Improved Support for SQL Procedural Language

IBM System i
COMMON Luxembourg 2007

Jarek Miszczyk, IBM Rochester

IBM Information Management software

Simplify Your IT.

Agenda

- Brief Introduction to DB2 SQL Procedural Language
- Faster Execution with Expression Evaluator
- Mastering Nested Compound Statement Handlers
- Implicit Schema Qualification for Static and Dynamic SQL Statements

DB2 SQL Procedural Language

- Language similar to SQL Server T-SQL and Oracle PL/SQL
- Subset of SQL/PSM standard
  - Database Language SQL - Part 4: Persistent Stored Modules (ISO/IEC 9075)
  - Include SQL procedures, SQL functions and Feature P01, "Stored modules" support

Basic Structure of a SQL Procedure

```
CREATE PROCEDURE <procname>  [( {optional parameters} )]
[optional procedure attributes]
BEGIN [atomic]
 ... statements ...
END
```

For Example:

```
CREATE PROCEDURE DB2USER.COF_DELETE(IN P_ID INTEGER )
LANGUAGE SQL
SPECIFIC DB2USER.COF_DELETE
P1 : BEGIN
     DELETE FROM COFFEES WHERE ID = P_ID ;
END P1
```

Basic Structure - Parameters

CREATE PROCEDURE <procname>
[({optional parameters})]
[optional procedure attributes]
BEGIN [atomic]
... statements ...
END

A parameter definition consists of three parts:
- **Mode** defines if the parameter is input (IN), output (OUT), or both(INOUT)
- **Parameter** name
- **Data Type** is the SQL data type and size

For example:

```
CREATE PROCEDURE new_salary
                (IN empnr INTEGER,
                 INOUT salary DECIMAL(9,2),
                 OUT update_ts TIMESTAMP)
```

Basic Structure - Attributes

CREATE PROCEDURE <procname>
[({optional parameters})]
[optional procedure attributes]
BEGIN [atomic]
... statements ...
END

LANGUAGE SQL

RESULT SETS <n>
specifies max number of result sets that can be returned from procedure.

SPECIFIC <unique_name>
A qualified or unqualified name that uniquely identifies the procedure, can be used to control procedures "short" name when procedure name greater than 10 characters

FENCED | NOT FENCED

[CONTAINS | READS | **MODIFIES**] **SQL**

NOT DETERMINISTIC | **DETERMINISTIC**

[**OLD |** NEW] **SAVEPOINT LEVEL**
NEW causes savepoint to automatically be created upon entry of the stored procedure.

COMMIT ON RETURN [YES | **NO**]
YES - DB2 issues a commit if the procedure successfully returns. Result Set cursors must be declared WITH HOLD to be usable after the commit operation. ATOMIC compound statement requires COMMIT ON RETURN YES.

Basic Structure - Procedure Body

CREATE PROCEDURE <procname>
[({optional parameters})]
[optional procedure attributes]
BEGIN [atomic]
procedure statement; [repeatable]
END

Compound statement
Statement that groups other statement together in an SQL procedure. Compound statements can be nested within each other.

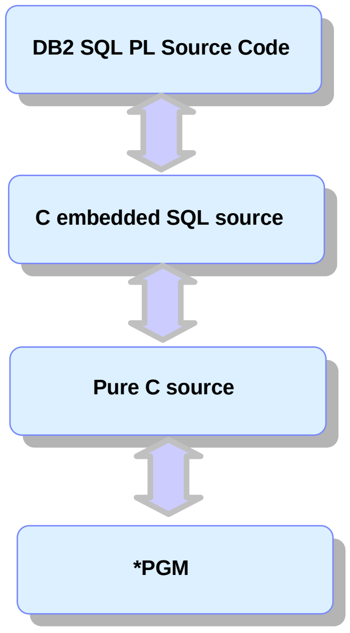
ATOMIC
indicates that if an error occurs, all SQL statements in the compound statement group will be rolled back.
If ATOMIC specified, COMMIT or ROLLBACK cannot be specified in the stored procedure

NOT ATOMIC
indicates that an error does NOT cause statements to be rolled back

For example:

```
CREATE PROCEDURE DB2USER.COF_SELECT()
 RESULT SETS 1
 LANGUAGE SQL
 SPECIFIC DB2USER.COF_SELECT
P1 : BEGIN NOT ATOMIC
-- Declare cursors
DECLARE DB2_SP_SQL1 CURSOR FOR
     SELECT * FROM DB2USER . COFFEES ;
-- Cursor left open for client application.
OPEN DB2_SP_SQL1 ;
END P1
```

Multiphase process to create stored procedure executable



- Development system requirements
  - Openness Includes (5769-SS1)
  - *DB2 SQL Development Kit requirement eliminated in V5R2*
  - *C compiler requirement eliminated in V5R1*
- Performance Considerations
  - Generated C code is not as efficient as user-written code

