

> Native SQL Stored Procedures

Robert Andresen

Principal Consultant

IBM Certified Database Administrator, DB2 9 for z/OS

IBM Certified SOA Associate

ITIL Foundation Certified

Cell: +1 630 254-0168

Robert.Andresen@ca.com

Agenda

- > Native vs. external SQL procedures
- > Native SQL procedure definition
- > Versioning
- > Native SQL procedure execution
- > Deploying a native SQL procedure to other servers
- > DB2/DSN/SQL: changes to commands
- > Testing, error handling and debugging
 - Sample Rexx exec to test native SQL procedure
- > Migrating external to native SQL procedures

External SQL procedures

- > New in DB2 V5
- > Required SQL code plus C code
- > Multiple step prepare
- > Runs in WLM environment

Native SQL procedures

- > New in DB2 9 NFM
- > Simpler builds
 - No C/C++ compile
 - Single step DDL
- > Better performance
 - executed in DB2 DBM1 address space, not WLM
- > zIIP eligible
 - Saves on software licensing costs
- > Enhanced support for SQL PL
 - FOR loops
 - Nested compound statements
 - Data types BIGINT, BINARY, VARBINARY & DECFLOAT

SYSIBM.SYSENVIRONMENT

- > New table in DB2 9
- > Environment information
 - Index on expressions
 - Native SQL procedure
- > ENVID is unique environment identifier
- > SYSIBM.SYSROUTINES column TEXT_ENVID points to SYSENVIRONMENT

Authorization

> CREATE PROCEDURE SQL

- CREATEIN for the schema
- SYSADM or SYSCTRL

Native SQL Procedures

- > Compiled into run time structures
- > Bound when created
- > Loaded into EDM Pool at execute time
 - Large packages
 - Watch EDM Pool failures

Sample RETURNDEPTSALARY

- > In IBM SQL Reference
- > Modified to use DSN8910.EMP table

Sample native SQL

```
CREATE PROCEDURE RETURNDEPTSALARY
  (IN DEPTNUMBER CHAR(3),
   OUT DEPTSALARY DECIMAL(15,2),
   OUT DEPTBONUSCNT INT)
LANGUAGE SQL
READS SQL DATA
P1: BEGIN
  DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
  DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
  DECLARE TOTAL_SALARY DECIMAL(15,2) DEFAULT 0;
  DECLARE BONUS_CNT INT DEFAULT 0;
  DECLARE END_TABLE INT DEFAULT 0;
  DECLARE C1 CURSOR FOR
    SELECT SALARY, BONUS FROM DSN8910.EMP
    WHERE WORKDEPT = DEPTNUMBER;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET END_TABLE = 1;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SET DEPTSALARY = NULL;
  OPEN C1;
  FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
```

Sample native SQL, cont.

```
WHILE END_TABLE = 0 DO
  SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY +
  EMPLOYEE_BONUS;
  IF EMPLOYEE_BONUS > 0 THEN
    SET BONUS_CNT = BONUS_CNT + 1;
  END IF;
  FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
END WHILE;
CLOSE C1;
SET DEPTSALARY = TOTAL_SALARY;
SET DEPTBONUSCNT = BONUS_CNT;
END P1
```

Batch prepare

```
/**
/** Step 3: Prepare routine as a native SQL procedure
/**      -> Also generates a package called DSN8.DSN8ES3_RS_TBL
/**
//PH066S03 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
    DSN SYSTEM(S91A)
    RUN PROGRAM(DSNTEP2) PLAN(DSNTEP91) +
        LIB('S91A.RUNLIB.LOAD') PARM('/SQLTERM(%)' )
    END
//SYSIN DD DISP=SHR,
//      DSN=ANDRO16.JCL.CNTL2(NSQLDSAL)
//      DD *

%
```

CREATE Changes

- > Native SQL procedures
 - without FENCED or
 - external WLM address space
 - loaded by zOS
 - EXTERNAL
- > VERSION: application life cycle
- > LANGUAGE SQL: optional

