Welcome to:

DB2 SQL Procedure Language

Unit Objectives

After completing this unit, you should be able to:

- Describe the DB2 SQL Procedure Language
- Identify valid SQL statements for a procedure
- Code process control statements for a procedure

SQL Stored Procedures

- **Based on ANSI/ISO standard language SQL/PSM**
- **Simple language which includes:**
  - Features from block-structured languages
  - Exception handling
  - Familiar to Sybase, Oracle, Informix, Microsoft SQL Server programmers

SQL Procedure Language (1 of 3)

- **SQL Procedures support:**
  - Multiple parameters: input, output, input/output
  - Returning multiple output result sets to client
- **SQL Procedures are defined in DB2 catalog.**
- **SQL Procedure source is stored in DB2 catalog.**
- **SQL Procedure language is folded to upper case.**
  - Exception: delimited values

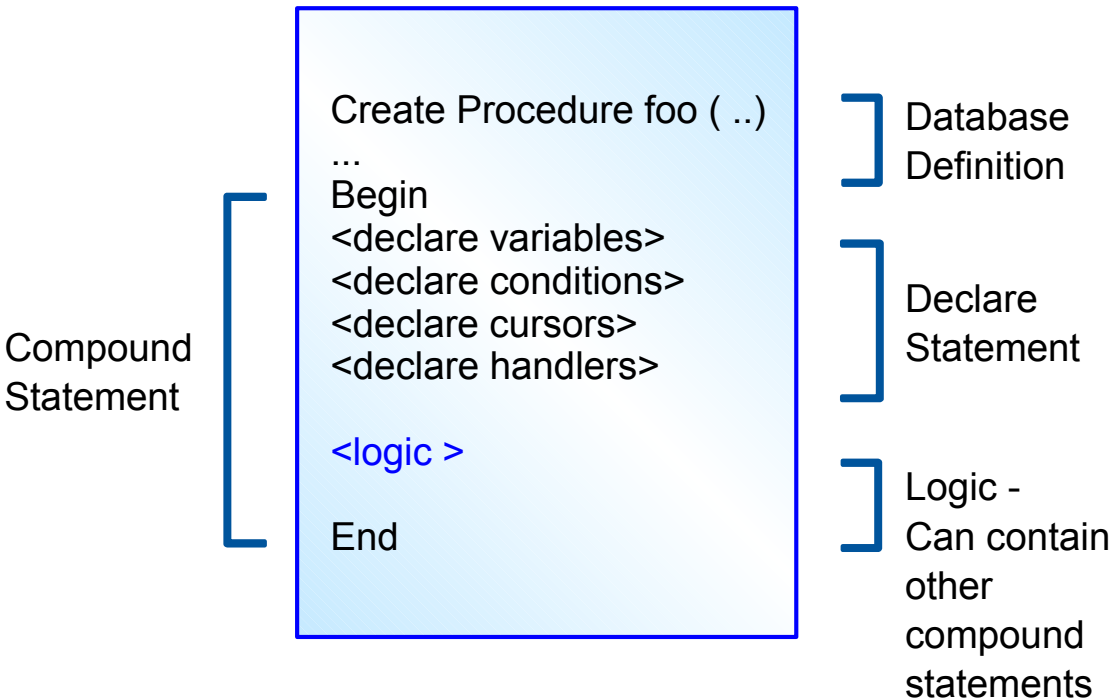
SQL Procedure Language (2 of 2)

```
CREATE PROCEDURE DB2ADMIN.Sample1 ( IN in_Dept INT )
 RESULT SETS 1
 LANGUAGE SQL
-------------------------------------------------------------------------
-- SQL Stored Procedure
-------------------------------------------------------------------------
P1: BEGIN
 DECLARE  r_error    int default  0;
 DECLARE  SQLCODE    int default 0;
 DECLARE CONTINUE HANDLER FOR SQLWARNING, SQLEXCEPTION, NOT FOUND
   BEGIN
    SET r_error = SQLCODE;
   END;
  BEGIN
  DECLARE cursor1 CURSOR WITH RETURN FOR
    SELECT   DEPTNAME,     MANAGER,  LOCATION
    FROM   ORG
    WHERE
      DEPTNUMB = in_Dept;
  -- Cursor left open for client application
  OPEN cursor1;
 END;
END P1
```

