Writing a Db2 COBOL Program

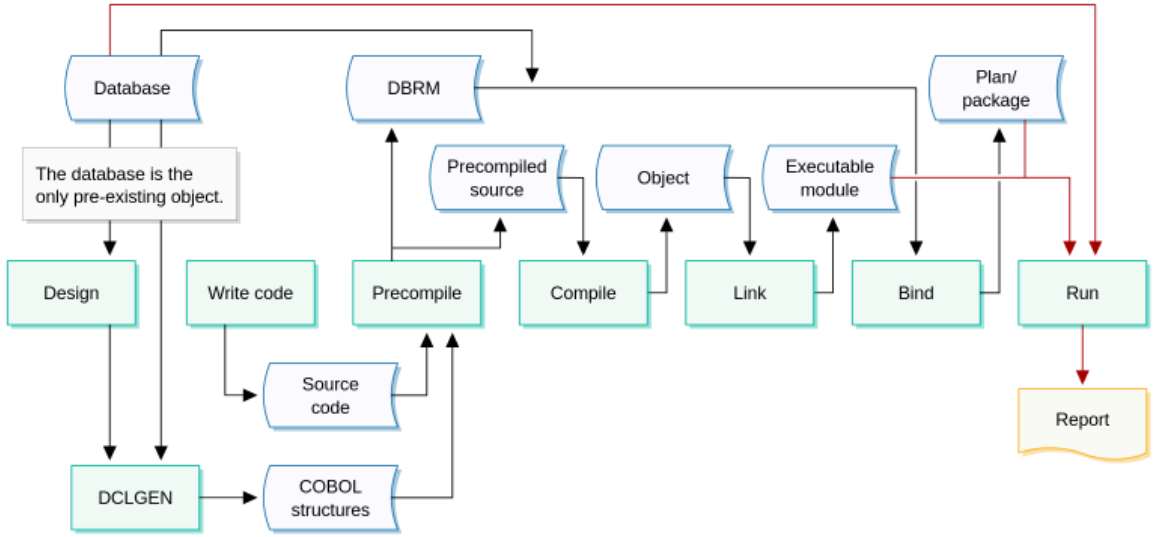
Objectives

Writing a Db2 COBOL Program

This module will follow the steps involved in creating a Db2 COBOL program and running it on the mainframe.

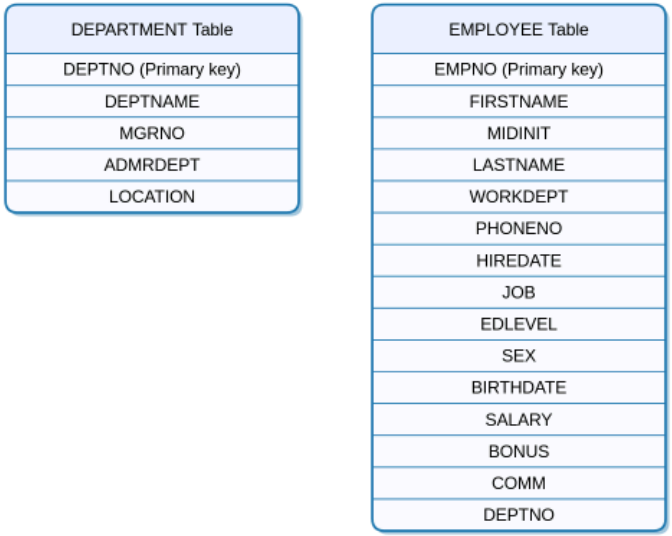
After completing this module you will be able to:

- Write and Build a Program That Issues Simple SQL Statements to Db2



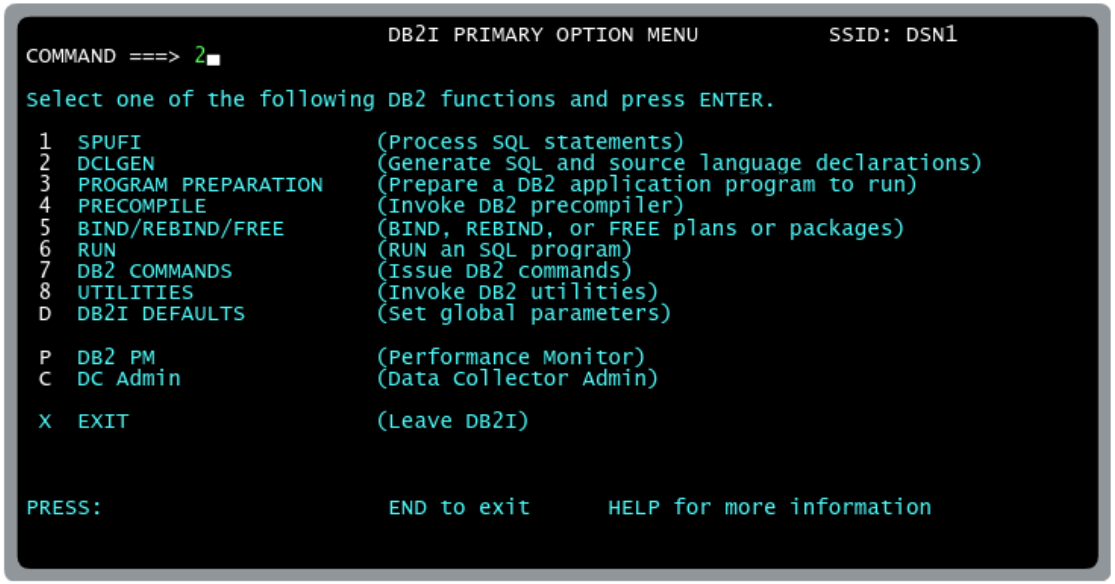
This module steps through the building and running of a Db2 report program in COBOL.

The TSO mainframe environment will be used to do this, but you could use a development environment such as Rational Developer for System z (IBM) or Visual COBOL (Micro Focus).



Take a look at the structure of the database you will be using. It is very simple and consists of the two tables shown. To write a program you must know the structure of these tables as well as the COBOL type that is equivalent to the columns you will use.

It is possible to work this out from the documentation, but the best way is to make Db2 give it to you using DCLGEN.

