

The SQL Procedure Language (SQL PL)

Tony Andrews
Themis Education

tandrews@themisinc.com

www.themisinc.com

Coding a SQL PL Procedure

- An SQL procedure consists of:
 - CREATE PROCEDURE header
 - BEGIN statement
 - Body (SQL procedural statements and / or DB2 SQL statements)
 - END statement
- Comments within an SQL procedure:
 - -- for a single line comment
 - /* to start */ end multiple-lines comments
- Statements end with semicolon

A SQL PL Header

```
CREATE PROCEDURE SPA80
```

Parameters

```
(IN P_DNO CHAR(3)  
,OUT P_CNT SMALLINT  
,OUT P_SUMSAL DECIMAL(11,2)  
,OUT P_RETCODE INTEGER )
```

```
VERSION V1
```

- Procedure Name
 - 128 byte max length
 - Unique within Schema / Collection
 - Schema / Collection ID will be supplied when create is deployed
- Parameters
 - 128 byte max length
 - Can be IN bound, OUT bound or INOUT (both directions)
 - Used to pass data between procedure and caller
 - Cannot specify a default value
- Versioning
 - 64 EBCDIC bytes max length
 - If using versioning do not use the default V1 naming convention

A SQL PL Header

```
LANGUAGE SQL
-- COMMON BIND OPTIONS
  CALLED ON NULL INPUT
  RESULT SETS 0
  QUALIFIER THEMIS1
  PACKAGE OWNER THMC04
  ASUTIME LIMIT 500000
  COMMIT ON RETURN NO
  CURRENT DATA NO
  DEGREE ANY
  WITHOUT EXPLAIN
  ISOLATION LEVEL CS
  VALIDATE BIND
```

...

...

Bind Options

A SQL PL Body

The body consists of 5 parts:

- SQL variable declarations
- Condition names
- Cursors
- Condition Handlers
- Code...

A SQL PL Body

...

...

P1: BEGIN

SET P_CNT = 0;

SET P_SUMSAL = 0;

SET P_RETCODE = 0;

Assign Values

SELECT COUNT (*), SUM (SALARY)

INTO P_CNT, P_SUMSAL

FROM EMP

WHERE DEPTNO = P_DNO;

END **P1**

Clear your
outbound
parms

SQL Procedure Statements

- DECLARE Statement
 - Assignment Statement
 - **CALL, GOTO, LEAVE, RETURN**
 - **IF, CASE, WHILE, LOOP, REPEAT, ITERATE, FOR**
 - Compound statement
 - **GET DIAGNOSTICS** statement
 - **SIGNAL, RESIGNAL** statements
 - SQL Statements
- Note: Successful Execution of any SQL statement will set
- SQLCODE variable value to 0 and
 - SQLSTATE variable value to '00000'.

Declaring SQL Variables

Syntax: DECLARE *SQL-variable-name data-type*
[DEFAULT *constant*] ;

- Same data types and lengths as DB2 table columns
- Parameter and variable names are not case sensitive
- SQL reserved word cannot be used as parameter name or variable name
- Variable name declarations must be first prior to other statements in the procedure body.
- Coding variable names and parameter names in a DB2 SQL statement **do not require** colon to precede them
- A declare statement ends with a semicolon

Assignment Statement

Assigns a value to an output parameter or to an SQL variable

```
SET var1 = 10;
```

```
SET var2 = ( SELECT count(*) FROM EMP );
```

```
SET var3 = NULL;
```

- Assignment statements conform to the SQL assignment rules
- The data type of the target and source must be compatible
- An assignment statement must end with a semicolon.

```
V10: SET var1 = 10, var3 = NULL;
```

SQL Variable Example

...

```
P1: BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;
    DECLARE V_LAST_PAID_DATE DATE;
    DECLARE V1 CHAR(25) DEFAULT 'NOT PAID';
    DECLARE V2 INTEGER;
    DECLARE V3 DECIMAL(9,2);
    DECLARE V4 DECIMAL(9,2) DEFAULT 0;

    SET V2 = 1000;
    SET V3 = 500.00;

    .
    .
    .
END P1
```

IF Statement

```
DECLARE v_grade CHAR(1);  
DECLARE v_a_count INTEGER;  
DECLARE v_b_count INTEGER;  
DECLARE v_invalid_count INTEGER;  
...  
UPDATE STUDENT SET GRADE = v_grade  
WHERE STUDENT_NO = var1;
```

```
IF v_grade = 'A' THEN  
    set v_a_count = v_a_count + 1;  
ELSEIF v_grade = 'B' THEN  
    set v_b_count = v_b_count + 1;  
ELSEIF . . . THEN . . .  
ELSE  
    Set v_invalid_count = v_invalid_count + 1;  
END IF;
```



Watch
Your
Punctuation!

