Agenda Key:31MA
Session Number:409094

DB2 for IBM i: SQL Stored Procedures

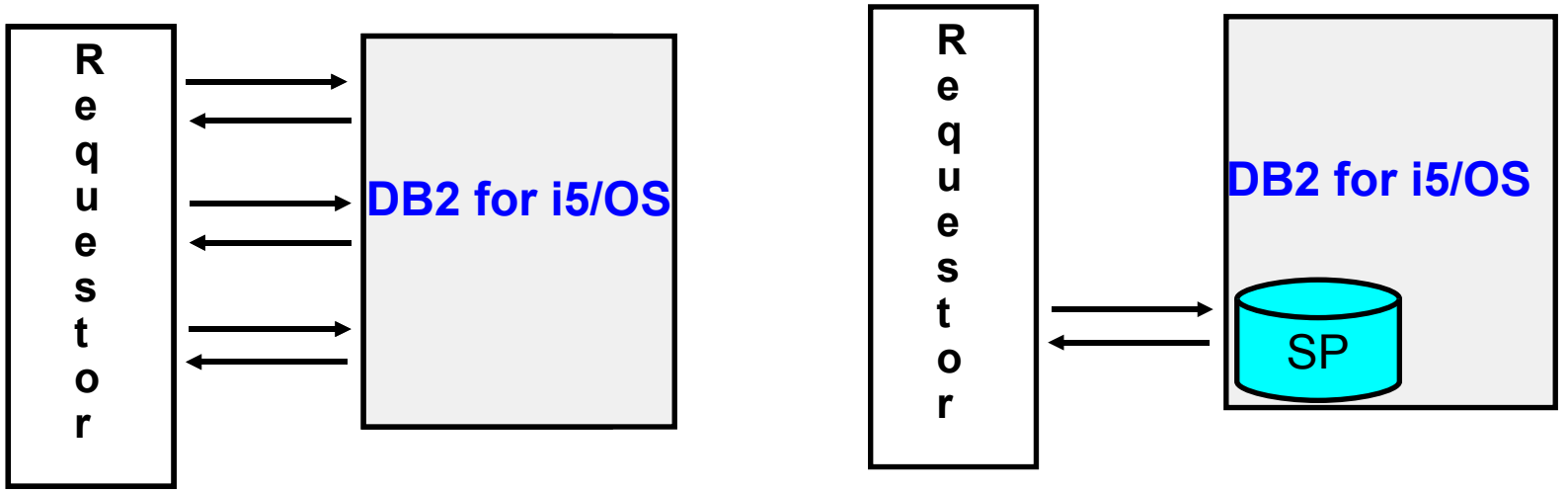
Tom McKinley (mac2@US.IBM.COM)
DB2 for IBM i consultant
IBM Lab Services

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 iSeries
  - Enables you to secure your data
  - iSeries adopted authority model can be leveraged
- Useful for moving host-centric applications to distributed applications

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

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) over and over

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

Stored Procedures

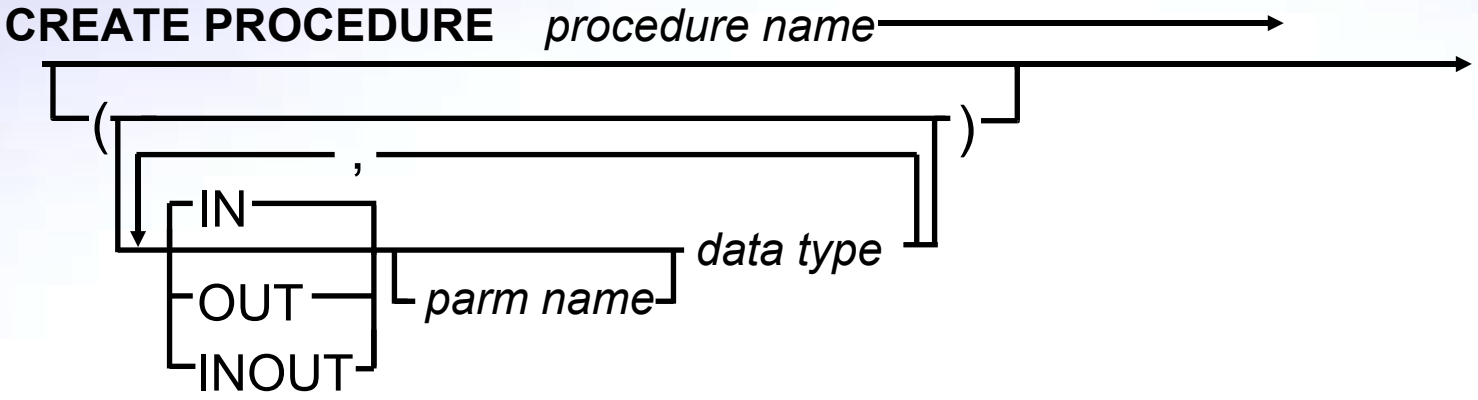
- DB2 for i5/OS supports two types of stored procedures

