

# Beyond the Basics with SQL



**Skip Marchesani**  
**System *i* Developer**

**Newton, NJ 07860**  
**Woodbury, VT 05681**

**smarches@warwick.net**

**www.System*i*Developer.com**

Copyright 2008 System i Developer, LLC



## Beyond the Basics with SQL

---

### Disclaimer:

This presentation may contain examples of code and names of companies or persons. The code is given for presentation purposes and has not been tested by IBM and/or Skip Marchesani. Therefore IBM and/or Skip Marchesani does not guarantee the reliability, serviceability, or function of the code and the code is provided "AS IS". IBM AND/OR SKIP MARCHESANI EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE AND MERCHANTABILITY. Any names appearing in this presentation are designed to be fictitious and IBM and Skip Marchesani makes no representations as to the accuracy of the names or data presented in accordance therewith.

### Reproduction:

This presentation is the property of Skip Marchesani and System i Developer, LLC. Permission is granted to make a limited number of copies of this material for non-commercial purposes, providing this page is included with all copies. Express written permission is required for making copies for other purposes.

System i, iSeries, AS/400, i5/OS, OS/400, DB2/400, DB2 UDB are registered trademarks of the IBM Corporation

Copyright 2008 System i Developer, LLC

## Overview

---

### An Important Goal for this Presentation

- An important goal for this presentation is to show you that SQL is just another programming language. The more you work and play with it, the more you realize the power of SQL and what it can do for you as an application development or database manipulation tool. With a little thought and creativity you will find you can use SQL for things that at first glance you did not think possible.
- This presentation is based on the SQL function available in V5R3 of OS/400, assumes you have a basic understanding of relational database concepts and SQL, and that you are familiar with using the SELECT, INSERT, UPDATE, and DELETE statements.



Copyright 2008 System i Developer, LLC

## Sample Tables Used in Examples

---

### • Employee Table - EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

### • Department Table - DEP

DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares



Copyright 2008 System i Developer, LLC

## SQL Syntax Used in Examples

---

- **All UPPER CASE - SQL required syntax**
- **All lower case - SQL parameter data supplied by end user**
- **One or more characters enclosed in single quotes ('S') is a literal used for comparison purposes and is supplied by the end user.**
- **Anything enclosed in [ ] is optional SQL syntax or optional parameter data**



Copyright 2008 System i Developer, LLC

## Overview

---

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5

Copyright 2008 System i Developer, LLC

## Overview Part 1

---

- **COUNT, SUM, and AVG for data analysis**
- **DEC to format numeric columns**
- **AS to name a derived column**
- **SUBSTR and CONCAT for character columns**
- **CAST and DIGITS to change data type**
- **Summarizing Data with SELECT**
- **Other Interesting Stuff**
- **Summary**
- **V5R3 and V5R4 SQL Information Sources**



Copyright 2008 System i Developer, LLC

---

# COUNT, SUM, and AVG for Data Analysis



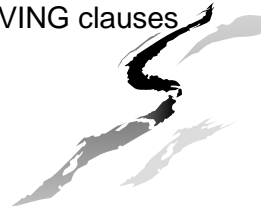
Copyright 2008 System i Developer, LLC

# Counting with SQL

---

## Types of Counting with SQL

- **Total number of rows in a table**
  - How many rows are in the table
- **Subset of rows in a table**
  - Based on selection criterion in WHERE clause
- **Distinct or unique occurrences of rows**
  - How many different (distinct or unique) values exist
- **Groups of rows**
  - Summarization of data using GROUP BY and HAVING clauses



Copyright 2008 System i Developer, LLC

# COUNT

---

## Two Types of COUNT

- **Column Function**
- **COUNT**
  - Result set is a large integer with precision (length) of 10
  - Max result set size is 10 digits with a limit of 2,147,483,647
- **COUNT BIG**
  - Max result set size is 32 digits
  - 9,999,999,999,999,999,999,999,999,999,999
  - aka COUNT REALLY REALLY BIG



Copyright 2008 System i Developer, LLC

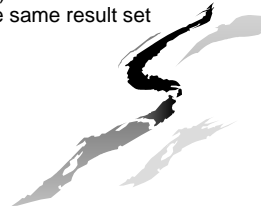
# COUNT...

---

## COUNT Syntax

- **COUNT(\*)**
  - Includes rows with null values in the count
- **COUNT(column\_name) or COUNT(ALL column\_name)**
  - Omits rows with null values from the count
- **COUNT(DISTINCT column\_name)**
  - Omits rows with duplicate or null values from the count

Note: If database tables are not defined with null capable columns, COUNT(\*) and COUNT(column\_name) or COUNT(ALL column\_name) will return the same result set



Copyright 2008 System i Developer, LLC

# COUNT...

---

## COUNT Examples

```
SELECT COUNT( * ) FROM emp
```

COUNT(*) 6
---------------

```
SELECT COUNT( * ) FROM emp  
WHERE dpt = 977
```

COUNT(*) 2
---------------



Copyright 2008 System i Developer, LLC

# COUNT...

---

## COUNT Examples...

```
SELECT COUNT( DISTINCT class ) FROM emp
```

COUNT(*)
4

```
SELECT dpt, COUNT( * ) FROM emp
GROUP BY dpt
ORDER BY dpt
```

DPT	COUNT(*)
901	1
911	2
977	2
990	1



Copyright 2008 System i Developer, LLC

# Summing or Adding with SQL

---

## SUM Column Function

- **Result set is the sum of a set of numbers with 1 to many numbers in the set**
  - Precision (length) is 31 with a scale (number of decimal positions) the same as the column being summed
- **Data type of column must be numeric**
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double



Copyright 2008 System i Developer, LLC

# SUM

---

## SUM Syntax

- **SUM(column\_name) or SUM(ALL column\_name)**
  - Sum the value found in the column for all rows selected
- **SUM(DISTINCT column\_name)**
  - Duplicate values are not included in the sum



Copyright 2008 System i Developer, LLC

# SUM...

---

## SUM Examples

```
SELECT SUM(sal) FROM emp
```

SUM(sal) 36,700
--------------------

```
SELECT SUM(sal) FROM emp  
WHERE dpt = 911
```

SUM(sal) 14,500
--------------------



Copyright 2008 System i Developer, LLC

# Averaging Data with SQL

---

## AVG Column Function

- **Result set is the average of a set of numbers with 1 to many numbers in the set**
  - Precision (length) is 31 with a scale (number of decimal positions) equal to 31 minus defined column length
    - Translation: Lots of decimal positions
- **Data type of column must be numeric**
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double



Copyright 2008 System i Developer, LLC

# AVG

---

## AVG Syntax

- **AVG(column\_name) or AVG(ALL column\_name)**
  - Average the value found in the column for all rows selected
- **AVG(DISTINCT column\_name)**
  - Duplicate values are not included in the average



Copyright 2008 System i Developer, LLC

# AVG...

---

## AVG Examples

**SELECT AVG(sal) FROM emp**

AVG(sal)
6,116.6666666666666666666666666666

**SELECT AVG(sal) FROM emp  
WHERE dpt = 911**

AVG(sal)
7,250.0000000000000000000000000000

- Ugly to look at!



