



# DB2 for i: SQL Stored Procedures

*Kent Milligan, IBM Rochester  
IBM i – DB2 & ISV Enablement*

*THE NEW POWER EQUATION*

© 2009 IBM Corporation



## What is a Stored Procedure?

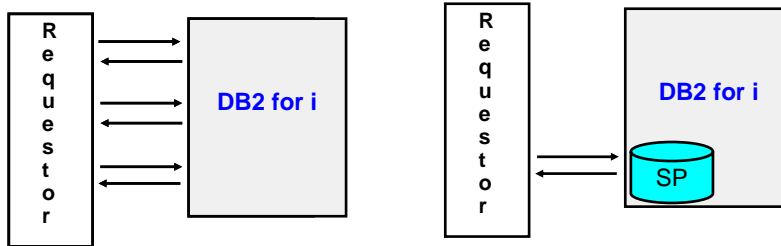
- **Just a called program**
  - Called from SQL-based interfaces via SQL CALL statement
- **Supports input and output parameters**
  - Result sets on some interfaces
- **Follows security model of IBM i**
  - Enables you to secure your data
  - IBM i adopted authority model can be leveraged
- **Useful for moving host-centric applications to distributed applications**

*THE NEW POWER EQUATION*

© 2009 IBM Corporation

## What is a Stored Procedure?

- Performance savings in distributed computing environments by dramatically reducing the number of flows (requests) to the database engine
  - One request initiates multiple transactions and processes



- Performance improvements further enhanced by the option of providing result sets back to ODBC, JDBC, .NET & CLI clients

*THE NEW POWER EQUATION*

## Recipe for a Stored Procedure...

- 1 Create it
 

```
CREATE PROCEDURE total_val (IN Member# CHAR(6),
                             OUT total DECIMAL(12,2))

LANGUAGE SQL
BEGIN
  SELECT SUM(curr_balance) INTO total
  FROM accounts
  WHERE account_owner=Member# AND
         account_type IN ('C','S','M')
END
```
- 2 Call it (from an SQL interface – SQL CALL) over and over
 

```
CALL total_val('123456', :balance)
```

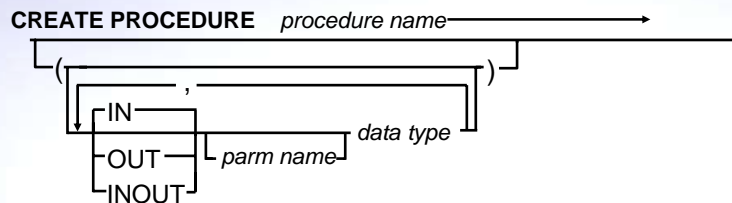
*THE NEW POWER EQUATION*

## Stored Procedures

- **DB2 for i supports two types of stored procedures**
  - EXTERNAL
    - Register high-level language program(RPG, Java, C, etc) as a stored procedure
    - Procedure may or may not use SQL
  - SQL
    - Entire procedure is coded with SQL
    - Based on SQL Standard (PSM)
      - Natural fit for SQL programmers on IBM i
      - Similar to PL/SQL and T-SQL
      - Same language used for SQL Triggers & Functions
- **SQL CREATE PROCEDURE statement used for both types**

*THE NEW POWER EQUATION*

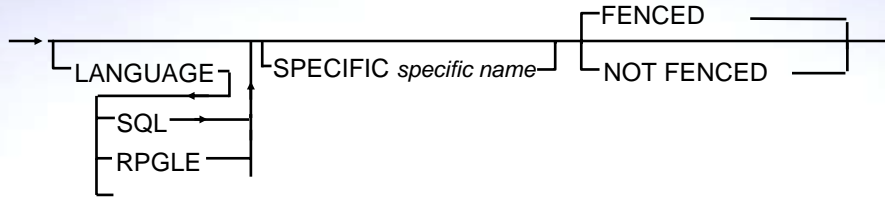
## Create Procedure options



- Procedure name + number of parameters make a unique signature
- Up to 1024 parameters can be passed (pre-V5R4 limit is 253)

*THE NEW POWER EQUATION*

## Create Procedure options



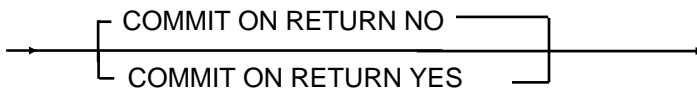
- **LANGUAGE** - Language of procedure (SQL, C, RPG...)
- **SPECIFIC** – unique "short" name for SQL procedures
  - Automatically generated if not specified
  - Can be used to control C program object name
- **FENCED/NOT FENCED** – No effect on procedures
  - NOT FENCED can improve performance of SQL Functions