1. EXTERNAL
   - Register high-level language program(RPG, Java, C, etc) as a stored procedure
   - Procedure may or may not use SQL

2. SQL
   - Entire procedure is coded with SQL
   - follows SQL Standard (PSM)
   - Allows ‘normal’ DDL/DML SQL in addition to procedural statements

– SQL CREATE PROCEDURE statement used for both types

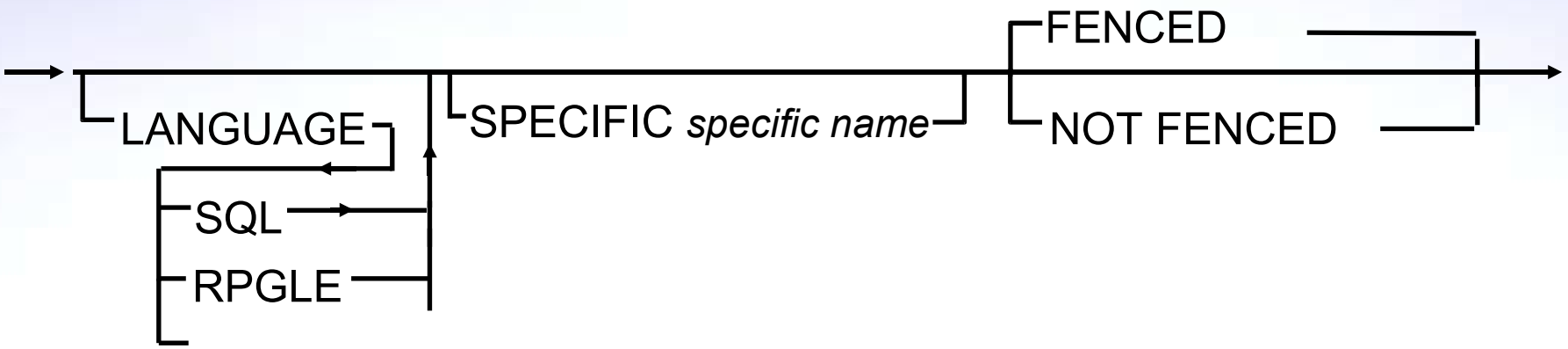
Create Procedure options



- Procedure name + number of parameters make a unique signature
  - Can have multiple procedures with the same name!
    - Ex: call lib1/proc1(parm1)… call lib1/proc1(parm1, parm2)

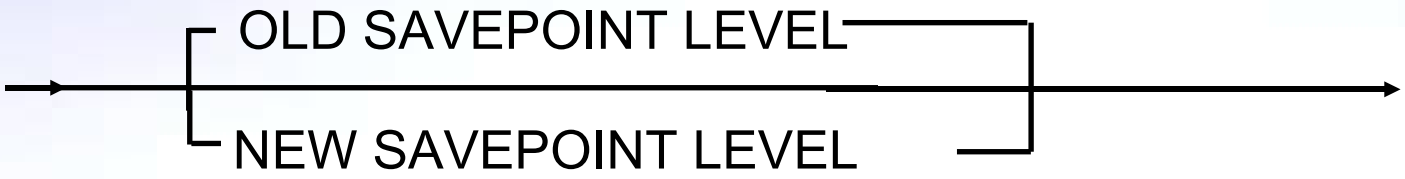
Note: Up to 1024 arguments supported (253 prior to V5R4)

Create Procedure options



- LANGUAGE - Language of procedure (SQL, C, RPG…)
- SPECIFIC – unique "short" name for SQL procedures
  - Automatically generated if not specified
- FENCED/NOT FENCED – DB2 family compatibility only. No effect

