(in_taxfreem, ""))
  pgmt_i = -1;
else
  pgmt_i = 0;

if (!strcmp(in_taxfreey, ""))
  pgyr_i = -1;
else
  pgyr_i = 0;

/************************************** UPDATE VPERS01 TABLE *******
/* WITH NEW TAX CLASS, NEW TAX FREE MONTHLY VALUE, and */
/* NEW TAX FREE YEARLY VALUE. */
/*********************************************************************************/
EXEC SQL

/******************** IF ROW FOUND  */
/* NO ACTION */
/******************** IF ROW NOT FOUND */
/* SAY SO IN A MESSAGE */
/*********************************************************************************/
if (sqlca.sqlcode == 100)
  fprintf(report, "%s\n\n", msg12);

/******************** OTHERWISE CALL ERROR ROUTINE  */
/******************** */
/******************** */
/**%*% %%%%%%%%%%%%%%%%%%% USED FOR TESTING RESTARTABILITY %%%%%%%%%%%%%%%%%%%%%%%%*/
inpz++;
  if (inpz > 50)
    {
    fprintf(report, "%s\n\n", msg13);
    exit(12);
  }
/**%*% %%%%%%%%%%%%%%%%%%% USED FOR TESTING RESTARTABILITY %%%%%%%%%%%%%%%%%%%%%%%%*/
/**??* ******************************************************/
/* PERFORM CHECKPOINT AFTER 20 ROWS VIA SUBROUTINE */
/*******************************************************************/
if (Input_counter > 19)

/*****************************************************************/
/* READ NEXT ROW FROM Input-FILE */
/* INCREMENT Input_counter BY 1 */
/*******************************************************************/

fscanf(cardin, fmt1,   type,
       in_empno,
       in_taxclass,
       in_taxfreem,
       in_taxfreey,
       rest);

Input_counter++;

} /* END WHILE */

return;
DBERROR: sql_err();

} /* END NORMAL_ROUTINE; */
/*******************************************************************/
/*--*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* SUBROUTINE CHECKPOINT_ROUTINE */
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
void checkpoint_routine()
{

/*******************************************************************/
/* INITIALIZE Input_counter TO 0 */
/* PUT RESTART INFO INTO RTEXT */
/*******************************************************************/

Input_counter = 0;

strcpy(pgmid,"TP01");

strcpy(rtext,""");
strcat(rtext,type);
strcat(rtext,in_empno);
strcat(rtext,in_taxclass);
strcat(rtext,in_taxfreem);
strcat(rtext,in_taxfreey);
strcat(rtext,rest);

EXEC SQL
VALUES (CURRENT_TIMESTAMP)
INTO :timestamp;
/**?**一个小时已经过去。*/
/* IF THERE WE HAVE ALREADY PUT A ROW INTO THE RESTART TABLE, */
/* UPDATE RESTART TABLE WITH THE MOST RECENT INFO */
/**?**一个小时的过早 */
if (restart_switch != 0)
{
    EXEC SQL
    
    EXEC SQL
}
/**?**一个小时的过早 */
/* IF NO ROWS IN RESTART TABLE */
/* SET RESTART_SWITCH TO '1'. */
/* INSERT INFO INTO RESTART TABLE */
/**?**一个小时的过早 */
else
{
    restart_switch = 1;
    EXEC SQL
    
    EXEC SQL
    if (sqlca.sqlcode == 0)
    {
        fprintf(report, "%s\n", msg14);
        /* MESSAGE FROM CHECKPOINT PROCESSING */
        fprintf(report, "%s %s\n", msg01, pgmid);
        fprintf(report, "%s %s\n", msg02, timestamp);
        fprintf(report, "%s %s\n", msg03, rtext);
    }

    return;
    DBERROR: sql_err();

}    /* end of CHECKPOINT_ROUTINE; */
/**?**一个小时的过早 */
/*--*/
/**?**一个小时的过早 */
/* SQL error handler */
/*************************************************************/
void sql_err() {