Result sets

> DYNAMIC RESULT SETS 1

- result set returned to caller

> DECLARE c1 CURSOR WITH RETURN FOR

- declares cursor c1 associated with result set

> OPEN c1 as last statement

- Allows caller to retrieve data from cursor c1

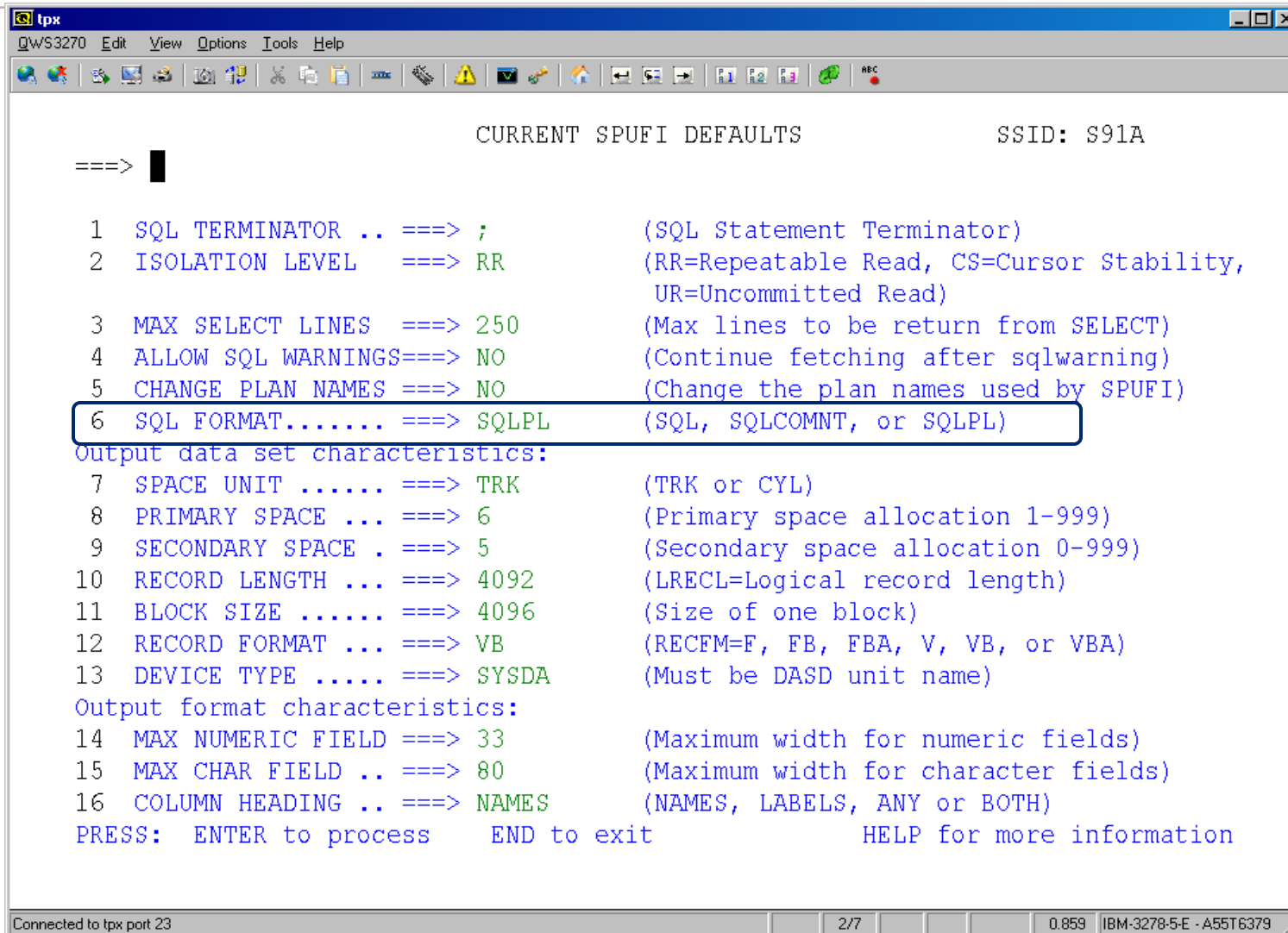
SQLFORMAT

> Used by SPUFI/DSNTEP2/DSNTEP4

> How to process SQL to pass onto PREPARE

- SQL: As in prior releases, multi-line SQL converted to single line buffer, comments removed
- SQLCOMNT: Keeps comments and multi-line format to assist in diagnostics, adds LF after comments where none is found
- SQLPL: similar to SQLCOMNT, adds LF to lines ending without token split, retains format of SQL procedure stored in DB2 catalog

SPUFI default options:



The screenshot shows a terminal window titled 'tpx' with a menu bar (QWS3270, Edit, View, Options, Tools, Help) and a toolbar. The main display area shows the 'CURRENT SPUFI DEFAULTS' for 'SSID: S91A'. A cursor is at the prompt '==>'. A list of 16 default options is displayed, with the 6th option, 'SQL FORMAT..... ==> SQLPL', highlighted by a blue rectangular box. Below the list, there are sections for 'Output data set characteristics:' and 'Output format characteristics:'. At the bottom, instructions read 'PRESS: ENTER to process END to exit HELP for more information'. The status bar at the bottom indicates 'Connected to tpx port 23', '2/7', '0.859', and 'IBM-3278-5-E - A55T6379'.

```
tpx
QWS3270 Edit View Options Tools Help

CURRENT SPUFI DEFAULTS                      SSID: S91A

==> █

1  SQL TERMINATOR .. ==> ;                  (SQL Statement Terminator)
2  ISOLATION LEVEL   ==> RR                  (RR=Repeatable Read, CS=Cursor Stability,
                                           UR=Uncommitted Read)
3  MAX SELECT LINES  ==> 250                (Max lines to be return from SELECT)
4  ALLOW SQL WARNINGS==> NO                 (Continue fetching after sqlwarning)
5  CHANGE PLAN NAMES ==> NO                 (Change the plan names used by SPUFI)
6  SQL FORMAT..... ==> SQLPL               (SQL, SQLCOMNT, or SQLPL)
Output data set characteristics:
7  SPACE UNIT ..... ==> TRK                (TRK or CYL)
8  PRIMARY SPACE ... ==> 6                 (Primary space allocation 1-999)
9  SECONDARY SPACE . ==> 5                 (Secondary space allocation 0-999)
10 RECORD LENGTH ... ==> 4092              (LRECL=Logical record length)
11 BLOCK SIZE ..... ==> 4096              (Size of one block)
12 RECORD FORMAT ... ==> VB                (RECFM=F, FB, FBA, V, VB, or VBA)
13 DEVICE TYPE ..... ==> SYSDA            (Must be DASD unit name)
Output format characteristics:
14 MAX NUMERIC FIELD ==> 33                (Maximum width for numeric fields)
15 MAX CHAR FIELD .. ==> 80                (Maximum width for character fields)
16 COLUMN HEADING .. ==> NAMES            (NAMES, LABELS, ANY or BOTH)
PRESS: ENTER to process  END to exit      HELP for more information

Connected to tpx port 23  2/7  0.859  IBM-3278-5-E - A55T6379
```

CREATE from SPUFI

```
BROWSE      ANDR016.SPUFI.OUT
Command ==>
***** Top of
-----+-----+-----+-----+-----+-----+-----
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
  LANGUAGE SQL
  MODIFIES SQL DATA
  UPDATE DSN8910.EMP
    SET SALARY = SALARY * RATE
    WHERE EMPNO = EMPLOYEE_NUMBER
-----+-----+-----+-----+-----+-----+-----
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
-----+-----+-----+-----+-----+-----+-----
DSNE614I AUTOCOMMIT IS NO; NO CHANGES COMMITTED
-----+-----+-----+-----+-----+-----+-----
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 9
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 18
***** Bottom
```


DSNTEP2

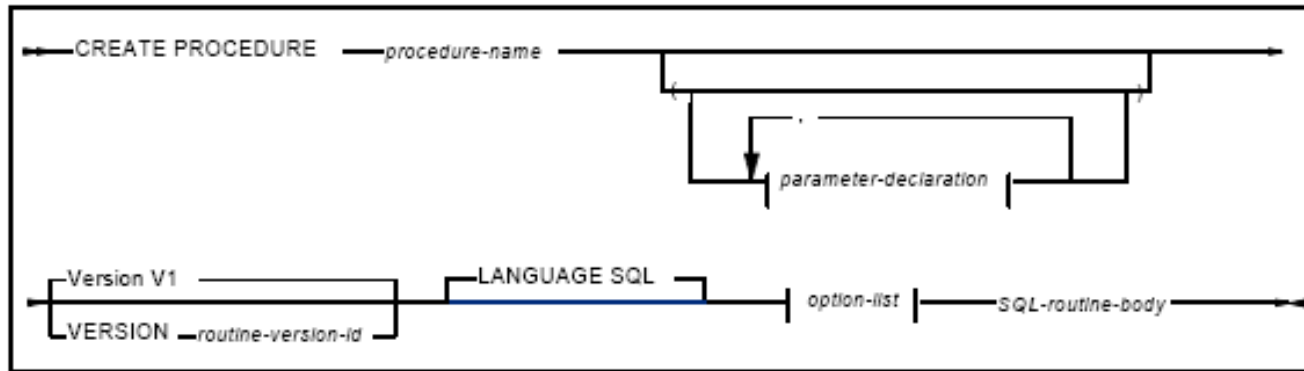
> Specify SQLFORMAT:

```
//SYSTSIN DD *  
DSN SYSTEM(DB9A)  
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP91) +  
LIB('DSN910.RUNLIB.LOAD') +  
PARMS (' /SQLFORMAT (SQLPL) , SQLTERM (%) ' )  
END
```