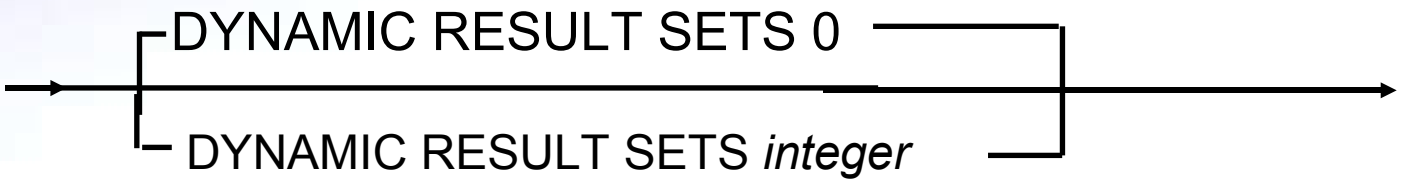
Create Procedure options



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

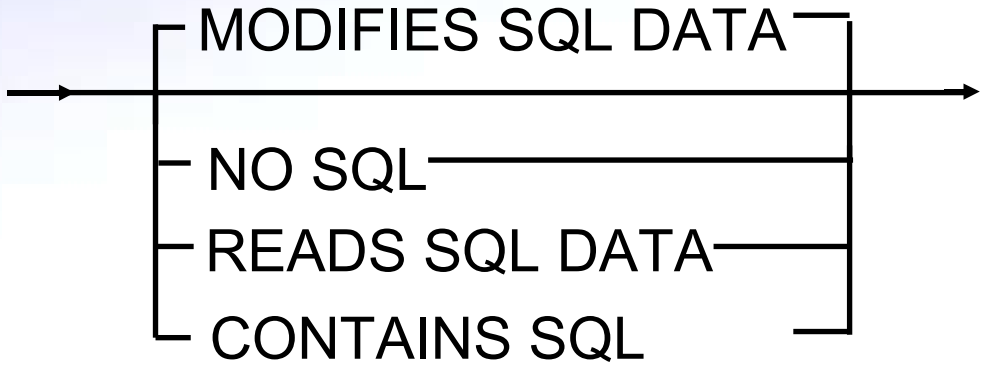
- COMMIT ON RETURN YES - DB2 issues a commit if the procedure successfully returns
  - Work done by both the invoker and procedure is committed
  - Result Set cursors must be declared WITH HOLD to be usable after the commit operation

Create Procedure options



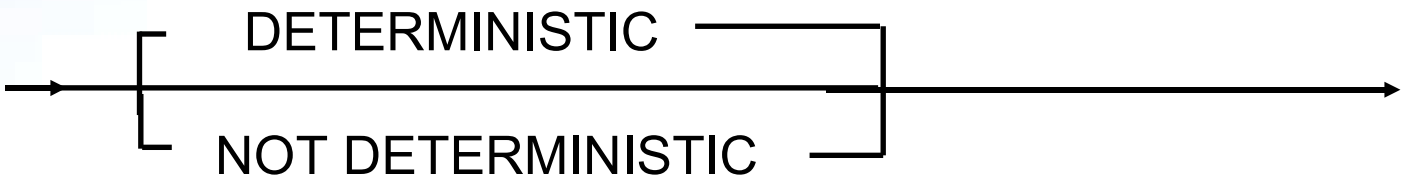
- RESULT SETS specifies max number of result sets that can be returned from procedure.
  - Only returned to ODBC, JDBC, .NET & CLI clients
  - more on result sets later on...

Create Procedure options



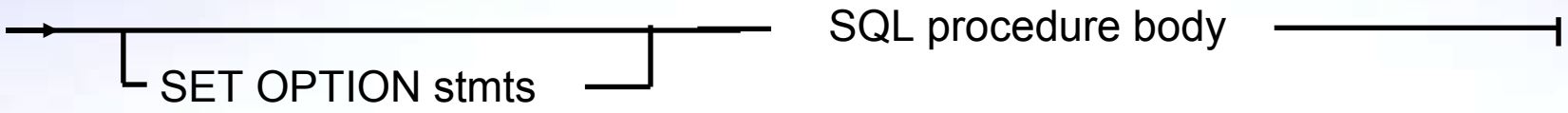
- MODIFIES SQL DATA – Most any SQL statement allowed
- READS SQL DATA – Read Only statements
- CONTAINS SQL – Simple local statements (SET, DECLARE)
- NO SQL – No SQL allowed (external procedures only)

Create Procedure options



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

Create Procedure options



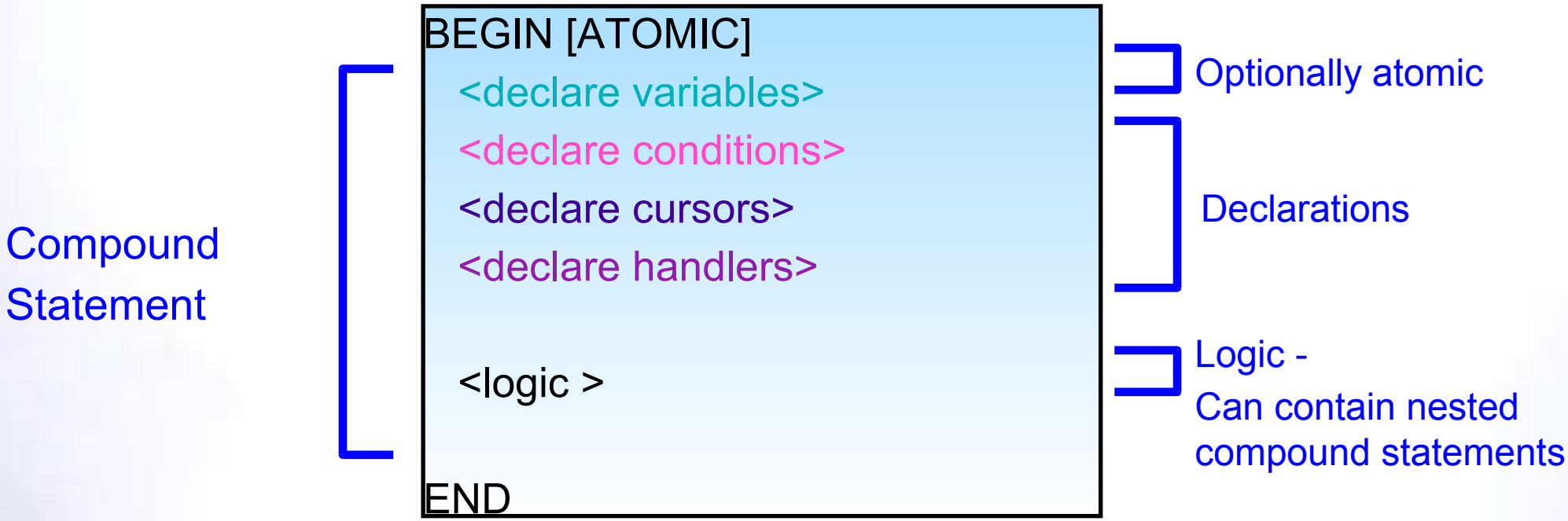
- SET OPTION - set processing options
  - Naming option (*SQL vs *SYS), sort-sequence, SQL path, debug…
  - Example: SET DBGVIEW=*STMT, USRPRF=*USER