#include <sqlenv.h>
define DATA_LEN 80

char error_message_buffer[512];
short buffer_size = sizeof(error_message_buffer);
short int rc;
int i;

fprintf(report, "%s %i\n", msg04, sqlca.sqlcode);
/* Format the sqlca */
rc = sqlaintp(error_message_buffer, buffer_size, DATA_LEN, &sqlca);
if (rc == 0)
  fprintf(report, "%s %hi\n", msg08, rc);
else
  /* fprintf(report, error_message_buffer); */     /* Print formatted */
  fprintf(report, "%s\n", error_message_buffer);
/* Attempt to rollback any work already done */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
x = 1;
pgmend();
return;
} /* end of sql_err */

pgmend()
{

if (x == 1)
{
  EXEC SQL ROLLBACK;

  if (sqlca.sqlcode != 0)         /* If rollback fails */
    fprintf(report, "%s %i\n", msg06, sqlca.sqlcode); /* display error msg */
} else
  EXEC SQL COMMIT;
fprintf(report, "%s\n", msg15);
fclose(cardin);
fclose(report);
exit(0);
return(0);
}
Appendix E. Restart: Expected Program Output

The following is a hard copy listing of the expected output from the restart program.

First run:

MSG07I - NORMAL START

MSG14I - COMMIT SUCCESSFUL

CHECKPOINT : PGMID = TP01
    TIMESTAMP = 2001-11-16-17.00.02.021001
    RTEXT = PUPDATE0000000203100010380012456  ***** FIRST CHECKPOINT *****

MSG14I - COMMIT SUCCESSFUL

CHECKPOINT : PGMID = TP01
    TIMESTAMP = 2001-11-16-17.00.02.041000
    RTEXT = PUPDATE0000000403100010380012456  ***** SECOND CHECKPOINT *****

MSG13I - ERROR FORCED - RESTART....

Second run:

MSG10I - RESTARTING

TP012001-11-16-16.56.08.125000PUPDATE0000000403100010380012456  ***** SECOND CHECKPOINT *****

%MSG14I - COMMIT SUCCESSFUL

CHECKPOINT : PGMID = TP01
    TIMESTAMP = 2001-11-16-16.56.15.976001
    RTEXT = PUPDATE000000060100000000000000

MSG15I - END OF PROGRAM
Appendix F. Restart Sample Solution

/* Restart Program                                       */
/**************************************************************/
/* THIS BATCH PROGRAM READS IN EMPLOYEE DATA CONCERNING THEIR TAX */
/* CLASS WITH MONTHLY AND YEARLY TAX EXEMPTIONS AND INSERTS THE */
/* DETAILS ON TO A DB2 TABLE. EVERY 20 INSERTS ARE COMMITTED AND */
/* CHECKPOINTED BY PUTTING THE DETAILS ON TO A RESTART TABLE.    */
/**************************************************************/
/* changed May 2005                                      */
/* in_variable lengths                                  */
/* * compare only first 16 bytes for restarting          */
/* */
/* PGM FLOW                                             */
/* */
/* +---------------------------------------------------------------+ */
/* ! READ RESTART-TABLE (SQL)                             ! */
/* +---------------------------------------------------------------+ */
/* ! IS THERE A ROW IN THE RESTART TABLE?                ! */
/* ! ! YES ! NO                                             !NEITHER! */
/* ! +---------------------------------------------------------------+ */
/* ! ! ***** 1) (RESTART_ROUTINE) ! NORMAL_ROUTINE          ! */
/* ! +---------------------------------------------------------------+ */
/* ! ! ***** 2) (NORMAL_ROUTINE)                           ! */
/* ! +---------------------------------------------------------------+ */
/* ! ! DELETE THE ROW IN THE RESTART TABLE (SQL)          ! */
/* ! +---------------------------------------------------------------+ */
/* ! ! DELETE OK                                           ! */
/* ! ! YES ! NO                                             ! */
/* ! +---------------------------------------------------------------+ */
/* ! ! END OF PGM                                         ! */
/* ! +---------------------------------------------------------------+ */
/* ! 1) SUBROUTINE RESTART_ROUTINE *************************** */
/* +---------------------------------------------------------------+ */
/* ! ! READ Input-FILE                                      */
/* ! +---------------------------------------------------------------+ */
/* ! ! EOF Input-FILE?                                      */
/* ! ! NO ! YES                                            ! */
/* ! +---------------------------------------------------------------+ */
/* ! ! ERROR ROUTINE                                       ! */
/* ! ! % +ERROR ROUTINE !                                  ! */
/* ! ! ! ABNORMAL PROGRAM END                              ! */
/* ! +---------------------------------------------------------------+ */
/* ! DO UNTIL (EOF OR THE DATA FROM THE RESTART TABLE      ! */
/* ! ! = THE DATA FROM THE Input-FILE) ! */
/* ! +---------------------------------------------------------------+ */
/* ! END OF ROUTINE                                        ! */
/* ! +---------------------------------------------------------------+ */
/* ! 2) SUBROUTINE NORMAL_ROUTINE *************************** */
/* +---------------------------------------------------------------+ */
/* ! READ Input-FILE (FIRST RECORD IF NORMAL OR       */
/* ! NEXT RECORD IF A RESTART)                   */
/* +---------------------------------------------------------------+ */
/* ! DO WHILE (Input-FILE NOT EOF)                   */
/* +---------------------------------------------------------------+ */
/* ! ! UPDATE EMPLOYEE TABLE (SQL)                 */
/* +---------------------------------------------------------------+ */
/* ! ! EMPLOYEE FOUND?                                 */
/* +---------------------------------------------------------------+ */
/* ! ! YES ! NO                                         */
/* +---------------------------------------------------------------+ */
/* ! ! % ! ERROR MESSAGE                                */
/* +---------------------------------------------------------------+ */
/* ! ! IS NUMBER OF RECORDS READ > CHECKPOINT LIMIT?    */
/* +---------------------------------------------------------------+ */
/* ! ! YES ! NO                                         */
/* +---------------------------------------------------------------+ */
/* ! ! ***** 3) (CHECKPOINT_ROUTINE)                     */
/* +---------------------------------------------------------------+ */
/* ! ! READ NEXT RECORD FROM Input-FILE               */
/* +---------------------------------------------------------------+ */
/* ! ! INCREMENT NUMBER OF RECORDS READ                */
/* +---------------------------------------------------------------+ */
/* 3) SUBROUTINE CHECKPOINT_ROUTINE ******************************* */
/* +---------------------------------------------------------------+ */
/* ! FILL UP RESTART STRUCTURE WITH DETAILS OF THE RECORD */
/* +---------------------------------------------------------------+ */
/* ! INSERT/UPDATE TO RESTART TABLE (SQL)              */
/* +---------------------------------------------------------------+ */
/* ! INSERT/UPDATE OK                                  */
/* ! YES ! NO                                         */
/* +---------------------------------------------------------------+ */
/* ! DB2 COMMIT                                       */
/* ! YES ! NO                                         */
/* +---------------------------------------------------------------+ */
/* ! COMMIT OK                                        */
/* ! YES ! NO                                         */
/* +---------------------------------------------------------------+ */
/* ! END OF ROUTINE                                    */
/* +---------------------------------------------------------------+ */
/* 4) CALL SUBROUTINE DBERRORP ***************************** */
/* +---------------------------------------------------------------+ */
/* ! ERRORTEXT = ????????????? ERROR MESSAGE            */
/* ! CALL DBERRORP ERROR-Routine                      */
/* ! CALL PLIRETC(12) SHOW RETURN-CODE                */
/* ! RETURN ABNORMAL PROGRAM END                      */
/* +---------------------------------------------------------------+ */
/* ! TABLE / MEMBER NAME STRUCTURE NAME FIELD NAMES */
/* ! VIEW */
/* VRESTART  MRESTART  SRESTART  PGMD */
/* TIMESTAMP */
/* RTEXT */
/* */
/* VPERS01  MPERS01  SPERS01  EMPNO */
/* TAXCLASS */
/* TAXFREEM */
/* TAXFREEY */
/* */

/*****************************************************************************/
/*/--*/

