**Creating a Database using SQL**

# Step 1:

On the ISPF menu, enter P for products.

A screen shot of a smart phone

Description automatically generated

Enter D to access DB2 Subsystem.A screenshot of a cell phone

Description automatically generated

Then on the DB2 menu, enter 2 to execute SQL statements.

A screenshot of a cell phone

Description automatically generated

# Step 2:

In the Execute SQL Statement menu, enter 1 to edit/run SQL statements.

A screenshot of a cell phone

Description automatically generated

The next screen will allow us to enter SQL commands. Enter the following command below to create a new database. ‘LIT0002’ is the name of our database we want to create and ‘UNILITSG’ is the name of our storage group. A storage group is a collection of all the tablespaces within a database. Basically, the database is saved in a storage group.

A screen shot of a computer

Description automatically generated

Once we have our SQL statements entered, we press F3 to exit, which brings us to this screen below.

Enter 1 and hit the start key to execute our SQL statement.

A screenshot of a cell phone

Description automatically generated

Once the SQL is executed, red text will appear to tell us it has successfully executed the statements.

A picture containing black, screen, room

Description automatically generated

# Step 3:

Confirm execution. Go back to the DB2 menu and enter 1 for DB2 system catalog.

A screenshot of a cell phone

Description automatically generated

Enter D on the primary command line and enter our ID for Owner field at the bottom in order to view databases belonging to the owner ID.

A screenshot of a cell phone

Description automatically generated

In the next screen, we will see a list of databases belonging to our owner ID. As we can see below, our ‘PROJECT1’ database is listed below confirming that the SQL statements worked.

A screen shot of a monitor

Description automatically generated

**Creating Tables Using SPUFI**

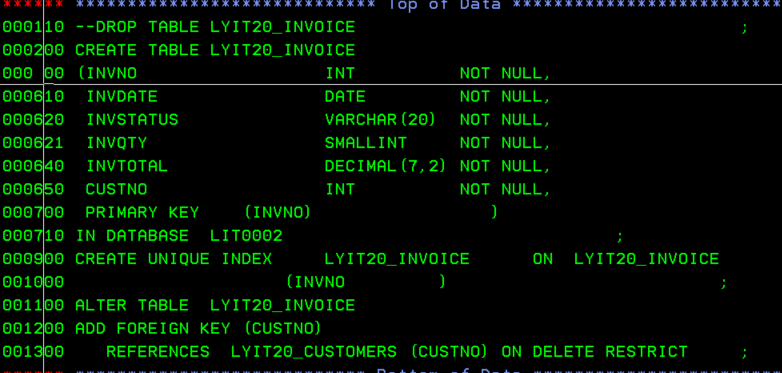
SPUFI (SQL Processing Using File Input) is a tool that allows us to execute files containing several SQL statements at once.

# Step 1:

The first thing we need to do is create our files containing SQL to create a customers table and a invoice table. The file below contains SQL to create the customer table using the Create command. We then initialise the columns that will be placed in the table. The data types used in DB2 are: Char, Varchar, Decimal, Integer, Float, Date and BLOB (Binary Large Object). As seen below, we use Varchars for our data types for the customer data. The difference between Char and Varchar is that Char is fixed length and Varchar is variable length meaning Char always uses the same amount of storage space per entry, while Varchar only uses the amount necessary to store the actual text. However, if you know the exact length of Char, then Char will execute with a bit more speed. In this example, we will be using Varchar. The next thing we do is select our primary key. In this instance, it will be CUSTNO as seen below.

A picture containing monitor, sitting, clock, screen

Description automatically generated

The same process applies for the Invoice table except in this instance, we will be using a foreign key in order to connect our two tables, customers and invoice. Our primary key for Invoice is INVNO and our foreign key will be CUSTNO.

# Step 2:

The next step is to create our load files which will contain our customer and invoice data to load into our tables. We will be using the INSERT command to insert the data into our tables.