IF Statement

```
IF v_grade = 'A' THEN  
    set v_a_count = v_a_count + 1;  
    set v_counter = v_counter + 1;  
ELSEIF v_grade = 'B' THEN  
    set v_b_count = v_b_count + 1;  
    set v_counter = v_counter + 1;  
ELSEIF . . . THEN . . .  
ELSE  
    Set v_invalid_count = v_invalid_count + 1;  
END IF;
```

CASE Statement

1. Simple CASE: Testing for value of variable

```
CASE v_grade
  WHEN 'A' THEN set v_a_count = v_a_count + 1;
                   set v_counter = v_counter + 1;
  WHEN 'B' THEN set b_count = b_count + 1;
                   set v_counter = v_counter + 1;
  ELSE set v_invalid_count = v_invalid_count + 1 ;
END CASE;
```

2. Searched CASE: Testing for TRUE condition

```
CASE
  WHEN v_edlevel < 12 THEN
    set v_no_diploma = v_no_diploma + 1;
  WHEN v_edlevel > 11 and v_edlevel < 16 THEN
    set v_high_school = v_high_school + 1;
END CASE;
```

LOOP, LEAVE & REPEAT

```
FETCH_LOOP: LOOP
    FETCH CURSOR1 INTO VAR1, VAR2, VAR3 ;
    IF SQLCODE = 100 THEN
        LEAVE FETCH_LOOP;
    END IF;
    <process values returned by cursor>
END LOOP;
```

```
REPEAT
    FETCH CURSOR1 INTO VAR1, VAR2, VAR3;
    IF SQLCODE = 100 THEN SET V_EOF = 'Y' ;
    ELSE
        <process values returned by cursor>
    END IF;
    UNTIL V_EOF = 'Y'
END REPEAT;
```

WHILE Statement

```
DECLARE SQLCODE          INTEGER DEFAULT 0;  
DECLARE V_EOF            CHAR(1) DEFAULT 'N';
```

...

```
WHILE (V_EOF = 'N') DO  
    FETCH CURSOR1 INTO VAR1, VAR2, VAR3;  
    IF SQLCODE = 100 THEN  
        SET V_EOF = 'Y';  
    ELSE  
        <process a row> . . . ;  
    END IF;  
END WHILE;
```

FOR Statement

```
FOR FOR_ROUTINE AS
  CURSOR1 CURSOR FOR
    SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
      FROM EMP
     WHERE DEPTNO = p_deptno
    ORDER BY SALARY DESC
DO
  SET v_numrows = v_numrows + 1;
  SET v_salarytotal = v_salarytotal + SALARY;
END FOR;
```

Native Only!

Variables

Column from
CURSOR1

GOTO Statement

The GOTO transfers control to a labeled statement. The labeled statement and the GOTO statement must be in the same scope.

```
IF V_SERVICE < 10000 THEN
    GOTO EXIT_RTN ;
END IF ;
...;
...;
EXIT_RTN:
    BEGIN
        SET P_RETURN_CODE = V_SERVICE ;
    END ;
```

ITERATE Statement

The ITERATE statement causes the flow of control to return to the beginning of a labeled loop.

```
WHILE_ROUTINE :  
    WHILE (MORE_RESULT = 0) DO  
        FETCH CURSOR1 INTO VAR1, VAR2, VAR3;  
        SET MORE_RESULT = SQLCODE;  
        IF VAR3 < 0 THEN  
            ITERATE WHILE_ROUTINE;  
        END IF ;  
        . . . ;  
        . . . ;  
    END WHILE;
```

Compound Statement

label:

BEGIN { NOT ATOMIC or ATOMIC }

[SQL-variable-declaration ;]

[Declare Cursor statement ;]

[condition-declaration ;]

[return-code-declaration ;]

[handler-declaration ;]

SQL-procedure-statement; ...