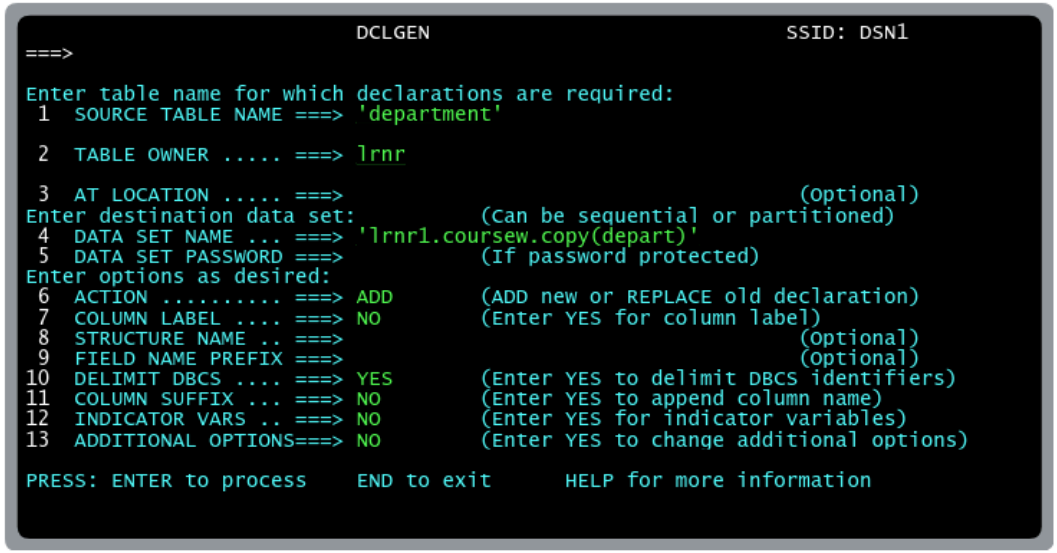


Step 1 of 3

You will produce the DCLGEN for the DEPARTMENT table first. You will then produce the DCLGEN for the EMPLOYEE table.

Here is the TSO Db2 menu.

Type **2** to select the DCLGEN option and **press** **Enter**.



Step 2 of 3

Now you will specify the table you require and the data set that will contain the generated structure. Note that because you previously set the Db2 language to COBOL, you must fully qualify the output data set or Db2 will try to append a COBOL qualifier to the data set name.

Type **'DEPARTMENT'** in the SOURCE TABLE NAME field.
Type **LRNR** in the TABLE OWNER field because you are using the schema or owner of LRNR.
Type **'LRNR1.COURSEW.COPY(DEPART)'** in the DATA SET NAME field to nominate a data set or member to contain the structure.

Press **Enter** when you have finished.

```
******************************************************************
* DCLGEN TABLE(LRNR.DEPARTMENT)                                  *
*        LIBRARY(LRNR1.COURSEW.COPY(DEPART))                     *
*        LANGUAGE(COBOL)                                         *
*        QUOTE                                                   *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
******************************************************************
     EXEC SQL DECLARE LRNR.DEPARTMENT TABLE
     ( DEPTNO                        CHAR(3) NOT NULL,
       DEPTNAME                      VARCHAR(29) NOT NULL,
       MGRNO                         CHAR(6),
       ADMRDEPT                      CHAR(3) NOT NULL,
       LOCATION                      CHAR(16)
     ) END-EXEC.
******************************************************************
* COBOL DECLARATION FOR TABLE LRNR.DEPARTMENT                    *
******************************************************************
 01  DCLDEPARTMENT.
     10 DEPTNO               PIC X(3).
     10 DEPTNAME.
        49 DEPTNAME-LEN      PIC S9(4) USAGE COMP.
        49 DEPTNAME-TEXT     PIC X(29).
     10 MGRNO                PIC X(6).
     10 ADMRDEPT             PIC X(3).
     10 LOCATION             PIC X(16).
******************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5       *
******************************************************************
```

Here is the generated structure that contains comments specifying the table, library, and language. More importantly, it contains an SQL DECLARE statement and a COBOL record structure.

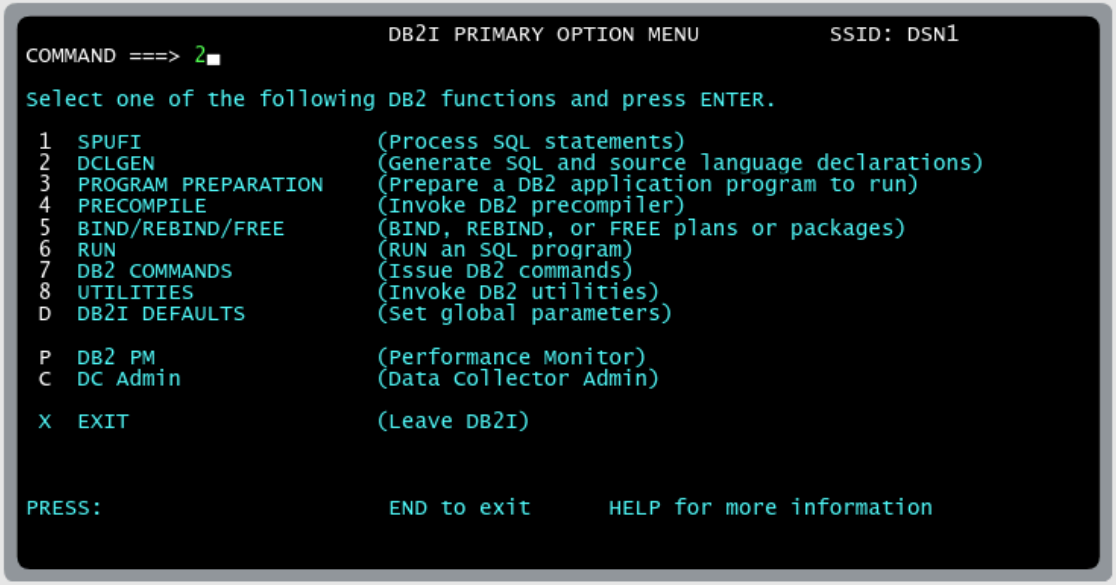
These will be included in this structure in your COBOL program for processing by the Db2 precompiler.

Note: You will not copy this as the COBOL compile will not understand the EXEC SQL statements.

Question 1 of 2

You will now generate the DCLGEN for the EMPLOYEE TABLE.

Type the correct option and **press** **Enter**.