A picture containing sitting, green, black, screen

Description automatically generatedA picture containing green, phone

Description automatically generated

# Step 3:

To access SPUFI we will need to go to the DB2 Subsystem as seen below and enter I for DB2 products. Press enter to continue.

A screenshot of a cell phone

Description automatically generated

Then enter 1 for SPUFI.

A screenshot of a cell phone

Description automatically generated

In the next screen, we will enter our data set name for the file we want to execute. Since we want to create the customers table, we enter the DSN for the file which is CRCUST. Next for the output dataset name we enter RESULTS and enter yes for the processing options as seen below. Hit enter to continue.

A screenshot of a cell phone

Description automatically generated

The next screen is warning screen, we ignore this screen by hitting enter to continue.

A screenshot of a cell phone screen with text

Description automatically generated

The next screen is SPUFI DEFAULTS which will be automatically filled out. We will hit enter once again to continue.

A screenshot of a cell phone

Description automatically generated

The next screen is a quick edit screen to make last minute changes to out CRCUST file. Since we don’t need to make changes, we hit F3 to exit and continue the execution of the file.

A picture containing monitor, sitting, clock, screen

Description automatically generated

This brings us back to the first SPUFI screen. We hit enter one more time to execute the SQL.

A screenshot of a cell phone

Description automatically generated

The next and final screen is the results of our execution. As shown below, we can see that our SQL was run successfully when the SQLCODE = 0. We have created our Customers table and the same process applies to creating our Invoice table.

A screenshot of a computer

Description automatically generated

# Step 4:

In this step, we will load in our data into the Customers and Invoice tables. This is the same process as in step 3 except the SQL statements will be different. We will be using the INSERT command to load the data into a table.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA picture containing sitting, green, black, screen

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

If we want to check if the data has been loaded into our tables successfully, we can go back to the DB2 menu, enter 2 to use the interactive SQL.

A screenshot of a cell phone

Description automatically generated

Enter 1 to run SQL statements.

A screenshot of a cell phone

Description automatically generated

Enter the SQL statement below to view all data from the Customers table

A clock sitting in the dark

Description automatically generated

A picture containing monitor, black, sitting, screen

Description automatically generated

Once we have our tables created and populated with data, we can move onto the next stage which is putting together a Cobol program using embedded SQL to access our data in the tables and produce an output report.

**Run DCLGEN Utility**

When we develop a Cobol program that gets data from a DB2 table, we include a description of rows in the table called a host structure. DB2 can develop the table structures for us from the data definitions used in the database. This is done by a utility that comes with DB2 called DCLGEN, which stands for Declarations Generator. This makes the life easier for the programmer as coding the table structures can be quite tedious.

# Step 1:

On the DB2 Subsystem menu, Enter I for DB2 products and hit enter.

A screenshot of a cell phone

Description automatically generated

Then enter 2 for DCLGEN and hit enter.

A screenshot of a cell phone

Description automatically generated

Next, we enter our table which will be LYIT20\_CUSTOMERS in the source table name field and put our ID for table owner. Then in the data set name field, we put where we want to store the generated Cobol source code. Hit enter to run.

A screenshot of a cell phone

Description automatically generated

If successful we will get a message screen saying member added.

A close up of a screen

Description automatically generated

# Step 2:

Check our generated Cobol source code.

A picture containing sitting, phone

Description automatically generated

The same process applies for the INVOICE table.

**Establish a DB2 Environment**

The next thing we need to do is develop our Cobol program and the JCL (Job Control Language). In order to connect our Cobol program and Db2, we will need to create a Pre-compile, Compile, Bind and Run JCLs. You can create four individual JCL programs or just one JCL containing the four steps. In this tutorial, we will be using just one JCL containing the four steps. So far on our Zeus system, we have two partitioned datasets, DB2 and DB2.SRCINC. DB2 contains our SQL files to create tables and populate the tables. DB2.SRCINC contains our DCLGEN Cobol source code files containing the table structures for our tables in Cobol source code to use in our Cobol programs. We need to create four more partitioned datasets: 1. DB2.DBRMLIB (Holds a Db2 load member) 2. DB2.IBMCOB (Holds our Cobol programs) 3. DB2.JCL (Holds our JCLs 4. DB2.LOAD (Holds our cobol load members). We should now have the following in our DB2 qualifier environment as shown below:

A screenshot of a cell phone

Description automatically generated

**Create the JCL**

JCL stands for Job Control Language. It is a scripting Language which is used for multiple purposes and one of the most important features of JCL is to run COBOL programs. The JCL compiles and executes Cobol programs.