Copyright 2008 System i Developer, LLC

---

# DEC to Format Numeric Results



Copyright 2008 System i Developer, LLC

# DEC to Format Numeric Results

---

## DEC or DECIMAL Scalar Column Function

- **Result set is the representation of a number in decimal format (1234.56)**
  - Precision (length) and scale (number of decimal positions) are defined as part of the DEC function
- **Data type of column must be numeric**
  - Decimal - packed
  - Numeric - zoned
  - Smallint
  - Integer
  - Bigint
  - Float
  - Real
  - Double



Copyright 2008 System i Developer, LLC

# DEC

---

## DEC or DECIMAL Syntax

- **DEC(expression, precision, scale, [ 'decimal-character' ])**



Copyright 2008 System i Developer, LLC



## To Truncate or Round?

---

What is the Correct Answer?

```
SELECT DEC(AVG(sal),7,2) FROM emp
```

DEC 6,116.66
-----------------

**6,116.66**    or    **6,116.67**



Copyright 2008 System i Developer, LLC

## Rounding a Result

---

### ROUND Scalar Column Function

- Returns a numeric value rounded to some number of places to the right or left of the decimal point
- **ROUND(expression, decimal-position)**
  - Postive number - round to right of decimal point
  - Negative number - round to left of decimal point



Copyright 2008 System i Developer, LLC

## Rounding a Result...

---

### ROUND Scalar Column Function

```
SELECT DEC(ROUND(AVG(sal),2),7,2)
FROM emp
```

DEC 6,116.67
-----------------



Copyright 2008 System i Developer, LLC

## What's the Difference?

---

### Column Headings Reflect the Function

```
SELECT DEC(AVG(sal),7,2) FROM EMP
```

DEC 6,116.66
-----------------

```
SELECT DECIMAL(AVG(sal),7,2) FROM emp
WHERE dpt = 911
```

DEC 7,250.00
-----------------

- Easy to fix



Copyright 2008 System i Developer, LLC

---

# AS to Name a Derived Column



Copyright 2008 System i Developer, LLC

## AS to Name a Derived Column

---

### AS Operator

- **Used to**
  - Rename an existing column
  - Name a derived column
- **Considerations**
  - Name cannot be qualified
  - Name does not have to be unique



Copyright 2008 System i Developer, LLC

# AS

---

## AS Syntax

- AS column\_name



Copyright 2008 System i Developer, LLC

# AS...

---

## DEC or DECIMAL Examples

```
SELECT DEC(AVG(sal),7,2) FROM EMP
```

```
DEC  
6,116.67
```

```
SELECT DECIMAL(AVG(sal),7,2) FROM emp  
WHERE dpt = 911
```

```
DEC  
7,250.00
```



Copyright 2008 System i Developer, LLC

## AS...

---

### AS Examples

```
SELECT DEC(AVG(sal),7,2) AS avgsal  
FROM EMP
```

AVGSAL 6,116.67
--------------------

```
SELECT DECIMAL(AVG(sal),7,2) AS avg911sal  
FROM emp  
WHERE dpt = 911
```

AVG911SAL 7,250.00
-----------------------



Copyright 2008 System i Developer, LLC

## AS...

---

### AS Can Be Implied

```
SELECT DEC(AVG(sal),7,2) avgsal  
FROM EMP
```

AVGSAL 6,116.67
--------------------

```
SELECT DECIMAL(AVG(sal),7,2) avg911sal  
FROM emp  
WHERE dpt = 911
```

AVG911SAL 7,250.00
-----------------------



Copyright 2008 System i Developer, LLC

---

# SUBSTR and CONCAT for Character Columns



Copyright 2008 System i Developer, LLC

## SUBSTR for a Character Column

---

### SUBSTR or SUBSTRING Scalar Function

- **Result is a subset or substring of a character column**
  - Start position and length of resulting string is defined in the function
- **Data type must be character**
  - Fixed or variable length - including large object (BLOB)
  - Single or double byte



Copyright 2008 System i Developer, LLC

## SUBSTR...

---

### SUBSTR or SUBSTRING Syntax

- **SUBSTR(string\_expression, start\_position, length)**
- **SUBSTRING(string\_expression FROM start\_position FOR length)**



Copyright 2008 System i Developer, LLC

## SUBSTR...

---

### SUBSTR or SUBSTRING Examples

```
SELECT dpt, SUBSTR(dnm,1,4) AS sname
FROM dep
```

DPT	SNAME
901	Acco
977	Manu
911	Sale
990	Spar

DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares



Copyright 2008 System i Developer, LLC

# CONCAT for a Character Column

---

## CONCAT Scalar Function

- **Combines or concatenates two string expressions together**
- **Result is a single string consisting of the first string expression, followed by the second string expression**
- **Data type must be character**
  - Fixed or variable length - including large object (BLOB)
  - Single or double byte



Copyright 2008 System i Developer, LLC

# CONCAT...

---

## CONCAT Syntax - Industry Standard

- **CONCAT(string\_expression\_01, string\_expression\_02)**

## CONCAT Syntax - Non Standard

- **string\_01 || string\_02 || string\_03**
  - || = double pipe (upper case backward slash)



Copyright 2008 System i Developer, LLC

## CONCAT...

---

### CONCAT Examples - Simple CONCAT

```
SELECT nbr, nam,
       CONCAT(sex, nam) AS sexnam
FROM emp
```

NBR	NAM	SEXNAM
10	Ed	MEd
20	Heikki	MHeikki
30	John	MJohn
40	Mike	MMike
50	Marcela	FMarcela
60	Frank	MFrank



Copyright 2008 System i Developer, LLC

## CONCAT...

---

### CONCAT Examples - CONCAT with SUBSTR

```
SELECT dpt,
       CONCAT(SUBSTR(dnm,1,4), SUBSTR(dnm,1,4))
       AS double
FROM dep
```

DPT	DOUBLE
901	AccoAcco
977	ManuManu
911	SaleSale
990	SparSpar



Copyright 2008 System i Developer, LLC

## CONCAT...

---

### CONCAT Examples - CONCAT with a Literal

```
SELECT dpt,
       CONCAT(SUBSTR(dnm,1,4), 'Stuff')
       AS stuff
FROM dep
```

DPT	STUFF
901	AccoStuff
977	ManuStuff
911	SaleStuff
990	SparStuff



Copyright 2008 System i Developer, LLC

## CONCAT...

---

### CONCAT Examples - Concat with 3 Strings

```
SELECT dpt,
       CONCAT(CONCAT(SUBSTR(dnm,1,4), '-'), 'Stuff')
       AS hyphen
FROM dep
```

DPT	HYPHEN
901	Acco-Stuff
977	Manu-Stuff
911	Sale-Stuff
990	Spar-Stuff



Copyright 2008 System i Developer, LLC

## CONCAT...

---

### CONCAT Examples - Concat with 3 Strings

```
SELECT dpt,  
       SUBSTR(dnm,1,4) || '-' || 'Stuff'  
       AS hyphen  
FROM dep
```

DPT	HYPHEN
901	Acco-Stuff
977	Manu-Stuff
911	Sale-Stuff
990	Spar-Stuff



Copyright 2008 System i Developer, LLC

## SUBSTR & CONCAT

---

### SUBSTR & CONCAT with Numeric Columns

- Can be used with numeric columns but data type must be changed from numeric to character



Copyright 2008 System i Developer, LLC

---

# CAST and DIGITS to Change Data Type



Copyright 2008 System i Developer, LLC

## CAST to Change Data Type

---

### CAST Operation

- **Converts the existing data type of the cast expression to the specified data type**
  - Numeric to character
  - Character to numeric
  - Numeric to numeric
  - Character to character
  - See SQL Reference for supported CAST from/to data types
- **When converting numeric to character**
  - Leading zeros are truncated
  - Decimal point ignored and digits to the right are part of result string
  - Digits are left justified

Copyright 2008 System i Developer, LLC

# DIGITS to Change Data Type

---

## DIGITS Scalar Function

- **Only converts numeric to character data type**
- **When converting numeric to character**
  - Leading zeros are NOT truncated
  - Decimal point ignored and digits to the right are part of result string
  - Digits are left justified



Copyright 2008 System i Developer, LLC

# CAST and DIGITS

---

## CAST Syntax

- **CAST(expression AS data\_type)**

## DIGITS Syntax

- **DIGITS(numeric\_expression)**



Copyright 2008 System i Developer, LLC

## CAST and DIGITS...

---

### CAST and DIGITS Example

```
SELECT nbr, nam,
       SUBSTR(CAST(dpt AS CHAR(3)),2,2) AS chardpt
FROM emp
```

NBR	NAM	CHARDPT
10	Ed	11
20	Heikki	01
30	John	77
40	Mike	77
50	Marcela	11
60	Frank	90

```
SELECT nbr, nam,
       SUBSTR(DIGITS(dpt),2,2) AS chardpt
FROM emp
```

