COBOL - Database Interface

As of now, we have learnt the use of files in COBOL. Now, we will discuss how a COBOL program interacts with DB2. It involves the following terms –

- Embedded SQL
- DB2 Application Programming
- Host Variables
- SQLCA
- SQL Queries
- Cursors

Embedded SQL

Embedded SQL statements are used in COBOL programs to perform standard SQL operations. Embedded SQL statements are preprocessed by the SQL processor before the application program is compiled. COBOL is known as the **Host Language**. COBOL-DB2 applications are those applications that include both COBOL and DB2.

Embedded SQL statements work like normal SQL statements with some minor changes. For example, the output of a query is directed to a predefined set of variables which are referred as **Host Variables**. An additional INTO clause is placed in the SELECT sta

DB2 Application Programming

Following are rules to be followed while coding a COBOL-DB2 program −

- All the SQL statements must be delimited between **EXEC SQL** and **ENDEXEC.**.
- SQL statements must be coded in Area B.
- All the tables that are used in a program must be declared in the WorkingStorage Section. This is done by using the **INCLUDE** statement.
- All SQL statements other than INCLUDE and DECLARE TABLE must appear in the Procedure Division.

Host Variables

Host variables are used for receiving data from a table or inserting data in a table. Host variables must be declared for all values that are to be passed between the program and the DB2. They are declared in the Working-Storage Section.

Host variables cannot be group items, but they may be grouped together in host structure. They cannot be **Renamed** or **Redefined**. Using host variables with SQL statements, prefix them with a **colon (:).**.

Syntax

Following is the syntax to declare host variables and include tables in the Working-St

Advertisement

```
[
    EXEC SQL BEGIN DECLARE SECTION
    END-EXEC.

    01 STUDENT-REC.
       05 STUDENT-ID PIC 9(4).
       05 STUDENT-NAME PIC X(25).
       05 STUDENT-ADDRESS X(50).
    EXEC SQL END DECLARE SECTION
    END-EXEC.
```

SQLCA

SQLCA is a SQL communication area through which DB2 passes the feedback of SQL execution to the program. It tells the program whether an execution was successful or not. There are a number of predefined variables under SQLCA like **SQLCODE** which contains the error code. The value '000' in SQLCODE states a successful execution.

Syntax

Following is the syntax to declare an SQLCA in the Working-Storage section −

```
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL
    INCLUDE SQLCA
    END-EXEC.
```

SQL Queries

Lets assume we have one table named as 'Student' that contains Student-Id, Student-Name, and Student-Address.

The STUDENT table contains the following data −

Advertisement

```
S                                    Student Address
1                                    Hyderabad
1                                        Delhi
1                                        Mumbai
1                                        Lucknow
```

Th **SELECT** query in a COBOL program −

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   EXEC SQL
      INCLUDE SQLCA
   END-EXEC.

   EXEC SQL
      INCLUDE STUDENT
   END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION
   END-EXEC.
      01 WS-STUDENT-REC.
         05 WS-STUDENT-ID PIC 9(4).
         05 WS-STUDENT-NAME PIC X(25).
         05 WS-STUDENT-ADDRESS X(50).
   EXEC SQL END DECLARE SECTION
   END-EXEC.

PROCEDURE DIVISION.
   EXEC SQL
      SELECT STUDENT-ID, STUDENT-NAME, STUDENT-ADDRESS
      INTO :WS-STUDENT-ID, :WS-STUDENT-NAME, WS-STUDENT-ADDRESS FROM
STUDENT
      WHERE STUDENT-ID=1004
   END-EXEC.