- Most interesting options for SQL Procedures are:
  - USRPRF for adopted authority (defaults to *OWNER)
  - DBGVIEW=*SOURCE for creating debuggable version of SQL Procedure

SQL Compound Statement

- Group statements together
  BEGIN ATOMIC or **NOT ATOMIC**
  SQL procedure statement; [repeatable]
  END

- 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



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 (/* ... */)

Conditional Constructs

- **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
```

- **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;
```

Looping Constructs

- FOR - for each row of a query

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

  - SELECT statement columns can accessed directly

Other looping Constructs

- Other loop types
  - 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

Looping examples

Example: LOOP

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

Example: REPEAT

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

Example: WHILE

```
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!

Getting Feedback

- 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...
GET DIAGNOSTICS update_counter = ROW_COUNT;
 ...
```

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 '*xxyzz*'
    - SQLWARNING, NOT FOUND, SQLEXCEPTION

Error Handling Example

```
CREATE PROCEDURE proc1()
  …
 BEGIN
  DECLARE row_not_fnd CONDITION FOR '02000'; -- row not found
  condition
  DECLARE CONTINUE HANDLER FOR row_not_fnd
    SET at_end='Y'; -- set local variable at_end
  …

 END
```

Signaling errors

Send error or warning with optional message text

- **SIGNAL**

  SIGNAL *condition info* SET *assign value*;

  - condition info
    - Defined **condition** (prior discussion)
    - SQLSTATE ‘xxyzz’
  - Assign value
    - MESSAGE_TEXT, TABLE_NAME….
    - Values that can be retrieved via GET DIAGNOSTICS

- **RESIGNAL**

  RESIGNAL [*condition info* SET *assign value*];

  - Use within handler
  - Can just RESIGNAL (within brackets isoptional)
  - condition info
    - Defined **condition** (prior discussion)
    - SQLSTATE ‘xxyzz’
  - Assign value
    - MESSAGE_TEXT, TABLE_NAME….
    - Values that can be retrieved via GET DIAGNOSTICS

Signal Example

```
CREATE PROCEDURE Chg_Salary(IN i_empno CHAR(6),
                                          IN i_change DEC(9,2) )
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
```

RETURN statement

Return status of procedure invocation

RETURN <optional integer value>;

- Communicates success/failure status to caller
  - If no return statement not specified, then...
    - If SQLCODE >= 0, then return value set to a value of 0
    - If SQLCODE < 0, then return value set to -1
- Accessing the return value
  - Returned in SQLERRD[0]
  - when invoked by another procedure
    GET DIAGNOSTICS statusvar = RETURN_STATUS;
  - **"?=CALL <procedure name>"** syntax common in ODBC and JDBC

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;
```

EXAMPLE: SQL Stored Procedure

```
CREATE PROCEDURE CREDITPGM
( IN i_perinc DECIMAL(3,2), IN i_group CHAR(1),
  INOUT o_numrec INTEGER)
LANGUAGE SQL
--Update the customer's credit limit in the input customer group
--with the input percentage increase, return the number of updated
--customers as output
BEGIN ATOMIC
    DECLARE proc_cusnbr CHAR(5);
    DECLARE proc_cuscrd DECIMAL(11,2);
    DECLARE numrec INTEGER;
    DECLARE at_end INTEGER DEFAULT 0;
    DECLARE not_found CONDITION FOR '02000';
    DECLARE CONTINUE HANDLER FOR not_found SET at_end=1;
    DECLARE c1 CURSOR FOR
            SELECT cusnbr, cuscrd FROM ordapplib/customer
            WHERE cusgroup = i_group;
    ... continued...
```

SQL Procedure Example (cont'd)