Copyright 2008 System i Developer, LLC



## CAST and DIGITS...

---

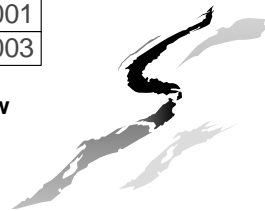
### CAST and DIGITS Example

```
SELECT * FROM empd
ORDER BY nbr
```

NBR	NAM	CLS	SEX	DPT	SAL	EDATE
10	Ed	5	M	911	7,000	11,152,000
20	Heikki	2	M	901	6,000	03,122,002
30	John	5	M	977	3,200	05,152,001
40	Mike	4	M	977	6,500	10,152,000
50	Marcela	3	F	911	7,500	12,012,001
60	Frank	2	M	990	6,500	09,152,003

**Note: Neither Interactive SQL nor RUN SQL Scripts will show leading zeros in EDATE**

Copyright 2008 System i Developer, LLC



## CAST and DIGITS...

---

### CAST and DIGITS Example

```
SELECT * FROM empd
ORDER BY edate
```

NBR	NAM	CLS	SEX	DPT	SAL	EDATE
20	Heikki	2	M	901	6,000	03,012,002
30	John	5	M	977	3,200	05,152,001
60	Frank	2	M	990	6,500	09,152,003
40	Mike	4	M	977	6,500	10,152,000
10	Ed	5	M	911	7,000	11,152,000
50	Marcela	3	F	911	7,500	12,012,001

**Note: Neither Interactive SQL nor RUN SQL Scripts will show leading zeros in EDATE**

Copyright 2008 System i Developer, LLC

## CAST and DIGITS...

---

### CAST Truncates Leading Zeros

```
SELECT nbr, nam, edate,
       CONCAT(SUBSTR(CAST(edate AS CHAR(8)),5,4),
             SUBSTR(CAST(edate AS CHAR(8)),1,4))
       AS hire_date
FROM empd ORDER BY hire_date
```

NBR	NAM	EDATE	HIRE_DATE
30	John	5152001	001 5152
20	Heikki	3012002	002 3012
60	Frank	9152003	003 9152
40	Mike	10152000	20001015
10	Ed	11152000	20001115
50	Marcela	12012001	20011201

**CAST turned 05152001 into 5152001\_**

Copyright 2008 System i Developer, LLC

## CAST and DIGITS...

---

### DIGITS Preserves Leading Zeros

```

SELECT nbr, nam, edate,
       CONCAT(SUBSTR(DIGITS(edate),5,4),
              SUBSTR(DIGITS(edate),1,4)) AS hire_date
FROM empd
ORDER BY hire_date

```

NBR	NAM	EDATE	HIRE_DATE
40	Mike	10152000	20001015
10	Ed	11152000	20001115
30	John	5152001	20010515
50	Marcela	12012001	20011201
20	Heikki	3012002	20020301
60	Frank	9152003	20030915



Copyright 2008 System i Developer, LLC

---

## Summarizing Data with SELECT



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT

---

### The SELECT statement is an *Either Or* Proposition

- *Either* return detail rows in the result set
- *Or* return summarized data in the result set
- A single SQL statement has no capability to do
  - Detail
  - Detail
  - Level break
  - Detail
  - Level break
- Query Manager **CAN** do above
  - SQL based query product



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### SELECT Statement Clauses

**SELECT . . . . . (columns, \*, or expressions)**  
**FROM . . . . . (tables or views)**  
**WHERE . . . . . (row selection criteria)**  
**GROUP BY. . . (row summarization criteria)**  
**HAVING. . . . . (GROUP BY selection criteria)**  
**ORDER BY. . . (column ordering criteria)**



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### GROUP BY Clause

- Defines grouping or summary criteria for SELECT
- Format

```
SELECT fldA, fldE, FUNCTION(fldG), FUNCTION(fldH)
   FROM table-name
   WHERE selection-criteria
   GROUP BY fldE, fldA
   ORDER BY fldA, fldE
```

- If a column referenced in the column list of the SELECT statement is not operated on by a function, that column must be referenced as part of the grouping criteria in the GROUP BY clause

Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### GROUP BY Clause...

- For each department list the total salary and total number of employees

```
SELECT dpt, SUM(sal), COUNT ( * )
   FROM emp
   GROUP BY dpt
   ORDER BY dpt
```

DPT	SUM(SAL)	COUNT(*)
901	6,000	1
911	14,500	2
977	9,700	2
990	6,500	1



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### HAVING Clause...

- Defines **GROUP BY** selection criteria
- **Format**

```
SELECT fldA, fldE, FUNCTION(fldG), FUNCTION(fldH)
   FROM table-name
   WHERE selection-criteria
   GROUP BY fldE, fldA
   HAVING group-by-selection-criteria
   ORDER BY fldA, fldE
```

Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### HAVING Clause...

- **For those departments that have more than one employee, list the total salary and total number of employees**

```
SELECT dpt, SUM(sal) AS saltot,
        COUNT(*) AS emptot
   FROM emp
   GROUP BY dpt
   HAVING COUNT(*) > 1
   ORDER BY dpt
```

DPT	SALTOT	EMPTOT
911	14,500	2
977	9,700	2



Copyright 2008 System i Developer, LLC

---

# Other Interesting Stuff



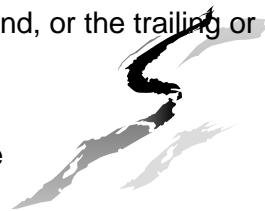
Copyright 2008 System i Developer, LLC

## Other Interesting Stuff

---

### Some Other Interesting SQL Scalar Functions

- **CASE**
  - Allows selection and substitution of a value based on a condition
- **RRN**
  - Returns the relative record number of a row in a table
- **HEX**
  - Returns the hexadecimal representation of a value
- **RTRIM (Right Trim)**
  - Removes blanks or hexadecimal zeros from the end, or the trailing or right side, of a string expression
- **LENGTH**
  - Returns the length of a number or character value



Copyright 2008 System i Developer, LLC

## Other Interesting Stuff...

---

### New V5R3 Scalar Functions

- **INSERT**

- Inserts a number or character string into an existing string and optionally overlays specific positions in the existing string with the insert string

- **REPLACE**

- Replaces all occurrences of a search string in a number or character string with a replacement string



Copyright 2008 System i Developer, LLC

## CASE for Substitution of a Value

---

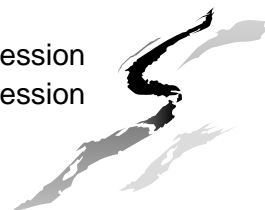
### Two forms of CASE Syntax

- **Simple WHEN clause**

```
CASE expression
  WHEN expression THEN result-expression
  WHEN expression THEN result-expression
  ...
  ELSE result-expression
END CASE
```

- **Searched WHEN clause**

```
CASE
  WHEN search-condition THEN result expression
  WHEN search-condition THEN result expression
  ...
  ELSE result-expression
END CASE
```



Copyright 2008 System i Developer, LLC

## CASE...

---

### Two forms of Case...

- **Simple WHEN clause**

```
SELECT nbr, nam,  
       CASE dpt  
         WHEN 901 THEN 'Accounts'  
         WHEN 911 THEN 'Sales'  
         WHEN 977 THEN 'Manufact'  
         WHEN 990 THEN 'Spares'  
         ELSE 'No Name'  
       END CASE  
FROM emp ORDER BY nbr
```



Copyright 2008 System i Developer, LLC

## CASE...

---

### Two forms of Case...

- **Searched WHEN clause**

```
SELECT nbr, nam,  
       CASE  
         WHEN dpt = 901 THEN 'Accounts'  
         WHEN dpt = 911 THEN 'Sales'  
         WHEN dpt = 977 THEN 'Manufact'  
         WHEN dpt = 990 THEN 'Spares'  
         ELSE 'No Name'  
       END CASE  
FROM emp ORDER BY nbr
```



Copyright 2008 System i Developer, LLC

## CASE...

---

### Two forms of Case...

- **Results Set for both CASE Examples**

NBR	NAM	CASE
10	Ed	Sales
20	Heikki	Accounts
30	John	Manufact
40	Mike	Manufact
50	Marcela	Sales
60	Frank	Spares