V5R4 Expression Evaluator

The need for Expression Evaluator

- The SQL PL supports standard programming constructs such as loops (FOR), conditions (IF THEN) and assignments (SET i=j +1)
- Before V5R4 these simple constructs were executed as queries against a "dummy" system table QSYS2/QSQPTABL
- Negative performance impact on "tight looping" procedures
  - The SQL statement goes through the entire SQL stack
    - Statement is parsed
    - Access plan stored in *PGM is validated, and re-planned if necessary
    - Open Data Path created or reused
  - Potentially large number of reusable ODPs over the QSYS2/QSQPTABL table
    - ODPs processing takes CPU and ODPs increase memory foot print

V5R4 Expression Evaluator

- A fast path evaluator for 'table-less' SQL scalar expressions
  - Scalar expressions in procedural statements (e.g. IF, SET)
  - No need for Open/Fetch/Close processing
  - No ODPs associated with simple statements
- Early performance tests show up to 30% better performance
  - The performance gains depend on the number of tightly looping statements in a procedure
  - Caching of expressions improves performance on consecutive operations
  - Your mileage will vary!
- The expression evaluator is disabled for scalar operations that:
  - refer LOBs
  - invoke UDFs
- Column QVC1E for record ID 1000 in database monitor traces indicates whether expression evaluator was used to run the procedural statement
  - Statements converted to in-line C not recorded in dbmon

Expression Evaluator Example - 1 of 3

```
create procedure justice_for_all(out o_number_of_raises int, out o_cost_of_raises decimal(9,2))
language sql
proc_body:
begin
 declare v_avg_tenure int;
 declare v_avg_compensation decimal(9,2);
 declare v_number_of_raises int;
 declare v_cost_of_raises decimal(9,2);

 set  v_avg_tenure = 0;
 set v_avg_compensation = 0.0;


 select avg(year(current_timestamp) - year(hiredate)), decimal(avg(salary + bonus +comm),9,2)
  into   v_avg_tenure, v_avg_compensation
  from employee;

 set v_number_of_raises = 0;
 set v_cost_of_raises = 0.0;
 for_loop:
     FOR each_row AS c1 CURSOR FOR
              SELECT year(current_timestamp) - year(hiredate) as tenure,
                             salary+ bonus + comm as compensation
              FROM employee
              DO
              IF tenure > v_avg_tenure and compensation < v_avg_compensation
             THEN
                  UPDATE employee SET salary = salary + (v_avg_compensation - compensation)
                  WHERE CURRENT OF c1;
                 SET v_number_of_raises = v_number_of_raises + 1;
                 SET v_cost_of_raises = v_cost_of_raises + (v_avg_compensation - compensation);
             END IF;
      END FOR;
 SET o_number_of_raises = v_number_of_raises;
 SET o_cost_of_raises =v_cost_of_raises;
END proc_body;
```

Expression Evaluator Example - 2 of 3

- FOR loop execution until first update

Database Monitor Trace for V5R3

```
QQC21  QQ1000
OP     DECLARE C1 CURSOR FOR SELECT YEAR ( CURRENT_TIMESTAMP ) - YEAR ( HIREDATE )
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
SI     SELECT 1 INTO : H FROM QSYS2 . QSQPTABL WHERE : H : H > : H : H AND : H : H < : H : H
UP     UPDATE EMPLOYEE SET SALARY = SALARY + ( : H : H - : H : H ) WHERE CURRENT OF C1
SV     SET : H : H = : H : H + ( : H : H - : H : H )
```

Database Monitor Trace for V5R4

```
QQC21  QVC21  QQ1000
OP     N      DECLARE C1 CURSOR FOR SELECT YEAR ( CURRENT_TIMESTAMP ) - YEAR ( HIREDATE )
UP     N      UPDATE EMPLOYEE SET SALARY = SALARY + ( : H : H - : H : H ) WHERE CURRENT OF C1
SV     Y      SET : H : H = : H : H + ( : H : H - : H : H )
```

Expression Evaluator Example - 3 of 3

- List ODPs in a job after stored procedure execution

List of ODPs on V5R3

```
                         Member/     Record      File    I/O
File        Library      Device      Format      Type    Count
SYSROUTINE  QSYS2        SYSRO00001  SYSRO00001  PHY         5
QASQRESL    QSYS2        QASQRESL    QASQRESL    LGL         9
QSQPTABL    QSYS2        QSQPTABL    FORMAT0001  PHY        90
EMPLOYEE    SAMPLE       EMPLOYEE    FORMAT0001  PHY        89
QSQPTABL    QSYS2        QSQPTABL    FORMAT0001  PHY        89
EMPLOYEE    SAMPLE       EMPLOYEE    FORMAT0001  PHY         9
EMPLOYEE    SAMPLE       EMPLOYEE                PHY       469
```