*THE NEW POWER EQUATION*

## Create Procedure options



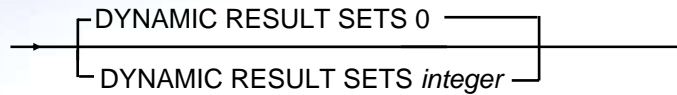
- **NEW SAVEPOINT LEVEL** – new savepoint (transaction level) created on entry



- **COMMIT ON RETURN YES** - DB2 issues a commit if the procedure successfully returns
  - Work done by both the invoker and procedure is committed
  - Result Set cursors must be declared WITH HOLD to be usable after the commit operation

*THE NEW POWER EQUATION*

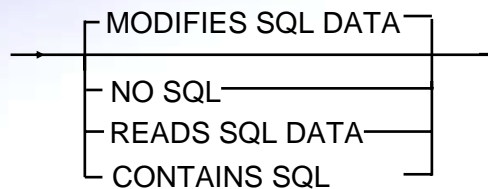
## Create Procedure options



- **RESULT SETS** specifies max number of result sets that can be returned from procedure.
  - Only returned to ODBC, JDBC, .NET & CLI clients
  - more on result sets later on...

*THE NEW POWER EQUATION*

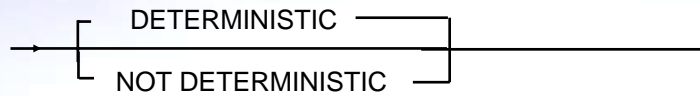
## Create Procedure options



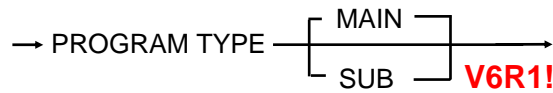
- **MODIFIES SQL DATA** – Most any SQL statement allowed
- **READS SQL DATA** – Read Only statements
- **CONTAINS SQL** – Simple local statements (SET, DECLARE)
- **NO SQL** – No SQL allowed (external procedures only)

*THE NEW POWER EQUATION*

## Create Procedure options



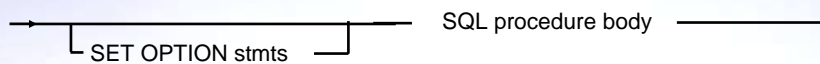
- **DETERMINISTIC** - procedure will always return the same result from successive calls with identical input arguments



- **PROGRAM TYPE** – SUB enables DB2 to create service program object. Program object created by default.

*THE NEW POWER EQUATION*

## Create Procedure options



- **SET OPTION** - set processing options
  - Naming option (\*SQL vs \*SYS), sort-sequence, SQL path, debug...
  - Example: SET DBGVIEW=\*STMT, USRPRF=\*USER
- **Most interesting options for SQL Procedures are:**
  - USRPRF for adopted authority (defaults to \*OWNER)
  - DBGVIEW for creating debuggable version of SQL Procedure
    - \*SOURCE enables SQL statement-level debug

*THE NEW POWER EQUATION*

## SQL Stored Procedure Example

```
CREATE PROCEDURE proc1 (IN Emp# CHAR(4),IN NwLvl INT)
LANGUAGE SQL Proc1_Src:
BEGIN

    DECLARE CurLvl INT;
    SELECT edlevel INTO CurLvl FROM emptbl
    WHERE empno=Emp#;

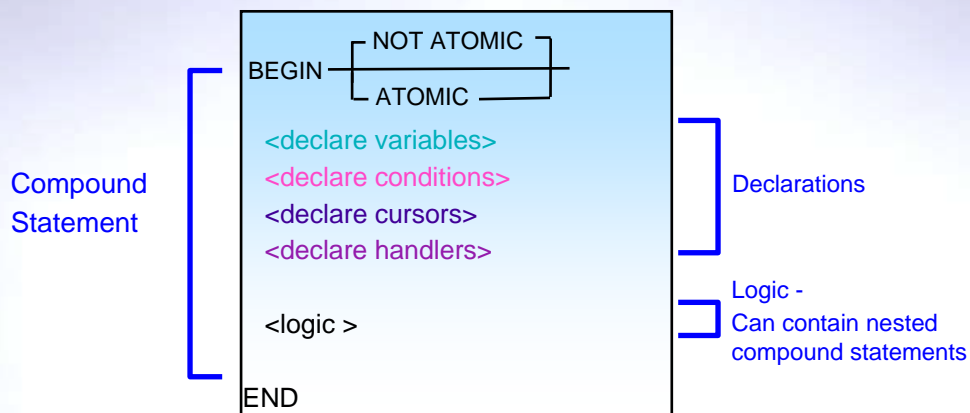
    IF NwLvl > CurLvl THEN
        UPDATE emptbl SET edlevel=NwLvl,
            salary=salary + (salary*0.05) WHERE empno=Emp#;
    END IF;

END
```