```
...
SET numrec = 0;

OPEN c1;
FETCH c1 INTO proc_cusnbr, proc_cuscrd;

WHILE at_end = 0 DO
    SET proc_cuscrd = proc_cuscrd +(proc_cuscrd * i_perinc);

    UPDATE ordapplib/customer SET cuscrd = proc_cuscrd
    WHERE CURRENT OF c1;

    SET numrec = numrec + 1;
    FETCH c1 INTO proc_cusnbr, proc_cuscrd;
END WHILE;

SET o_numrec = numrec;
CLOSE c1;
END
```

Catalog Considerations

SYSPROCS & SYSPARMS catalogs

- updated when an SQL Stored Procedure is created
- SYSPROCS contains procedure details
- Being registered in SYSPROCS defines program as a procedure

Moving Procedures into Production

Save & restore program object that gets created by DB2

- C program object
- 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

Moving Procedures into Production

- DB2 does try 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

Performance Considerations

- Code generation enhanced in V5R4
  - recreate older SQL procedures for maximum benefit
- Package multiple operations in a single procedure rather than multiple procedures
  - tradeoff with reuse packaging
- SQL Procedure considerations
  - Generated C code with embedded SQL will not be as efficient as user-written code
  - Local variable suggestions
    - Declare local variables as not null
    - Use integer instead of decimal precision with 0
    - Avoid data conversions (same data type, length and scale in assignments)
  - Consider moving handlers for a specific condition/statement within a nested compound statement