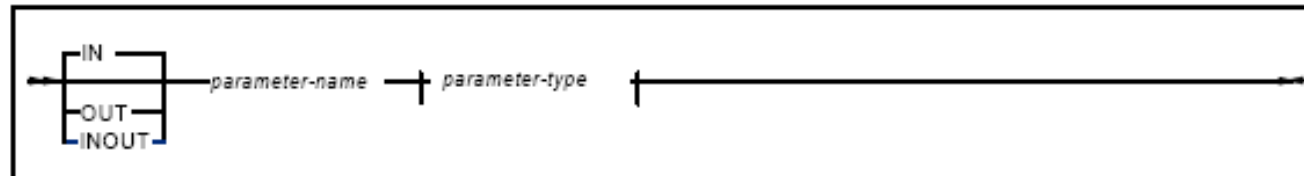
Successful Output

```
SET DEPTSALARY = TOTAL_SALARY;  
    SET DEPTBONUSCNT = BONUS_CNT;  
END P1  
%  
RESULT OF SQL STATEMENT:  
DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION  
DSNT418I SQLSTATE      = 00000 SQLSTATE RETURN CODE  
DSNT416I SQLERRD       = 0  0  0  -1  0  0 SQL DIAGNOSTIC  
INFORMATION  
DSNT416I SQLERRD       = X'00000000'  X'00000000'  X'00000000'  
X'FFFFFFFF'  X'00000000'  X'00000000' SQL DIAGNOSTIC  
INFORMATION  
CREATE      SUCCESSFUL  
PAGE      1
```

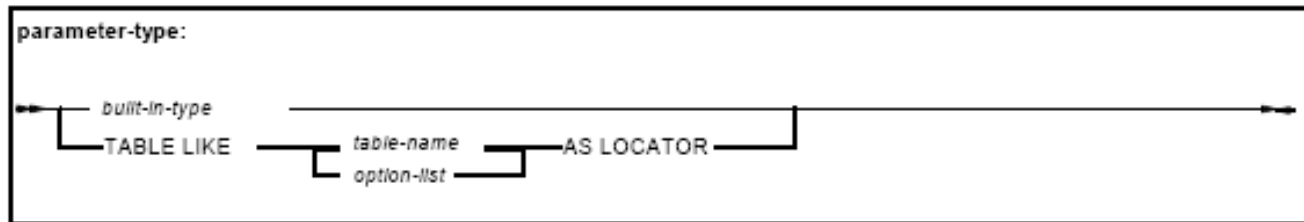
Create PROCEDURE Syntax, Pt. 1



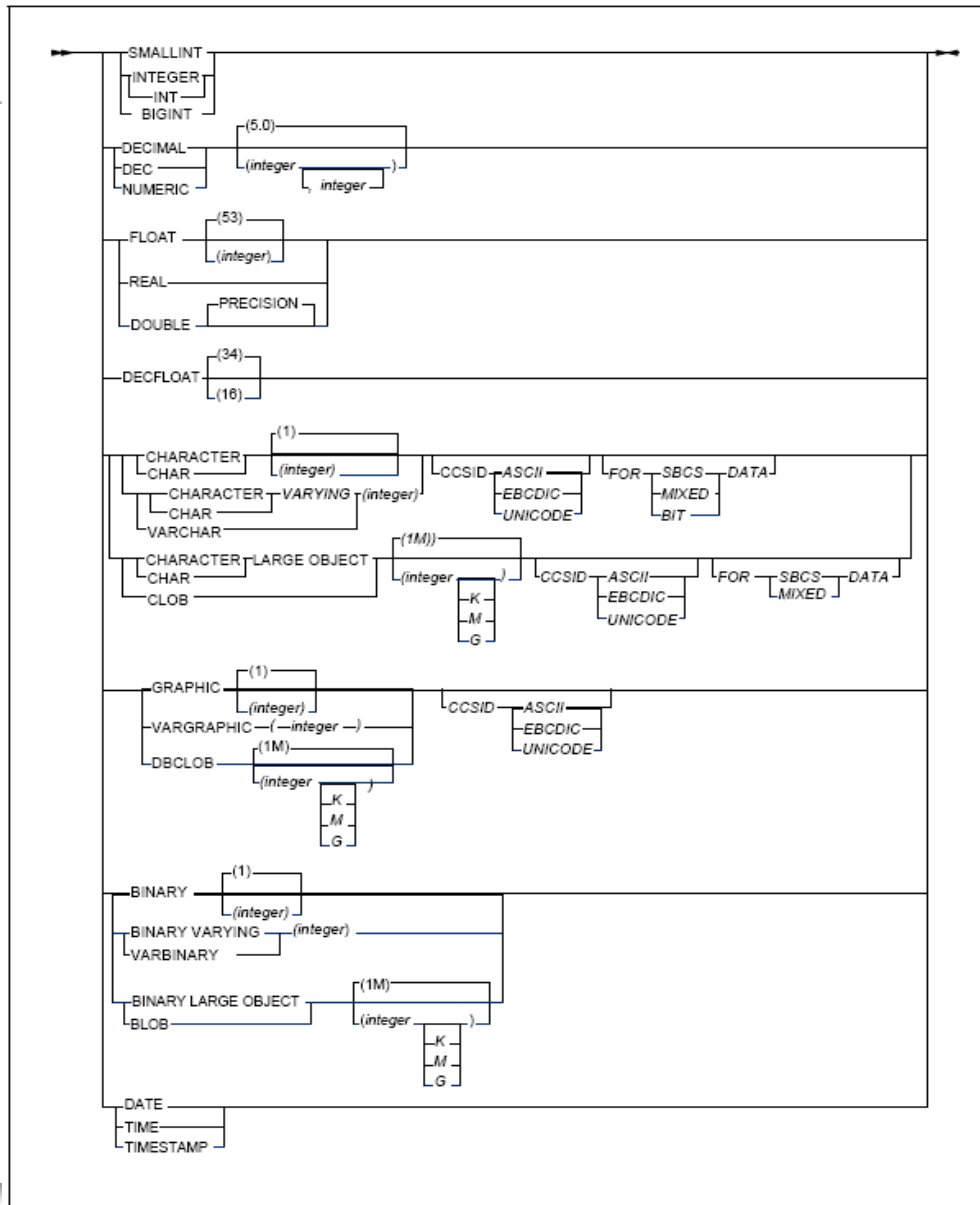
parameter-declaration:



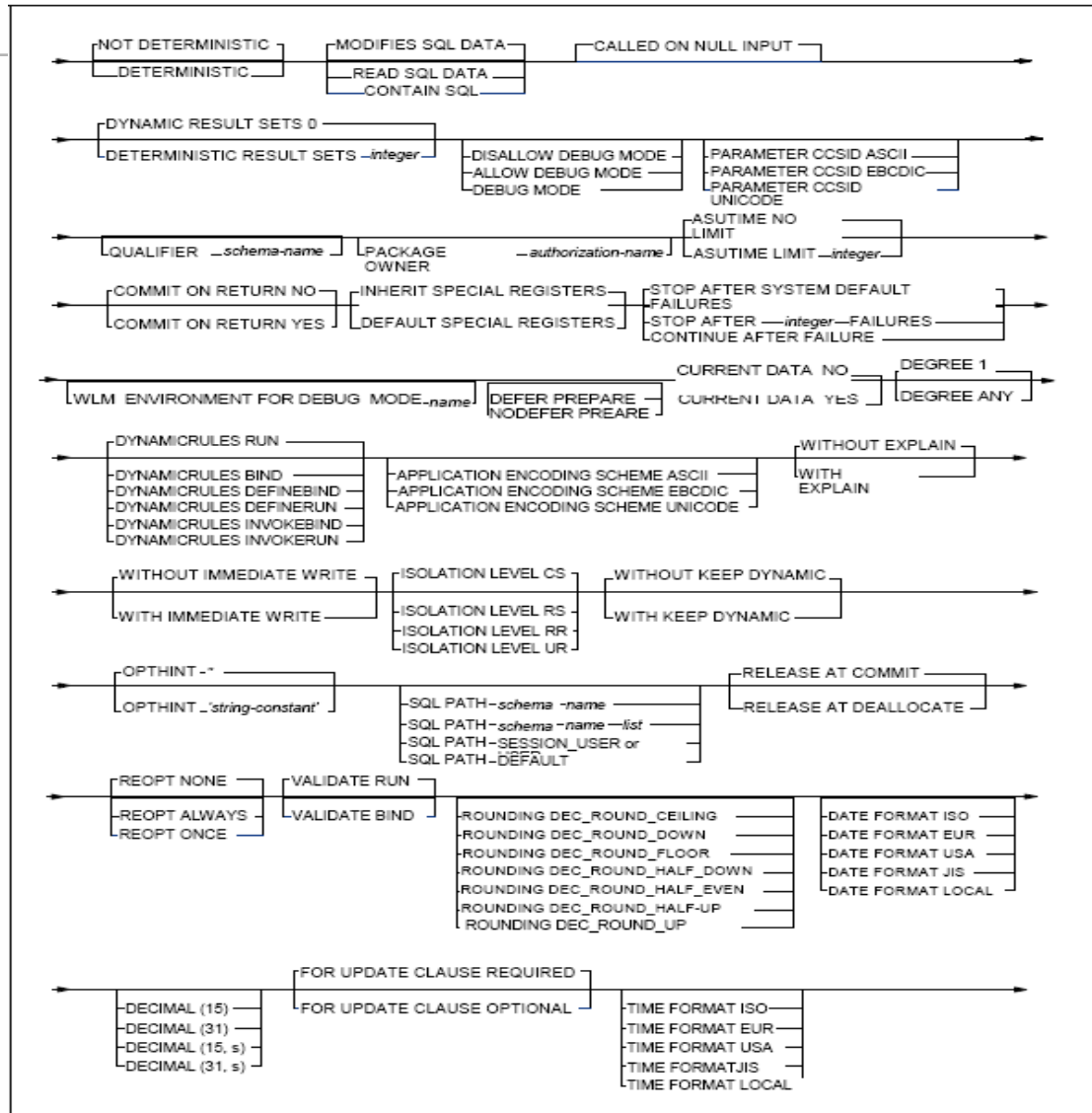
parameter-type:



Create Syntax, built-in type:



Create syntax, option list



FOR SQL control

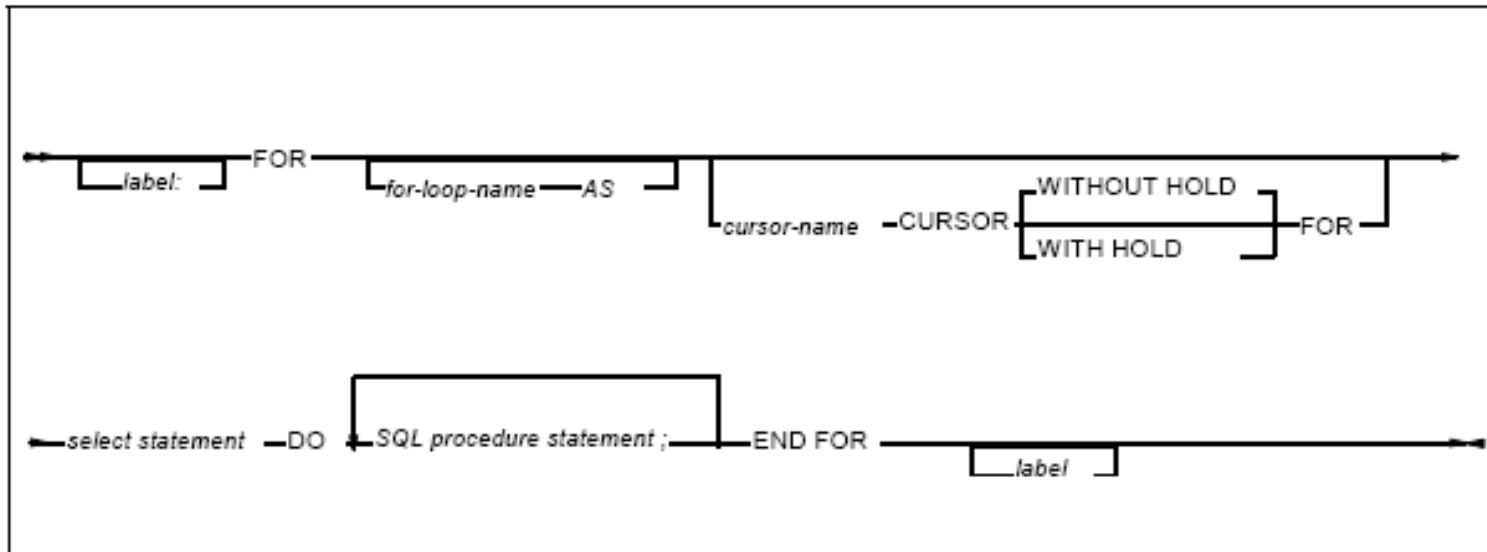
> FOR SQL control statement

- FOR loop with embedded SELECT statement
- Iterating cursor over the result set

> Example:

```
CREATE PROCEDURE CALC_SALARY(OUT SUM INTEGER)
VERSION V1
LANGUAGE SQL
READS SQL DATA
BEGIN
SET SUM = 0;
FOR V1 AS
C1 CURSOR FOR
SELECT SALARY FROM STAFF
DO SET SUM = SUM + V1.SALARY;
END FOR;
```

FOR SQL syntax



Extended GOTO

- > Allows for branching out of the current compound statement to different levels within the same scope
 - If the GOTO statement is in a condition handler, the target label must be defined in that same condition handler
 - If the GOTO statement is not in a condition handler, the target label must not be defined in a condition handler
- > If a GOTO branches out of a compound statement, all open cursors declared in that compound statement are closed.
 - Except: cursors that return a result set.
 - The same is true for nested compound statements.