- DB2 for i uses the source to create a C program object with the IBM i pre-compiler and compiler commands

*THE NEW POWER EQUATION*

## SQL Compound Statement



- **ATOMIC**
  - all statements succeed or are rolled back.
  - COMMIT or ROLLBACK cannot be specified in the procedure
  - must also be created with COMMIT ON RETURN YES
- **NOT ATOMIC – no guarantee or atomicity**

*THE NEW POWER EQUATION*

## Basic Constructs

- **DECLARE – define variable. Initialized when procedure is called**

```
DECLARE v_midinit, v_edlevel CHAR(1);
DECLARE v_ordQuantity INT DEFAULT 0;
DECLARE v_enddate DATE DEFAULT NULL;
```

- Uninitialized variables are set to NULL

- **SET - assigning a value parameter or variable**

```
SET total_salary = emp_salary + emp_commission;
SET total_salary = NULL;
SET loc_avgsalary = (SELECT AVG(salary) FROM employees);
```

- **Comments - two types**

- Two consecutive hyphens (--)
- Bracketed comments (/ \* ... \*/)

*THE NEW POWER EQUATION*

## Basic Constructs

- **Call statement - for invoking stored procedures**
- **CALL ProcedureName(Parm1, Parm2, etc);**
  - Up to 1024 arguments allowed on CALL statement (253 prior to V5R4)
  - A parameter can contain SQL parameter, SQL variable, constant, special register, or NULL
  - Expressions not yet supported as parameters
- **Provides a mechanism for accessing system functions and APIs from an SQL Stored Procedure**

*THE NEW POWER EQUATION*

## Conditional Constructs

- IF statement
 

```
IF rating=1 THEN SET price=price * 0.95;
ELSEIF rating=2 THEN SET price=price * 0.90;
                        ELSE SET price=price * 0.80;
END IF;
```

- CASE Expression

- **First form:**

```
CASE workdept
  WHEN 'A00' THEN
    UPDATE department
      SET deptname = 'ACCOUNTING';
  WHEN 'B01' THEN
    UPDATE department
      SET deptname = 'SHIPPING';
  ...
  ELSE UPDATE department
      SET deptname = 'UNKNOWN';
END CASE;
```

- **Second form:**

```
CASE
  WHEN vardept='A00' THEN
    UPDATE department
      SET deptname = 'ACCOUNTING';
  WHEN vardept='B01' THEN
    UPDATE department
      SET deptname = 'SHIPPING';
  ...
  ELSE UPDATE department
      SET deptname = 'UNKNOWN';
END CASE;
```

*THE NEW POWER EQUATION*

## Looping Constructs

- **FOR statement - execute a statement for each row of a table**

```
FOR loopvar AS
  loopcursor CURSOR FOR
  SELECT firstname, middinit, lastname FROM emptbl
  DO
    SET fullname=lastname||', ' || firstname||' ' || middinit;
    INSERT INTO namestbl VALUES( fullname );
  END FOR;
```

- **Allows columns in FOR SELECT statement to be accessed directly without host variables**
- **Cursor can be used in WHERE CURRENT OF... operation**

*THE NEW POWER EQUATION*

## Looping Constructs

- **Other looping constructions**
  - LOOP - repeat forever
    - Use LEAVE statement to end loop
  - REPEAT...UNTIL – exit condition checked at end
  - WHILE – exit condition checked on entry  
END LOOP;
- **Loop control statements**
  - LEAVE – leaves/ends loop
  - ITERATE – go to top of loop

*THE NEW POWER EQUATION*

## Looping examples

LOOP Example -

```
fetch_loop:
LOOP
  FETCH cursor1 INTO
    v_firstname, v_lastname;
  IF SQLCODE <> 0 THEN
    LEAVE fetch_loop;
  END IF;
  ...
END LOOP;
```

REPEAT Example -

```
r_loop:
REPEAT
  FETCH cursor1 INTO
    v_firstname, v_lastname;
  ...
UNTIL SQLCODE <> 0
END REPEAT;
```

WHILE Example -

```
while_loop:
WHILE at_end=0 DO
  FETCH cursor1 INTO
    v_firstname, v_lastname;
  IF SQLCODE <> 0 THEN
    SET at_end = 1;
  END IF;
  ...
END WHILE;
```

NOTE: Though they look similar,  
each example works differently!