```
BEGIN
  DECLARE CONTINUE HANDLER
              FOR SQLSTATE ' 23504'...
  ...
  DELETE FROM master WHERE id=1;
  ...
```

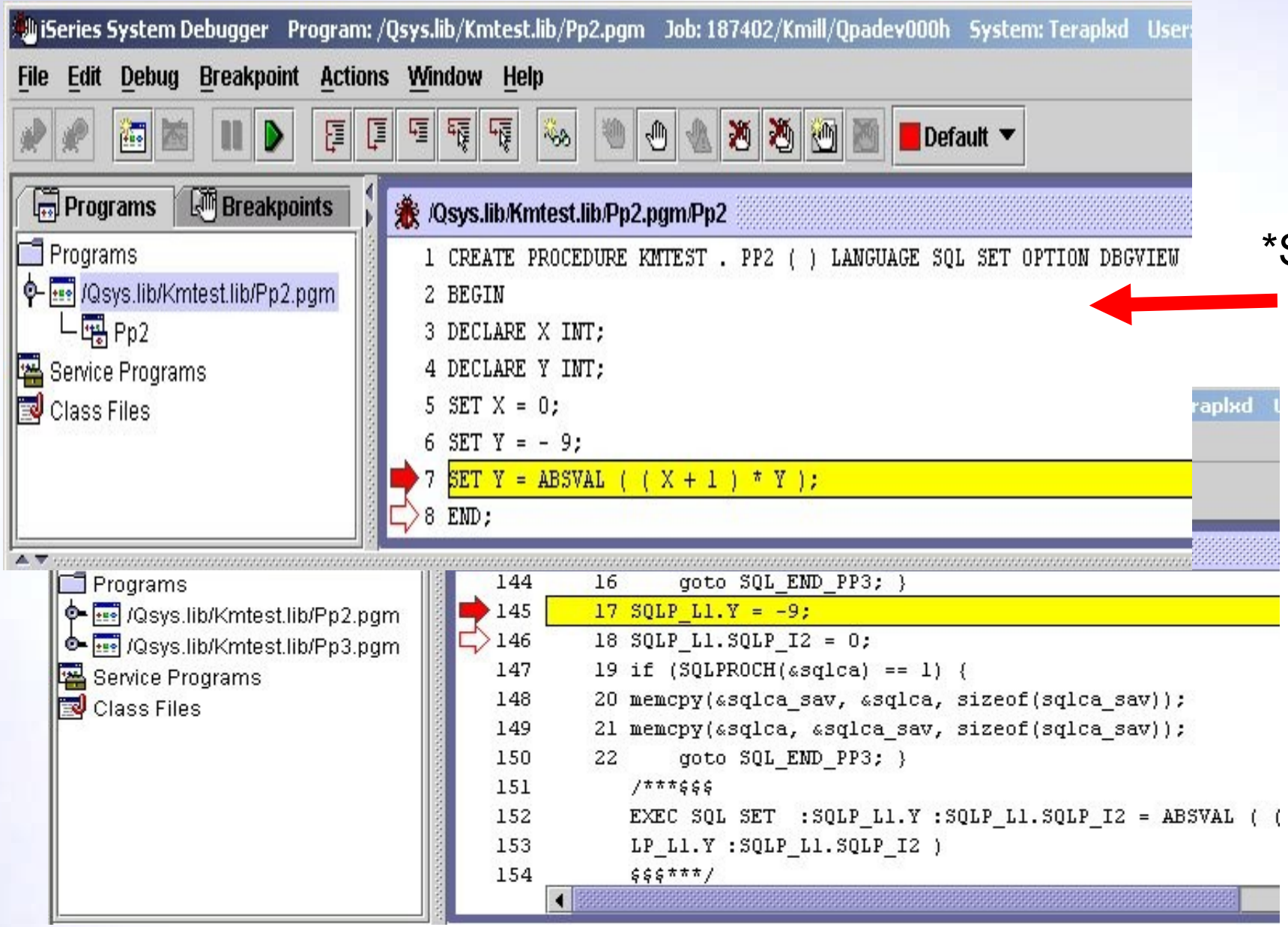
```
BEGIN
 ...
  BEGIN
      DECLARE CONTINUE HANDLER FOR
                                SQLSTATE ' 23504'...
      DELETE FROM master WHERE id=1;
   END
...
```

Tool Considerations

- iSeries Navigator
  - A simple graphical editor for developing SQL Procedures
  - Runtime debug by displaying contents of result sets and output parameters
  - Integration with System i Graphical Debugger

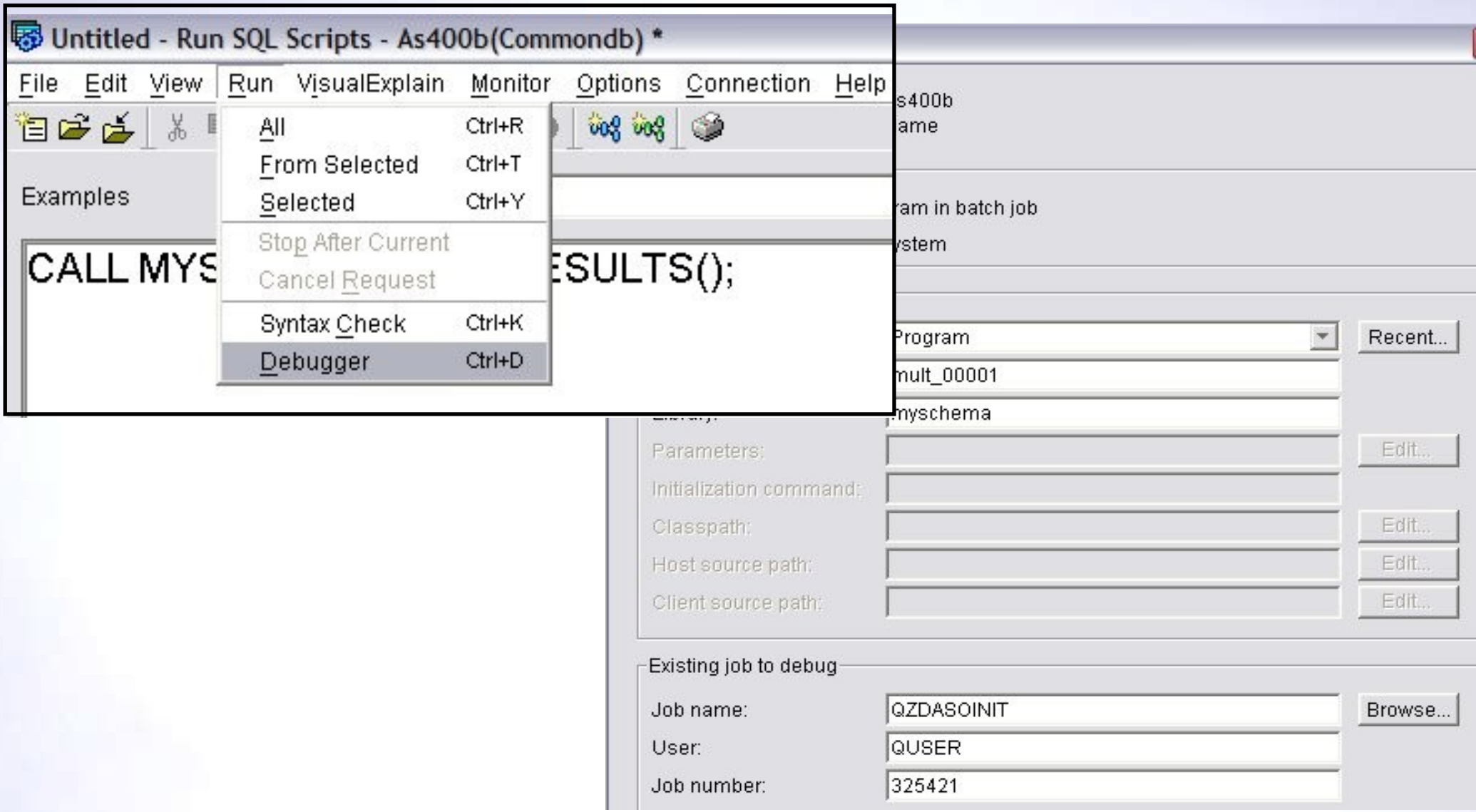
- DB2 Developer Workbench
  - More advanced graphical editor
  - Embedded in DB2 Express, which is available for download at: ibm.com/software/data/db2/ad/dwb.html
    - Need to setup a DRDA connection to with the DB2 Connect component that's part of DB2 Personal Developer Edition
    - Also need to setup a JDBC connection with the iSeries Toolbox JDBC driver

SQL Procedure Debug



*SOURCE view

V5R3 Debugger Integration

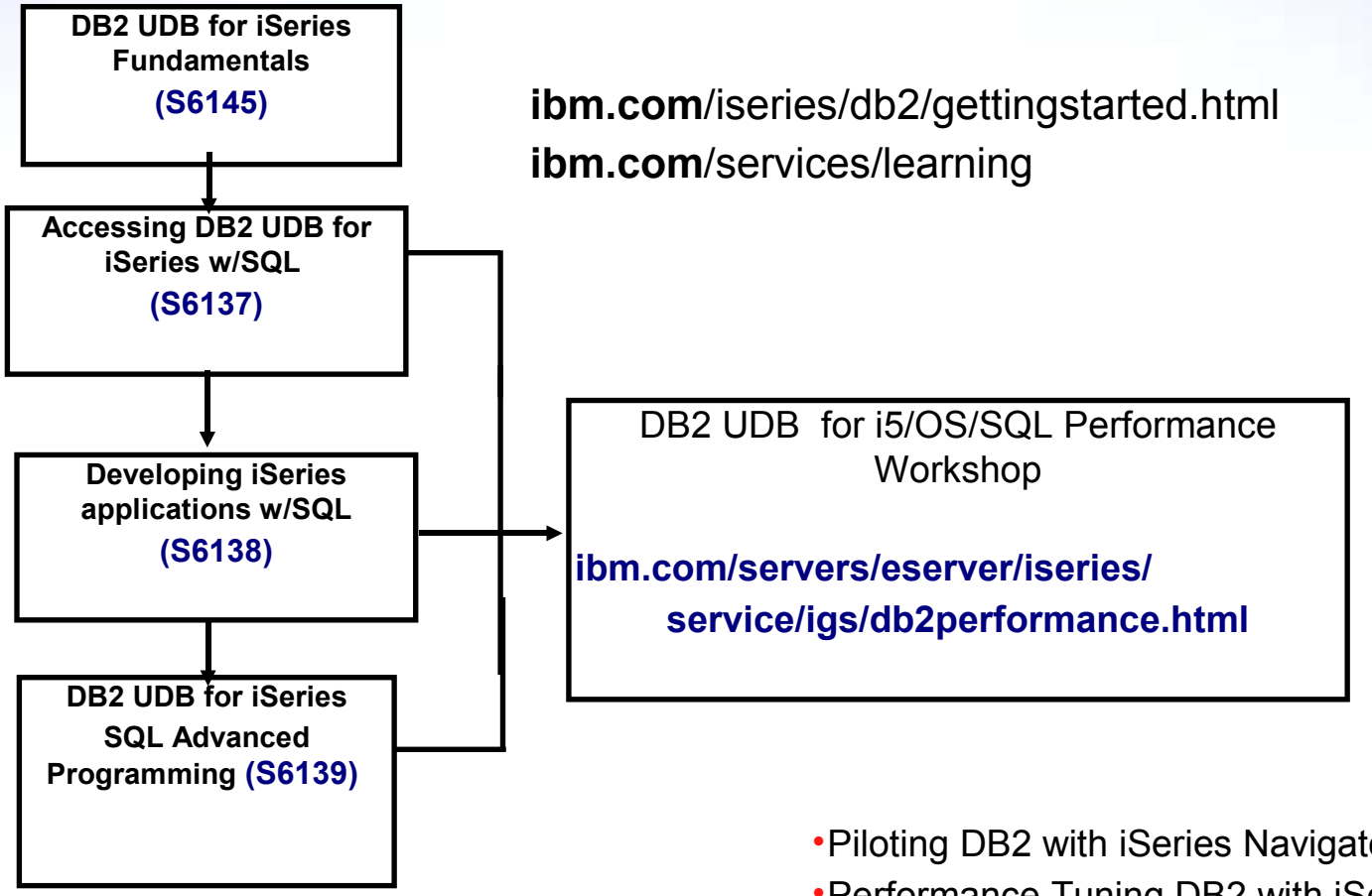


>>>>> Graphical Debugger White Paper:
ibm.com/servers/enable/site/education/abstracts/sqldebug_abs.html

Additional Information

- **DB2 for i5/OS home page - http://ibm.com/iseries/db2**
- **Newsgroups**
  - USENET: comp.sys.ibm.as400.misc, comp.databases.ibm-db2
  - iSeries Network (NEWS/400 Magazine) SQL & DB2 Forum - http://www.iseriesnetwork.com/Forums/main.cfm?CFApp=59