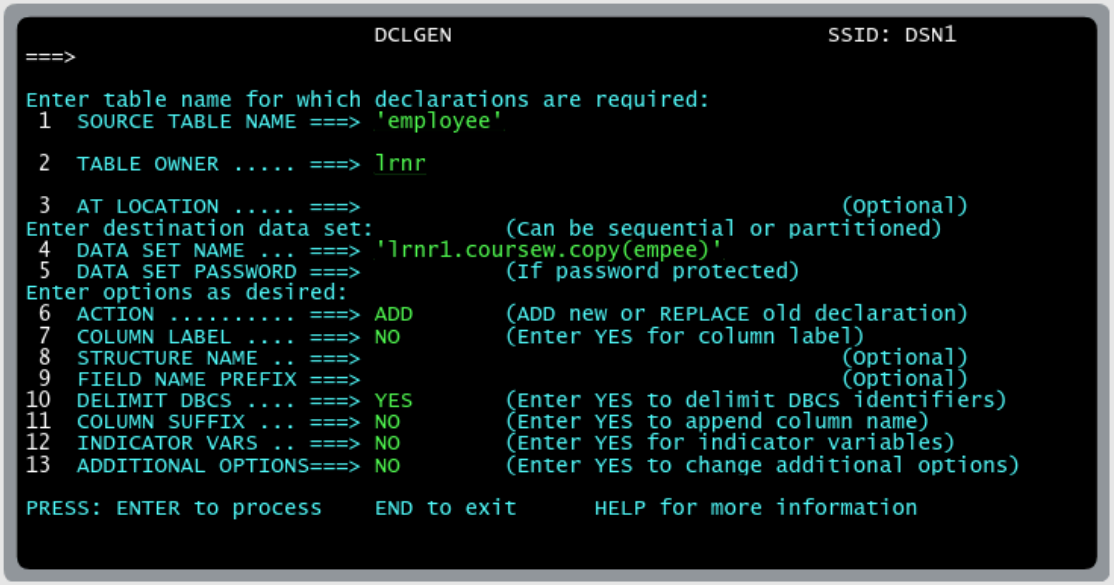


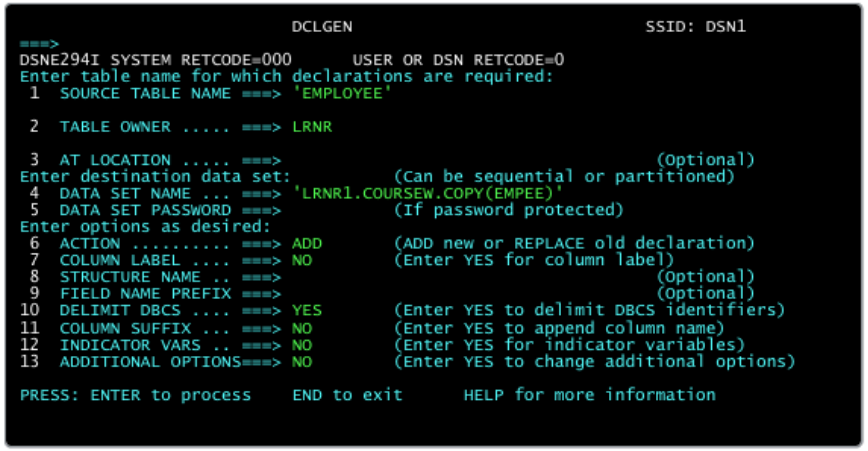
Question 2 of 2

You now need to supply the following information to the DCLGEN:

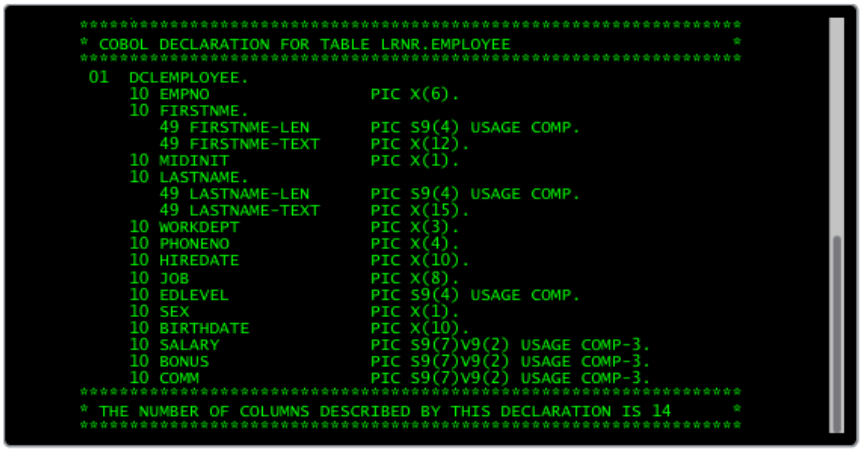
- The source table is EMPLOYEE. This needs to be qualified.
- The table owner is LRNR
- The destination data set is LRNR1.COURSEW.COPY(EMPEE). This also needs to be qualified.

Type the correct options in the relevant fields and **press** **Enter** when you have finished.





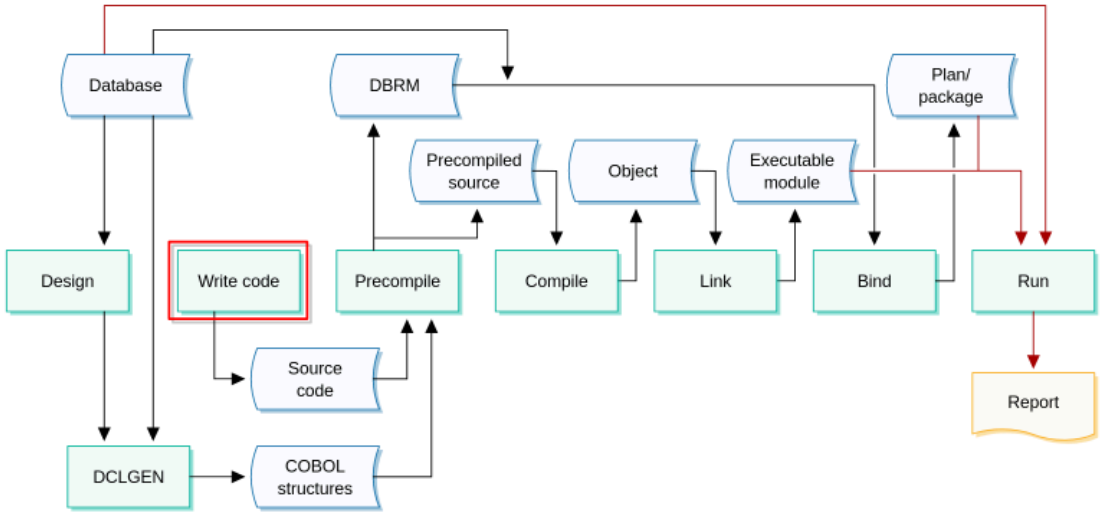
You have successfully generated the DCLGEN structure. It will be in the data set 'LRNR1.COURSEW.COPY(EMPEE)'.



```
************************************************************************
* COBOL DECLARATION FOR TABLE LRNR.EMPLOYEE                            *
************************************************************************
 01  DCLEMPLOYEE.
     10 EMPNO                PIC X(6).
     10 FIRSTNME.
        49 FIRSTNME-LEN      PIC S9(4) USAGE COMP.
        49 FIRSTNME-TEXT     PIC X(12).
     10 MIDINIT              PIC X(1).
     10 LASTNAME.
        49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
        49 LASTNAME-TEXT     PIC X(15).
     10 WORKDEPT             PIC X(3).
     10 PHONENO              PIC X(4).
     10 HIREDATE             PIC X(10).
     10 JOB                  PIC X(8).
     10 EDLEVEL              PIC S9(4) USAGE COMP.
     10 SEX                  PIC X(1).
     10 BIRTHDATE            PIC X(10).
     10 SALARY               PIC S9(7)V9(2) USAGE COMP-3.
     10 BONUS                PIC S9(7)V9(2) USAGE COMP-3.
     10 COMM                 PIC S9(7)V9(2) USAGE COMP-3.
************************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 14            *
************************************************************************
```