SQL Procedure Language (3 of 3)

- **An SQL Procedure consists of:**
  - A CREATE PROCEDURE statement
    - LANGUAGE = SQL
  - A procedure body which may include:
    - Compound statement(s): BEGIN … END
    - Declaration statements
    - Assignment statements
    - Conditional statements
    - Iterative control structure: LOOPs, and so forth
    - Exception Handling
    - CALL another stored procedure

Structure (1 of 2)



Structure (2 of 2)

- **An SQL Procedure can be:**
  - A single statement

```
CREATE PROCEDURE Sample1 (OUT Parm1 CHAR(10))
LANGUAGE SQL
SET Parm1 = 'value1'
```

  - A compound statement

```
CREATE PROCEDURE Sample2 (OUT Parm1 CHAR(10),
    OUT Parm2 CHAR(10))
LANGUAGE SQL
BEGIN
  SET Parm1 = 'value1' ;
  SET Parm2 = 'value2';
END
```

  - Or nested compound statements

SQL Procedure Language Statements

- Not limited to stored procedures
- Some platform differences
- Facilitate application solution
- Add business logic capability to SQL language

Compound Statements

- **Can have labels:** **P1: BEGIN … END P1**
- **Can be atomic:** **BEGIN ATOMIC**
- **Can contain:**
  - Declarations
  - Procedural statements
  - Other compound statements

Example - Nested Compound Statements

```
CREATE PROCEDURE ADMIN.Proc1 (out p1_a integer,  out p2_a integer)
  LANGUAGE SQL
P1: BEGIN
 declare a integer default 5;
 declare c1 cursor with return for select * from staff;
   P2: BEGIN
    declare a integer default 7;
    declare c1 cursor with return for select * from department;
    open c1;
    set p2_a = a;
   END P2;
 open c1;
 set p1_a = a;
END P1
```

Compound Statements

- **Order of statements in compound statement**
  - SQL variables and condition declarations
  - Cursor declarations
  - Handler declarations
  - Procedure body statements

- **Terminating statements with ;**
  - Procedure body has no terminating character
  - Statement nested within other statements ends with ;

Example - ";"

```
CREATE PROCEDURE foo
    ( out day_Of_Year int )
LANGUAGE SQL
----------------------------------------
-- SQL Stored Procedure
----------------------------------------
P1: BEGIN
  DECLARE c_Date DATE;
  SET c_Date = CURRENT DATE;
  SET day_of_Year = dayofyear(c_Date);
END P1
```

Declarations (1 of 2)

- **Local variables:**

```
DECLARE var_name  datatype [ DEFAULT value];
  Ex:    DECLARE my_var INTEGER DEFAULT 6;
```

  - Default value is NULL
  - Variable name is folded to upper case
  - Rules for ambiguous names
    - Check if column name (table exists)
    - Check if SQL variable / parameter name
    - Assume to be a column name
    - **Note:** OS/390 and z/OS will check if variable / parameter name else assume a column
      - Qualify with table name to force

Declarations (2 of 2)

- **Condition declaration:**
  - `DECLARE not_found CONDITION FOR SQLSTATE '02000';`
- **Local cursor declaration:**
  - `DECLARE c1 CURSOR FOR select * from staff;`
    - WITH RETURN TO CLIENT / WITH RETURN TO CALLER
- **Handler declaration:**
  - `DECLARE EXIT HANDLER FOR SQLEXCEPTION...;`

Example "Cursors"

```
CREATE PROCEDURE Cur_Samp
  (IN v_name VARCHAR(254), OUT v_job VARCHAR(5))
LANGUAGE SQL
P1: BEGIN
    DECLARE c1 CURSOR FOR
      SELECT JOB FROM STAFF WHERE NAME = v_name;
    OPEN c1;
    FETCH c1 INTO v_job;
END P1
```

Assignments

- **Syntax:**

```
SET lv_name = expression;
SET lv_name = NULL;
```

- **Example:**

```
SET salary = salary + salary*0.1;
SET init_salary = NULL;
SET salary = (select salary from employee
              where empno = lv_emp_num);
```