- **Education Resources - Classroom & Online**
  - http://ibm.com/iseries/db2/gettingstarted.html
  - http://ibm.com/servers/enable/site/education/ibo/view.html?oc#db2
  - http://ibm.com/servers/enable/site/education/ibo/view.html?wp#db2
- **DB2 for i5/OS Publications**
  - Online Manuals: http://www.iseries.ibm.com/db2/books.htm
  - Porting Help: http://ibm.com/servers/enable/site/db2/porting.html
  - DB2 UDB for iSeries Redbooks (http://ibm.com/redbooks)
    - Stored Procedures, Triggers, and UDFs on DB2 UDB for iSeries (SG24-6503)
    - SQL Performance Diagnosis with Database Monitor (SG24-6654)
  - ***SQL/400 Developer's Guide*** by Paul Conte & Mike Cravitz
    - http://as400network.com/str/books/Uniquebook2.cfm?NextBook=183

Education Roadmap



ibm.com/iseries/db2/gettingstarted.html

ibm.com/services/learning

- Self-study iSeries Navigator tutorials for DB2 at:

http://ibm.com/servers/enable/site/education/ibo/view.html?oc#db2

- Piloting DB2 with iSeries Navigator
- Performance Tuning DB2 with iSeries Navigator & Visual Explain
- Integrating XML and DB2 for i5/OS

Thank You

Appendix:
Result sets

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
  - Client Access ODBC driver
  - SQL CLI
  - Toolbox JDBC driver
  - Native JDBC driver
  - DB2 Connect
  - iSeries DRDA Connections
- Result sets are returned via open SQL cursors

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;
```