The EMPLOYEE table contains more columns than DEPARTMENT. These are all defined here, although you may not reference them all.

Scroll the window to view all the code.



The process of writing the COBOL code and the Db2 SQL is often easier when addressed separately. It is particularly useful to write and test the Db2 SQL in an interactive environment and then copy it into a COBOL program.

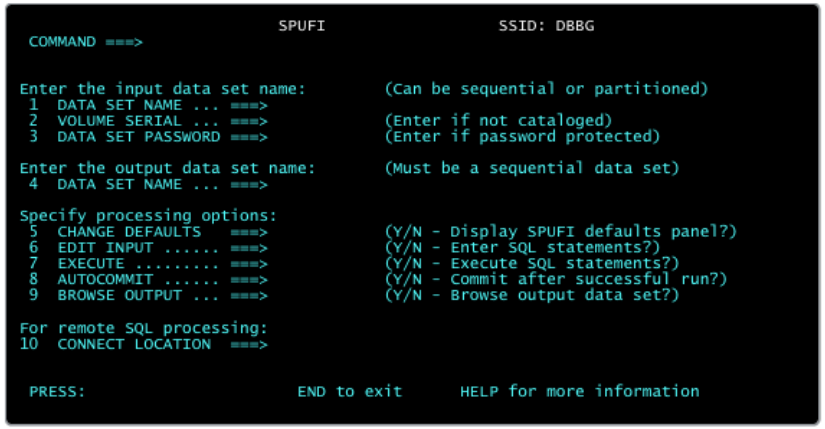
For this exercise you will be writing a COBOL program to discover how many department and employee rows are in the database. You also want to display the results, first by ID and then by department.

```
       PROCEDURE DIVISION.

       0000-MAINLINE.

                 PERFORM 200-OPEN-FILES.

                 PERFORM 300-GET-DATA.

                 PERFORM 400-PROCESS-FILE.

                 PERFORM 900-CLOSE-FILES.

                 STOP RUN.

       200-OPEN-FILES.

           OPEN    OUTPUT  REPORT-FILE.

       300-GET-DATA.
      * DB2 CODE WILL GO HERE
           MOVE '5'                    TO WS-NUMBER-OF-DEPTS.
           MOVE '10'                   TO WS-NUMBER-OF-EMPLOYEES.
           MOVE '111'                  TO WS-DEPARTMENT-NO.
           MOVE 'EXAMPLE'              TO WS-DEPARTMENT-NAME.
```

As you are only looking at the Db2 code, you will start with a COBOL program that already has most of what you need.

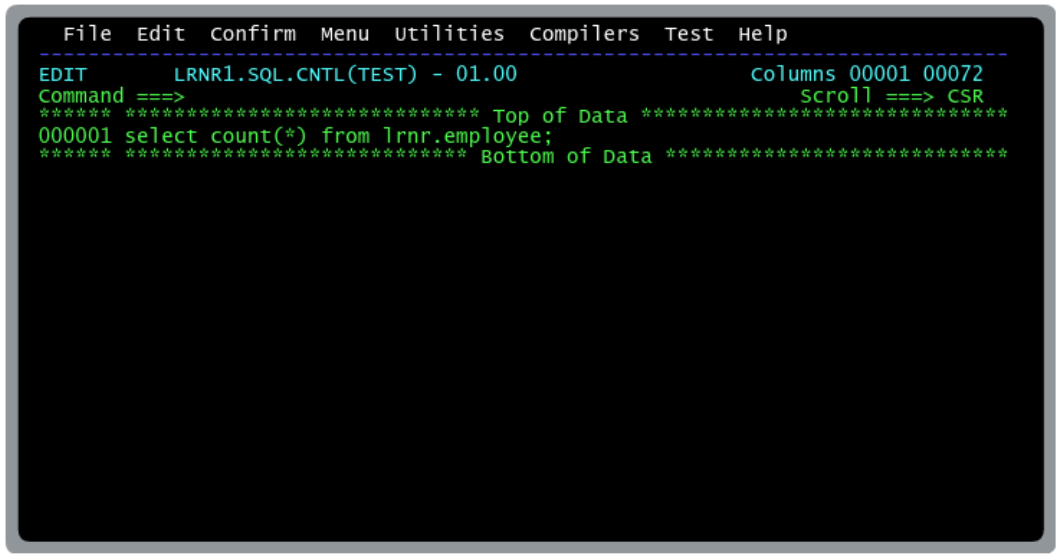
This COBOL program places literal values in the report where you will soon be retrieving data from Db2. It has also been compiled and test-run successfully.

Scroll the window to view all the code.



It is a good idea to test all SQL statements separately using a utility such as SPUFI or IBM Data Studio. COBOL programs may not detect an error if the SQL is incorrect. Testing SQL also shows the format and number of results returned, making it easier to process these results.