END { label }

Compound Statement

P1 :

BEGIN

```
DECLARE SQLCODE INTEGER ;  
DECLARE C1 CURSOR WITH RETURN ..... ;  
INSERT INTO AUDIT VALUES ( PARMX, PARMY, PARMZ) ;  
IF (SQLCODE = -803) THEN  
    ... ;  
ELSE  
    ... ;  
END IF ;  
INSERT ... ;
```

END P1 ;

SQL Statements

Most DB2 SQL statements are supported :

- SELECT INTO
- DECLARE CURSOR / OPEN / FETCH / CLOSE
- INSERT
- UPDATE
- DELETE
- MERGE

Sample Stored Procedure

```
CREATE PROCEDURE SPA80 (OUT P_CNT1 SMALLINT
                        ,OUT P_SUMSAL DECIMAL(11,2)
                        ,OUT P_RETCODE INTEGER
                        )
    VERSION V1
    ASUTIME 500000
    ISOLATION LEVEL CS
    VALIDATE BIND
    PACKAGE OWNER DBTHM80
    QUALIFIER THEMIS1
    RESULT SETS 0
    LANGUAGE SQL

P1: BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;

    SELECT COUNT(*) , SUM(SALARY)
        INTO P_CNT1, P_SUMSAL
        FROM EMP;

    SET P_RETCODE = SQLCODE;
END P1
```

Returning Result Sets

```
CREATE PROCEDURE SPB80 (OUT P_RETCODE INTEGER)

    VERSION V1                ASUTIME 500000
    ISOLATION LEVEL CS        VALIDATE BIND
    PACKAGE OWNER DBTHM80    QUALIFIER THEMIS1
    RESULT SETS 1           LANGUAGE SQL

P1: BEGIN
    DECLARE SQLCODE INTEGER DEFAULT 0;

    DECLARE CURSOR1 CURSOR WITH RETURN FOR
        SELECT EMPNO, LASTNAME, MIDINIT, FIRSTNME,
               SALARY, DEPTNO
        FROM EMP
        ORDER BY DEPTNO, EMPNO
        FOR FETCH ONLY;

    OPEN CURSOR1;

    SET P_RETCODE = SQLCODE;
END P1
```

Processing a Cursor

```
P1: BEGIN
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE V_EOC CHAR(1) DEFAULT 'N';
  DECLARE V_SAL DECIMAL(9,2);
  DECLARE C1 CURSOR FOR
    SELECT SALARY FROM EMP
    WHERE DEPTNO = P_DEPTNO;

  OPEN C1;

  SET P_SUM = 0;
  REPEAT
    FETCH C1 INTO V_SAL;
    IF SQLCODE = 100 THEN
      SET V_EOC = 'Y';
    END IF;
    SET P_SUM = P_SUM + V_SAL;
  UNTIL V_EOC = 'Y'
  END REPEAT;

  CLOSE C1;
END P1
```


Capturing SQLCODE & SQLSTATE

```
P1: BEGIN
    DECLARE SQLCODE    INTEGER DEFAULT 0;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';

    ...
    FETCH CURSOR1 INTO V1, V2;
    IF SQLCODE = 100 THEN
        SET EOF = 'Y';
```