*THE NEW POWER EQUATION*

## Feedback & Error Handling

- **GET DIAGNOSTICS**
  - Retrieve information about last statement executed
    - Row\_count, return\_status, error status...
  - CURRENT or STACKED
    - CURRENT – statement that was just executed
    - STACKED – statement before error handler was entered
      - Only allowed within error handler
  - Example:
 

```
DECLARE update_counter INTEGER;
...
UPDATE orders SET status='LATE'
WHERE ship_date < CURRENT DATE;
GET DIAGNOSTICS update_counter = ROW_COUNT;
...
```
- **Example:**

*THE NEW POWER EQUATION*

## Feedback & Error Handling

- **SQLSTATE & SQLCODE accessed by declaring variables with those names**
  - DB2 will automatically update these variables after each statement
  - **NOTE:** Every procedural statement is potentially an SQL statement. These variables may need to be saved after every statement
  - Sample usage:
 

```
DECLARE SQLSTATE CHAR(5);
DECLARE SQLCODE INTEGER;

DELETE FROM tablex WHERE col1=100;
IF SQLSTATE='02000' THEN ....
```

*THE NEW POWER EQUATION*

## Feedback & Error Handling

### Conditions and Handlers

- **CONDITION**
  - DECLARE *condition name* CONDITION FOR *string constant*;**
  - Allows alias for cryptic SQLSTATE
  - Condition name must be unique within the Stored Procedure
  
- **HANDLER**
  - DECLARE *type* HANDLER FOR *condition*;**
  - *Type*
    - UNDO - rollback statements in compound statement (must be ATOMIC)
    - CONTINUE – continue processing
    - EXIT – exit compound statement
  - *Condition*
    - Defined condition (above)
    - SQLSTATE 'xyzzy'
    - SQLWARNING, NOT FOUND, SQLEXCEPTION

*THE NEW POWER EQUATION*

## Feedback & Error Handling Example

```
CREATE PROCEDURE proc1()
...
BEGIN
  -- row not found condition
  DECLARE row_not_fnd CONDITION FOR '02000';

  DECLARE CONTINUE HANDLER FOR row_not_fnd
    SET at_end='Y'; -- set local variable at_end
  ...
  DELETE FROM tablex WHERE hiredate >= '04/01/2011';

END
```

*THE NEW POWER EQUATION*

## Feedback & Error Handling

- SIGNAL & RESIGNAL should be used to pass back error or status to the invoker
  - **SIGNAL:** **SIGNAL** *condition info* **SET** *assign value*;
    - Condition info – condition name or SQLSTATE 'xyzz'
    - SET clause provides ability to pass along additional diagnostic information
      - MESSAGE\_TEXT most commonly used
      - Values that can be retrieved via GET DIAGNOSTICS
  - **RESIGNAL:** **RESIGNAL** [*condition info* **SET** *assign value*];
    - Can be used only within handler
    - Can just RESIGNAL – “bracket” info is options
    - Condition info – condition name or SQLSTATE 'xyzz'
    - SET clause provides ability to pass along additional diagnostic information
      - MESSAGE\_TEXT most commonly used
      - Values that can be retrieved via GET DIAGNOSTICS
- SIGNAL/RESIGNAL information is copied back to the SQLCA of the stored procedure invoker
  - **EXAMPLE:** VB program could retrieve the SQLSTATE and message text via the Connection object (Conn.Error(i).SQLSTATE & Conn.Error(i).Description)

*THE NEW POWER EQUATION*

## Signal & Resignal Example

```
CREATE PROCEDURE Change_Salary(IN i_empno CHAR(6),
                              IN i_change DEC(9,2) )
  SPECIFIC CHGSAL LANGUAGE SQL
BEGIN

  DECLARE EXIT HANDLER FOR SQLSTATE '38S01'
    RESIGNAL SQLSTATE '38S01'
    SET MESSAGE_TEXT = 'CHGSAL: Change exceeds limit.';

  DECLARE EXIT HANDLER FOR SQLSTATE '02000'
    SIGNAL SQLSTATE '38S02'
    SET MESSAGE_TEXT = 'CHGSAL: Invalid employee nbr.';

  -- check, if the new compensation within the limit
  IF (i_change > 25000)
    THEN SIGNAL SQLSTATE '38S01';
  END IF;

  UPDATE employee SET salary=v_salary + i_salary WHERE empno = i_empno;

END
```