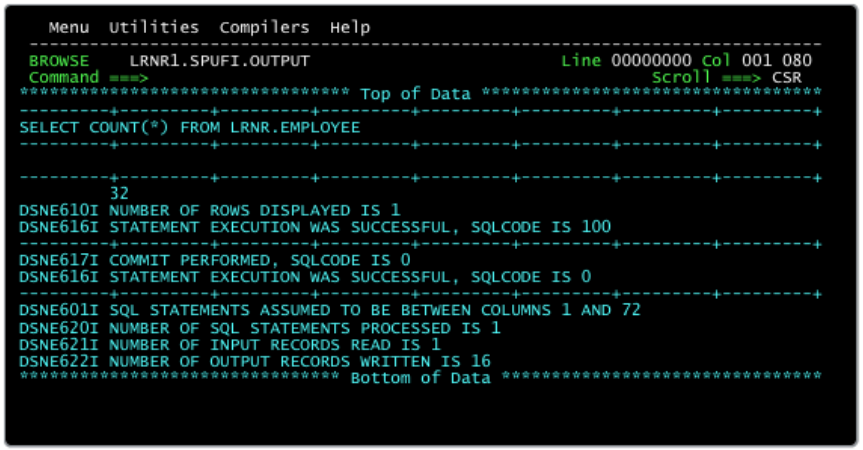
All programs must check the SQL return code, and handle any errors found.



Step 1 of 2

You can also use SPUFI to test your SQL. Although you use the LRNR qualifier in SPUFI, you will remove it in your program. The qualifier will then be set in the OWNER parameter of the BIND. This allows the one program to be bound and run against different owners or schemas.

Type the SQL to count the number of rows in LRNR.EMPLOYEE and **press** **PF3**.



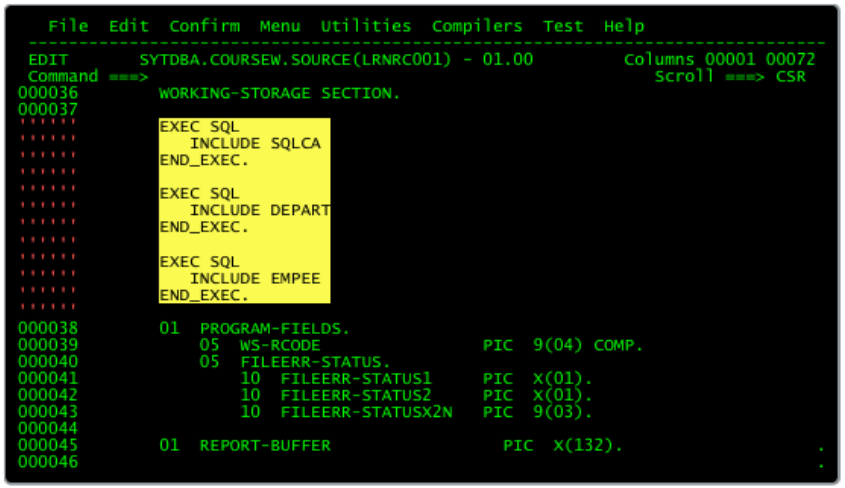
You can see that your SQL has completed successfully, returning a single value of 32.

Not only can you use Db2 tools such as SPUFI and IBM Data Studio to prototype your SQL statements, but it is possible to use properly configured desktop tools such as MS ACCESS.

```
SELECT DEPTNO, DEPTNAME  FROM DEPARTMENT
WHERE DEPTNO = (SELECT MIN(DEPTNO) FROM DEPARTMENT)
```

```
SELECT DEPTNO, DEPTNAME  FROM DEPARTMENT
WHERE DEPTNO = (SELECT MIN(DEPTNO) FROM DEPARTMENT)

---------+---------+---------+---------+---------+---------+---------+---------+
DEPTNO  DEPTNAME
---------+---------+---------+---------+---------+---------+---------+---------+
A00     SPIFFY COMPUTER SERVICE DIV.
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The next SQL statement we require is one that lists the DEPTNO and DEPTNAME column values from the lowest numbered DEPTNO in the DEPARTMENT table. This is shown here.



As you can see here, SQL INCLUDE statements have been used to do the following:

- Provide the declaration of the SQLCA and
- Include the declarations from DCLGEN (DEPART and EMPEE)

```
01  PROGRAM-FIELDS.
    05  WS-ZERO                 PIC 9 VALUE 0.
    05 WS-NUMBER-OF-DEPTS       PIC S9(4) USAGE COMP.
    05 WS-NUMBER-OF-EMPLOYEES   PIC S9(4) USAGE COMP.

01  REPORT-BUFFER                PIC  X(132).

01  WS-RESULT-FIELDS.
    05 WS-TOTALS-RECORD.
       10 FILLER                          PIC X(30)
             VALUE 'NUMBER OF DEPARTMENTS'.
       10 WS-NOD-REP                      PIC 9(6).
       10 FILLER                          PIC X(27)
             VALUE '   NUMBER OF EMPLOYEES'.

       10 WS-NOE-REP                   PIC 9(6).

    05 WS-DEPARTMENT-RECORD.
     10 FILLER                        PIC X(20)
                              VALUE 'FIRST DEPARTMENT:'.
     10 WS-DEPARTMENT-NO                 PIC 9(6).
     10 FILLER                        PIC X(3) VALUE SPACES.
     10 WS-DEPARTMENT-NAME            PIC x(40).
     10 FILLER                        PIC X(66) VALUE SPACES.
```

The SQLCA and DCLGEN declarations are now in place, and the following highlighted areas have been added:

- Host variable declarations. These are required because the SQL statements that will be executed will produce data
- Simple error handling that will display SQLCODE and SQLSTATE values if there is an SQL error, and then branch to code that will result in a program dump being taken.

Scroll the window to view all the code.

```
      * DB2 CODE WILL GO HERE
           EXEC SQL
             WHENEVER SQLERROR GOTO 390-SQL-ERROR
           END-EXEC.

           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-DEPTS FROM DEPARTMENT
           END-EXEC.

           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-EMPLOYEES FROM EMPLOYEE
           END-EXEC.