NOTE: SQLERROR if more than one row

Control Flow Statements (1 of 2)

- **CASE statement**
  - Select execution path based on multiple conditions
- **IF statement**
  - Select execution path based on evaluation of conditions
- **LOOP statement**
  - Execute statements multiple times
- **REPEAT statement**
  - Execute statements until condition is true

Control Flow Statements (2 of 2)

- **WHILE statement**
  - Execute statements while condition is true
- **FOR statement**
  - Execute statements for each row of a table
- **ITERATE**
  - Transfers flow on control to labeled block or loop
- **LEAVE statement**
  - Transfer control out of loop or block for FOR, LOOP, REPEAT or WHILE

Conditional Statements

- **Syntax:**

```
IF cond1 THEN statement ;
ELSEIF cond2 THEN statement  ;
  ELSE statement ;
END IF;
```

- **Example:**

```
IF  rating = 1 THEN
  UPDATE EMPLOYEE SET salary = salary*1.10 WHERE empno = i_num;
ELSEIF  rating = 2 THEN
  UPDATE EMPLOYEE SET salary = salary*1.05 WHERE empno = i_num;
ELSE
  UPDATE EMPLOYEE SET salary = salary*1.03 WHERE empno = i_num;
END IF;
```

CASE Statement (1 of 2)

- **Simple CASE statement:**

```
CREATE PROCEDURE foo ( IN v_workdept CHAR(3))
LANGUAGE SQL
P1: BEGIN
   CASE v_workdept
      WHEN 'A00'
         THEN UPDATE department
         SET deptname = 'DATA ACCESS 1';
      WHEN 'B01'
         THEN UPDATE department
         SET deptname = 'DATA ACCESS 2';
      ELSE UPDATE department
         SET deptname = 'DATA ACCESS 3';
   END CASE
END P1
```

CASE Statement (2 of 2)

- **Searched CASE statement:**

```
CREATE PROCEDURE foo ( IN v_workdept CHAR(3))
LANGUAGE SQL
P1: BEGIN
     CASE
        WHEN v_workdept = 'A00'
           THEN UPDATE department
           SET deptname= 'DATA ACCESS 1';
        WHEN v_workdept = 'B01'
           THEN UPDATE department
           SET deptname = 'DATA ACCESS 2';
        ELSE UPDATE department
           SET deptname = 'DATA ACCESS 3';
     END CASE;
END P1
```

LOOP Statement

- **Syntax:**

```
[label] LOOP
            SQL-procedure-statement(s);
        END LOOP  [label]
```

- **Example:**

```
fetch_loop:
   LOOP
      FETCH c1 INTO v_firstname, v_lastname;
      SET counter = counter + 1;
      IF counter = 51 THEN
         LEAVE fetch_loop;
      END IF;
   END LOOP fetch_loop;
```

FOR Statement

- **Syntax:**

```
[label] FOR for-loop-name AS [cursor-name CURSOR FOR]
  select-statement DO
    SQL-procedure-statement(s);
        END FOR [label]
```

- **Example:**

```
DECLARE fullname CHAR(40);
FOR v1 AS c1 CURSOR FOR
  select firstnme, midinit, lastname FROM employee
DO
  SET fullname = lastname || ',' || firstnme || ',' midinit;
  INSERT INTO tname VALUE (fullname)
END FOR;
```

Other Control Flow Statements

- **REPEAT Statement**

```
ftch_loop2:
  REPEAT
    FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
  UNTIL  SQLCODE <> 0 END REPEAT ftch_loop2;
```

- **WHILE Statement**

```
WHILE at_end = 0 DO
  FETCH c1 INTO v_firstnme, v_midinit, v_lastname;
  IF SQLCODE = 100 THEN SET at_end = 1;
  END IF;
END WHILE;
```

Error Handling

- **SQL procedure terminates if an SQL error occurs unless a handler is declared**
  - Warning: data truncation on a set is an SQL error
- **SQLSTATE and SQLCODE**
  - Access requires explicit declaration, for example:
    - `DECLARE SQLSTATE CHAR(5) DEFAULT '00000';`
    - `DECLARE SQLCODE INTEGER DEFAULT 0;`