List of ODPs on V5R4

```
                         Member/     Record      File    I/O
File        Library      Device      Format      Type    Count
SYSROUTINE  QSYS2        SYSRO00001  SYSRO00001  PHY         5
QASQRESL    QSYS2        QASQRESL    SYSRO00001  LGL         9
EMPLOYEE    SAMPLE       EMPLOYEE    FORMAT0001  PHY        89
EMPLOYEE    SAMPLE       EMPLOYEE    FORMAT0001  PHY         9
EMPLOYEE    SAMPLE       EMPLOYEE                PHY       469
```

Condition Handlers in Nested Compound Statements

Compound Statements Revisited

```
CREATE PROCEDURE <procname>
 [( {optional parameters} )]
[optional procedure attributes]
BEGIN [atomic]
 SQL statement; [repeatable]
END
```

- Compound Statement consists of BEGIN/END block and any number of SQL statements contained within the block
- DB2 for iSeries supports nested compound statements
  - used to scope constructs such as variables, cursor declarations, and condition handlers
  - only constructs within the same or enclosing compound statement are visible

The general structure of a compound statement:

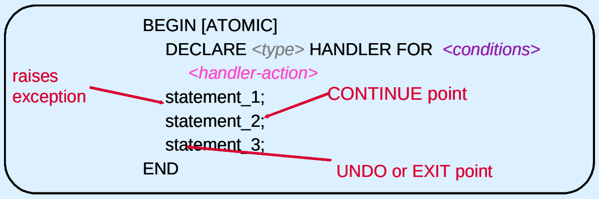
```
BEGIN
<variable declarations>
<condition declarations>
<cursor declarations>
<condition handler declarations>
<SQL statement list/procedure logic>
END
```

Condition Handler

```
BEGIN
<variable declarations>
<condition declarations>
<cursor declarations>
<condition handler declarations>
<SQL statement list/procedure logic>
END
```

- Condition Handler is an SQL statement executed when an exception or completion condition occurs within the compound statement
- A condition handler must specify
  - Handled conditions
  - Where to resume execution (CONTINUE, EXIT or UNDO)
  - Action to perform to handle the condition
    - Action can be any SQL statement including a compound statement
- Scope of the handler is limited to the containing compound statement

Three types of condition handlers:



```
BEGIN [ATOMIC]
   DECLARE <type> HANDLER FOR  <conditions>
      <handler-action>
   statement_1;
   statement_2;
   statement_3;
END
```

SQLSTATE

- SQLSTATE is a five-character string contained in DB2 Communications Area (DB2 CA)
  - Access from SQL PL requires explicit declaration
    - DECLARE SQLSTATE CHAR(5); -- must be outer most scope
- Automatically set by DB2 after each SQL operation
- SQLSTATE values
  - '00000', success
  - '01XXX', warning
  - '02000', no data was found on select/fetch, insert, update or delete
  - Everything else unsuccessful

Handler Conditions

- General conditions:
  - SQLWARNING, SQLEXCEPTION, NOT FOUND
- Can assign name to specific condition:
  - DECLARE not_found CONDITION FOR SQLSTATE '02000';
- SQLSTATE values
  - User-defined
    - Classes that begin with ‘7’ through ‘9’, or ‘I’ through ‘Z’ may be defined. Use any sub-class.
  - Reserved for DB2
    - Classes that begin with ‘0’ through ‘6’, or ‘A’ through ‘H’ are reserved for the database

Condition Declaration

BEGIN
<variable declarations>
<condition declarations>
<cursor declarations>
<condition handler declarations>
<SQL statement list/procedure logic>
END

- Condition name allows user friendly alias (e.g., duplicate_key) to be associated with more-cryptic SQLSTATE values such as '02505'
- Condition name must be unique within the Stored Procedure

For example:

```
DECLARE DUPLICATE_KEY CONDITION FOR SQLSTATE '02505';
DECLARE EXIT HANDLER FOR DUPLICATE_KEY
  BEGIN
   GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT;
   INSERT INTO jm_debug ( SQLTEXT, T1 )
   VALUES ( 'Level2-Exit Handler: Error message: ' ||
          SQLERRM, CURRENT TIMESTAMP) WITH NC;
  END ;
```

SIGNAL and RESIGNAL Statements

```
BEGIN
DECLARE too_many_rows CONDITION
        FOR SQLSTATE '70001';
DECLARE EXIT HANDLER FOR too_many_rows
  BEGIN
  INSERT INTO jm_debug
   VALUES('Too_many_rows invoked..');
  RESIGNAL;
  END;
SELECT count(*) INTO v_num_rows
  FROM item_fact;
IF v_num_rows > upper_limit THEN
   SIGNAL too_many_rows
     SET MESSAGE_TEXT =
     'No of row limit exceeded.';
END
```

- RESIGNAL statement resignals an exception condition
  - Can be coded only as part of the condition handler
  - In the Simplest form has no parameters and used to reissue the same condition that caused the handler to be invoked