Copyright 2008 System i Developer, LLC

## RRN to Find Relative Record Number

---

### RRN Scalar Function

- **Result set is the relative record number of a row**
  - Length is 15 with zero decimal positions

### RRN Syntax

- **RRN(table\_name)**



Copyright 2008 System i Developer, LLC

# RRN

---

## RRN Example

```
SELECT nbr, nam, RRN(ax) AS rec
FROM emp ax ORDER BY nbr
```

NBR	NAM	REC
10	Ed	2
20	Heikki	1
30	John	5
40	Mike	4
50	Marcela	3
60	Frank	6



Copyright 2008 System i Developer, LLC

# HEX to Find Hexadecimal Value

---

## HEX Scalar Function

- **Result set is the hexadecimal representation of a value**
  - Length is twice the defined length with a max length of approx 32K

## HEX Syntax

- **HEX(expression)**



Copyright 2008 System i Developer, LLC

# HEX

---

## HEX Example

```
SELECT nbr, nam, HEX(nam) AS hex_nam
FROM emp ORDER BY nbr
```

NBR	NAM	HEX_NAM
10	Ed	C58440404040404040
20	Heikki	C88589929289404040
30	John	D19688954040404040
40	Mike	D48992854040404040
50	Marcela	D48199838593814040
60	Frank	C69981959240404040



Copyright 2008 System i Developer, LLC

# RTRIM to Remove Trailing Blanks

---

## RTRIM Scalar Function

- **Result set is the RTRIM expression with trailing blanks or hexadecimal zeros removed**
  - Length of the result set is the length of the expression minus the number of blanks or hex zeros removed

## RTRIM Syntax

- **RTRIM(expression)**



Copyright 2008 System i Developer, LLC

# RTRIM

---

## RTRIM Example

```
SELECT nbr, nam, RTRIM(nam) AS tnam
FROM emp ORDER BY nbr
```

NBR	NAM	TNAM
10	Ed	Ed
20	Heikki	Heikki
30	John	John
40	Mike	Mike
50	Marcela	Marcela
60	Frank	Frank

- What's the difference between NAM and TNAM?



Copyright 2008 System i Developer, LLC

# LENGTH to Determine the Length of a Value

---

## LENGTH Scalar Function

- Result set is the length of the string in the expression including any blanks or zeros
  - Result set length is 10 digits with zero decimals

## LENGTH Syntax

- LENGTH(expression)



Copyright 2008 System i Developer, LLC

# LENGTH

---

## LENGTH Example

```
SELECT nbr, nam, LENGTH(nam) AS len1
FROM emp ORDER BY nbr
```

NBR	NAM	LEN1
10	Ed	10
20	Heikki	10
30	John	10
40	Mike	10
50	Marcela	10
60	Frank	10

- Why is the length for each NAM string = 10?



Copyright 2008 System i Developer, LLC

# Combining RTRIM and LENGTH

---

## How Many Non Blank Characters are in Each Name?

```
SELECT nbr, nam, LENGTH(RTRIM(nam)) AS len2
FROM emp ORDER BY nbr
```

NBR	NAM	LEN2
10	Ed	2
20	Heikki	6
30	John	4
40	Mike	4
50	Marcela	7
60	Frank	5



Copyright 2008 System i Developer, LLC

## INSERT to Overlay Positions In a String

---

### INSERT Scalar Function - New for V5R3

- **Result set is the source string with the insert string placed into the source string at the specified start position, and optionally specified positions in the source string overlaid with the insert string**
  - Result set length is length of the source string plus length of the insert string minus the number of positions overlaid (if any)
  - Result set length cannot exceed the maximum length of the data type
  - Data type of the source string and insert string must be compatible

### INSERT Syntax

- **INSERT(source\_string, start, length, insert\_string)**

Copyright 2008 System i Developer, LLC

## INSERT

---

### INSERT Example with No Overlay

```
SELECT nbr, nam, INSERT(nam, 3, 0, 'xy') AS insrt1
FROM emp ORDER BY nbr
```

NBR	NAM	INSRT1
10	Ed	Edxy
20	Heikki	Hexyikki
30	John	Joxyhn
40	Mike	Mixyke
50	Marcela	Maxyrcela
60	Frank	Frxyank



Copyright 2008 System i Developer, LLC

## INSERT...

---

### INSERT Example Overlaying Positions 3 and 4

```
SELECT nbr, nam, INSERT(nam, 3, 2, 'xy') AS insrt2
FROM emp ORDER BY nbr
```

NBR	NAM	INSRT2
10	Ed	Edxy
20	Heikki	Hexyki
30	John	Joxy
40	Mike	Mixy
50	Marcela	Maxyela
60	Frank	Frxyk

- **Easy way to update part of a character string!**



Copyright 2008 System i Developer, LLC

## REPLACE - Based on a Search String

---

### REPLACE Scalar Function - New for V5R3

- **Result set is the source string with all occurrences of the search string replaced with the replace string**
  - Result set length is length of the source string plus length of the replace string minus the length of the search string
  - Result set length cannot exceed the maximum length of the data type
  - Data type of the source string and replace string must be compatible
  - If no match with the search string, the source string is returned unchanged as the result set

### REPLACE Syntax

- **REPLACE(source\_string, search\_string, replace\_string)**

Copyright 2008 System i Developer, LLC

# REPLACE

---

## REPLACE Example

```
SELECT nbr, nam, REPLACE(nam, 'ik', 'qqq') AS replc1
FROM emp ORDER BY nbr
```

NBR	NAM	REPLC1
10	Ed	Ed
20	Heikki	Heqqqki
30	John	John
40	Mike	Mqqqqe
50	Marcela	Marcela
60	Frank	Frank

- Easy way to update part of a character string!



Copyright 2008 System i Developer, LLC

## Summary

---

### I Didn't Know You Could Do that with SQL!

- COUNT, SUM, and AVG for data analysis
- DEC to format numeric columns
- AS to name a derived column
- SUBSTR and CONCAT for character columns
- CAST and DIGITS to change data type
- Summarizing Data with SELECT
- CASE
- HEX and RRN
- RTRIM and LENGTH
- INSERT and REPLACE - New for V5R3



Copyright 2008 System i Developer, LLC

## Summary...

---

### I Didn't Know You Could Do that with SQL...

- From this presentation you should have a better understanding of SQL as a programming language.
- The more you work and play with it, the more you realize the power of SQL and what it can do for you as an application development or database manipulation tool.
- With a little thought and creativity you will find you can use SQL for things that at first glance you did not think possible.



Copyright 2008 System i Developer, LLC

## V5R3 and V5R4 SQL Information Sources

---

- **iSeries Information Center Publications - Web or CD**
  - SQL Reference
  - SQL Programming
  - Embedded SQL Programming
  - Query Manager Use
  - SQL Messages and Codes
- **To access Info Center on the Web**
  - <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp>
  - <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>
    - In left scroll bar
      - Click on iSeries Information Center ...
      - Click on Database
      - Click on Printable PDFs
      - Use right scroll bar to scroll down to above SQL publication
- **DB2 UDB for iSeries on the Web**
  - <http://www.ibm.com/servers/eserver/iseries/db2/>

Copyright 2008 System i Developer, LLC



---

# Part 2

Copyright 2008 System i Developer, LLC

## Overview

---

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5

Copyright 2008 System i Developer, LLC

## Overview Part 2

---

- **Working with Edit Characters**
- **Basic Subselect**
- **Scalar Subselect**
- **Subselect and INSERT, UPDATE, & DELETE**
- **Subselect and CREATE TABLE**
- **Subselect and Derived Table**
- **Summarizing Data with SELECT**
- **Identifying Potential Duplicate Rows**
- **Search by Sound with SOUNDEX**
- **Summary**
- **V5 SQL Information Sources**



Copyright 2008 System i Developer, LLC

## SQL Syntax Used in Examples

---

- **All UPPER CASE - SQL required syntax**
- **All lower case - SQL parameter data supplied by end user**
- **One or more characters enclosed in single quotes ('S') is a literal used for comparison purposes and is supplied by the end user.**
- **Anything enclosed in [ ] is optional SQL syntax or optional parameter data**



Copyright 2008 System i Developer, LLC

---

# Working with Edit Characters



Copyright 2008 System i Developer, LLC

## Working with Edit Characters

---

### Transferring data to/from i5, iSeries, or AS/400

- **When data is transferred *from* a client PC to an iSeries it could contain edit characters that may not be compatible with the DB2 UDB (DB2/400) data format.**
- **When data is transferred *to* a client PC from an iSeries it might lack edit characters that may be required by the PC end user.**
- **Examples:**
  - Telephone number: 973-555-1212
  - Social Security number: 132-54-7698
  - Date: 12/25/03
  - Time: 10:30 am



Copyright 2008 System i Developer, LLC

## Working with Edit Characters...

---

### The Question

- Can SQL be used to *easily* remove extraneous edit characters or insert required edit characters.

