**COBOL-DB2**

**Steps to create COBOL-DB2 Program:**

* **Create all the necessary tables**
* **Create DCLGEN**
  + Goto ISPF application -> 15 -> 2 (DCLGEN) -> Give source table name and in which dataset (PS/PDS member) you want to store the DCLGEN
* **Pre-compile (DSNHPC):**
  + Syntax errors has been checked for COBOL as well as SQL code, but the syntax checking is done based on the DCLGEN for SQL code. The actual SQL code syntax errors are checked in Bind process.
  + Separates Modified Source NON-SQL (COBOL) and Modified Source SQL (SQL) into 2 separate files.
  + The SQL statements are kept under **DBRM (DataBase Request Module).**
  + DBRM contains list of SQL statements which is derived from COBOL-DB2 source program.
  + Then this DBRM will be used as input to the Bind process, and this will generate output as a Plan/Package.
* **Compile and link edit (Compile COBOL code) (IGYCRCTL):**
  + The main objective of compile is to convert the COBOL source code into load Module.
  + When the COBOL source code is given to the compiler, it first checks the syntax errors in the code.
  + Then the source code is converted into machine level language which is 0’s and 1’s that is kept under **Object Module.**
  + Then link edit happens, this will take all the required object modules and combine them into a single **Load Module.**
  + The load Module will contain the executable codes which is ready for execution.

Then this Load module is kept into a library called LOADLIB, it can be private library or system defined library. We can use JCLLIB or STEPLIB to access the load module in JCL.

* **DB2Bind (IKJEFT01):**
  + Bind performs the authorization check.
  + Checks the syntax errors of the SQL of DB2 cataloged table. This is more important because the syntax check done in pre-compile step is done using DCLGEN.
  + Provides the runtime instructions for the SQL statements which is present in DBRM. Each SQL statement is parsed (gone through) and best access method is chosen.
  + It also does DB2 optimization.
  + DBRM is given as input to create Plan/Package.
  + **PLAN:**
    - It is an executable module which contains the access path which is produced by the DB2 optimizer
    - It consists of one or more DBRM or packages.
    - DB2 catalog contains the details of the PLAN.
  + **PACKAGE:**
    - A package is a single, bound DBRM only. Package also contains the access paths.
    - Packages are not directly executable; it needs to be included in a PLAN and we need to execute the PLAN.
* **Execute/Run the program**
  + At runtime, the program runs SQL using the package created during the bind step, fetching data from DB2 as needed.
* **JCL to compile COBOL-DB2 program**

**//JOBCARD**

**//\***

**//STEP10 EXEC DB2COBCL,**

**// COPYLIB=NUHID.XXX.COPYLIB /\*COPYBOOK LIB**

**// DCLGLIB=NUHID.XXX.DBRMLIB /\*DBRM LIB**

**// LOADLIB=NUHID.XXX.LOADLIB /\*LOADLIB LIB**

**// SRCLIB=NUHID.XXX.COBOL /\*SOURCE LIB**

**// MEMBER=PGM-NAME /\*SOURCE CODE MEMBER**

**Structure of COBOL-DB2 Program:**

* **SQLCA:** It is used to check the return code of all embedded SQL statements which are used in this particular program.
* **EXEC SQL**

**INCLUDE DCLGEN-NAME**

**END-EXEC.**

Basically, this is the name of the DCLGEN member. Most of the time, we should keep the name of the table and the DCLGEN member as same. If not, please code your DCLGEN member name.