- SIGNAL statement causes error or warning condition to be returned with the specified SQLSTATE & optional message text
  - Message text can be up to 70 bytes in length, longer messages will be truncated without warning
- If a handler for the signaled exception exists, exception is handled and control transferred to handler

Non-trivial example - 1 of 3

```
CREATE PROCEDURE SQLTUTOR.P_NESTED_TEST ( IN P_TABLE_NAME VARCHAR(128),
                                  OUT P_ERROR_IND_OUT CHARACTER(1) )
LANGUAGE SQL
SPECIFIC P_NESTED_TEST
Level_1 :
BEGIN -- Main Procedure Body; Level-1 Compound Statement
DECLARE V_REF_CURSOR_TEXT VARCHAR ( 1024 ) ;
DECLARE V_SQL_STMT_EXEC1 VARCHAR ( 1024 ) ;
DECLARE SQLERRM VARCHAR ( 4000 ) DEFAULT '' ;
DECLARE V_AVG_PRICE DOUBLE PRECISION ;
DECLARE V_ROWS_INSERTED INTEGER DEFAULT 0 ;
DECLARE OBJECT_NOT_FOUND CONDITION FOR SQLSTATE '42704' ;
DECLARE COFFEES_QUERY_FAILED CONDITION FOR SQLSTATE '70010' ;
DECLARE COFFEES_UNKNOWN_AVG_PRICE CONDITION FOR SQLSTATE '70019' ;
DECLARE COFFEES_INSERT_FAILED CONDITION FOR SQLSTATE '70020' ;
DECLARE C_GET_COFFEES CURSOR FOR V_CUR_STMT ;
-- Exit handler scoped to the main procedure body
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
 GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
 SET P_ERROR_IND_OUT = 'Y' ;
 INSERT INTO JM_DEBUG ( SQLTEXT )
 VALUES ( 'Level_1-Exit Handler for sqlexception: Message : '
          || SQLERRM ) WITH NC ;
 RESIGNAL ;
END ;
-- Level_1 compound statement body starts here
SET V_REF_CURSOR_TEXT = 'SELECT avg(price)  FROM ' || TRIM ( P_TABLE_NAME ) ;
PREPARE V_CUR_STMT FROM V_REF_CURSOR_TEXT ;
INSERT INTO JM_DEBUG ( SQLTEXT )
       VALUES ( 'Level_1-Main Procedure Body: V_CUR_STMT prepared' );
-- Level_2_1 compound statement continued on the next page
```

Non-trivial example - 2 of 3

```
-- Level_2_1 compound statement
Level_2_1:
  BEGIN
    -- exit handler scoped to compound statement Level_2_1
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
      BEGIN
        GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
        INSERT INTO JM_DEBUG ( SQLTEXT )
               VALUES ('     Level_2_1-Exit Handler for Select: Message: '
               || SQLERRM ) WITH NC ;
        SIGNAL COFFEES_QUERY_FAILED SET MESSAGE_TEXT = SQLERRM ;
      END ;
    -- continue handler scoped to  scoped to compound statement Level_2_1
    DECLARE CONTINUE HANDLER FOR COFFEES_UNKNOWN_AVG_PRICE
      BEGIN
        GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
        INSERT INTO JM_DEBUG ( SQLTEXT )
          VALUES ('     Level_2_1-Handler for SQLSTATE 70019: Message: '
          || SQLERRM ) WITH NC ;
        SET V_AVG_PRICE = 0.0 ;
       END ;
    -- Level_2_1 compound statement body starts here
    OPEN C_GET_COFFEES ;
    FETCH C_GET_COFFEES INTO V_AVG_PRICE ;
      CLOSE C_GET_COFFEES ;
    IF V_AVG_PRICE IS NULL THEN
      SIGNAL COFFEES_UNKNOWN_AVG_PRICE
             SET MESSAGE_TEXT = 'Unknown avg price of coffee.' ;
    END IF ;
    INSERT INTO JM_DEBUG ( SQLTEXT )
    VALUES ( '     Level_2_1 - Body: v_avg_price = '|| TRIM(CHAR(V_AVG_PRICE))) WITH NC ;
  END Level_2_1;
  -- Level_1 body resumes here
```