For running COBOL DB2 program, specialised IBM utility is used in the JCL and program; DB2 region and required parameters are passed as input to the utility.

The below steps are followed in running a COBOL-DB2 program. **NOTE**: The following 4 steps are used in a single JCL. Our JCL should contain a Job Card at the top followed by the 4 steps below:

1. The first step is to run the precompile on the source program. This produces two output files. The first is a modified source program in which each of the SQL statements has been translated into the Cobol statements that invoke the appropriate DB2 interface functions. Although the precompile leaves the original SQL statement in the source program, it converts them to comments so they will be ignored by the Cobol compiler. The second file produced by the precompile is a database request module, or DBRM. It contains information about how your program will use DB2 and will be used as input in a later step of this development process. This module is stored in DB2.DBRMLIB.

A screen shot of a smart phone

Description automatically generated

1. After the precompile is finished, the Cobol compiler compiles the modified source program into an object module. Then, the linkage editor links the object module with other required modules including DB2 interface modules. This produces a load module which is stored in DB2.LOAD.

A picture containing sitting, green, black, table

Description automatically generated

1. Before the load module can be run/executed, DB2 must bind the program. This bind procedure uses the DBRM that was created by the precompile step to check all the DB2 functions used in the program to make sure they are valid and that we are authorized to perform them. In addition, this procedure selects the most efficient way for DB2 to implement the functions your program requests.

A picture containing sitting, green, screen, black

Description automatically generated

1. Finally, the load module is run/executed which gives us our output.

A picture containing monitor, sitting, green, table

Description automatically generated

Here is the full JCL below:

//LIT0002 JOB UNIVER,CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID

//\*

//JOBLIB DD DISP=SHR,DSN=DB2.V10R1.SDSNLOAD

//DCLGEN EXEC PGM=IKJEFT01

//SYSTSPRT DD SYSOUT=\*

//SYSTSIN DD \*

DSN SYSTEM(DB11)

DCLGEN TABLE(LYIT20\_CUSTOMERS) -

ACTION(REPLACE) -

LIBRARY('LIT0002.LYIT11.DB2.SRCINC(CUSTOMER)') -

LANGUAGE(IBMCOB) -

QUOTE -

LABEL(YES)

END

/\*

//\*

//PCL EXEC PROC=COBPCL10,

// MEMBER=DB2COBOL,

// DBRMLIB=LIT0002.LYIT11.DB2.DBRMLIB,

// SRCCOB=LIT0002.LYIT11.DB2.IBMCOB,

// SRCINC=LIT0002.LYIT11.DB2.SRCINC,

// DSTLOAD=LIT0002.LYIT11.DB2.LOAD

//\*

//BIND EXEC PGM=IKJEFT01

//STEPLIB DD DSN=DB2.V10R1.SDSNLOAD,DISP=SHR

//DBRMLIB DD DSN=LIT0002.LYIT11.DB2.DBRMLIB,DISP=SHR

//SYSTSPRT DD SYSOUT=\*

//SYSTSIN DD \*

DSN SYSTEM(DB11)

BIND PLAN(LIT0002) MEMBER(DB2COBOL) ACTION(REPLACE) -

VALIDATE(BIND) ISOLATION(CS)