           MOVE WS-NUMBER-OF-DEPTS     TO WS-NOD-REP.
           MOVE WS-NUMBER-OF-EMPLOYEES TO WS-NOE-REP.
           MOVE '111'                  TO WS-DEPARTMENT-NO.
           MOVE 'EXAMPLE'              TO WS-DEPARTMENT-NAME.
       390-SQL-ERROR.
           DISPLAY 'SQLCODE ' SQLCODE.
           DISPLAY 'SQLSTATE' SQLSTATE.
           PERFORM  990-ABEND.
```

Step 2 of 2

```
SELECT COUNT(*) FROM DEPARTMENT
SELECT COUNT(*) FROM EMPLOYEE
```

You will need to modify these statements to indicate that the results are to be stored in the WS-NUMBER-OF-DEPTS and WS-NUMBER-OF-EMPLOYEES host variables. You may have noticed that code has already been added to move these two values to the report display fields. **Type** the code into the blank fields and **press** **Enter** when you have finished.

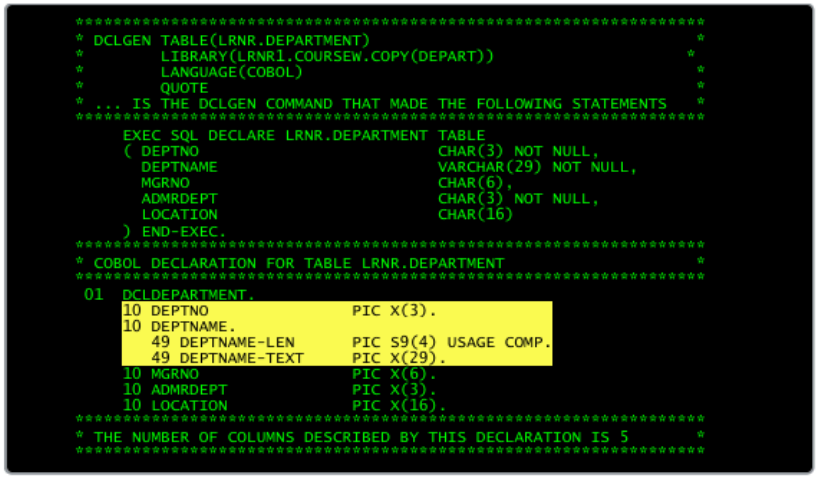
That is incorrect

```
           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-DEPTS FROM DEPARTMENT
           END-EXEC.

           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-EMPLOYEES FROM EMPLOYEE
           END-EXEC.
....

           MOVE WS-NUMBER-OF-DEPTS     TO WS-NOD-REP.
           MOVE WS-NUMBER-OF-EMPLOYEES TO WS-NOE-REP.
           MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
           MOVE DEPTNAME-TEXT          TO WS-DEPARTMENT-NAME.

      390-SQL-ERROR.
....
```

We are now going to add the last section of SQL code that deals with obtaining the DEPTNO and DEPTNAME values from the department with the lowest code. The code looks like this:

```
SELECT DEPTNO, DEPTNAME FROM DEPARTMENT
WHERE DEPTNO = (SELECT MIN(DEPTNO) FROM DEPARTMENT)
```

You must retrieve the values into the host variables provided by the DCLGEN.



```
************************************************************************
* DCLGEN TABLE(LRNR.DEPARTMENT)                                        *
*        LIBRARY(LRNR1.COURSEW.COPY(DEPART))                           *
*        LANGUAGE(COBOL)                                               *
*        QUOTE                                                         *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS         *
************************************************************************
     EXEC SQL DECLARE LRNR.DEPARTMENT TABLE
     ( DEPTNO                        CHAR(3) NOT NULL,
       DEPTNAME                      VARCHAR(29) NOT NULL,
       MGRNO                         CHAR(6),
       ADMRDEPT                      CHAR(3) NOT NULL,
       LOCATION                      CHAR(16)
     ) END-EXEC.
************************************************************************
* COBOL DECLARATION FOR TABLE LRNR.DEPARTMENT                          *
************************************************************************
 01  DCLDEPARTMENT.
     10 DEPTNO                PIC X(3).
     10 DEPTNAME.
        49 DEPTNAME-LEN       PIC S9(4) USAGE COMP.
        49 DEPTNAME-TEXT      PIC X(29).
     10 MGRNO                 PIC X(6).
     10 ADMRDEPT              PIC X(3).
     10 LOCATION              PIC X(16).
************************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5             *
************************************************************************
```

The department number is straightforward with the variable DEPTNO defined; however, because DEPTNAME is defined in Db2 as a VARCHAR, Db2 will return a structure consisting of the length and the value of the column. Your COBOL program could use the fields independently or use the structure, but it must cater for the length field. You could code either of the following:

```
INTO :DEPTNAME-LEN :DEPTNAME-TEXT
or
INTO :DEPTNAME
```

When you have successfully retrieved the value, you can use the DEPTNAME-TEXT field in your COBOL program.

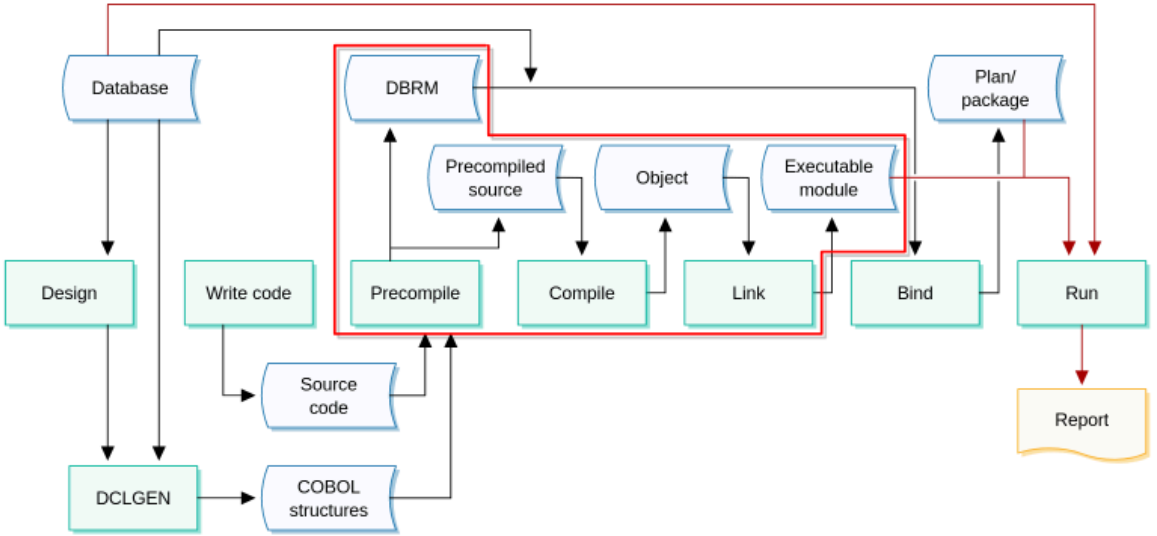
```
           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-DEPTS FROM DEPARTMENT
           END-EXEC.

           EXEC SQL
                SELECT COUNT(*) INTO :WS-NUMBER-OF-EMPLOYEES FROM EMPLOYEE
           END-EXEC.