Non-trivial example - 3 of 3

```
  -- Level_1 body resumes here
INSERT INTO JM_DEBUG(SQLTEXT)
       VALUES ( 'Level_1-Resuming processing after end of Level_2_1 compound statement.' )
WITH NC ;
SET V_SQL_STMT_EXEC1 = 'INSERT INTO ' || TRIM ( P_TABLE_NAME )
    || ' VALUES(10, ''Colombian Supreme'', 10, 9.95, 1000, 1000)' ;
-- Level_2_2 compound statement
Level_2_2:
BEGIN
    -- exit handler scoped to compound statement Level_2_1
    DECLARE EXIT HANDLER FOR OBJECT_NOT_FOUND
      BEGIN
        GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
        INSERT INTO JM_DEBUG (SQLTEXT)
               VALUES ( '     Level_2-2-Handler for Insert: Message: '
               || SQLERRM ) WITH NC ;
        SIGNAL COFFEES_INSERT_FAILED SET MESSAGE_TEXT = SQLERRM ;
      END;
      -- Level_2_2 compound statement body starts here
      EXECUTE IMMEDIATE V_SQL_STMT_EXEC1 ;
      GET DIAGNOSTICS V_ROWS_INSERTED = ROW_COUNT ;
      INSERT INTO JM_DEBUG (SQLTEXT)
         VALUES ( '     Level_2-2-Main Body: ' || TRIM(CHAR(V_ROWS_INSERTED))
             || ' row(s) inserted in COFFEES.' ) WITH NC ;
   END Level_2_2;
-- Level_1 body resumes here
INSERT INTO JM_DEBUG (SQLTEXT)
VALUES ( 'Level_1-Resuming processing after end of Level_2_2 compound statement.' ) WITH NC ;
SET P_ERROR_IND_OUT = 'N' ;
END level_1;
```

Test Case 1 - 1 of 4

Stored Procedure Invocation

```
-- Table COFFEES is empty at first
CALL P_NESTED_TEST('COFFEES', ' ');
```

```
-- Level_2_1 compound statement body starts here
   OPEN C_GET_COFFEES ;
   FETCH C_GET_COFFEES INTO V_AVG_PRICE ;
   CLOSE C_GET_COFFEES ;
   IF V_AVG_PRICE IS NULL THEN
     SIGNAL COFFEES_UNKNOWN_AVG_PRICE
       SET MESSAGE_TEXT = 'Unknown avg price of coffee.';
   END IF ;
```

Table COFFEES is empty.
Execution proceeds until IF statement in compound statement Level_2_1 is reached.
The value of V_AVG_PRICE is NULL (unknown) at that point.

Content of JM_DEBUG
Level_1-Main Procedure Body: V_CUR_STMT prepared

Test Case 1 - 2 of 4

```
Level_2_1:
 BEGIN

   -- Level_2_1 compound statement body starts here
   OPEN C_GET_COFFEES ;
   FETCH C_GET_COFFEES INTO V_AVG_PRICE ;
     CLOSE C_GET_COFFEES ;
   IF V_AVG_PRICE IS NULL THEN
     SIGNAL COFFEES_UNKNOWN_AVG_PRICE
        SET MESSAGE_TEXT =
        'Unknown avg price of coffee.' ;
   END IF ;
...
 END Level_2_1;
```

Since V_AVG_PRICE has value of NULL the SIGNAL COFFEES_UNKNOWN_AVG_PRICE is fired.
This custom error condition corresponds to SQLSTATE '70019'

Content of JM_DEBUG
Level_1-Main Procedure Body: V_CUR_STMT prepared

Test Case 1 - 3 of 4

```
Level_2_1:
 BEGIN
  ...
 -- continue handler scoped to  scoped to compound
 -- statement Level_2_1
 DECLARE CONTINUE HANDLER
           FOR COFFEES_UNKNOWN_AVG_PRICE
   BEGIN
     GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
     INSERT INTO JM_DEBUG ( SQLTEXT )
     VALUES('Level_2_1-Handler for SQLSTATE 70019: Message: '
         || SQLERRM ) WITH NC ;
     SET V_AVG_PRICE = 0.0 ;
   END ;
   -- Level_2_1 compound statement body starts here
  ...
   IF V_AVG_PRICE IS NULL THEN
     SIGNAL COFFEES_UNKNOWN_AVG_PRICE
       SET MESSAGE_TEXT = 'Unknown avg price of coffee.' ;
   END IF ;
...
 END Level_2_1;
```

DB2 runtime tries to locate the handler for the particular condition within the same compound statement.

Condition Handler found and invoked.

V_AVG_PRICE set to 0.0

Content of JM_DEBUG
Level_1-Main Procedure Body: V_CUR_STMT prepared
Level_2_1-Handler for SQLSTATE 70019: Message: Unknown avg price of coffee.

Test Case 1 - 4 of 4

```
   -- Level_1 body resumes here
INSERT INTO JM_DEBUG(SQLTEXT)
       VALUES ( 'Level_1-Resuming processing after end of Level_2_1 .....
SET V_SQL_STMT_EXEC1 = 'INSERT INTO ' || TRIM ( P_TABLE_NAME )
    || ' VALUES(10, ''Colombian Supreme'', 10, 9.95, 1000, 1000)' ;
-- Level_2_2 compound statement
Level_2_2:
    BEGIN
      ...
      -- Level_2_2 compound statement body starts here
      EXECUTE IMMEDIATE V_SQL_STMT_EXEC1 ;
      GET DIAGNOSTICS V_ROWS_INSERTED = ROW_COUNT ;
      INSERT INTO JM_DEBUG (SQLTEXT)
         VALUES ( '     Level_2-2-Main Body: ' || TRIM(CHAR(V_ROWS_INSERTED))
              || ' row(s) inserted in COFFEES.' ) WITH NC ;
   END Level_2_2;
-- Level_1 body resumes here
 INSERT INTO JM_DEBUG (SQLTEXT)
 VALUES ( 'Level_1-Resuming processing after end of Level_2_2 compound .....
 SET P_ERROR_IND_OUT = 'N' ;
END level_1;
```

The execution successfully continues until the end of the main procedure body is reached.