   IF SQLCODE = 0
      DISPLAY WS-STUDENT-RECORD
   ELSE DISPLAY 'Error'
```

Advertisement

JC –

```
/                                    = A,MSGCLASS = C
/
/                                    RMLIB,DISP = SHR
```

```
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD *
   DSN SYSTEM(SSID)
   RUN PROGRAM(HELLO) PLAN(PLANNAME) -
   END
/*
```

When you compile and execute the above program, it produces the following result −

```
1004 Chulbul Pandey     Lucknow
```

The following example shows the usage of **INSERT** query in a COBOL program −

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   EXEC SQL
   INCLUDE SQLCA
   END-EXEC.

   EXEC SQL
   INCLUDE STUDENT
   END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION
   END-EXEC.
      01 WS-STUDENT-REC.
         05 WS-STUDENT-ID PIC 9(4).
         05 WS-STUDENT-NAME PIC X(25).
                                    50).




P



                                    DENT-NAME.
                                    -ADDRESS.
```

Advertisement

```
   EXEC SQL
      INSERT INTO STUDENT(STUDENT-ID, STUDENT-NAME, STUDENT-ADDRESS)
      VALUES (:WS-STUDENT-ID, :WS-STUDENT-NAME, WS-STUDENT-ADDRESS)
   END-EXEC.

   IF SQLCODE = 0
      DISPLAY 'Record Inserted Successfully'
      DISPLAY WS-STUDENT-REC
   ELSE DISPLAY 'Error'
   END-IF.
STOP RUN.
```

JCL to execute the above COBOL program –

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS = A,MSGCLASS = C
//STEP001  EXEC PGM = IKJEFT01
//STEPLIB  DD DSN = MYDATA.URMI.DBRMLIB,DISP=SHR
//SYSPRINT DD SYSOUT = *
//SYSUDUMP DD SYSOUT = *
//SYSOUT   DD SYSOUT = *
//SYSTSIN  DD *
   DSN SYSTEM(SSID)
   RUN PROGRAM(HELLO) PLAN(PLANNAME) -
   END
/*
```

When you compile and execute the above program, it produces the following result –

```
Record Inserted Successfully
1005 TutorialsPoint     Hyderabad
```

Th Advertisement **UPDATE** query in a COBOL program –

```
   INCLUDE SQLCA
   END-EXEC.

   EXEC SQL
   INCLUDE STUDENT
   END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION
   END-EXEC.
      01 WS-STUDENT-REC.
         05 WS-STUDENT-ID PIC 9(4).
         05 WS-STUDENT-NAME PIC X(25).
         05 WS-STUDENT-ADDRESS X(50).
   EXEC SQL END DECLARE SECTION
   END-EXEC.

PROCEDURE DIVISION.
   MOVE 'Bangalore' TO WS-STUDENT-ADDRESS.
   EXEC SQL
      UPDATE STUDENT SET STUDENT-ADDRESS=:WS-STUDENT-ADDRESS
      WHERE STUDENT-ID = 1003
   END-EXEC.

   IF SQLCODE = 0
      DISPLAY 'Record Updated Successfully'
   ELSE DISPLAY 'Error'
   END-IF.
STOP RUN.
```

JCL to execute the above COBOL program −

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS = A,MSGCLASS = C
//STEP001  EXEC PGM = IKJEFT01
/                         RMLIB,DISP = SHR
/
/
/
/


                          E) -
```

```
   END
/*
```

When you compile and execute the above program, it produces the following result −

```
Record Updated Successfully
```

The following **example** shows the usage of **DELETE** query in a COBOL program −

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
WORKING-STORAGE SECTION.

   EXEC SQL
   INCLUDE SQLCA
   END-EXEC.

   EXEC SQL
   INCLUDE STUDENT
   END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION
   END-EXEC.
      01 WS-STUDENT-REC.
         05 WS-STUDENT-ID PIC 9(4).
         05 WS-STUDENT-NAME PIC X(25).
         05 WS-STUDENT-ADDRESS X(50).
   EXEC SQL END DECLARE SECTION
   END-EXEC.

PROCEDURE DIVISION
```

Advertisement

```
                                    -ID
```

```
      DISPLAY 'Record Deleted Successfully'
   ELSE DISPLAY 'Error'
   END-IF.
STOP RUN.
```

JCL to execute the above COBOL program −

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS = A,MSGCLASS = C
//STEP001  EXEC PGM = IKJEFT01
//STEPLIB  DD DSN = MYDATA.URMI.DBRMLIB,DISP=SHR
//SYSPRINT DD SYSOUT = *
//SYSUDUMP DD SYSOUT = *
//SYSOUT   DD SYSOUT = *
//SYSTSIN  DD *
   DSN SYSTEM(SSID)
   RUN PROGRAM(HELLO) PLAN(PLANNAME) -
   END
/*
```