### The Answer

- With a little thought and some creativity the answer is *YES!*



Copyright 2008 System i Developer, LLC

## Case Study

---

### PC Name and Address File

- Excel spreadsheet
- Character (alpha) phone number
  - Format is Char(12) AAA-EEE-NNNN
    - AAA = Area Code
    - EEE = Phone Exchange
    - NNNN = Phone Number
  - Hyphens for editing included as part of the data
- PC file to be imported into DB2 UDB database



Copyright 2008 System i Developer, LLC

## Case Study...

---

### PC File to be Imported into DB2 UDB Database

- **Name and Address Table - NAMEMSTR**
- **Decimal (numeric) phone number - PHONE**
  - Format is Dec(10,0) AAEEEEENNNN
    - AAA = Area Code
    - EEE = Phone Exchange
    - NNNN = Phone Number
  - Hyphens are not valid in a numeric column



Copyright 2008 System i Developer, LLC

## Case Study...

---

### PC File Layout

- **IDNBR** dec
- **FNAME** char
- **LNAME** char
- **COMPNY** char
- **ADDRL1** char
- **ADDRL2** char
- **CITY** char
- **STATE** char
- **ZIP** char
- **PHONEA** char  
AAA-EEE-NNNN

### DB2 UDB Table Format

- **IDNBR** dec
- **FNAME** char
- **LNAME** char
- **COMPNY** char
- **ADDRL1** char
- **ADDRL2** char
- **CITY** char
- **STATE** char
- **ZIP** char
- **PHONE** dec  
AAEEEEENNNN
- **Other columns**



Copyright 2008 System i Developer, LLC

## Case Study...

---

### The Question:

- How can SQL be used to remove the hyphens in the character phone number and then change it to a decimal data type?



Copyright 2008 System i Developer, LLC

## Removing the Hyphens

---

### Import the PC File

- Use Excel *Transfer Data to iSeries* to import PC file into a work table in DB2 UDB
  - Assign import file name as **NAMEWORK**
  - Assign phone number column name as **PHONEA**
  - Verify that phone number is defined as Char(12)

Note: *Transfer Data To/From iSeries* is an Excel plug-in that is part of the licensed product for File Transfer within iSeries Access Express. When installing iSeries Access, you must select the option to install File Transfer to have the Excel plug-in installed.



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Import the PC File and Add PHONE Column...

- Add second phone number column with **PHONE** as name and data type of **Dec(10,0)** to match same column in **NAMEMSTR**

```
ALTER TABLE NAMEWORK  
ADD COLUMN PHONE DEC (10 , 0)  
NOT NULL WITH DEFAULT
```



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Format of PHONEA is Char(12) AAA-EEE-NNNN

- Area code is in positions 1 thru 3
- 1st hyphen is in position 4
- Phone exchange is in positions 5 thru 7
- 2nd hyphen is position 8
- Phone number is in positions 9 thru 12



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Update PHONE in NAMEWORK

- Update the column PHONE by removing the hyphens from PHONEA and then changing its data type to Dec(10,0)

```
UPDATE NAMEWORK SET PHONE =  
    CAST(CONCAT(CONCAT(SUBSTR(PHONEA,1,3),  
        SUBSTR(PHONEA,5,3)),  
        SUBSTR(PHONEA,9,4)) AS DEC(10,0))
```



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Update PHONE in NAMEWORK

- Alternate non standard CONCAT syntax

```
UPDATE NAMEWORK SET PHONE =  
    CAST(SUBSTR(PHONEA,1,3) ||  
        SUBSTR(PHONEA,5,3) ||  
        SUBSTR(PHONEA,9,4) AS DEC(10,0))
```



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### CONCAT Builds PHONE w/o Hyphens - Dec(10,0)

- Join two character strings into a single string
- **Right (or 2nd) CONCAT**
  - AAA + EEE = AAEEEE
  - 123 + 567
- **Left (or 1st) CONCAT**
  - AAEEEE + NNNN = AAEEEENNNN
  - 123567 + 9101112

### CAST Function

- **Changes Char(12) to Dec(10,0)**
  - Assuming valid numeric characters in the column



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Insert/Add NAMEWORK Rows into NAMEMSTR

- **PHONE** exists as a 10 position numeric field with no hyphens
- **NAMEWORK** rows can be inserted into **NAMEMSTR** using a **SUBSELECT**
  - More on SUBSELECT later

```
INSERT INTO NAMEMSTR (IDNBR, FNAME, LNAME,
COMPNY, ADDR1, ADDR2, CITY, STATE,
ZIP, CONTRY, PHONE)
SELECT IDNBR, FNAME, LNAME, COMPNY,
ADDR1, ADDR2, CITY, STATE, ZIP,
CONTRY, PHONE
FROM NAMEWORK
```



Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Considerations for Inserting NAMEWORK Rows

- **PHONEA not included when inserting rows from NAMEWORK into NAMEMSTR**
- **Column names in INTO clause and SUBSELECT must be listed in the same position or sequence**
  - One for one, positional relationship between columns in the INTO clause and the columns in the SUBSELECT
  - Column names do not have to be the same but column attributes (data type) must be compatible
- **For name and address and possibly other table types, and prior to doing insert, consider checking NAMEWORK for duplicates that already exist in NAMEMSTR**
  - More on this later

Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Can It be Done as a One Step Process?

- **YES - Include the CONCAT, SUBSTR, and CAST of PHONEA as part of the SUBSELECT in the INSERT statement**

```

INSERT INTO NAMEMSTR (IDNBR, FNAME, LNAME,
                     COMPNY, ADDRL1, ADDRL2, CITY, STATE, ZIP,
                     CONTRY, PHONE)
  SELECT IDNBR, FNAME, LNAME, COMPNY, ADDRL1,
         ADDRL2, CITY, STATE, ZIP, CONTRY,
         CAST(CONCAT(CONCAT(SUBSTR(PHONEA,1,3),
                               SUBSTR(PHONEA,5,3)),
              SUBSTR(PHONEA,9,4)) AS DEC(10,0))
  FROM NAMEWORK
  
```

Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### There's an Easier Way in V5R3

- **REPLACE Scalar Function**
- **Syntax**

**REPLACE(source\_string, search\_string, replace\_string)**

```
SELECT idnbr, fname, lname, compny, addr1,
       addr2, city, state, zip, contry,
       REPLACE( phonea, '-', '' ) AS phone
FROM namework
```

Two single quotes next to each other (not a double quote)

Copyright 2008 System i Developer, LLC

## Removing the Hyphens...

---

### Can It be Done as a One Step Process?

- **YES - Include the REPLACE as part of the SUBSELECT in the INSERT statement**

```
INSERT INTO namemstr (idnbr, fname, lname,
                    compny, addr1, addr2, city, state, zip,
                    contry, phone)
SELECT idnbr, fname, lname, compny, addr1,
       addr2, city, state, zip, contry,
       CAST(REPLACE( phonea, '-', '' ) AS DEC(10,0))
FROM namework
```

Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability

---

### The Next Question:

- How can SQL be used to insert hyphens in the numeric phone number so the DB2 UDB table can be exported to an Excel file on the PC?



Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

**Format of PHONE is Dec(10,0) AAEEENNNN**

- Area code is in positions 1 thru 3
- Phone exchange is in positions 4 thru 6
- Phone number is in positions 7 thru 10



Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### Create Export File Using Interactive SQL - STRSQL

- **SELECT** required columns and use **CONCAT**, **SUBSTR**, and **CAST** to change **PHONE** data type to **Char(10)** and insert hyphens in the appropriate positions, and create **PHONEA**
- Use **F13** then **Option 1** to direct **SELECT** output to a file.

```
SELECT IDNBR, FNAME, LNAME, COMPNY, ADDR1,
       ADDR2, CITY, STATE, ZIP, CONTRY,
       CONCAT(CONCAT(CONCAT(CONCAT(
           SUBSTR(CAST(PHONE AS CHAR(10)),1,3), '-'),
           SUBSTR(CAST(PHONE AS CHAR(10)),4,3), '-'),
           SUBSTR(CAST(PHONE AS CHAR(10)),7,4)) AS PHONEA
FROM NAMEMSTR
```

Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### Create Export File Using Interactive SQL - STRSQL

- **Alternate non standard CONCAT syntax**
- Use **F13** then **Option 1** to direct **SELECT** output to a file.

```
SELECT IDNBR, FNAME, LNAME, COMPNY, ADDR1,
       ADDR2, CITY, STATE, ZIP, CONTRY,
       (SUBSTR(CAST(PHONE AS CHAR(10)),1,3) || '-' ||
        SUBSTR(CAST(PHONE AS CHAR(10)),4,3) || '-' ||
        SUBSTR(CAST(PHONE AS CHAR(10)),7,4)) AS PHONEA
FROM NAMEMSTR
```

Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### SUBSTR(CAST(PHONE AS CHAR(10)),start,length)

- **PHONE** is a numeric field - Dec(10,0), and only a character field can be operated on by CONCAT and SUBSTR.
- **PHONE** must be CAST or converted to a character field each time it is operated on by CONCAT or SUBSTR



Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### CONCAT Builds PHONEA with Hyphens - Char(12)

- **Right outer (or 4th) CONCAT**
  - SUBSTR(CAST(PHONE AS CHAR(10)),1,3), '-')
  - AAA-
- **Right inner (or 3rd) CONCAT**
  - SUBSTR(CAST(PHONE AS CHAR(10)),4,3),
  - AAA-EEE
- **Left inner (or 2nd) CONCAT**
  - '-',
  - AAA-EEE-
- **Left outer (or 1st) CONCAT**
  - SUBSTR(CAST(PHONE AS CHAR(10)),7,4)
  - AAA-EEE-NNNN



Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### There's an Easier Way in V5R3

- **INSERT Scalar Function**
- **Syntax**

```
INSERT(source_string, start, overlay length, insert_string)
```

```
SELECT idnbr, fname, lname, compny, addr1,
       addr2, city, state, zip, contry,
       INSERT(
         INSERT(CAST(phone AS CHAR(10)), 4, 0, '-' ),
         8, 0, '-' ) AS phonea
FROM namemstr
```



Copyright 2008 System i Developer, LLC

## Inserting Hyphens for Readability...

---

### Format of PHONEA is Char(12) AAA-EEE-NNNN

- **Area code is in positions 1 thru 3**
- **1st hyphen is in position 4**
- **Phone exchange is in positions 5 thru 7**
- **2nd hyphen is position 8**
- **Phone number is in positions 9 thru 12**



Copyright 2008 System i Developer, LLC

---

# Basic Subselect



Copyright 2008 System i Developer, LLC

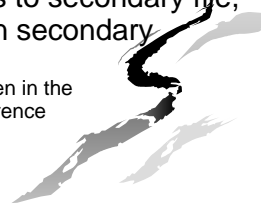
## Basic Subselect

---

### What is a Subselect (aka Subquery)?

- **Capability to nest 1 to 256 (V5R3) SELECT statements inside a SELECT, INSERT, UPDATE, DELETE or CREATE TABLE statement**
- **Provides significantly enhanced function**
- **Can reference two or more tables or views within a single, compound SQL statement without having to do a join of the tables or views involved**
  - Similar to RPG pgm that reads primary file, chains to secondary file, and retrieves or updates rows in primary based on secondary

Note: The term subquery and subselect are often used interchangeably - even in the IBM documentation! Since the difference is subtle (we won't go into the difference here), and to simplify discussion, only the term subselect will be used in this presentation.



Copyright 2008 System i Developer, LLC

## Sample Tables - Subselect Examples

---

- **Employee Table - EMP**

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

- **Department Table - DEP**

DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares



Copyright 2008 System i Developer, LLC

## SELECT with Simple Subselect

---

### Using Subselect to Build a Selection List

- **SELECT** all employees that work in a department that has a name beginning with 'S' (upper case S)
- **DNM** - Department name is only in DEP (Department Master Table) and is not in EMP (Employee Master Table)
- Is there an easy way to do this without doing an inner join between EMP and DEP?

**YES - There Is!**



Copyright 2008 System i Developer, LLC

## SELECT with Simple Subselect...

---

### Using Subselect to Build a Selection List...

- **Yes! - Subselect is the solution - JOIN not required**

```
SELECT nbr, nam, dpt, sal FROM emp
WHERE dpt IN
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
ORDER BY nbr
```

NBR	NAM	DPT	SAL
10	Ed	911	7,000
50	Marcela	911	7,500
60	Frank	990	6,500

Copyright 2008 System i Developer, LLC

## SELECT with Simple Subselect...

---

### Subselect Terminology

- **First or left most SELECT is the primary or outer select**

```
SELECT nbr, nam, dpt, sal FROM emp
WHERE dpt IN
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
ORDER BY nbr
```

- **Second or right most SELECT is the subselect and is also called the inner select**

```
SELECT nbr, nam, dpt, sal FROM emp
WHERE DPT in
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
ORDER BY nbr
```

Copyright 2008 System i Developer, LLC

## Using Subselect with IN Predicate

---

### Using the IN Predicate

- **Compares a column value in the WHERE clause on the left side of the IN predicate with a set or list of values on the right side of the IN predicate**
- **Subselect provides the list of values for the IN predicate**

```
WHERE dpt IN
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
```

- When subselect is used on the right side of the IN predicate, only **single** column is allowed in the column list for the SELECT
  - dpt
- Multiple rows can be returned by subselect
  - Builds selection list



Copyright 2008 System i Developer, LLC

## Using Subselect with IN Predicate...

---

### How Does the IN Predicate Work with Subselect?

- **SQL resolves (evaluates and executes) the subselect first**

```
SELECT nbr, nam, dpt, sal FROM emp
      WHERE dpt IN
            (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
      ORDER BY nbr
```

DEP	
DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares

Copyright 2008 System i Developer, LLC

## Using Subselect with IN Predicate...

---

### How Does the IN Predicate Work with Subselect?...

- **Subselect retrieves all rows that satisfy selection criteria in WHERE clause**
- **Retrieved rows become selection list for IN predicate when SQL executes outer select**

```
SELECT nbr, nam, dpt, sal FROM emp
WHERE dpt IN (911, 990)
ORDER BY nbr
```

NBR	NAM	DPT	SAL
10	Ed	911	7,000
50	Marcela	911	7,500
60	Frank	990	6,500

DEP	
DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares

Copyright 2008 System i Developer, LLC

## Basic Subselect - Quick Recap

---

### Maximum of 256 subselects per SQL statement

- **Also can be referred to as inner selects**
- **Nest within**
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - CREATE TABLE - V5R2
- **Performance**
  - As the number of subselects within an SQL statement is increased, the longer it will take for that SQL statement to execute

Copyright 2008 System i Developer, LLC

---

# Scalar Subselect



Copyright 2008 System i Developer, LLC

## Scalar Subselect

---

### What is a Scalar Subselect?

- **A Scalar Subselect is used in place of an SQL expression or function**

```
SELECT fld1, fld2, (SELECT flda FROM tablea...)  
FROM table1...
```

```
UPDATE table1  
SET fld1 = SELECT flda FROM tablea...
```