Compound statement Level_2_2 inserts a new row into COFFEES.

Content of JM_DEBUG
Level_1-Main Procedure Body: V_CUR_STMT prepared
Level_2_1-Handler for SQLSTATE 70019: Message: Unknown avg price of coffee.
Level_2_1 - Body: v_avg_price = 0E0
Level_1-Resuming processing after end of Level_2_1 compound statement.
Level_2-2-Main Body: 1 row(s) inserted in COFFEES.
Level_1-Resuming processing after end of Level_2_2 compound statement.

Test Case 2

```
CALL P_NESTED_TEST('COFFEES', ' ');
```

```
Level_1 :
...
-- Exit handler scoped to the main procedure body
  LARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
 GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
 SET P_ERROR_IND_OUT = 'Y' ;
 INSERT INTO JM_DEBUG (SQLTEXT) VALUES( 'Level_1-Exit Handler ...
 RESIGNAL ;
END ;
Level_2_2:
    -- exit handler scoped to compound statement Level_2_1
     CLARE EXIT HANDLER FOR OBJECT_NOT_FOUND
    ...
      -- Level_2_2 compound statement body starts here
      EXECUTE IMMEDIATE V_SQL_STMT_EXEC1 ;
      GET DIAGNOSTICS V_ROWS_INSERTED = ROW_COUNT ;
      INSERT INTO JM_DEBUG (SQLTEXT) VALUES( '     Level_2-2-Main..
   END Level_2_2;
-- Level_1 body resumes here
...
END level_1;
```

The execution proceeds until the EXECUTE IMMEDIATE statement is reached.

The row with the key value 10 in the first column already exists.

DB2 runtime throws a duplicate key exception

First, the DB2 runtime tries to locate a handler for this specific condition in the compound statement Level_2_2.

Then, the runtime searches for a general handler, which also does not exist in Level_2_2.

Control returns to the main procedure body with the error condition pending. The runtime locates the general handler for SQL exceptions and invokes it.

Content of JM_DEBUG
Level_1-Main Procedure Body: V_CUR_STMT prepared
Level_2_1 - Body: v_avg_price = 9.9499999999999993E0
Level_1-Resuming processing after end of Level_2_1 compound statement.
Level_1-Exit Handler for sqlexception: Message : Duplicate key value specified.

Lessons Learned from Test Cases

- Condition handlers are scoped to the compound statement in which they are declared
- Handlers for specific SQLSTATEs are invoked BEFORE handlers for general conditions (SQLEXCEPTION, SQLWARNING, NOT FOUND)
- Unhandled conditions are returned to the upper level compound statement - eventually to caller
- Custom SQLSTATE and a handler can be used to deal with user-defined error conditions in SQL PL

Implicit Schema Qualification for
Static and Dynamic SQL Statements

Default Schema

- Default schema is used to implicitly qualify unqualified database object names
  - Alias, constraint, external program, index, nodegroup, package, sequence, table, trigger, and view names
- Default Schema for static SQL statements
  - For SQL naming (*SQL), the default schema is set to the authorization ID (user profile name) in effect ***when the stored procedure is created.***
  - For system naming (*SYS), the default schema is set to the job's library list (*LIBL).
- Default Schema for dynamic SQL statements:
  - For SQL naming (*SQL), the default schema is set to the authorization ID (user profile name) in effect ***when the stored procedure is executed.***
  - For system naming (*SYS), the default schema is set to the job's library list (*LIBL).

Implicit Qualification Example

```
CREATE PROCEDURE TestDynSQL ( )
     LANGUAGE SQL
MainBody: BEGIN
DECLARE STMT VARCHAR ( 128 ) ;
DECLARE SQLERRM VARCHAR ( 256 ) DEFAULT '';
DECLARE ErrorIndicator CHAR(1) DEFAULT 'N';
Static_Stmt: BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
      GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
      INSERT INTO jm_debug ( SQLTEXT )
        VALUES ( 'Static Statement Failed: '||SQLERRM) WITH NC;
      SET ErrorIndicator = 'Y';
    END;
  INSERT INTO jm_debug VALUES('Static Statement Succeeded.');
END Static_Stmt;
Dynamic_Statement: BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
    GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
   INSERT INTO jm_debug ( SQLTEXT )
     VALUES ( 'Dynamic Statement Failed: '||SQLERRM) WITH NC;
   SET ErrorIndicator = 'Y';
  END;
  SET STMT = 'INSERT INTO jm_debug
      VALUES(''Dynamic Statement Succeeded.'')';
  PREPARE S1 FROM STMT ;
  EXECUTE S1;
END Dynamic_Stmt;
IF ErrorIndicator = 'Y' THEN
    SIGNAL SQLSTATE '70000'  SET
       message_text='There were errors. Check jm_debug for details.';
END IF;
END  MainBody;
```

The stored procedure attempts to insert two rows into the jm_debug table.