Condition Handlers

- **Condition Handling**
  - Compound statement contains any number of handlers
  - A condition handler must specify:
    - A set of conditions it is prepared to handle
    - Action to perform to handle the condition
    - Where to resume the execution
      - EXIT, CONTINUE, UNDO
  - Action can be any SQL statement or compound statement

Condition Handlers - Example

```
BEGIN [ATOMIC]
  DECLARE type HANDLER FOR conditions
  handler-action
  statement_1; raises exception
  statement_2; CONTINUE point
  statement_3;
END UNDO or EXIT point
```

Error Conditions

- **Conditions raised**
  - Implicitly by the system via error situation
  - Explicitly via SIGNAL or RESIGNAL
    - *SIGNAL*
      - Signal an error or warning condition.
      - It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text.
    - *RESIGNAL*
      - Resignal an error or warning condition.
      - It causes an error or warning to be returned with the specified SQLSTATE, along with optional message text.

Example "SIGNAL/RESIGNAL"

```
CREATE PROCEDURE divide ( IN numerator INTEGER,
            IN denominator INTEGER, OUT result INTEGER)
  LANGUAGE SQL
  BEGIN
    DECLARE overflow CONDITION FOR SQLSTATE '22003';
    DECLARE CONTINUE HANDLER FOR overflow
       RESIGNAL SQLSTATE '22375' ;
    IF denominator = 0 THEN
       SIGNAL overflow;
    ELSE
       SET result = numerator / denominator;
    END IF;
  END
```

Example "Exception Handling"

```
CREATE PROCEDURE ITERATOR()  LANGUAGE SQL
BEGIN
  DECLARE at_end INTEGER DEFAULT 0;
  DECLARE not_found CONDITION FOR SQLSTATE '02000';
    DECLARE c1 CURSOR FOR ....;
  DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
    OPEN c1;
    ftch_loop1: LOOP
      FETCH c1 INTO v_dept, v_deptname, v_admdept;
      IF at_end = 1 THEN
        LEAVE ftch_loop1;
      ELSEIF v_dept = 'D01' THEN
        ITERATE ftch_loop1;
      END IF;
      INSERT INTO department (deptno, deptname, admrdept)
      VALUES ( 'NEW', v_deptname, v_admdept);
    END LOOP;
    CLOSE c1;
END
```