*THE NEW POWER EQUATION*

## Feedback & Error Handling

- **RETURN** statement can be used to communicate high-level success/failure status to caller
  - **RETURN <optional integer value>;**
  - If no return statement not specified, then...
    - If SQLCODE  $\geq 0$ , then return value set to a value of 0
    - If SQLCODE  $< 0$ , then return value set to -1
- **Accessing the return value**
  - when invoked by another procedure  
**GET DIAGNOSTICS statusvar = RETURN\_STATUS;**
  - **"?=CALL <procedure name>"** syntax common in ODBC and JDBC
  - Returned in SQLERRD[0]

*THE NEW POWER EQUATION*

## RETURN Example

```

CREATE PROCEDURE ModAgency(IN agencyVID INTEGER,
  IN agencyNUM INTEGER, IN agencyID INTEGER, IN agentNID INTEGER)
...
BEGIN
...
  SET CurrentDT = CURRENT_TIMESTAMP;

  UPDATE agency
  SET agency_vid=agencyVID, agency_num=agencyNUM,
  agent_NID=agentNID, updated_date=CurrentDT
  WHERE agency_ID = agencyID;

  GET DIAGNOSTICS rcount = ROW_COUNT;
  IF (rcount <> 1) THEN
    GOTO UPD_FAILURE;
  ELSE
    GOTO SUCCESS;
  END IF;
...

  SUCCESS:      RETURN 0;
  INS_FAILURE:  RETURN 900;
  UPD_FAILURE:  RETURN 901;
END;

```

*THE NEW POWER EQUATION*



## Moving Procedures into Production

- **Remember: DB2 implements SQL procedures using C programs**
  - QSYS2/SYSPROCS catalog updated each time that an SQL procedure is created
  - Program object location found in EXTERNAL\_NAME column
- **Approach: Save & Restore program object that gets created by DB2**
  - C program object is tagged so that DB2 can recognize it as an SQL Stored Procedure (in most cases - see next chart)
  - Procedure created in restore library regardless of the CREATE PROCEDURE source
  - If SPECIFIC specified on Create Procedure statement, then it must be unique - system only generates unique specific name when it's not specified
  - Program restore always completes, the database catalogs might not get updated

*THE NEW POWER EQUATION*

© 2009 IBM Corporation



## Moving Procedures into Production

- **DB2 tries to automatically recognize the C program on the restore as an SQL Stored Procedure, but there are exceptions....**
  - If DB2 does not find a matching procedure in the catalogs, then the C program is registered as an SQL Stored Procedure
  - If DB2 finds **one** procedure with the same name (differences in parameters ignored), catalog entries for the existing procedure are dropped and the new program object is registered as an SQL Stored Procedure.
  - If DB2 finds one or more procedures with the same name and a different signature (ie, different parms), then the restored program will be registered as a procedure with the same name *(and possibly overlay the program object for the existing procedure)*
    - When parameters have changed it is probably best to drop the existing procedure before the restore

*THE NEW POWER EQUATION*

© 2009 IBM Corporation

## Tool Considerations

- **System i Navigator**
  - A simple graphical editor for developing SQL Procedures
  - Runtime debug by displaying contents of result sets and output parameters
  - Integration with IBM i Graphical Debugger
  
- **IBM Data Studio Developer**
  - More advanced graphical editor, also includes graphical debugger
  - Download trial at:  
[ibm.com/software/data/db2/express/download.html](http://ibm.com/software/data/db2/express/download.html)
  - Connectivity requirements
    - DB2 Connect license
    - Requires a JDBC connection with the System i Toolbox JDBC driver

*THE NEW POWER EQUATION*

## System i Navigator – Debugger Integration

The screenshot displays the 'Run SQL Scripts' menu in System i Navigator. The menu items and their keyboard shortcuts are:

- All (Ctrl+R)
- From Selected (Ctrl+T)
- Selected (Ctrl+Y)
- Stop After Current
- Cancel Request
- Syntax Check (Ctrl+K)
- Debugger (Ctrl+D)

The background window shows configuration fields for a job named 'QZDASOINIT' with user 'QUSER' and job number '325421'. Other fields include 'Program', 'Initialization command', 'Classpath', 'Host source path', and 'Client source path', each with an 'Edit...' button.