Static insert SQL statement is used. The errors in the Static_Stmt compound statement are intercepted and handled by the continue handler.

A dynamic SQL statement is used to insert a second row.

The errors are handled by the continue handler in the Dynamic_Stmt compound statement .

Test Case 3

```
Static_Stmt: BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  ...
✓ INSERT INTO jm_debug VALUES('Static Statement Succeeded.');
END Static_Stmt;
Dynamic_Statement: BEGIN
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
    GET DIAGNOSTICS EXCEPTION 1 SQLERRM = MESSAGE_TEXT ;
   INSERT INTO jm_debug ( SQLTEXT )
     VALUES ( 'Dynamic Statement Failed: '||SQLERRM) WITH NC;
   SET ErrorIndicator = 'Y';
  END;
  SET STMT = 'INSERT INTO jm_debug
      VALUES(''Dynamic Statement Succeeded.'')';
  PREPARE S1 FROM STMT ;
✗ EXECUTE S1;
END Dynamic_Stmt;
IF ErrorIndicator = 'Y' THEN
    SIGNAL SQLSTATE '70000'  SET
       message_text='There were errors. Check jm_debug for
details.';
END IF;
END  MainBody;
```

Invocation:

The stored procedure is created by user DB2ADMIN.

The jm_debug table resides in the DB2ADMIN schema.

The naming convention in effect is *SQL.

User called DB2GURU with (*ALLOBJ) invokes the stored procedure.

DB2GURU sets the PATH :

`SET PATH = DB2ADMIN;`

Excution:

Static statement succeeds.

The dynamic statement fails.

DB2 uses the runtime authorization ID to resolve the unqualified name.

No jm_debug table in schema DB2GURU.

DB2 signals an SQL exception.

Content of JM_DEBUG
Static Statement Succeeded.
Dynamic Statement Failed: JM_DEBUG in DB2GURU type *FILE not found.

Implicit Qualification - New Options on CREATE PROCEDURE

- Default collection (DFTRDBCOL)
  - Specifies schema used for unqualified names of tables, views, indexes, and SQL packages
  - Applies only to static SQL statements.
- Dynamic default collection (DYNDFTCOL)
  - Specifies if default schema name specified for DFTRDBCOL is also used for dynamic statements
- By default, these two attributes are set to: DFTRDBCOL(*NONE), DYNDFTCOL(*NO)
  - Default DB2 behavior observed in Test Case 3
  - DFTRDBCOL(*NONE) means no default schema has been set

```
CREATE PROCEDURE TestDynSQL ( )
   LANGUAGE SQL
   SPECIFIC testdynsql
   MODIFIES SQL DATA
   SET OPTION DYNDFTCOL=*YES
BEGIN
…
END
```

Test Case 4

```
CREATE PROCEDURE TestDynSQL ( )
   LANGUAGE SQL
   SET OPTION DYNDFTCOL=*YES

Static_Stmt: BEGIN
   ...
✓ INSERT INTO jm_debug VALUES('Static Statement Succeeded.');
END Static_Stmt;
Dynamic_Statement: BEGIN
   ...
  SET STMT = 'INSERT INTO jm_debug
      VALUES(''Dynamic Statement Succeeded.'')';
 ✓PREPARE S1 FROM STMT ;
  EXECUTE S1;
END Dynamic_Stmt;
...
END  MainBody;
```

Procedure Creation:
To avoid hardcoding, the DFTRDBCOL keyword has been deliberately omitted.
The naming convention in effect is *SQL.
The DFTRDBCOL is inherited from the current environment.
At deployment default schema set to DB2ADMIN.
The program object has the following attributes:
DFTRDBCOL=DB2ADMIN
DYNDFTCOL=*YES

Execution:
Both static and dynamic unqualified statements are implicitly qualified with DB2ADMIN.
The stored procedure completes successfully

Content of JM_DEBUG
Static Statement Succeeded.
Dynamic Statement Succeeded.