(2) Handle not_found
set at_end = 1

(1) Row not found
(3) Continue execution

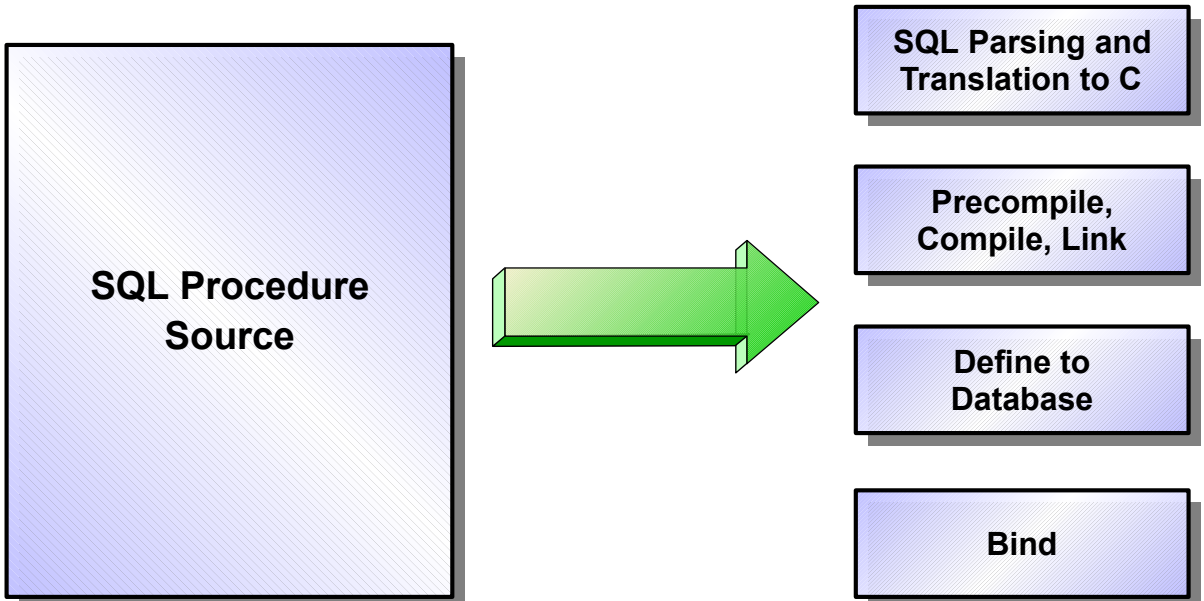
Nested SP (return to caller) Example

- **Client A calls X**
  - X opens C1 (return to caller)
  - X call Y
    - Y opens C2 (return to caller)
    - Y cannot access C1
    - Y calls Z
      - Z opens C3 (return to caller)
      - Z cannot access C1, C2
      - Z returns
    - Y can access C3
    - Y returns
  - X can access C2
  - X cannot access C3
  - X returns
- **A can access C1**
- **A cannot access C2, C3**

Nested SP (return to client) Example

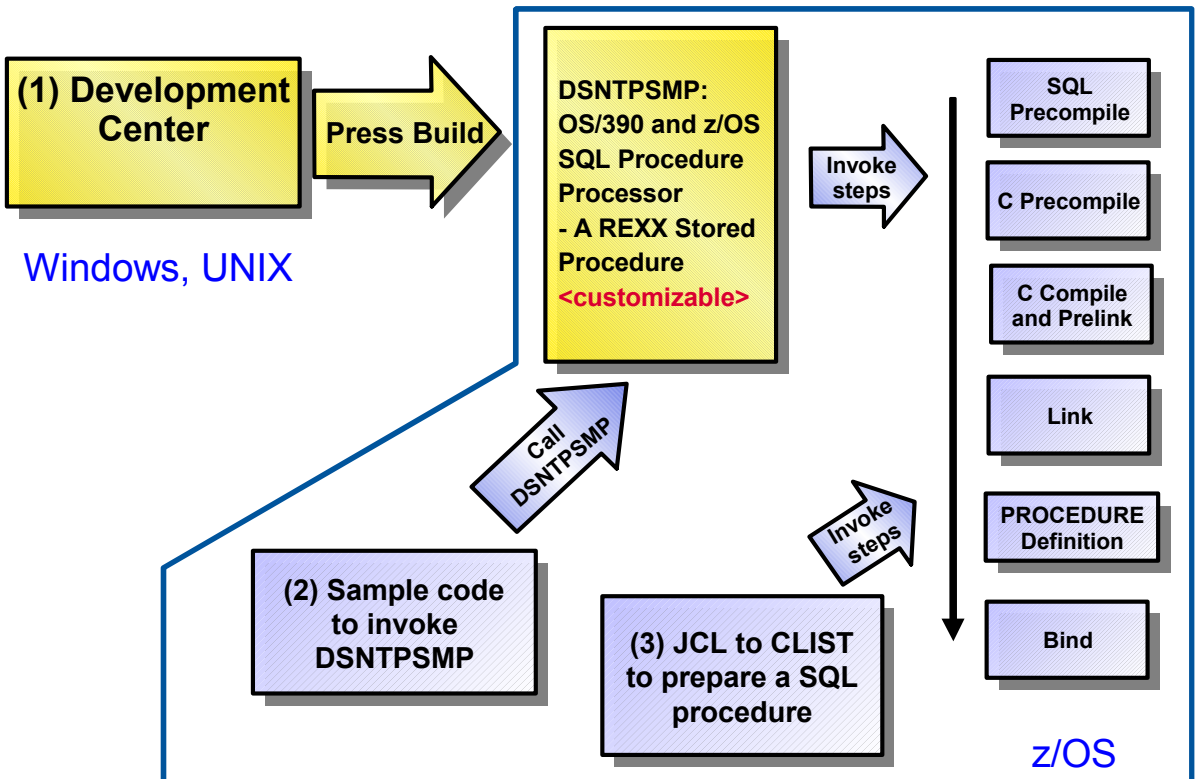
- **Client A calls X**
  - X opens C1 (return to client)
  - X call Y
    - Y opens C2 (return to client)
    - Y cannot access C1
    - Y calls Z
      - Z opens C3 (return to client)
      - Z cannot access C1, C2
      - Z returns
    - Y cannot access C3
    - Y returns
  - X cannot access C2, C3
  - X returns
- **A can access C1, C2, C3**