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.

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

- 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.

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;
```

Result Set Example

CALL RegionCustList(16)

call kmsample/RegionCustList(16)

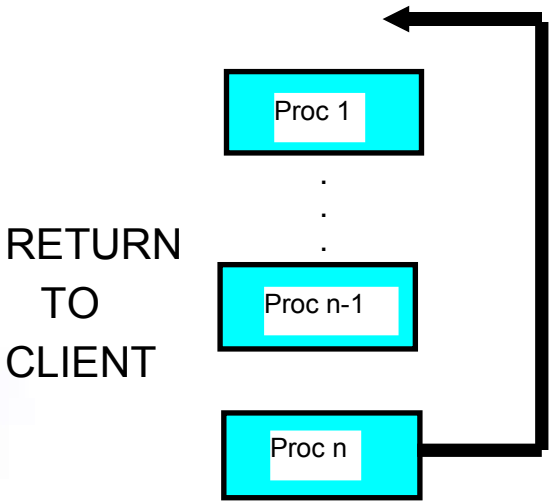
EMPNO	FIRSTNME	LASTNAME
000050	JOHN	GEYER
000060	IRVING	STERN
000070	EVA	PULASKI
000090	EILEEN	HENDERSON
000130	DELORES	QUINTANA
000150	BRUCE	ADAMSON
000170	MASATOSHI	YOSHIMURA
000190	JAMES	WALKER
000200	DAVID	BROWN
000260	SYBIL	JOHNSON
000320	RANDY	BROWN
000340	JASON	GOETZ
200170	KIM	GRIFFITH
200340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

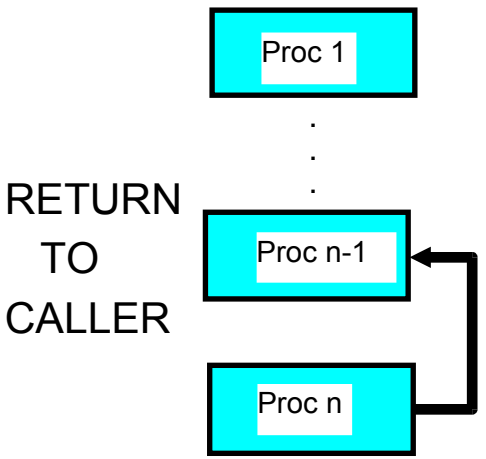
Considerations

- 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





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
  - iSeries Navigator SQL Script Center best tool for debugging result sets and output parameter values

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 */
     ...
```

IBM DB2 for IBM i Consulting and Services

- ✓ Database modernization (look for upcoming Workshop)
- ✓ DB2 Web Query
- ✓ Database design, features and functions
- ✓ DB2 SQL performance analysis and tuning
- ✓ Data warehousing review and assessment
- ✓ DB2 for IBM i education and training

Contact: Tom McKinley mac2@us.ibm.com
IBM Systems and Technology Group
Rochester, MN USA

Trademarks and Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.
Trademarks of International Business Machines Corporation in the United States, other countries, or both can be found on the World Wide Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and are used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

The customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Prices are suggested U.S. list prices and are subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.