           EXEC SQL
                SELECT DEPTNO, DEPTNAME
                INTO :DEPTNO, :DEPTNAME FROM DEPARTMENT
                WHERE DEPTNO = (SELECT MIN(DEPTNO) FROM DEPARTMENT)
           END-EXEC.

           MOVE WS-NUMBER-OF-DEPTS     TO WS-NOD-REP.
           MOVE WS-NUMBER-OF-EMPLOYEES TO WS-NOE-REP.
           MOVE DEPTNO                 TO WS-DEPARTMENT-NO.
           MOVE DEPTNAME-TEXT          TO WS-DEPARTMENT-NAME.

       390-SQL-ERROR.                                                    .
```

You can see in the added code that values are being saved to the host variable DEPTNO and host structure DEPTNAME. At the bottom of this screen you can see that the DEPTNAME-TEXT value is being moved to the report display field.

```
           INCLUDE DEPART
       END-EXEC.

       EXEC SQL
           INCLUDE EMPEE
       END-EXEC.

       EXEC SQL
           INCLUDE SQLCA
       END-EXEC.

   01  PROGRAM-FIELDS.
       05  WS-ZERO                 PIC 9 VALUE 0.
       05  WS-RCODE                PIC  9(04) COMP.
       05  FILEERR-STATUS.
           10  FILEERR-STATUS1     PIC  X(01).
           10  FILEERR-STATUS2     PIC  X(01).
           10  FILEERR-STATUSX2N   PIC  9(03).
       10 WS-NUMBER-OF-DEPTS       PIC S9(4) USAGE COMP.
       10 WS-NUMBER-OF-EMPLOYEES   PIC S9(4) USAGE COMP.

   01  REPORT-BUFFER                 PIC  X(132).

   01  WS-RESULT-FIELDS.
       05 WS-TOTALS-RECORD.
```

Here is your completely coded program. Take some time to study it before continuing.

Scroll the window to view all the code.



You have your COBOL source and now you need to produce an executable program.

The extra step required for a Db2 program is the precompile. The latest versions of the IBM COBOL compiler integrate this step, but logically it is the first step in the process.

The precompile checks and converts all code enclosed in an EXEC SQL ... END-EXEC clause to COBOL code. It also produces a DBRM that contains all the SQL for use in the bind step.

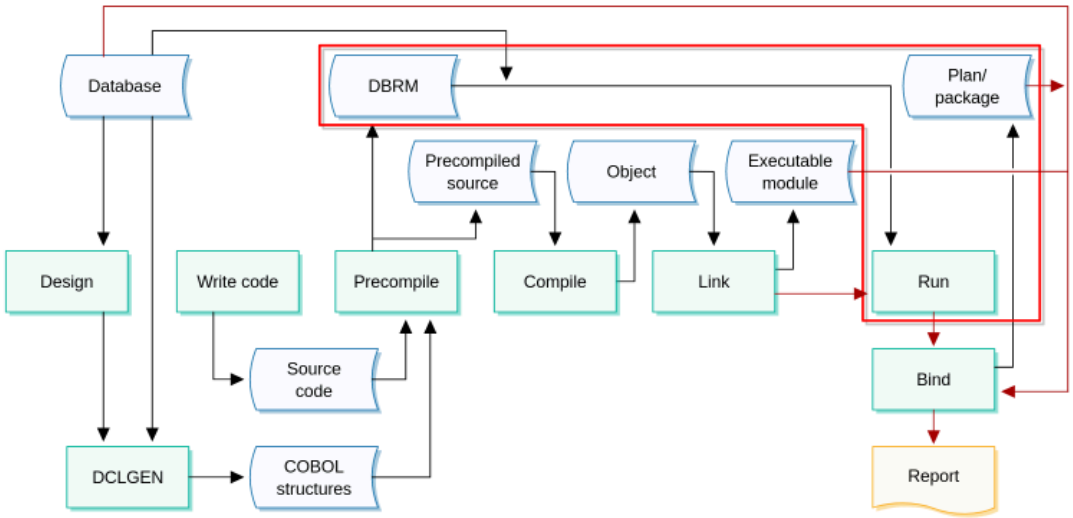
```
//LRNRCB  JOB (9999,0000),'COMPILE DB2 BATCH',
//         TIME=1,
//         CLASS=A,
//         MSGCLASS=O,
//         NOTIFY=&SYSUID
//*
//*********************************************************************
//*  DB2PC     DSNHPC     DB2   PRE-COMPILE                            *
//*  COB       IGYCRCTL   COBOL II COMPILE                           *
//*  LKED      IEWL       LINKEDIT                                   *
//*********************************************************************
//*
//DB2PC   EXEC PGM=DSNHPC,REGION=2048K,
//         PARM=('APOST, DATE(ISO),HOST(IBMCOB),SOURCE,TIME(ISO),XREF')
//DBRMLIB  DD  DSN=LRNR.COURSEW.DBRMLIB(LNRCD002),DISP=SHR
//SYSIN    DD  DSN=LRNR.COURSEW.SOURCE(LNRCD002),DISP=SHR
//SYSLIB   DD  DSN=LRNR.COURSEW.COPY,DISP=SHR
//SYSCIN   DD  DSN=&&DSNHOUT,DISP=(NEW,PASS),
//             UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//*
```

Many installations will have facilities such as JCL procedures, workbenches, source management systems, and job management systems that enable and control the process of turning a source file into an executable and running it.

In the background, however, they will be performing the processes you will see in the next few pages.

Shown here is JCL to precompile, compile, and link for your program. Note the DBRM produced by the precompile.

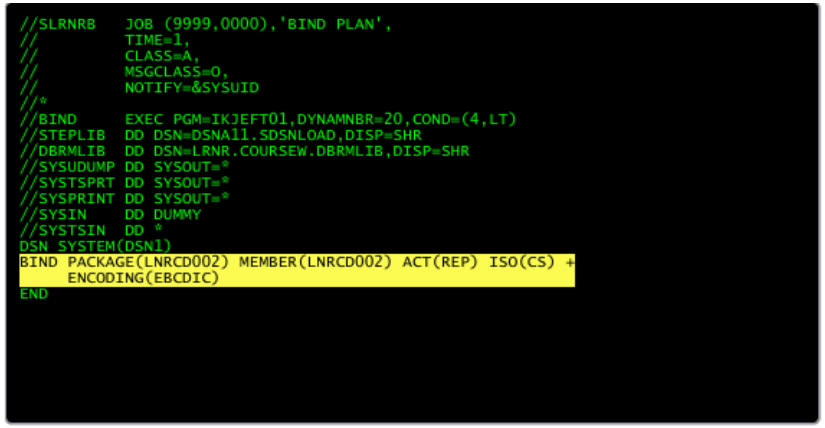
Scroll the window to view all the code.



After precompiling, compiling, and linking, you will have an executable program. Now you must tell the Db2 system about the SQL you are going to run, allow the system to determine the optimal access path, and verify the existence of all the objects referenced and your security access to them.

This is done in the bind step which references the DBRM and the target Db2 system.

Note: The one program can be bound against many Db2 systems.