**>>>> Graphical Debugger White Paper:**  
[ibm.com/servers/enable/site/education/abstracts/sqldebug\\_abs.html](http://ibm.com/servers/enable/site/education/abstracts/sqldebug_abs.html)

*THE NEW POWER EQUATION*

## SQL Procedure Debug

```

1 CREATE PROCEDURE KMTST . PP2 ( ) LANGUAGE SQL SET OPTION DBGVIEW
2 BEGIN
3 DECLARE X INT;
4 DECLARE Y INT;
5 SET X = 0;
6 SET Y = - 9;
7 SET Y = ABSVAL ( ( X + 1 ) * Y );
8 END;

144      goto SQL_END_PP3; }
145 17 SQLP_L1.Y = -9;
146 18 SQLP_L1.SQLP_I2 = 0;
147 19 if (SQLPROCH(%sqlca) == 1) {
148 20 memcpy(%sqlca_sav, %sqlca, sizeof(sqlca_sav));
149 21 memcpy(%sqlca, %sqlca_sav, sizeof(sqlca_sav));
150 22 goto SQL_END_PP3; }
151 /****
152 EXEC SQL SET :SQLP_L1.Y :SQLP_L1.SQLP_I2 = ABSVAL ( (
153 LP_L1.Y :SQLP_L1.SQLP_I2 )
154 ****/

```

*THE NEW POWER EQUATION*

## SQL Procedure \*SOURCE Debug Tips

- Accessing SQL variables & parameters

Parameters:

EVAL P22.PARM1

Variables:

EVAL SP

EVAL SP.X

EVAL \*SP.Z :S 5

```

CREATE PROCEDURE p22(IN parm1 INTEGER)
LANGUAGE SQL
SET OPTION DBGVIEW=*SOURCE
sp: BEGIN
DECLARE x,y INT;
DECLARE z CHAR(5);
SET x = parm1;
SET y =-9;
SET y = absval((x+1)*y);
SET z = 'ABCDE';
END;

```

>>>> *Graphical Debugger White Paper:*

[ibm.com/servers/enablen/site/education/abstracts/sqldebug\\_abs.html](http://ibm.com/servers/enablen/site/education/abstracts/sqldebug_abs.html)

*THE NEW POWER EQUATION*

## Result Sets & Procedures

- **Stored procedures in combination with result sets can drastically reduce network trips by returning blocks of results**
- **Stored procedures that return result sets can only be called by the following interfaces**
  - System i Access ODBC, OLE DB & ADO.NET middleware
  - SQL CLI
  - Toolbox JDBC driver
  - Native JDBC driver
  - DB2 Connect
  - IBM i DRDA Connections
- **Result sets are returned via open SQL cursors**

*THE NEW POWER EQUATION*

## Result Set Example

CALL RegionCustList(16)

EMPNO	FIRSTNAME	LASTNAME
000050	JOHN	GEYER
000060	IRVING	STERN
000070	EVA	PULASKI
000090	EILEEN	HENDERSON
000130	DELORES	QUINTANA
000150	BRUCE	ADAMSON
000170	MASATOSHI	YOSHIMURA
000190	JAMES	WALKER
000200	DAVID	BROWN
000260	SYBIL	JOHNSON
000320	RANDY	BROWN
000340	JASON	GOETZ
200170	KIM	GRIFFITH
200340	ROY	ALONZO

*THE NEW POWER EQUATION*



## SQL Procedures - Result Sets (Standard)

```
CREATE PROCEDURE RegionCustList ( IN Region# INTEGER )
  RESULT SET 1
  LANGUAGE SQL

BEGIN
  --Take the inputted region number, Region# and
  --return the set of customers from that region
  --via a result set

  DECLARE c1 CURSOR WITH RETURN TO CALLER FOR
    SELECT custnum, firstname,lastname
      FROM custtable WHERE region = Region#;

  OPEN c1;

  END;
```

*THE NEW POWER EQUATION*

© 2009 IBM Corporation



## SQL Procedures - Result Sets (Proprietary)

```
CREATE PROCEDURE RegionCustList ( IN Region# INTEGER )
  RESULT SET 1
  LANGUAGE SQL

BEGIN
  --Take the inputted region number, Region# and
  --return the set of customers from that region
  --via a result set

  DECLARE c1 CURSOR FOR
    SELECT custnum, firstname,lastname
      FROM custtable WHERE region = Region#;

  OPEN c1;

  SET RESULT SETS CURSOR c1;

  END;
```