/*****************************************************************************/
/* *** PSEUDOCODE *** */
/* READ RESTART TABLE (SQL)
/* SELECT */
/* IF THERE IS A ROW IN THE RESTART TABLE */
/* REPEAT UNTIL (END OF FILE (Input)) OR */
/* (DATA FROM Input FILE = DATA FROM RESTART TABLE) */
/* READ Input FILE */
/* IF END OF FILE (Input) */
/* END PGM ABNORMALLY */
/* CALL RESTART_ROUTINE */
/* IF THERE IS NO ROW IN THE RESTART TABLE */
/* CALL NORMAL_ROUTINE */
/* OTHERWISE CALL DBERRORP */
/* END PGM ABNORMALLY */
/* READ Input-FILE */
/* REPEAT UNTIL (END OF FILE (Input)) */
/* UPDATE VPERS01 TABLE (SQL) */
/* SELECT */
/* IF NO ROWS FOUND IN VPERS01 TABLE */
/* PUT OUT MESSAGE */
/* IF ROW(S) FOUND IN VPERS01 TABLE */
/* NO ACTION */
/* OTHERWISE CALL DBERRORP */
/* END PGM ABNORMALLY */
/* IF Input_COUNTER > CHECKPOINT-LIMIT */
/* INSERT/UPDATE ROW IN RESTART TABLE (SQL) */
/* IF INSERT/UPDATE NOT OK */
/* CALL DBERRORP */
/* END PGM ABNORMALLY */
/* CALL CHECKPOINT_ROUTINE */
/* READ Input FILE */
/* DELETE ROW IN RESTART TABLE (SQL) */
/* IF DELETE NOT OK */
/* CALL DBERRORP */
/* END PGM ABNORMALLY */
/* END OF PGM */

/*****************************************************************************/
/*/--*/

/*****************************************************************************/
/* Include C library definitions */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**************************************************************/
/* Input/Output files                                        */
/*   cardin = input request records                          */
/*   report = output phone report printed to terminal        */
/**************************************************************/
FILE *cardin;
FILE *report;

/**************************************************************/
/* Input transaction request from text file MUPDATE          */
/* DESCRIPTION OF DATA ELEMENTS                             */
/*    TYPE         TYPE 'MUPDATE'                            */
/*    in_empno     PERSONNEL NUMBER (EMPNO)                  */
/*    in_taxclass  TAX CLASS (TAX CLASS)                     */
/*    in_taxfreem  MONTHLY TAX EXEMPTION (TAXFREEM)          */
/*    in_taxfreey  YEARLY TAX EXEMPTION (TAXFREEY)           */
/*    REST         FILLER TO MAKE IT 80 BYTES                */
/**************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
char type[7];
char in_empno[9];    /*May 2005*/
char in_taxclass[2]; /*May 2005*/
char in_taxfreem[7]; /*May 2005*/
char in_taxfreey[7]; /*May 2005*/
char rest[49];       /*May 2005*/
EXEC SQL END DECLARE SECTION;
short restart_switch;
short Input_counter;
short inpz;
char restext[81];
short x;

static char fmt1[] = "%7c%9c%2c%7c%7c%49c";
static char fmt2[] = "%-4s%-26s%-80s\n";

static char msg01[] =
"CHECKPOINT : PGMID = ";
static char msg02[] =
"    TIMESTAMP = ";
static char msg03[] =
"    RTEXT = ";
static char msg04[] =
"MSG04I - ROLLING BACK, RETURN CODE IS:"
static char msg05[] =
"MSG05I - EITHER INPUT OR OUTPUT FILE CANNOT BE OPENED"
static char msg06[] =
"MSG06I - ROLLBACK FAILED, RETURN CODE IS:"
static char msg07[] =
"MSG07I - NORMAL START"
static char msg08[] =
"MSG08I - NEGATIVE RETURN CODE FROM SQLAINTP ";
static char msg09[] =
"MSG09I - ERROR DELETING FROM VRESTART";
static char msg10[] =
"MSG10I - RESTARTING";
static char msg11[] =
"MSG11I - ERROR RESTARTING";
static char msg12[] =
"MSG12I - PERS NUM NOT FOUND FOR UPDATE";
static char msg13[] =
"MSG13I - ERROR FORCED - RESTART....";
static char msg14[] =
"MSG14I - COMMIT SUCCESSFUL";
static char msg15[] =
"MSG15I - END OF PROGRAM"