Copyright 2008 System i Developer, LLC

## Scalar Subselect...

---

### Example

- **SELECT** all employees that work in a department that has a name beginning with 'S' (upper case S) and include the department name in the result set
- **DNM** - Department name is only in DEP (Department Master Table) and is not in EMP (Employee Master Table)
- Is there an easy way to do this without doing an inner join between EMP and DEP?
- **YES - There Is!**



Copyright 2008 System i Developer, LLC

## Scalar Subselect...

---

### Example...

- **Yes!** - Scalar Subselect is the solution - JOIN not required

```
SELECT nbr, nam, dpt, sal,
       (SELECT dnm FROM dep b
        WHERE a.dpt = b.dpt) AS dept_name
FROM emp a
WHERE dpt IN
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
ORDER BY nbr
```

NBR	NAM	DPT	SAL	DEPT NAME
10	Ed	911	7,000	Sales
50	Marcela	911	7,500	Sales
60	Frank	990	6,500	Spares

DEP	
DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares

Copyright 2008 System i Developer, LLC

## Scalar Subselect...

---

### What's this DEP B and EMP A Stuff?

- **Correlation Names**
  - Alternate name for a table or view referenced in an SQL statement
  - SQL supports long names - up to 128 characters
  - Temporary short name used as an alternate to the long name
- **Correlation name used to qualify column name**
  - TABLE-NAME.COLUMN-NAME
  - CORRELATION-NAME.COLUMN-NAME
- **When the same column name is used in two tables (or views), and there is a reference to that column name for each of these tables in an SQL expression, that column name must be qualified to resolve to the correct table**

Copyright 2008 System i Developer, LLC

## Scalar Subselect...

---

### What's this DEP B and EMP A Stuff...

- **Correlation Name Assigned in FROM Clause**

```
SELECT nbr, nam, dpt, sal,
       (SELECT dnm FROM dep b
        WHERE a.dpt = b.dpt) AS dept_name
FROM emp a
WHERE dpt IN
      (SELECT dpt FROM dep WHERE dnm LIKE 'S%')
ORDER BY nbr
```

NBR	NAM	DPT	SAL	DEPT NAME
10	Ed	911	7,000	Sales
50	Marcela	911	7,500	Sales
60	Frank	990	6,500	Spares

DEP	
DPT	DNM
901	Accounts
977	Manufact
911	Sales
990	Spares

Copyright 2008 System i Developer, LLC

---

# Subselect and INSERT, UPDATE, and DELETE



Copyright 2008 System i Developer, LLC

## Subselect and INSERT

---

### Using Subselect in the INSERT Statement

- **Create a workfile and populate with rows and columns selected from EMP**

```
CREATE TABLE empname
(number DEC (3,0) NOT NULL WITH DEFAULT,
name CHAR (10) NOT NULL WITH DEFAULT,
dept DEC (3,0) NOT NULL WITH DEFAULT)
```

```
INSERT INTO empname (number, name, dept)
SELECT nbr, nam, dpt FROM emp
```

NBR	NAM	DPT
10	Ed	911
20	Heikki	901
30	John	977
40	Mike	977
50	Marcela	911
60	Frank	990

Copyright 2008 System i Developer, LLC

## Subselect and INSERT...

---

### Using Subselect in the INSERT Statement

- Create a workfile and populate with rows and columns selected from EMP and DEP

```
INSERT INTO empname (number, name, dept, dptname)
  SELECT nbr, name, dpt,
         (SELECT dnm FROM dep b
          WHERE a.dpt = b.dpt) AS dptnam
FROM emp a
```

NBR	NAM	DPT	DPTNAM
10	Ed	911	Sales
20	Heikki	901	Accounts
30	John	977	Manufact
40	Mike	977	Manufact
50	Marcela	911	Sales
60	Frank	990	Spares

Copyright 2008 System i Developer, LLC

## Subselect and DELETE

---

### Using Subselect in the DELETE Statement

- Delete rows for employees that have been transferred to another division

```
DELETE FROM emp WHERE nbr IN
  (SELECT nbr FROM transfer)
```

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### TRANSFER

NBR	NAM	DPT
20	Heikki	901
30	John	977
50	Marcela	911

Copyright 2008 System i Developer, LLC

## Subselect and DELETE...

### Using Subselect in the DELETE Statement...

#### EMP - Before Delete

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### TRANSFER

NBR	NAM	DPT
20	Heikki	901
30	John	977
50	Marcela	911

#### EMP - After Delete

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
40	Mike	4	M	977	6,500
60	Frank	2	M	990	6,500

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE

### Simple Scalar Subselect in the UPDATE Statement

- Update each employee's salary to the amount listed in the new salary table

```
UPDATE emp aa
  SET sal = (SELECT nsal FROM newsal1 bb
            WHERE aa.nbr = bb.nbr)
```

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### NEWSAL1

NBR	NSAL
10	7,890
20	6,890
30	4,090
40	7,390
50	8,390
60	7,390

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE...

### Simple Scalar Subselect in the UPDATE Statement...

#### EMP - Before Update

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### NEWSAL1

NBR	NSAL
10	7,890
20	6,890
30	4,090
40	7,390
50	8,390
60	7,390

#### EMP - After Update

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,890
20	Heikki	2	M	901	6,890
30	John	5	M	977	4,090
40	Mike	4	M	977	7,390
50	Marcela	3	F	911	8,390
60	Frank	2	M	990	7,390

Each row in EMP must have a corresponding row in NEWSAL1

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE...

### 'Null Values Not Allowed' Error

- If each row in EMP does not have a corresponding row in NEWSAL1

```
UPDATE emp aa
  SET sal = (SELECT nsal FROM newsal1 bb
            WHERE aa.nbr = bb.nbr)
```

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### NEWSAL2

NBR	NSAL
10	7,520
20	6,750
30	3,720
60	7,020

Error Msg: *Null values not allowed in column or variable SAL - WHY?*

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE...

### 'Null Values Not Allowed' Error - Why?

#### EMP - Before Update

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### NEWSAL2

NBR	NSAL
10	7,520
20	6,750
30	3,720
60	7,020

#### EMP - After Update (assumes no commitment control)

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,520
20	Heikki	2	M	901	6,750
30	John	5	M	977	3,720
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

Employees 10, 20, and 30 are updated based on NEWSAL2. When employee 40 read, there is no corresponding NSAL value in NEWSAL2. Therefore SQL attempts to change SAL to a null value which is not allowed. Update process ends with error at that point and employees 50 and 60 are not processed.

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE...

### Avoiding the 'Null Values Not Allowed' Error

- Add WHERE clause with second subselect to UPDATE

```
UPDATE emp aa
  SET sal = (SELECT nsal FROM newsal1 bb
            WHERE aa.nbr = bb.nbr)
  WHERE aa.nbr IN (SELECT nbr FROM newsal2)
```

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### NEWSAL2

NBR	NSAL
10	7,520
20	6,750
30	3,720
60	7,020

Copyright 2008 System i Developer, LLC

## Subselect and UPDATE...

---

### Avoiding the 'Null Values Not Allowed' Error...

**EMP - Before Update**

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

**NEWSAL2**

NBR	NSAL
10	7,520
20	6,750
30	3,720
60	7,020

**EMP - After Update**

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,520
20	Heikki	2	M	901	6,750
30	John	5	M	977	3,720
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	7,020

Copyright 2008 System i Developer, LLC

---

## Subselect and CREATE TABLE



Copyright 2008 System i Developer, LLC

## Subselect and CREATE TABLE

---

### V5R2 Scalar Subselect

```
CREATE TABLE workfile AS
  (SELECT a.dpt, SUM(sal) AS totals,
   (SELECT dnm FROM dep b
    WHERE a.dpt = b.dpt) AS dnm
   FROM emp a
   GROUP BY a.dpt, 3)
WITH DATA
```

Dept	Total Salary	Dept Name
901	6,000	Accounts
911	14,500	Sales
977	9,700	Manufact
990	6,500	Spares



Copyright 2008 System i Developer, LLC

## Subselect and CREATE TABLE...

---

### CREATE TABLE Syntax Using V5R2 Scalar Subselect

- Each derived column defined in the scalar subselect within the CREATE TABLE statement *must* be given a name using the AS operator
- CREATE TABLE statement *must* be ended with one of the following two clauses
  - WITH DATA
    - Table is populated with rows and columns that match table definition and selection criterion
  - WITH NO DATA
    - Table is created as empty table with no rows



Copyright 2008 System i Developer, LLC

## Subselect and CREATE TABLE...

---

### CREATE TABLE and V5R2 Scalar Subselect

- **Provides SQL with missing native functions**
  - CPYF - with or without data
  - CRTDUPOBJ - with or without data
  - Field Reference File support

### CREATE TABLE table\_name AS (SELECT...)...

- *Subset* of fields in reference table used in created table

### CREATE TABLE table\_name LIKE reference\_table\_name...

- *All* fields in reference table used in created table

- **One Stop Shopping - create table and populate with data**
  - Not available with native interface!