*THE NEW POWER EQUATION*

© 2009 IBM Corporation

## Returning Result Sets



- After specifying a non-zero value on the **RESULT SET** clause there are several ways for a procedure to return result sets. (SQL Development Kit required)
  - SET RESULT SETS statement (non-standard) used to identify result sets - cursor or array
    - SET RESULT SETS CURSOR x2;
    - SET RESULT SETS ARRAY :tperf FOR :rowcntr ROWS (external only)
  - If SET RESULT SETS statement not specified
    - If no cursors use WITH RETURN clause, then any cursor still open at the end of the procedure is identified as a result set
    - If any cursors have specified WITH RETURN, then any cursor specifying WITH RETURN that's left open at the end of an [SQL procedure](#) is identified as a result set.
- If multiple result sets, then result sets returned in the ordered specified on SET RESULT SETS or in the order that the cursors are opened.

```
DECLARE c2 CURSOR WITH RETURN TO CALLER
FOR SELECT * FROM SYSTABLES;
```

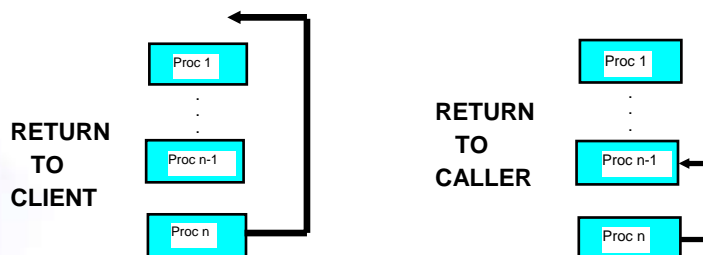
*THE NEW POWER EQUATION*

© 2009 IBM Corporation

## Considerations



- **Result Set Consumer Control**
  - RETURN TO [CLIENT](#)  
DECLARE c1 CURSOR WITH RETURN TO CLIENT FOR SELECT \* FROM t1  
  
SET RESULT SETS WITH RETURN TO CLIENT FOR CURSOR x1
  - RETURN TO [CALLER](#)  
DECLARE c1 CURSOR WITH RETURN TO CALLER FOR SELECT \* FROM t1  
  
SET RESULT SETS WITH RETURN TO CALLER FOR  
ARRAY :array1 FOR :hv1 ROWS



*THE NEW POWER EQUATION*

© 2009 IBM Corporation

## Considerations

- **Result Set Considerations:**
  - If result set returned via cursor, rows are returned starting with the current position of the cursor
  - Typical call processing (ODBC)
    - Execute the stored procedure, then use SQLBindcol and SQLFetch against CALL statement handle
    - If multiple result sets then use SQLMoreResults to move to the next the result set
  - System i Navigator SQL Script Center best tool for debugging result sets and output parameter values

*THE NEW POWER EQUATION*

## Invoker Code - Result Sets Example (ODBC)

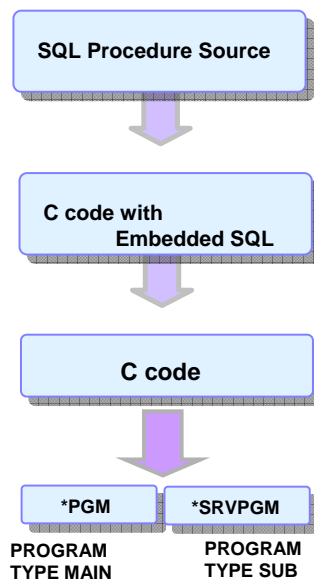
```
strcpy(stmttxt, "CALL Proc1(?,?,?)");
rc= SQLPrepare(hstmt ,stmttxt, SQL_NTS);
.....
/* Ready procedure parameters */
rc=SQLBindParameter( hstmt, 1, SQL_PARAM_INPUT, SQL_INTEGER,SQL_INTEGER,
                    sizeof( Nmpd_Year),0, ( SQLPOINTER ) &Nmpd_Year,
                    sizeof( Nmpd_Year), ( SQLINTEGER * ) &Nmi_PcbValue );
.....
/* call the procedure */
rc =SQLExecute(hstmt );
.....
/* Bind columns for the results */
rc = SQLBindCol( Hnd_Hstmt, 1, SQL_CHAR, (SQLPOINTER ) Chr_Supplier_Name,
                sizeof( Chr_Supplier_Name ), (SQLINTEGER * ) &Nmi_PcbValue );
.....
/* Scroll thru the results */
while ( rc == SQL_SUCCESS )
{
    rc = SQLFetch( hstmt );
    if ( rc == SQL_SUCCESS )
        { /* Print current results */
            ...
        }
}

/* check for more result sets */
rc = SQLMoreResults( hstmt );
if (rc <> SQL_NO_DATA_FOUND)
{ /* process the next result set */
    ...
}
```