/** Function prototypes **********************************************/
void restart_routine(void);
void normal_routine(void);
void checkpoint_routine(void);
void sql_err(void);
/** Function prototypes **********************************************/

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;

/**************************** Variables that map to the VPERS01 and RESTART tables are declared */
/**************************** Variables that map to the VPERS01 and RESTART tables are declared */
/**************************** Variables that map to the VPERS01 and RESTART tables are declared */

  /* Variables for VPERS01 table */
  char empno[9];
  char taxclass[2];
  float taxfreem; /*TAXFREEM DECIMAL(7, 2),*/
  float taxfreey; /*TAXFREEY DECIMAL(7, 2) */

  /* Variables for RESTART table */
  char pgmid[4];
  char timestamp[26];
  char rtext[81];

  /* Indicator Variables */
  short pgmt_i;
  short pgyr_i;

  /* Variables for connect statement */
  char userid[9];
char password[9];
char database[9];

EXEC SQL END DECLARE SECTION;

/******************************/
**** SQL error handling ****/
**** SQL warning handling ****/
/******************************/

EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER SQLERROR GO TO DBERROR;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

/**********************************************************************/
/* Main routine */
/* */
/* */
/* Read one record at a time from the input file. */
/**********************************************************************/
int main()
{
  float dummy=2;
  /* Open the input and output files */
  cardin = fopen("MUPDATE", "r");
  report = fopen("REPORT", "w");
  if (cardin == NULL || report == NULL) {
    fprintf(report, "%s\n", msg05);
    exit(0);
  }

  47502606

  strcpy(userid,"udba");
  strcpy(password,"udba");
  strcpy(database,"EDDB");

  47502606

  Code the connect statement */
  /* using the host variables */
  /**********************************************************************/
EXEC SQL CONNECT TO :database; /* USER :userid USING :password;*/

/**********************************************************************
/* UPDATE VPERS01 DATA                                               */
***********************************************************************/
    restart_switch = 0;
    x = 0;
    taxfreem = 0;
    taxfreey = 0;
    strcpy(in_taxfreem,""); /*May 2005*/
    strcpy(in_taxfreey,""); /*May 2005*/

/**********************************************************************
/* SET UP WHERE STATEMENT FOR RESTART TABLE                         */
/* READ RESTART TABLE - DETERMINE IF THERE ARE ANY ROWS              */
/* FOR THE pgmid.                                                    */
***********************************************************************/
    strcpy(pgmid,"TP01");

EXEC SQL
    SELECT *
    INTO  :pgmid, :timestamp, :rtext
    FROM  VRESTART
    WHERE PGMID = :pgmid;

/**********************************************************************
/* IF THERE IS A ROW IN RESTART                                       */
/* DO THE RESTART PROCESSING                                          */
**********************************************************************/
    if (sqlca.sqlcode == 0)
        restart_routine();

/**********************************************************************
/* If no row in restart table                                       */
***********************************************************************/
    if (sqlca.sqlcode == 100)
        fprintf(report, "%s\n\n", msg07);

/**********************************************************************
/* IF NO ROW IN RESTART                                              */
/* DO NORMAL PROCESSING                                              */
/* NORMAL PROCESSING VIA SUBROUTINE                                   */
**********************************************************************/
    normal_routine();

/**********************************************************************
/* DELETE ROW FROM RESTART TABLE AND CHECK FOR ERRORS                */
***********************************************************************/
EXEC SQL
    DELETE FROM VRESTART
    WHERE PGMID = :pgmid;

    if (sqlca.sqlcode < 0)
        fprintf(report, "%s\n\n", msg09);
/*********************************************************************/
/* END OF PROCESSING                                                */
/*********************************************************************/
pgmend();
DBERROR: sql_err();
return(0);
} /* end main */
/*********************************************************************/
/* END OF PROGRAM                                                   */
/*********************************************************************/
/*--*/
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
/* SUBROUTINE  RESTART_ROUTINE                                      */
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
void restart_routine()
{