END

/\*

//\*

//RUN EXEC PGM=IKJEFT01

//STEPLIB DD DISP=SHR,DSN=DB2.V10R1.SDSNEXIT

// DD DISP=SHR,DSN=DB2.V10R1.SDSNLOAD

//SYSTSPRT DD SYSOUT=\*

//SYSOUT DD SYSOUT=\*

//SYSTSIN DD \*

DSN SYSTEM(DB11)

RUN PROG(DB2COBOL) -

LIB('LIT0002.LYIT11.DB2.LOAD') -

PLAN(LIT0002)

END

//\*

//OREPORT DD SYSOUT=\*,DCB=(RECFM=FBA,LRECL=174,BLKSIZE=0)

**Cobol Program**

The Cobol program can access and modify the data in a DB2 database using SQL statements within the Cobol program. These statements can be referred to as embedded SQL statements, or embedded SQL.

How to code embedded SQL statements

* Each statement must start with EXEC SQL and end with END-EXEC.
* Code the lines of the SQL statement in columns 12 through 72.
* Don’t code an SQL statement in a Cobol copy book member.
* When one SQL statement immediately follows another, always code a period after the END-EXEC for the first statement.

Below is the embedded SQL we will be using in the Cobol Program. The INCLUDE statements are used to include our DCLGEN output such as Customer and Invoice. The third INCLUDE statement includes the SQL Communication Area which is a structure or collection of variables that is updated after each SQL statement executes. The last SQL statement is where we declare our cursor for the tables we will be accessing. We use the SELECT statement to query and gather data from the rows in the table. In this program, we will be using **INNER JOIN** to query data from two related tables such as Customer and Invoice using the foreign key. We use ORDER BY at the end of the statement to sort the rows by the invoice date.

A screenshot of a cell phone

Description automatically generated

**SQL CODES**

The SQLCODE field is the one that DB2 puts a return value in after each SQL statement is executed. This value indicates whether the SQL operation was successful or not. As a result, you need to check the value of the SQLCODE field after almost every SQL statement. Below is an example using the SQLCODE variable. **NOTE**: SQLCODE variable is already declared in SQLCA as seen above.

A picture containing sitting, monitor, black, green

Description automatically generated

Below is the meanings of each SQLCODE:

A picture containing holding, sitting, black, man

Description automatically generated

**SQL Cursor to process multiple rows**

To process multiple rows in a table in Cobol program, we must use a cursor. A cursor is a pointer that identifies the current row in the table. When we use a cursor, we work through a result table one row at a time, much as you read through a sequential file. The first thing we need to do is declare the cursor in order to use it in our program as seen below:

A screen shot of a computer

Description automatically generated

The next thing we need to do is open the cursor:

A picture containing sitting, monitor, black, green

Description automatically generated

Once the cursor is opened, we can now fetch the data using a FETCH statement. When DB2 processes a FETCH statement, it moves the cursor to the next row, therefore making it the current row. Then DB2 moves the contents of the current row into the host variables which are specified in the INTO clause seen below. It is good practice to initialise the host variables before each fetch to prevent concatenation of data or in other words, empty the host variables.

A screenshot of a cell phone

Description automatically generated

The last thing we need to do is close the cursor at the end of the program as seen below:

A picture containing monitor, screen, black, phone

Description automatically generated

To output our data from the host variables, we need to move the host variable values to the output report variables as seen below:

A screenshot of a cell phone

Description automatically generated

Once the values are moved to the output variables, we then write out the report line as seen below:

A screenshot of a cell phone

Description automatically generated

Below is our headings section where we have declared the output variables along with spaces in between each variable in order to have the correct layout for the report.

A screenshot of a cell phone

Description automatically generated

Below is the middle section of our OREPORT:

A screenshot of a cell phone

Description automatically generated

Notice how we have repeated the headings after 10 lines, to do this we need line and page counters in our program as shown below:

A screenshot of a computer

Description automatically generated

In our main procedure below, we can see that using an IF statement to check if the line count is bigger than the lines on page which is 10, then we print the headings again. We also calculate the total invoice amounts for our totals section.

A screenshot of a cell phone

Description automatically generated