SQL Procedures - Under the Covers

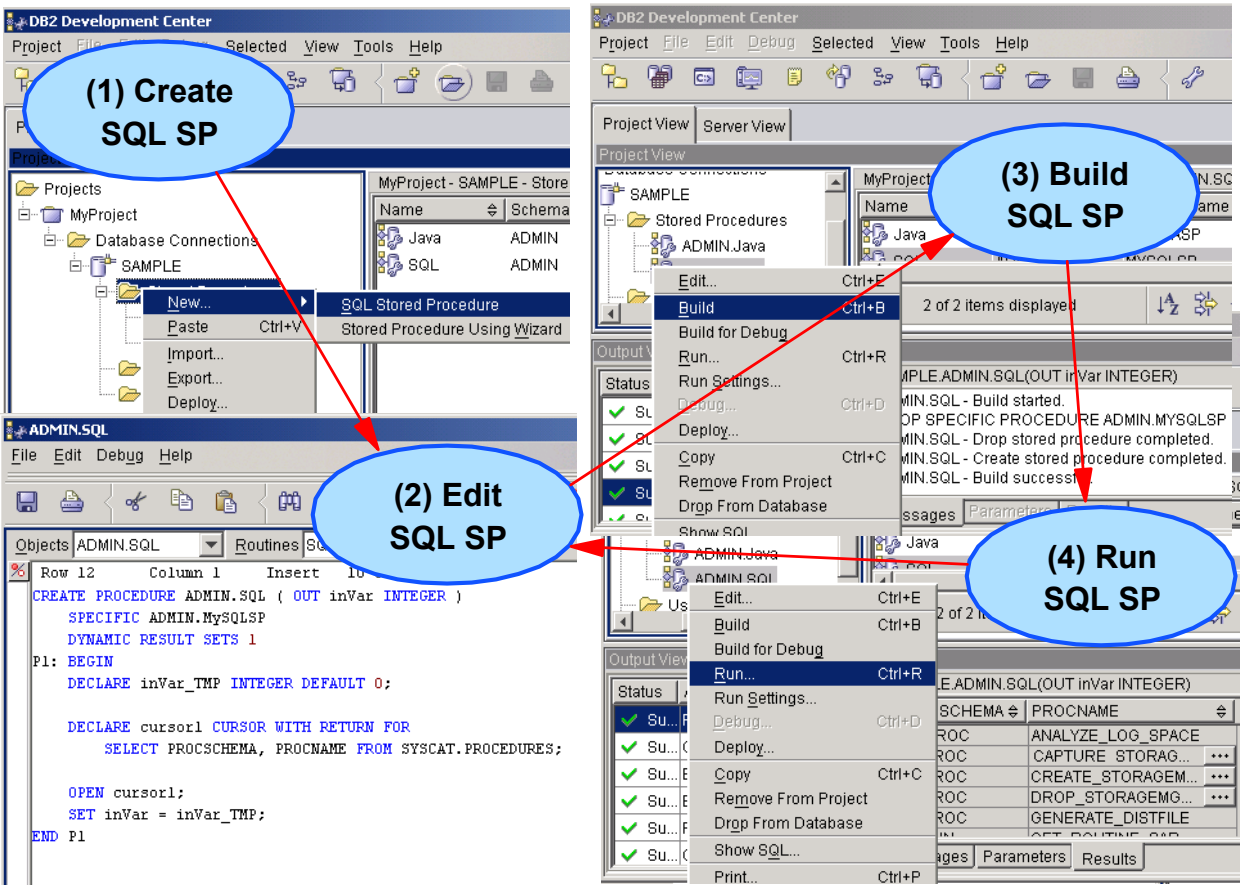


Executable Stored Procedure is compiled C!

Creating OS/390 and z/OS SQL Procedures



Using the Development Center



Unit Summary

Since completing this unit, you should be able to:

- Describe the DB2 SQL Procedure Language
- Identify valid SQL statements for a procedure
- Code process control statements for a procedure