/*********************************************************************/
/* READ Input FILE UNTIL                                           */
/* DATA FROM Input-FILE = DATA FROM RESTART TABLE                  */
/*********************************************************************/
fscanf(cardin, fmt1, type, in_empno, in_taxclass, in_taxfreem, in_taxfreey, rest);
strcpy(restext,"");
strcat(restext,type);
strcat(restext,in_empno);
strcat(restext,in_taxclass);
strcat(restext,in_taxfreem);
strcat(restext,in_taxfreey);
strcat(restext,rest);

/* compare only 16 bytes because restext read from vrestart table   */
/* has garbage at the end; the first 16 bytes are unique (May 2005) */

while (!feof(cardin))
{

    if (!strncmp(rtext,restext,16))
    {
    } /* RESTART PROCESSING - MESSAGE                               */
    /*********************************************************************/
    /*********************************************************************/
    fprintf(report, "%s\n", msg10);
    fprintf(report, fmt2, pgmid, timestamp, rtext);

    restart_switch = 1;
    return;
Student Exercises

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.
/* ASSIGN DATA FROM THE Input-FILE INTO THE SQL HOST VARIABLES */
sscanf(in_taxfreem," %f ",&taxfreem);
sscanf(in_taxfreey," %f ",&taxfreey);

if (!(strcmp(in_taxfreem,"")))
   pgmt_i = -1;
else
   pgmt_i = 0;

if (!(strcmp(in_taxfreey,"")))
   pgyr_i = -1;
else
   pgyr_i =0;

/* UPDATE VPERS01 TABLE */
/* WITH NEW TAX CLASS, NEW TAX FREE MONTHLY VALUE, and */
/* NEW TAX FREE YEARLY VALUE. */
EXEC SQL
UPDATE VPERS01
SET TAXCLASS = :in_taxclass,
    TAXFREEM = :taxfreem :pgmt_i,
    TAXFREEY = :taxfreey :pgyr_i
WHERE EMPNO = :in_empno;

if (sqlca.sqlcode == 100)
   fprintf(report, "%s\n\n", msg12);

/* OTHERWISE CALL ERROR ROUTINE */
/*--*/
/*%*/
/*%%%%%%%%%%%%%%%%%%%% USED FOR TESTING RESTARTABILITY %%%%%%%%%%%%%%*/
inpz++;
if (inpz > 50)
   {fprintf(report, "%s\n\n", msg13);
exit(12);
}
/*%*/
/*%%%%%%%%%%%%%%%%%%%% USED FOR TESTING RESTARTABILITY %%%%%%%%%%%%%%*/

/* PERFORM CHECKPOINT AFTER 20 ROWS VIA SUBROUTINE */
if (Input_counter > 19)
    checkpoint_routine();

fscanf(cardin, fmt1, type,
in_empno,
in_taxclass,
in_taxfreem,
in_taxfreey,
rest);

Input_counter++;
/* IF THERE WE HAVE ALREADY PUT A ROW INTO THE RESTART TABLE, */
/* UPDATE RESTART TABLE WITH THE MOST RECENT INFO */
/***************************************************************************/
if (restart_switch != 0)
{
  EXEC SQL
  UPDATE VRESTART
    SET   TIMESTAMP = :timestamp,
         RTEXT     = :rtext
    WHERE PGMID   = :pgmid;
}
/**??****************************************************************************
/* IF NO ROWS IN RESTART TABLE */
/* SET RESTART_SWITCH TO '1'. */
/* INSERT INFO INTO RESTART TABLE */
/***************************************************************************/
else
{
  restart_switch = 1;
  EXEC SQL
    INSERT INTO VRESTART
    VALUES (:pgmid, :timestamp, :rtext);
}
/**??****************************************************************************
/* DB2 COMMIT WITH CHECKING */
/***************************************************************************/
EXEC SQL COMMIT;
if (sqlca.sqlcode == 0)
{
  fprintf(report, "%s\n\n", msg14);
  /***************************************************************************/
  /* MESSAGE FROM CHECKPOINT PROCESSING */
  /***************************************************************************/
  fprintf(report, "%s %s\n", msg01, pgmid);
  fprintf(report, "%s %s\n", msg02, timestamp);
  fprintf(report, "%s %s\n\n", msg03, rtext);
}
return;
DBERROR: sql_err();
}  /* end of CHECKPOINT_ROUTINE; */
/***************************************************************************/
/*--*/