Extended GOTO example:

```
CREATE PROCEDURE GOTO()  
P1: BEGIN  
    DECLARE I, A INTEGER;  
    SET I = 1;  
  
    LAB1: SET A = 1;  
    BEGIN  
        LAB2: SET A = 2;  
        BEGIN  
            SET I = I + 1;  
            IF I < 3 THEN GOTO LAB1;  
            END IF;  
        END;  
    END;  
END P1#
```

Nested compound statements

- > Compound statement: grouping of other statements into an executable block
 - Delimited by BEGIN and END
 - SQL variables can be declared within a compound statement
- > Now can use:
 - A compound statement within a condition handler
 - Nested compound statements to define different scopes for SQL variables, cursors, condition names, and condition handlers
- > Scope considerations:
 - SQL variable declaration
 - Cursor definition
 - Condition name
 - Condition handler declarations

Ambiguous names

- > If a name is declared as a SQL variable and also exists as a column name:
- > External stored procedures use the name as to reference the variable
- > Native SQL stored procedures use the name to reference the table column
- > Qualify the column name to avoid this ambiguity

Name scoping table for compound statements

Type of name	Must be unique within...	Qualification allowed?	Can be referenced within...
SQL variable	The compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement.	Yes, it can be qualified with the label of the compound statement in which the variable was declared.	The compound statement in which it is declared, including any compound statements that are nested within that compound statement. When multiple SQL variables are defined with the same name you can use a label to explicitly refer to a specific variable that is not the most local in scope.
condition	The compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement.	No	The compound statement in which it is declared, including any compound statements that are nested within that compound statement. Can be used in the declaration of a condition handler, or in a SIGNAL or RESIGNAL statement. Note: When multiple conditions are defined with the same name there is no way to explicitly refer to the condition that is not the most local in scope.
cursor	The compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement.	No	The compound statement in which it is declared, including any compound statements that are nested within that compound statement. Note: When multiple cursors are defined with the same name there is no way to explicitly refer to the cursor that is not the most local in scope. However, if the cursor is defined as a result set cursor (that is, the WITH RETURN clause was specified as part of the cursor declaration), the invoking application can access the result set.
label	The compound statement that declared the variable, including any declarations in compound statements that are nested within that compound statement.	No	The compound statement in which it is declared, including any compound statements that are nested within that compound statement. Use a label to qualify the name of an SQL variable or as the target of a GOTO, LEAVE, or ITERATE statement.

Versioning

- > VERSION is an option on CREATE or ALTER PROCEDURE
- > Multiple versions may exist of a stored procedure
 - Any version may be set to be the active version
 - Only one version may be active at any time

> Add new version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC  
ADD VERSION REL9 . . .
```

> Activate a version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC ACTIVATE VERSION REL9#
```

Versioning, cont.

> Rebind an existing version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC REGENERATE ACTIVE VERSION#
```

> Replace active version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC  
REPLACE VERSION REL9 . . .
```

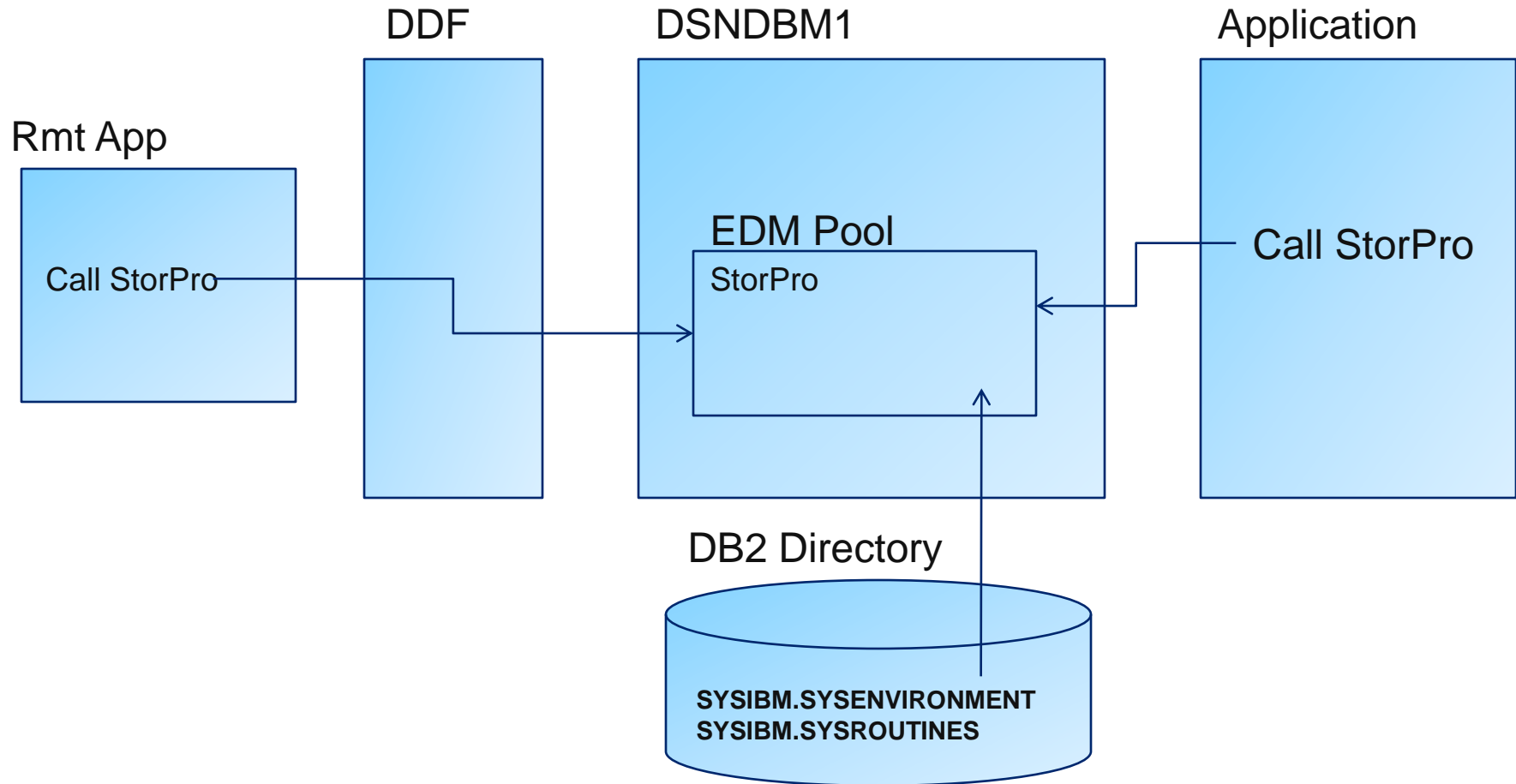
> Rebind an existing version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC REGENERATE ACTIVE VERSION#
```

> Drop an existing version:

```
ALTER PROCEDURE MY_NAT_SQL_PROC DROP VERSION REL8#
```

Calling a native SQL procedure



Sample Calls

Java:

```
CallableStatement cstmt =  
    con.prepareCall("CALL PAOLOR3.MEDIAN_RESULT_SET(?)");  
cstmt.registerOutParameter(1, Types.INTEGER);  
boolean hasResultSet = false;  
hasResultSet = cstmt.execute();
```

PL/I:

```
EXEC SQL CONNECT TO BETA;  
V1 = 528671;  
IV = -1;  
EXEC SQL CALL SUMARIZE(:V1,:V2 INDICATOR :IV);
```


Which version gets called?

> By default, procedure is identified by:

- Schema name
- Procedure name
- Number of parameters
- Current active version

> Application code may over-ride by setting special register:

- `SET CURRENT ROUTINE VERSION = 'version'`
- Be careful if you are calling multiple procedures that have the same version name or one procedure calls another
- Reset by setting register to an empty string

Deploying SQL procedures, old way

- > Distributing or installing a procedure created on one system to another system
- > Prior to V9, customers deployed SQL stored procedures by:
 - Copying over the load modules of the stored procedures
 - Copying DBRM for the stored procedure over
 - issuing a BIND PACKAGE
 - Issuing CREATE PROCEDURE to define the procedure

Deploy native SQL procedure in DB2 9

- > The DB2 enhanced support for deployment of native SQL procedures:
 - Install a native SQL procedure to a production system
 - Extended BIND PACKAGE command
 - New keyword DEPLOY.
- > Different from remote BIND package
 - logic of the procedure body will not be re-bound
 - No worries about unexpected behavior change

```
BIND PACKAGE ..... DEPLOY(collection-id.package-id) COPYVER(version-id) ...
```

DB2 Commands:

> START or STOP PROCEDURE:

- All versions of a procedure
- No way to target a single version

> REBIND PACKAGE:

- Only changeable bind option is EXPLAIN(NO/YES)
- REBIND PACKAGE only rebinds the SQL statements that included in the procedure
- Control statements in the procedure definition are not rebound

DB2 Commands 2

> COMMENT ON PROCEDURE

- extended to handle multiple versions

```
COMMENT ON PROCEDURE PAOLOR3.MEDIAN_RESULT_SET  
VERSION MEDIAN_V2  
IS 'THIS IS THE SECOND VERSION' ;
```

> GRANT and REVOKE

- Privileges granted or revoked are the same for all versions

> DROP statement

- SQL DROP will drop all versions of a SQL procedure and all associated packages
 - packages that are remotely bound are not dropped
- To drop only one version of a procedure, and only the package associated with that version

```
ALTER PROCEDURE...DROP VERSION routine-version-id
```

Native SQL procedures are not shown in DISPLAY PROCEDURE output

> Unless:

- If specific native SQL procedures have been stopped, the procedure names and status will be displayed, but the statistics will be all zeros

```
PROCEDURE STATUS ACTIVE QUED MAXQ TIMEOUT FAIL WLM_ENV  
MEDIAN_RESULT_SET  
STOPQUE 0 0 0 0 0  
NODIFF  
STOPQUE 0 0 0 0 0
```

- If all procedures in a schema are stopped:

```
----- SCHEMA=PAOLOR3  
DSNX9DIS PROCEDURES A - Z* STOP QUEUE
```

- A native SQL procedure that is currently being debugged will show under DISPLAY PROCEDURE command as in ACTIVE state. This is because the procedure is executing in a WLM environment.

Native SQL procedures special registers

> CURRENT DEBUG MODE

- DISALLOW
- ALLOW
- DISABLE

> CURRENT ROUTINE VERSION

- Set to specific version- affects all routines from then on
- Set to empty string to go back to default

Calling native SQL procedure from Rexx

```
SUBSYS = "S91A"

ADDRESS TSO "SUBCOM DSNREXX"      /* HOST CMD ENV AVAILABLE ? */
IF RC <> 0 THEN S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')

say 'About to connect...'
ADDRESS DSNREXX "CONNECT" SUBSYS
IF SQLCODE <> 0 THEN CALL SQLCA

say 'About to call SP...'
ADDRESS DSNREXX
/* Identify Stored procedure, define host variables */

STOPRO = "ANDRO16.RETURNDEPTSALARY"
DEPT    = "D11"
SALARY  = 123456789.99
BONUS   = 123456789

EXECSQL "CALL :STOPRO (:DEPT, :SALARY, :BONUS)"
IF SQLCODE <> 0 THEN CALL SQLCA

SAY "Stored Procedure: " STOPRO " seems to have worked!"
SAY "Department=" DEPT
SAY "Salary="      SALARY
SAY "Bonus="       BONUS
```

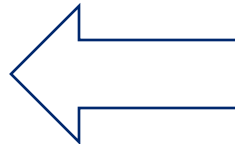

Rexx notes

- > Check if DSNREXX environment exists, if not
 - Create DSNREXX environment
- > Connect to DB2 and check return code
- > ADDRESS the DSNREXX environment
- > Build host variables
 - Rexx has no declare
- > Issue Call to SP, check return code

```
CREATE PROCEDURE RETURNDEPTSALARY
```

```
(IN DEPTNUMBER CHAR(3),  
OUT DEPTSALARY DECIMAL(15,2),  
OUT DEPTBONUSCNT INT)
```

```
EXECSQL "CALL :STOPRO  
        (:DEPT,  
        :SALARY,  
        :BONUS) "
```



- > Display returned values

Batch Rexx job:

```
//ANDRO16C  JOB  (10031), 'ANDRO16', CLASS=A, MSGCLASS=X,  
//          MSGLEVEL=(1,1), REGION=0M, NOTIFY=ANDRO16  
//* Batch TSO  
//STEP1     EXEC  PGM=IKJEFT01, PARM=(SPCALL, 'D11')  
//* Concatenate DB2 libraries  
//STEPLIB   DD  DISP=SHR, DSN=SYS2.DEMOED.LINKLIB  
//          DD  DISP=SHR, DSN=DB2.DB2910.SDSNLOAD  
//          DD  DISP=SHR, DSN=S91A.PRIVATE.SDSNEXIT  
//SYSPRINT  DD  SYSOUT=*  
//SYSTSIN   DD  DUMMY  
//SYSTSPRT  DD  SYSOUT=*  
//* Rexx library  
//SYSEXEC   DD  DSN=SALESUP.DEMO.REXX, DISP=SHR
```

Results of Rexx in batch:

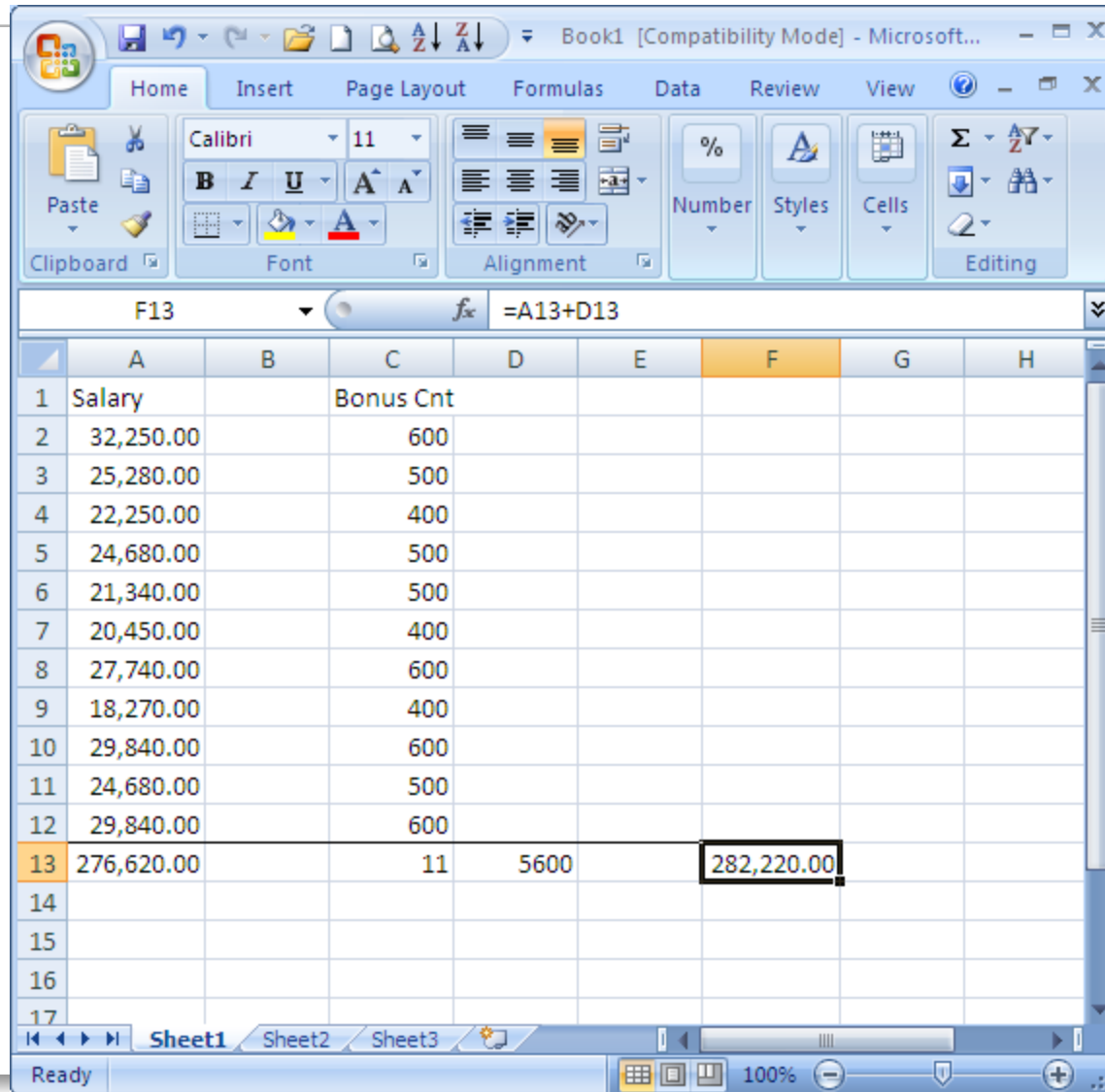
```
-----  
-----  
Jobname  Type      Jobnr Queue DDname  Stepname Procstep  
ANDRO16C JOB        1229 OUTP  SYSTSPRT STEP1  
-----  
-----  
  
...+....10...+....20...+....30...+....40...+....50...+....60...+....7!...+...  
.80...  
ACF0C038 ACF2 LOGONID ATTRIBUTES HAVE REPLACED DEFAULT USER ATTRIBUTES  
About to connect...  
About to call SP...  
Stored Procedure: seems to have worked!  
Department= D11  
Salary= 282220.00  
Bonus= 11  
READY  
END
```

Check Results, display table:

```
For Table    => DSN8910.EMP                > Row number=> 1 OF 11
Browse Mode => C                                Max Char  =>
SSID: S91A -----FETCH STATUS: COMPLETE-----
```

N:WORKDEPT	N:PHONENO	N:HIREDATE	N:JOB	N:EDLEVEL	N:SEX	N:BIRTHDATE	N:SALARY	N:BONUS	N
N D11	N 6423	N 1973-09-14	N MANAGER	N 16	N M	N 1945-07-07	N 32,250.00	N 600.00	N
N D11	N 4510	N 1972-02-12	N DESIGNER	N 16	N M	N 1947-05-17	N 25,280.00	N 500.00	N
N D11	N 3782	N 1977-10-11	N DESIGNER	N 17	N F	N 1955-04-12	N 22,250.00	N 400.00	N
N D11	N 2890	N 1978-09-15	N DESIGNER	N 16	N M	N 1951-01-05	N 24,680.00	N 500.00	N
N D11	N 1682	N 1973-07-07	N DESIGNER	N 17	N F	N 1949-02-21	N 21,340.00	N 500.00	N
N D11	N 2986	N 1974-07-26	N DESIGNER	N 16	N M	N 1952-06-25	N 20,450.00	N 400.00	N
N D11	N 4501	N 1966-03-03	N DESIGNER	N 16	N M	N 1941-05-29	N 27,740.00	N 600.00	N
N D11	N 0942	N 1979-04-11	N DESIGNER	N 17	N M	N 1953-02-23	N 18,270.00	N 400.00	N
N D11	N 0672	N 1968-08-29	N DESIGNER	N 18	N F	N 1948-03-19	N 29,840.00	N 600.00	N
N D11	N 2890	N 1978-09-15	N DESIGNER	N 16	N M	N 1951-01-05	N 24,680.00	N 500.00	N
N D11	N 0672	N 1968-08-29	N DESIGNER	N 18	N F	N 1948-03-19	N 29,840.00	N 600.00	N

Verify Answers:



Book1 [Compatibility Mode] - Microsoft...

Home Insert Page Layout Formulas Data Review View

Paste Clipboard Font Alignment Number Styles Cells Editing


















F13 $=A13+D13$

	A	B	C	D	E	F	G	H
1	Salary		Bonus Cnt					
2	32,250.00		600					
3	25,280.00		500					
4	22,250.00		400					
5	24,680.00		500					
6	21,340.00		500					
7	20,450.00		400					
8	27,740.00		600					
9	18,270.00		400					
10	29,840.00		600					
11	24,680.00		500					
12	29,840.00		600					
13	276,620.00		11	5600		282,220.00		
14								
15								
16								
17								

Sheet1 Sheet2 Sheet3

Ready 100%

Remember the EDM Pool:

R/EDMPOOL		EDM Pool - 30 Seconds						Row 640-656/660			
								Delta			
Pool	Total Pages	Free /Used	Pct Pool	1234567890	Full Fail	Requests	Loads	Loads	1234567890		
CT/PT	16996	16932	99.6		0						
CT		15	0.1								
PT		49	0.3								
SKEL	1280	656	51.3		0						
SKCT		46	3.6			0	0	0.0			
SKPT		578	45.2			90	0	0.0			
DBD	1250	1072	85.8		0						
used		178	14.2			45	0	0.0			
STMT	1250	760	60.8		0						
used		490	39.2			0	0	0.0			
STMT											
(2GB)	524287	524287	100.0		0						
CT		0	0.0								
PT		0	0.0								

Monitor CPU use

PROGRAM	TYPE	SQL	TIMEPCT	CPUPCT	INDB2_TIME	INDB2_CPU	GETPAGE	
DSNREXX	PKGE		6	69.31%	77.69%	00:05.916752	00:00.021759	67
RETURNND>	PROC		28	30.68%	22.30%	00:02.619462	00:00.006245	30

SQL_CALL	STMT#	SECT#	SQL	TIMEPCT	CPUPCT	INDB2_TIME	INDB2_CPU	
FETCH	00021	00002		2	74.35%	79.08%	00:01.947741	00:00.004939
FETCH	00028	00002		22	22.74%	13.38%	00:00.595789	00:00.000836
OPEN	00020	00002		2	2.89%	7.30%	00:00.075914	00:00.000456
CLOSE	00030	00002		2	.00%	.19%	00:00.000017	00:00.000012

Program Information

PROGRAM	-> RETURNDEPTSALARY	TYPE	-> PROC
SQL	-> 28	TIMEPCT	-> 30.68%
CPUPCT	-> 22.30%	INDB2_TIME	-> 00:02.619462
INDB2_CPU	-> 00:00.006245		

Buffer Manager Activity

GETPAGE	-> 30	GETPFAIL	-> 0
SYNCREAD	-> 19	SPFETCH	-> 0
LPFETCH	-> 0	DYNPFETCH	-> 1
PFPAGES	-> 16	PAGEUPDT	-> 0
IMWRITE	-> 0	HPREAD	-> 0
HPREADF	-> 0	HPREADPGS	-> 0
HPWRITE	-> 0	HPWRITEF	-> 0
EXPANS	-> 0	REOPT	-> 0

Error Handling and Debugging

- > DEBUG MODE
- > Compound statements within condition handlers
- > GET DIAGNOSTICS

DEBUG_MODE Column

- > SYSIBM.SYSROUTINES
- > ALLOW/DISALLOW/DISABLE
- > ALLOW DEBUG MODE debugging with the Unified Debugger technology
- > Included in IBM Data Studio and Developer Workbench (DWB) products
- > SET CURRENT DEBUG MODE = ALLOW#
- > WLM ENVIRONMENT FOR DEBUG MODE DB9AWLM

Compound statements within condition handlers

- > Multiple statements can now be written in a condition handler body by using a compound statement
 - BEGIN/END block
- > Old way: IF (1=1) THEN/END IF
 - IF clause resets SQLCODE and SQLSTATE

GET DIAGNOSTICS

- > GET DIAGNOSTICS SQL statement enhanced for use with native SQL stored procedures
- > DB2_LINE_NUMBER: new keyword, returns line number:
 - Where an error is found parsing a dynamic statement
 - Where an error is found in parsing, binding, or executing a CREATE or ALTER statement for a native SQL procedure
 - When a CALL statement invokes a native SQL procedure and the procedure returns with an error
 - This information is not returned for an external SQL procedure
 - Only useful if SQLPL format used

Stacked diagnostics area support

> APAR PK43524

> GET DIAGNOSTICS

- CURRENT
- STACKED

This APAR introduces support for requesting information from the stacked diagnostics area within a native SQL procedure.

In native SQL procedure when a handler is invoked, a copy is made of the diagnostics area that caused the handler to be activated. This copy becomes the current diagnostics area within the handler. The original diagnostics area that caused the handler to be activated is referred to as the stacked diagnostics area within the handler.

To explicitly request information from the stacked diagnostics area, specify the new STACKED keyword as part of the GET DIAGNOSTICS statement. The CURRENT keyword can be specified to refer to the current diagnostics area within the handler, but use of the keyword is optional.

Migrating external SQL to native SQL

- > External SQL procedure
 - FENCED or EXTERNAL keywords
 - Remove if explicitly specified
 - Both were defaults in V8
- > Drop the existing stored procedure
- > Run CREATE PROCEDURE statement again leaving both keywords omitted
- > Consider using SQLPL as your default format

Migration considerations

- > Can only migrate your external SQL stored procedures when your DB2 9 for z/OS runs in new function mode.
- > In case of a fallback from NFM to either ENFM* or CM*
 - No changes can be made to native SQL procedures
 - Existing native SQL procedures will still work

References:

- > SG24-7604-00 DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond (Draft Redbook)
- > SC18-9854-03 SQL Reference
- > Web resource: IBM® Information Management Software for z/OS® Solutions Information Center

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc.apsg/db2z_callspfmap.p.htm