*THE NEW POWER EQUATION*

## Performance Considerations

## Multi-phase Creation Process



- Debug Considerations
  - Use `DBGVIEW=*SOURCE` to debug “original” SQL code
- Performance Considerations
  - DB2 for i implements SQL procedural statements with one of the following methods:
    - 1) Pure C code
    - 2) Call to internal DB2 API
    - 3) Expression Evaluator (V5R4 & above)
    - 4) Generated SELECT statement
  - Coding choices and implementation directly impact what method DB2 can use
  - SRVPGM only supported by IBM i 6.1

## The need for Expression Evaluator

- SQL PL supports standard programming constructs such as loops (FOR), conditions (IF THEN) and assignments (SET a = b +1)
- Before V5R4 these simple constructs were often executed as SELECT against a "dummy" system table (QSQPPTABL)
- Negative performance impact on "tight looping" procedures
  - The SQL statement goes through the entire SQL stack
    - Statement is parsed
    - Access Plan needs to be created, stored, re-validated
    - Creation & reuse of Open Data Path (ODP)
  - Potentially large number of reusable ODPs over the QSQPPTABL table
    - ODPs processing takes CPU and ODPs increase memory foot print

## V5R4 Expression Evaluator

- A fast path evaluator for 'table-less' SQL expressions in procedures
  - Scalar expressions in procedural statements (e.g. IF, SET)
  - No need for Open/Fetch/Close processing
  - ODPs no longer required for simple expressions
- The expression evaluator is disabled for expression that:
  - Reference Tables, SET v1 = (SELECT COUNT(\*) FROM mytab)
  - Reference LOBs
  - Invoke UDFs
  - Require CQE (Classic Query Engine)
- 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!

## Tracking Expression Evaluator

- Usage of the expression evaluator tracked in database performance monitor
  - QVC1E field in the 1000 Record
  - Expressions implemented with C code are not collected in database monitor

## 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;

```



## DB2 6.1 Improves SQL Procedure Performance

- Expression Evaluator Improvements
  - Stream-lining of code path
  - Internal caching improvements
  - More usage of SQL Query Engine (SQE) - Upper, Lower, Sort Sequence, etc
  - Improved feedback in Database Monitor
- More usage of generated C code
- Expansion of Call cache for Stored Procedures
- Service program object support

\*\*\* ALL 6.1 ENHANCEMENTS REQUIRE DATABASE GROUP PTF#5 \*\*\*

## SQL Procedure Performance Resources

- Improving SQL Procedure Performance White Paper
  - <http://ibm.com/systems/i/db2/awp.html>
- System i Network article
  - <http://systeminetwork.com/article/coding-sql-routines-fast-track>
- Routines chapter in IBM SQL Programming Guide

## Additional Information



- **DB2 for i Websites**
  - Home Page: [ibm.com/systems/i/db2](http://ibm.com/systems/i/db2)
  - DeveloperWorks Zone: [ibm.com/developerworks/db2/products/db2i5OS](http://ibm.com/developerworks/db2/products/db2i5OS)
  - Porting Zone: [ibm.com/servers/enable/site/db2/porting.html](http://ibm.com/servers/enable/site/db2/porting.html)
- **Newsgroups**
  - USENET: comp.sys.ibm.as400.misc, comp.databases.ibm-db2
  - System i Network DB2 Forum -  
<http://systeminetwork.com/isnetforums/forumdisplay.php>
- **Education Resources - Classroom & Online**
  - [ibm.com/systemi/db2/gettingstarted.html](http://ibm.com/systemi/db2/gettingstarted.html)
  - [ibm.com/partnerworld/wps/training/i5os/courses](http://ibm.com/partnerworld/wps/training/i5os/courses)
- **DB2 for i Publications**
  - White Papers: [ibm.com/partnerworld/wps/whitepaper/i5os](http://ibm.com/partnerworld/wps/whitepaper/i5os)
  - Online Manuals: [ibm.com/systemi/db2/books.html](http://ibm.com/systemi/db2/books.html)
  - DB2 for i Redbooks ([ibm.com/systems/i/db2/redbooks.html](http://ibm.com/systems/i/db2/redbooks.html))  
[Stored Procedures, Triggers, and UDFs on DB2 UDB for iSeries \(SG24-6503\)](#)  
[OnDemand SQL Performance Analysis ... in V5R4 \(SG24-7326\)](#)

*THE NEW POWER EQUATION*

© 2009 IBM Corporation