/***************************************************************************/
/* SQL error handler */
/***************************************************************************/
void sql_err() {
  
#include <sqlenv.h>
#define DATA_LEN 80

char error_message_buffer[512];
short buffer_size = sizeof(error_message_buffer);
short int rc;
int i;

fprintf(report, "%s %i\n", msg04, sqlca.sqlcode);
/* Format the sqlca */
rc = sqlaintp(error_message_buffer, buffer_size, DATA_LEN, &sqlca);
if (rc == 0)
    fprintf(report, "%s %hi\n", msg08, rc);
else
    /* fprintf(report, error_message_buffer); */ /* Print formatted */
    fprintf(report, "%s\n", error_message_buffer);
/* Attempt to rollback any work already done */
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
x = 1;
pgmend();
return;
} /* end of sql_err */

pgmend()
{

if (x == 1)
{
    EXEC SQL ROLLBACK;

    if (sqlca.sqlcode != 0) /* If rollback fails */
        fprintf(report, "%s %i\n", msg06, sqlca.sqlcode); /* display error msg */
} else
    EXEC SQL COMMIT;
fprintf(report, "%s\n", msg15);
fclose(cardin);
fclose(report);
exit(0);
return(0);
## Appendix G. Editor Command Summaries

### OS/2 EPM Editor Command Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edit a new file</strong></td>
<td>EPM pgmname.SQC</td>
</tr>
<tr>
<td><strong>Save Program and Return to System</strong></td>
<td>F4</td>
</tr>
<tr>
<td><strong>Save Program and Stay in Edit</strong></td>
<td>F2</td>
</tr>
<tr>
<td><strong>Cancel Changes &amp; Return to System</strong></td>
<td>F3</td>
</tr>
<tr>
<td><strong>Activate EPM class profile</strong></td>
<td>Press ctrl-i, type profile on and press Enter, then alt-o and o to save</td>
</tr>
<tr>
<td><strong>Add Blank Lines</strong></td>
<td>Move cursor to start of line and press Enter</td>
</tr>
<tr>
<td><strong>Set Tabs</strong></td>
<td>Alt-o then r then s</td>
</tr>
<tr>
<td><strong>Tab to next position</strong></td>
<td>Tab Key</td>
</tr>
<tr>
<td><strong>Move display UP one screen</strong></td>
<td>PageUp</td>
</tr>
<tr>
<td><strong>Move display DOWN one screen</strong></td>
<td>PageDown</td>
</tr>
<tr>
<td><strong>Move display UP 'n' lines</strong></td>
<td>Use up arrow when at top or Drag scroll bar indicator on right with mouse</td>
</tr>
<tr>
<td><strong>Move display LEFT</strong></td>
<td>Use left arrow when at left edge of screen or Drag scroll bar indicator on bottom with mouse</td>
</tr>
<tr>
<td><strong>Move display RIGHT</strong></td>
<td>Use right arrow when at right edge or Drag scroll bar indicator at bottom with mouse</td>
</tr>
<tr>
<td><strong>Move cursor to end of line</strong></td>
<td>End</td>
</tr>
<tr>
<td><strong>Move cursor to start of line</strong></td>
<td>Home</td>
</tr>
<tr>
<td><strong>Find line containing a string</strong></td>
<td>Ctrl-s then fill in boxes</td>
</tr>
<tr>
<td><strong>Student Exercises</strong></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Split line at cursor</strong></td>
<td>Position cursor under character and press Enter</td>
</tr>
<tr>
<td><strong>Join lines</strong></td>
<td>Position cursor at end of first line and press alt-j</td>
</tr>
<tr>
<td><strong>Turn off C assist facility</strong></td>
<td>ctrl-i then type expand off and press Enter</td>
</tr>
<tr>
<td><strong>&quot;Global&quot; change</strong></td>
<td>ctrl-s then fill in boxes</td>
</tr>
<tr>
<td><strong>Highlight line(s)/text to edit</strong></td>
<td>Place cursor at starting point, hold shift and move cursor to highlight text or Place cursor at starting point and drag it with mouse</td>