```
//SLRNRB   JOB (9999,0000),'BIND PLAN',
//         TIME=1,
//         CLASS=A,
//         MSGCLASS=O,
//         NOTIFY=&SYSUID
//*
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DSNA11.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=LRNR.COURSEW.DBRMLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSTSIN  DD *
DSN SYSTEM(DSN1)
BIND PACKAGE(LNRCD002) MEMBER(LNRCD002) ACT(REP) ISO(CS) +
     ENCODING(EBCDIC)
END
```

The JCL example shown here is being used to build application package LNRCD002. The parameters of the BIND include:

- ACT which determines the action if a package of the same name already exists. In this example this new version will replace it.
- ISO which defines how isolated the application is in relation to other running applications. In this example, cursor stability (CS) indicates that application does not read a row that another process is using, until that process releases it.
- ENCODING indicates in this example that the encoding scheme for the package is EBCDIC.

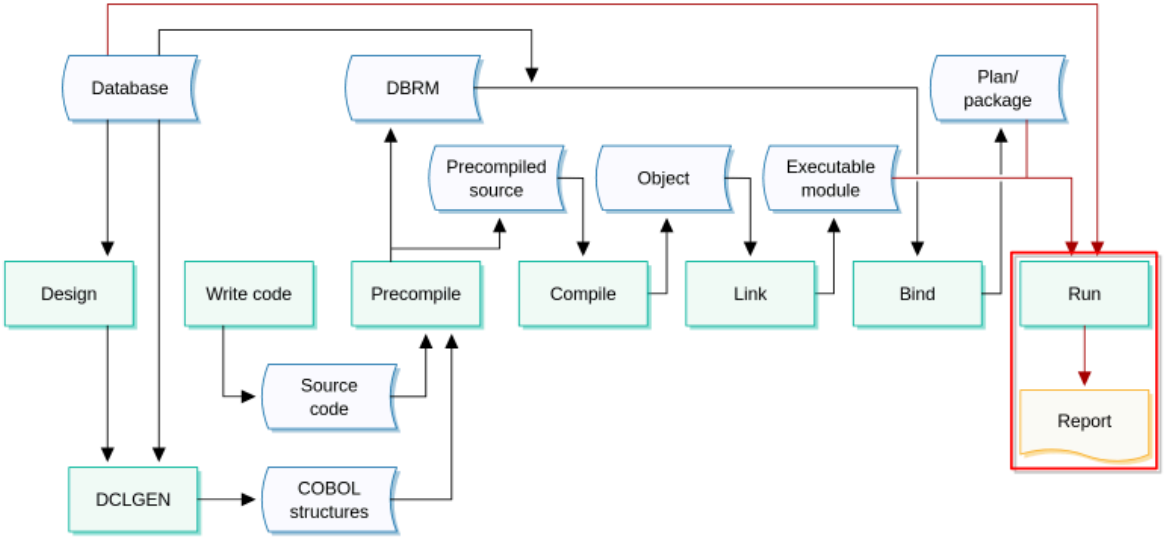
Many other parameters can be used for the bind process. You should reference the relevant IBM manual if you require more information.

```
//SLRNRB   JOB (9999,0000),'BIND PLAN',
//         TIME=1,
//         CLASS=A,
//         MSGCLASS=O,
//         NOTIFY=&SYSUID
//*
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DSNA11.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=LRNR.COURSEW.DBRMLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSTSIN  DD *
DSN SYSTEM(DSN1)
BIND PLAN(LRNR) PKLIST(LNRCD002.*) ACT(REP) ISO(CS) OWNER(LRNR)
END
```

Generally the Plan only needs to be bound once and related to specific packages. This should be run with care as all execution of the Plan is stopped while being bound.

Note: The OWNER parameter is used to set the owner or schema to any SQL where it is not explicitly coded in the program.

It is preferable not to code explicit owner within the program to allow the program to run in multiple environments and simplify schema changes.

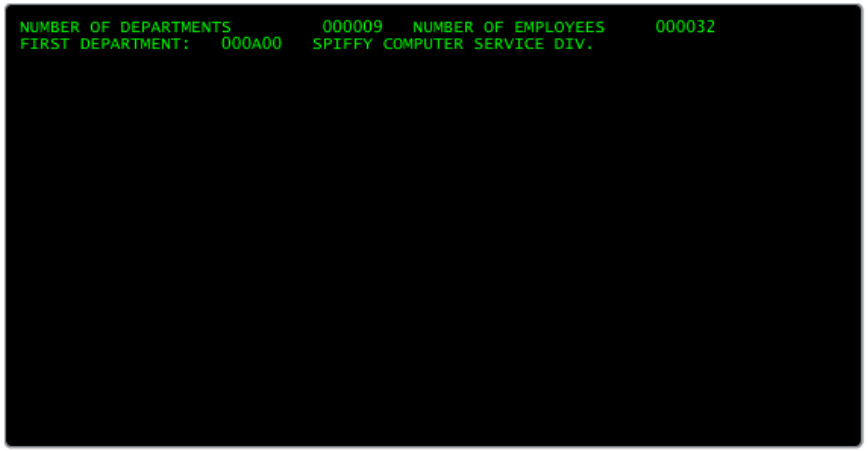


You can now run your program.

```
//LRNRDR  JOB (9999,0000),'RUNDB2 ',
//         TIME=1,
//         CLASS=A,
//         MSGCLASS=O,
//         NOTIFY=&SYSUID
//*
//RUNPROG  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN.SDSNLOAD,DISP=SHR
//REPT1    DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(DSN1)
RUN  PROGRAM(LNRCD002)  -
     PLAN(LRNR)     -
     LIBRARY('LRNR.COURSEW.LOADLIB')
END
/*
//*
```

Batch Db2 JCL is very simple as the program is run under the control of Db2.

The JCL tells the DSN processor about the Db2 subsystem, the program, the plan, and where to find the program (library). The JCL also allocates any non-Db2 data sets and inputs.



Running the program will produce the report shown here.