Unhandled Errors

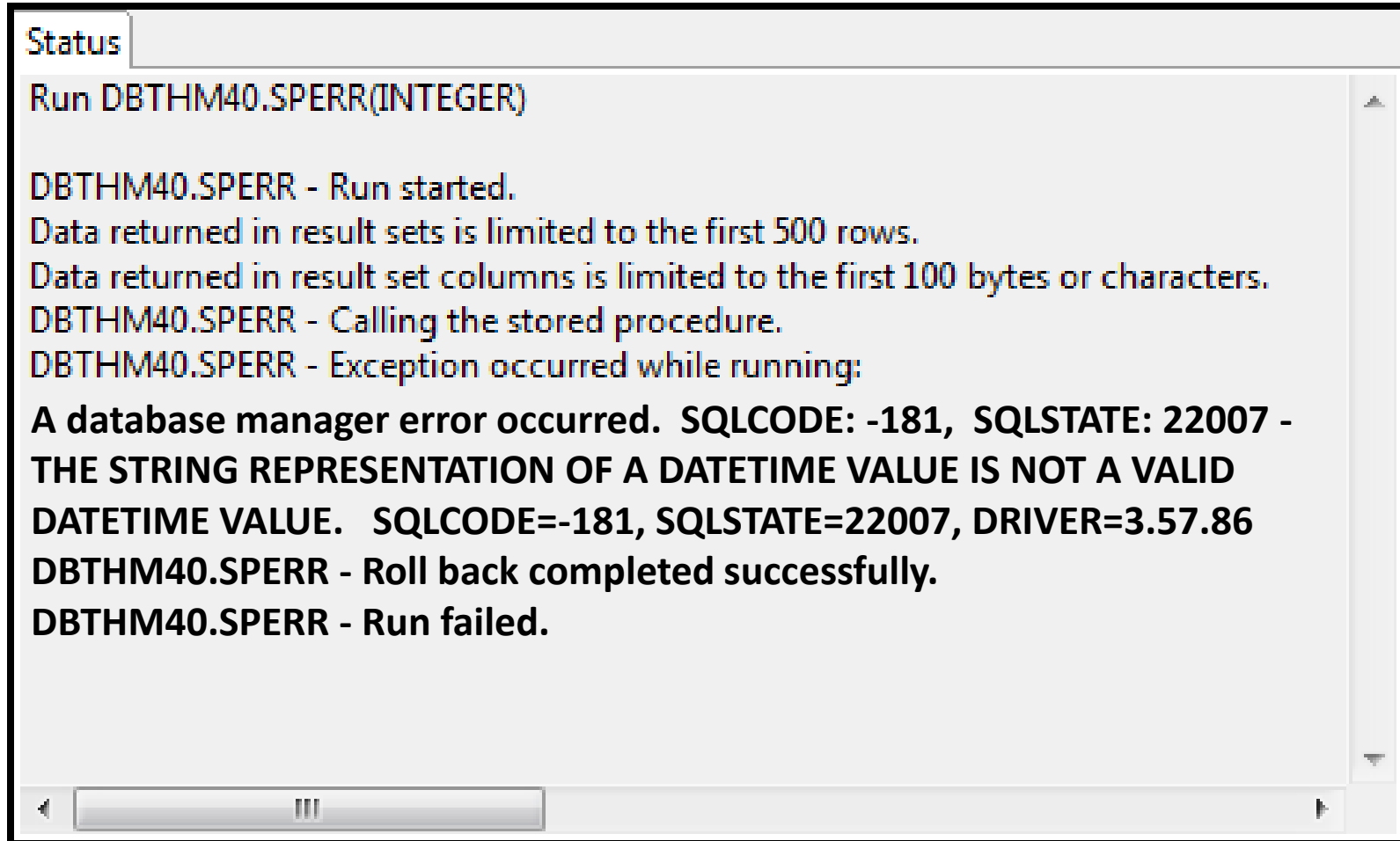
```
CREATE PROCEDURE SPERR (OUT P_RETCODE INTEGER)
  VERSION VERSION1
P1: BEGIN
  DECLARE SQLCODE          INTEGER DEFAULT 0;
  DECLARE V1               DATE;

  SET P_RETCODE = 0;
  SELECT DATE ('2016-12-32')
    INTO V1
    FROM SYSIBM.SYSDUMMY1;

  SET P_RETCODE = SQLCODE;
END P1
```

Execution Stops Here!!

Unhandled Errors



The screenshot shows a database console window with a title bar labeled 'Status'. The main area contains the following text:

```
Run DBTHM40.SPERR(INTEGER)

DBTHM40.SPERR - Run started.
Data returned in result sets is limited to the first 500 rows.
Data returned in result set columns is limited to the first 100 bytes or characters.
DBTHM40.SPERR - Calling the stored procedure.
DBTHM40.SPERR - Exception occurred while running:
A database manager error occurred. SQLCODE: -181, SQLSTATE: 22007 -
THE STRING REPRESENTATION OF A DATETIME VALUE IS NOT A VALID
DATETIME VALUE. SQLCODE=-181, SQLSTATE=22007, DRIVER=3.57.86
DBTHM40.SPERR - Roll back completed successfully.
DBTHM40.SPERR - Run failed.
```

The error message is highlighted in bold. The console window includes a scrollbar on the right and a status bar at the bottom with navigation icons.