When you compile and execute the above program, it produces the following result −

```
Record Deleted Successfully
```

Cursors

Cursors are used to handle multiple row selections at a time. They are data structures that hold all the results of a query. They can be defined in the Working-Storage Section or the Procedure Division. Following are the operations associated with Cursor −

- Declare
- Open

Advertisement

De

Cu king-Storage Section or the Procedure Division.
Th ent which is a nonexecutable statement.

```
EXEC SQL
   DECLARE STUDCUR CURSOR FOR
   SELECT STUDENT-ID, STUDENT-NAME, STUDENT-ADDRESS FROM STUDENT
   WHERE STUDENT-ID >:WS-STUDENT-ID
END-EXEC.
```

Open

Before using a cursor, Open statement must be performed. The Open statement prepares the SELECT for execution.

```
EXEC SQL
   OPEN STUDCUR
END-EXEC.
```

Close

Close statement releases all the memory occupied by the cursor. It is mandatory to close a cursor before ending a program.

```
EXEC SQL
   CLOSE STUDCUR
END-EXEC.
```

Fetch

Fetch statement identifies the cursor and puts the value in the INTO clause. A Fetch statement is coded in loop as we get one row at a time.

```
EXEC SQL
   FETCH STUDCUR
                           NT-NAME, WS-STUDENT-ADDRESS
E
```

Advertisement

Th cursor to fetch all the records from the
ST

```
I
F
```

```
DATA DIVISION.
   WORKING-STORAGE SECTION.

   EXEC SQL
   INCLUDE SQLCA
   END-EXEC.

   EXEC SQL
   INCLUDE STUDENT
   END-EXEC.

   EXEC SQL BEGIN DECLARE SECTION
   END-EXEC.
      01 WS-STUDENT-REC.
         05 WS-STUDENT-ID PIC 9(4).
         05 WS-STUDENT-NAME PIC X(25).
         05 WS-STUDENT-ADDRESS X(50).
   EXEC SQL END DECLARE SECTION
   END-EXEC.

   EXEC SQL
      DECLARE STUDCUR CURSOR FOR
      SELECT STUDENT-ID, STUDENT-NAME, STUDENT-ADDRESS FROM STUDENT
      WHERE STUDENT-ID >:WS-STUDENT-ID
   END-EXEC.

PROCEDURE DIVISION.
   MOVE 1001 TO WS-STUDENT-ID.
   PERFORM UNTIL SQLCODE = 100

   EXEC SQL
      FETCH STUDCUR
      INTO :WS-STUDENT-ID, :WS-STUDENT-NAME, WS-STUDENT-ADDRESS
```

JC –

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS = A,MSGCLASS = C
//STEP001  EXEC PGM=IKJEFT01
//STEPLIB  DD DSN=MYDATA.URMI.DBRMLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD *
   DSN SYSTEM(SSID)
   RUN PROGRAM(HELLO) PLAN(PLANNAME) -
   END
/*
```

When you compile and execute the above program, it produces the following result −

```
1001 Mohtashim M.       Hyderabad
1002 Nishant Malik      Delhi
1003 Amitabh Bachan     Mumbai
1004 Chulbul Pandey     Lucknow
```

Microsoft Azure Tutorial

Git Tutorial

Ethical Hacking Tutorial

Docker Tutorial

Kubernetes Tutorial

DSA Tutorial

Spring Boot Tutorial

SDLC Tutorial

Unix Tutorial

CERTIFICATIONS

Business Analytics Certification

Java & Spring Boot Advanced Certification

Data Science Advanced Certification

Cloud Computing And DevOps

Advanced Certification In Business Analytics

Artificial Intelligence And Machine Learning

DevOps Certification

Game Development Certification

Front-End Developer Certification

AWS Certification Training

Python Programming Certification

COMPILERS & EDITORS

Online Java Compiler

Online Python Compiler

Online Go Compiler

Online C Compiler

Online C++ Compiler

Advertisement

O

O

O

O

O

O

Change GDPR Consent

Advertisement