**INDENTIFICATION DIVISION**

**PROGRAM-ID. STRUCTDB2.**

**ENVIRONMENT DIVISION.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**EXEC SQL**

**INCLUDE SQLCA /\*SQL Communication Area\*/**

**END-EXEC.**

**PROCEDURE DIVISION.**

**MAIN-PARA.**

**EXEC SQL**

**EMBEDDED SQL**

**END-EXEC.**

**IF SQLCODE=0 /\*If embedded SQL return code is 0\*/**

**……**

**ELSE**

**……**

**END-IF.**

**STOP RUN.**

**Create a Table using COBOL-DB2:**

**INDENTIFICATION DIVISION**

**PROGRAM-ID. STRUCTDB2.**

**ENVIRONMENT DIVISION.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**EXEC SQL**

**INCLUDE SQLCA /\*SQL Communication Area\*/**

**END-EXEC.**

**PROCEDURE DIVISION.**

**MAIN-PARA.**

**EXEC SQL**

**CREATE TABLE INVESTOR(**

**ID INT,**

**INAME CHAR(20),**

**ISHARE SMALLINT**

**) IN DATABASE-NAME.TABLESPACE**

**END-EXEC.**

**IF SQLCODE=0 /\*If embedded SQL return code is 0\*/**

**DISPLAY ‘TABLE IS CREATED’**

**ELSE**

**DISPLAY ‘TABLE IS NOT CREATED’**

**END-IF.**

**STOP RUN.**

**Create a DML operation using COBOL-DB2:**

**INDENTIFICATION DIVISION**

**PROGRAM-ID. STRUCTDB2.**

**ENVIRONMENT DIVISION.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**EXEC SQL**

**INCLUDE SQLCA /\*SQL Communication Area\*/**

**END-EXEC.**

**PROCEDURE DIVISION.**

**MAIN-PARA.**

**EXEC SQL**

**INSERT INTO INVESTOR VALUES**

**(12345,’TOM’,10000)**

**END-EXEC.**

**IF SQLCODE=0 /\*If embedded SQL return code is 0\*/**

**DISPLAY ‘RECORD IS INSERTED’**

**ELSE**

**DISPLAY ‘RECORD IS NOT INSERTED’**

**END-IF.**

**STOP RUN.**

**Create a DML operation using Host Variables COBOL-DB2:**

**INDENTIFICATION DIVISION**

**PROGRAM-ID. STRUCTDB2.**

**ENVIRONMENT DIVISION.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**EXEC SQL**

**INCLUDE SQLCA /\*SQL Communication Area\*/**

**END-EXEC.**

**EXEC SQL**

**INCLUDE INVESTOR /\*NAME OF DCLGEN LIB MEMBER\*/**

**END-EXEC.**

**PROCEDURE DIVISION.**

**MAIN-PARA.**

**MOVE 12345 TO IID.**

**MOVE ‘SAM’ TO INAME.**

**MOVE 2000 TO ISHARE.**

**EXEC SQL**

**INSERT INTO INVESTOR VALUES**

**(:IID,:INAME,:ISHARE) /\*DCLGEN variables\*/**

**END-EXEC.**

**IF SQLCODE=0**

**DISPLAY ‘RECORD IS INSERTED’**

**ELSE**

**DISPLAY ‘RECORD IS NOT INSERTED’**

**END-IF.**

**STOP RUN.**

**CURSOR:**

* To handle multiple records in DB2 we use CURSOR.
* Let suppose we performed a select query, and based on where condition 3 records got selected but it won’t able to display 3 records. It’ll throw **-811 DB2** error as program is expecting 1 output but it got 3 outputs. So, to overcome that we need CURSOR.
* The idea behind using a cursor is to get multiple rows from the ‘SELECT’ query of the table.
* Singleton SQL containing an ‘INTO’ clause is used in those cases where the ‘SELECT’ query produces a single row only.
* If the ‘SELECT’ query produces multiple rows in the result set, then Cursors can be used to tackle this situation.
* So, Cursors are used to process multiple rows which are produced by the ‘SELECT’ query.
* How to use cursor:
  + **STEP-1: Declare the Cursor in WORKING-STORAGE SECTION.**
  + **STEP-2: Open the CURSOR in PROCEDURE DIVISION.**
  + **STEP-3: FETCH the CURSOR in PROCEDURE DIVISION. Fetch the records from the cursor one by one based on loop as per your requirements.**
  + **STEP-4: At end, CLOSE the CURSOR in PROCEDURE DIVISION.**
* **STEP-1: DECLARE CURSOR:**
  + **DECLARE cursor-name CURSOR FOR**

**{sql statements}**

* **STEP-2: OPEN CURSOR:**
  + **OPEN cursor-name**
  + ‘OPEN’ operation enables the pointer to the first record in the resultset.
* **STEP-3: FETCH RECORDS**
  + **FETCH cursor-name INTO {:host-var1, :host-var2, :host-var3,….}**
  + **Note:**
    - Fetch the records in a loop if you want to fetch all the records from the cursor until your required condition is met
    - We can also fetch the first record only based on the requirement of the project
    - Fetch returns one record at a time, so keep on fetching the records in a loop.
* **STEP-4: CLOSE CURSOR**
  + **CLOSE cursor-name**
  + This is used to close the cursor so that the resource used by the cursor are closed.

**CURSOR program:**

**INDENTIFICATION DIVISION**

**PROGRAM-ID. STRUCTDB2.**

**ENVIRONMENT DIVISION.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**EXEC SQL**

**INCLUDE SQLCA /\*SQL Communication Area\*/**

**END-EXEC.**

**EXEC SQL**

**INCLUDE COMPANY /\*DCLGEN LIB MEMBER\*/**

**END-EXEC.**

**EXEC SQL**

**DECLARE CR1 CURSOR FOR**

**SELECT C\_ID, C\_NAME, C\_SHARE**

**FROM COMPANY**

**END-EXEC.**

**PROCEDURE DIVISION.**

**MAIN-PARA.**

**EXEC SQL**

**OPEN CR1**

**END-EXEC.**

**IF SQLCODE=0**

**DISPLAY ‘OPEN CURSOR SUCCESSFUL’**

**ELSE**

**DISPLAY ‘OPEN NOT SUCCESSFUL’ SQLCODE**

**END-IF.**

**PERFORM FECTH-PARA UNTIL SQLCODE NOT EQUAL 0.**

**STOP RUN.**

**FETCH-PARA.**

**EXEC SQL**

**FETCH CR1 INTO :C-ID, :C-NAME, :C-SHARE /\*DCLGEN VARIABLES\*/**

**END-EXEC.**

**IF SQLCODE=0**

**DISPLAY C-ID, C-NAME, C-SHARE**

**END-IF.**