Handlers

DECLARE { EXIT
CONTINUE } HANDLER FOR { SQLEXCEPTION
SQLWARNING
NOT FOUND
SQLSTATE 'value' }

Exit Handler

```
P1: BEGIN
    ...
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            SET P_RETCODE = SQLCODE;
            GET STACKED DIAGNOSTICS CONDITION 1
                V_SQLMSG = MESSAGE_TEXT;
        END;
    ...
END P1
```

Variable
from the SP

Handler code
executes and
procedure
Exits

GET DIAGNOSTICS
keyword

GET DIAGNOSTICS

The SQL Stored Procedure will only receive values in SQLCODE and SQLSTATE, the other SQLCA values are not immediately available. Following are GET DIAGNOSTICS statements that will return additional SQLCA values:

GET DIAGNOSTICS CONDITION 1

```
var1 = ROW_COUNT,  
var2 = DB2_RETURNED_SQLCODE,  
var3 = MESSAGE_TEXT,  
var4 = DB2_REASON_CODE,  
var5 = RETURNED_SQLSTATE ,  
var6 = DB2_ROW_NUMBER,  
var7 = DB2_LINE_NUMBER;
```

var1 is DECIMAL(31,0) var2,4,6,7 are INTEGER
var3 is VARCHAR(32672) var5 is CHAR(5)

SQLCA DIAGNOSTICS

GET DIAGNOSTICS CONDITION 1

```
V_SQLSTATE          = RETURNED_SQLSTATE,  
V_SQLCODE_OUT      = DB2_RETURNED_SQLCODE,  
V_MESSAGE_TEXT     =MESSAGE_TEXT,  
V_MODULE_DETECTING_ERROR =  
                    DB2_MODULE_DETECTING_ERROR,  
V_SQLERRD_SET      = DB2_SQLERRD_SET,  
V_DB2_SQLERRD1     = DB2_SQLERRD1,  
V_DB2_SQLERRD2     = DB2_SQLERRD2,  
V_DB2_SQLERRD3     = DB2_SQLERRD3,  
V_DB2_SQLERRD4     = DB2_SQLERRD4,  
V_DB2_SQLERRD5     = DB2_SQLERRD5,  
V_DB2_SQLERRD6     = DB2_SQLERRD6
```

SIGNAL & RESIGNAL

```
DECLARE SQLSTATE CHAR(5);

DECLARE EXIT HANDLER FOR SQLSTATE VALUE '75001'
BEGIN
    SET P_RETCODE = SQLCODE;
    GET STACKED DIAGNOSTICS CONDITION 1
        P_MESSAGE_TEXT = MESSAGE_TEXT;
    RESIGNAL;
END;

IF P_INDATE < '1900-01-01' THEN
    SIGNAL SQLSTATE '75001'
        SET MESSAGE_TEXT = 'Input date is too old';
END IF;
```


SIGNAL & RESIGNAL Condition Handler

```
DECLARE INVALID_PARM_DATE CONDITION FOR SQLSTATE
                                VALUE '75001';
DECLARE EXIT HANDLER FOR INVALID_PARM_DATE
BEGIN
    SET P_RETCODE = SQLCODE;
    GET DIAGNOSTICS CONDITION 1
        P_MESSAGE_TEXT = MESSAGE_TEXT;
    RESIGNAL;
END;

IF P_IN_DATE != '2011-06-01' THEN
    SIGNAL INVALID_PARM_DATE
    SET MESSAGE_TEXT = 'Invalid Parm Date'
        CONCAT 'P_IN_DATE = '
        CONCAT P_IN_DATE;
END IF;
```

Note: No Semi Colon

Thank you for having me be a part of
your DB2 user group! I hope you
learned something new today!

Tony Andrews
Themis Education

tandrews@themisinc.com

www.themisinc.com