Access Authority - Options on CREATE PROCEDURE

- Object access authority is managed by USRPRF and DYNUSRPRF attributes
- User profile (USRPRF)
  - Specifies user profile used when the compiled routine is run, including the authority to each object in static SQL statements
  - The profile of *OWNER or *USER is used
- Dynamic user profile (DYNUSRPRF)
  - Specifies user profile used for dynamic SQL statements
  - The profile of program's user or program's owner is used
- By default, these parameters are set to USRPRF(*NAMING) and DYNUSRPRF(*USER)
  - Translates to USRPRF(*OWNER) and DYNUSRPRF(*USER) for SQL naming

```
CREATE PROCEDURE TestDynSQL ( )
   LANGUAGE SQL
   SPECIFIC testdynsql
   MODIFIES SQL DATA
   SET OPTION DYNDFTCOL=*YES, DYNUSRPRF=*OWNER
BEGIN
…
END
```

Test Case 4 Revisited 1 of 2

```
CREATE PROCEDURE TestDynSQL ( )
   LANGUAGE SQL
   SET OPTION DYNDFTCOL=*YES

Static_Stmt: BEGIN
   ...
✓ INSERT INTO jm_debug VALUES('Static Statement Succeeded.');
END Static_Stmt;
Dynamic_Statement: BEGIN
   ...
  SET STMT = 'INSERT INTO jm_debug
      VALUES(''Dynamic Statement Succeeded.'')';
  PREPARE S1 FROM STMT ;
✗ EXECUTE S1;
END Dynamic_Stmt;
...
END  MainBody;
```

Invocation:
The stored procedure is created by user DB2ADMIN.
The jm_debug table resides in the DB2ADMIN schema.
The naming convention in effect is *SQL.
*PGM object attributes:
DFTRDBCOL=DB2ADMIN
DYNDFTCOL=*YES
USRPRF(*OWNER)
DYNUSRPRF(*USER)
User called DB2USR (*USER) invokes the stored procedure.

Execution:
Both static and dynamic unqualified statements are implicitly qualified with DB2ADMIN.
Current user DB2USR has no access authority to JM_DEBUG therefore dynamic statement fails.

Content of JM_DEBUG
Static Statement Succeeded.
Dynamic Statement Failed: Not authorized to object JM_DEBUG in DB2ADMIN type *FILE.

Test Case 4 Revisited 2 of 2

```
CREATE PROCEDURE TestDynSQL ( )
   LANGUAGE SQL
    SET OPTION DYNDFTCOL=*YES, DYNUSRPRF=*OWNER

Static_Stmt: BEGIN
   ...
 ✓ INSERT INTO jm_debug VALUES('Static Statement Succeeded.');
END Static_Stmt;
Dynamic_Statement: BEGIN
  ...
  SET STMT = 'INSERT INTO jm_debug
      VALUES(''Dynamic Statement Succeeded.'')';
  PREPARE S1 FROM STMT ;
 ✓ EXECUTE S1;
END Dynamic_Stmt;
...
END  MainBody;
```

Also Required:

```
grant execute on procedure testdynsql to db2usr ;
GRTOBJAUT OBJ(DB2ADMIN) OBJTYPE(*LIB) USER(DB2USR)
AUT(*USE)
```

Invocation:
The stored procedure is created by user DB2ADMIN.
The jm_debug table resides in the DB2ADMIN schema.
The naming convention in effect is *SQL.
*PGM object attributes:
DFTRDBCOL=DB2ADMIN
DYNDFTCOL=*YES
USRPRF(*OWNER)
DYNUSRPRF(*OWNER)
User called DB2USR (*USER) invokes the stored procedure.

Execution:
Both static and dynamic unqualified statements are implicitly qualified with DB2ADMIN.
Owner user(DB2ADMIN) has access authority to JM_DEBUG. Both static and dynamic statements succeed..

Content of JM_DEBUG
Static Statement Succeeded.
Dynamic Statement Succeeded.

Software Prerequisites

- Always install the latest database group PTF
  - SF99502 for V5R2, SF99503 for V5R3, or SF99504 for V5R4

	OS/400 V5R2	i5/OS V5R3	i5/OS V5R4
Expression Evaluator	N/A	N/A	in base
Complex Nested Compund Statement	SI17232, SI17233	SI18929	in base
Implicit Qualification (DFTRDBCOL, DYNDFTCOL options)	SI16196 SI16197 SI16198	SI18022 SI18024 SI18025 SI18029	in base

Additional Info

- Improving SQL procedure performance
http://www-304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/VSHA-6WPNQL

- Improve Your Productivity with Significant Enhancements in DB2 SQL,
MC Press Online
http://www.mcpressonline.com/mc/1@1.VJfIcI48IGm.0@.6b2a798b

- DB2 SQL PL Essential Guide for DB2 UDB on Linux, UNIX, Windows, i5/OS, and z/OS,
IBM Press, ISBN 0-13-147700-5

- Cross-Platform DB2 Stored Procedures: Building and Debugging,
ITSO Redbook, SG24-5485

- Stored Procedures, Triggers and User Defined Functions on DB2 Universal Database for iSeries,
ITSO Redbook, SG24-6503