</tr>
<tr>
<td><strong>Delete line(s) or text</strong></td>
<td>1. Highlight line(s)/text to be deleted; 2. Press Delete</td>
</tr>
<tr>
<td><strong>Copy line(s) or text</strong></td>
<td>1. Highlight line(s)/text to be copied; 2. Press ctrl-insert (or Alt-e then c); 3. Position cursor to place you want to insert after; 4. Press shift-insert (or Alt-e then p).</td>
</tr>
<tr>
<td><strong>Move line(s) or text</strong></td>
<td>1. Highlight line(s)/text to be moved; 2. Press shift-delete (or Alt-e then t); 3. Position cursor to place you want to move after; 4. Press shift-insert (or Alt-e then p).</td>
</tr>
<tr>
<td><strong>Duplicate line(s) or text</strong></td>
<td>1. Highlight line(s)/text to be duplicated; 2. Press ctrl-insert (or Alt-e then c); 3. Position cursor to place you want to duplicate after; 4. Press shift-insert (or Alt-e then p); 5. Repeat steps 3 and 5 as many times as desired.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Copy lines/text from another file</strong></td>
<td>1. Open file to copy by pressing ctrl-o and filling in box with file to open; use windows to find file to open;</td>
</tr>
<tr>
<td></td>
<td>2. Highlight line(s)/text to be copied;</td>
</tr>
<tr>
<td></td>
<td>3. Press ctrl-insert (or Alt-e then c);</td>
</tr>
<tr>
<td></td>
<td>4. Go back to original file (the one you want to copy to);</td>
</tr>
<tr>
<td></td>
<td>5. Position cursor to place you want to copy after;</td>
</tr>
<tr>
<td></td>
<td>6. Press shift-insert (or Alt-e then p).</td>
</tr>
<tr>
<td><strong>Make a copy of file you are editing</strong></td>
<td>F7 then fill in new file name</td>
</tr>
<tr>
<td><strong>Change screen font</strong></td>
<td>Alt-o then r then s and select &quot;Fonts&quot; tab with mouse</td>
</tr>
<tr>
<td><strong>Exit menus or dialogs without saving</strong></td>
<td>ESC</td>
</tr>
</tbody>
</table>
UNIX vi Editor Command Summary

Edit a new file
vi pgmname.sqc

Save Program and Return to System
:wq -or- ZZ

Save Program and Stay in Edit
:w

Cancel Changes & Return to System
:q!

Enter insert mode
i

Return to command mode
ESC

Add a blank line BEFORE current line
O

Add a blank line AFTER current line
o

Undo last command
u

Repeat last command
.

Turn on line numbers
:set nu

Move display UP one screen
Ctrl-b

Move display DOWN one screen
Ctrl-f

Move display UP 'n' lines
##Ctrl-u

Move display DOWN 'n' lines
##Ctrl-d

Move display LEFT
h -or- left arrow

Move display RIGHT
l -or- right arrow

Move cursor up one line
k -or- up arrow

Move cursor down one line
j -or- down arrow

Move cursor to end of line
$

Move cursor to end of line and enter insert mode
A

Move cursor to start of line
^$

Move to TOP of program
H

Split line at cursor insert mode and press enter
Position cursor under character in insert mode and press Enter
Join lines J
Place cursor at end of first line, then press J

Replace/Overtype to end of line  R

Delete word   dw

Delete line   dd

Delete from cursor to end of line  D

Copy line(s)  :startline#,endline# co targetline#

Move line(s)  :startline#,endline# m targetline#

Copy lines from another file and insert after cursor  :r filename

Make a copy of the file you are editing  :w filename

Search DOWN for next occurrence of text  /text

Search UP for next occurrence of text  ?text

Go to a particular line  :line# G

Delete character at cursor  x

Copy line to buffer  Y

Pull line from buffer  P

Change word  cw