Copyright 2008 System i Developer, LLC

---

## Subselect and Derived Table



Copyright 2008 System i Developer, LLC

## Subselect and Derived Table

---

### What is a Derived a Table?

- A derived table is created in the FROM clause of a SELECT statement as part of the statement execution process
- Subselect can be used to derive a table in the FROM clause of a SELECT statement
- Example - SELECT the first row of a group of rows where SELECT DISTINCT or GROUP BY cannot be used to satisfy the selection criteria



Copyright 2008 System i Developer, LLC

## Subselect and Derived Table...

---

### SELECT the First Row of a Group...

- What is the first or lowest employee number in each department?

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

- SELECT DISTINCT or GROUP BY cannot be used to satisfy this selection criteria



Copyright 2008 System i Developer, LLC

## Subselect and Derived Table...

---

### SELECT the First Row of a Group...

- What is the first employee number in each department?

```
SELECT * FROM emp WHERE nbr IN
  (SELECT number FROM
    (SELECT dpt, MIN(nbr) AS number
     FROM emp
     GROUP BY dpt)
   AS first-row-table)
ORDER BY dpt
```



Copyright 2008 System i Developer, LLC

## Subselect and Derived Table...

---

### SELECT the First Row of a Group...

- What is the first employee number in each department?

#### EMP

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
40	Mike	4	M	977	6,500
50	Marcela	3	F	911	7,500
60	Frank	2	M	990	6,500

#### Result Set for SELECT Statement

NBR	NAM	CLS	SEX	DPT	SAL
10	Ed	5	M	911	7,000
20	Heikki	2	M	901	6,000
30	John	5	M	977	3,200
60	Frank	2	M	990	6,500



Copyright 2008 System i Developer, LLC

---

# Summarizing Data with SELECT



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT

---

The **SELECT** statement is an *Either Or* Proposition

- ***Either*** return detail rows in the result set
- ***Or*** return summarized data in the result set
- **A single SQL statement has no capability to do**
  - Detail
  - Detail
  - Level break
  - Detail
  - Level break
- **Query Manager CAN do above**
  - SQL based query product



Copyright 2008 System i Developer, LLC

# Summarizing Data with SELECT...

---

## SELECT Statement Clauses

**SELECT . . . . . (columns, \*, or expressions)**  
**FROM . . . . .(tables or views)**  
**WHERE . . . . .(row selection criteria)**  
**GROUP BY. . .(row summarization criteria)**  
**HAVING. . . . . (GROUP BY selection criteria)**  
**ORDER BY. . .(column ordering criteria)**



Copyright 2008 System i Developer, LLC

# Summarizing Data with SELECT

---

## GROUP BY Clause

- Defines grouping or summary criteria for SELECT
- Format

```
SELECT flda, fldc, FUNCTION(fldg), FUNCTION(fldh)
FROM table-name
WHERE selection-criteria
GROUP BY fldc, flda
ORDER BY fldc, flda
```

- If a column referenced in the column list of the SELECT statement is not operated on by a function, that column must be referenced as part of the grouping criteria in the GROUP BY clause

Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### GROUP BY Clause...

- For each department list the total salary and total number of employees

```
SELECT dpt, SUM(sal), COUNT ( * )
FROM emp
GROUP BY dpt
ORDER BY dpt
```

DPT	SUM(SAL)	COUNT(*)
901	6,000	1
911	14,500	2
977	9,700	2
990	6,500	1



Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### HAVING Clause...

- Defines **GROUP BY** selection criteria
- **Format**

```
SELECT flda, fldc, FUNCTION(fldg), FUNCTION(fldh)
FROM table-name
WHERE selection-criteria
GROUP BY fldc, flda
HAVING group-by-selection-criteria
ORDER BY fldc, flda
```

Copyright 2008 System i Developer, LLC

## Summarizing Data with SELECT...

---

### HAVING Clause...

- For those departments that have more than one employee, list the total salary and total number of employees

```
SELECT dpt, SUM(sal) AS saltot,  
        COUNT(*) AS emptot  
FROM emp  
GROUP BY dpt  
HAVING COUNT(*) > 1  
ORDER BY dpt
```

DPT	SALTOT	EMPTOT
911	14,500	2
977	9,700	2



Copyright 2008 System i Developer, LLC

---

## Identifying Potential Duplicate Rows



Copyright 2008 System i Developer, LLC

## Identifying Potential Duplicate Rows

---

### Inner Join for Identifying Potential Duplicate Rows

- Duplicate rows in a table can be a significant waste of time, money, and other resources - especially if the table contains contact or name and address information
- Inner join used because a resulting row is returned when a row in the inner (right) table matches a row in the outer (left) table



Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study

---

### Mailing List Table

- **Table Name:** MAILLIST
- **Total Rows:** 25,000+
- **Duplicate Rows:** Many - visual verification
- **Row Format:**

Column Name	Length	Data Type	Description
IDNBR	5.0	Dec	Unique ID Number
FNAME	15	Char	First Name
LNAME	30	Char	Last Name
COMPNY	50	Char	Company Name
ADDRL1	50	Char	Address Line 1
ADDRL2	50	Char	Address Line 2
CITY	25	Char	City
STATE	2	Char	State
ZIP	10	Char	Zip Code



Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study...

---

### Creating the Inner Join - Version 1

```

SELECT a.idnbr, a.fname, a.lname, a.compny, a.addr1,
       a.addr2, a.city, a.state, a.zip
FROM maillist a INNER JOIN maillist b
ON a.fname = b.fname
   AND a.lname = b.lname
   AND a.compny = b.compny
   AND a.addr1 = b.addr1
   AND a.addr2 = b.addr2
   AND a.city = b.city
   AND a.state = b.state
   AND a.zip = b.zip
   AND a.idnbr <> b.idnbr
ORDER BY a.lname, a.compny a.fname

```

- **No rows returned - Why?**

- Inconsistencies with address and other data

Copyright 2008 System i Developer, LLC



## Duplicate Row Case Study...

---

### Creating the Inner Join - Version 2

```

SELECT a.idnbr, a.fname, a.lname, a.compny, a.city
FROM maillist a INNER JOIN maillist b
ON a.state = b.state
   AND SUBSTR(a.fname,1,3) = SUBSTR(b.fname,1,3)
   AND SUBSTR(a.lname,1,3) = SUBSTR(b.lname,1,3)
   AND SUBSTR(a.compny,1,3) = SUBSTR(b.compny,1,3)
   AND SUBSTR(a.city,1,3) = SUBSTR(b.city,1,3)
   AND a.idnbr <> b.idnbr
ORDER BY a.lname, a.compny, a.fname

```

- **Four rows returned**

Copyright 2008 System i Developer, LLC



## Duplicate Row Case Study...

---

### Four Rows Returned

- Which rows are duplicates, which rows are NOT?

IDNBR	FNAME	LNAME	COMPNY	CITY
30368	Dan	Tuffy Jr	Tuffy Consulting	Redwing
30367	Daniel	Tuffy Sr	Tuffy Consulting	Redwing
03841	Jeff	Goeble	Kansas Farm Bureau	Kansas City
34057	Jeff	Goeble	Kansas Farm Bureau Services Inc	Kansas City

- Understand the term *Potential Duplicate Records*?
- Which of the two duplicate rows should be deleted?
- Why only four rows?
  - Variations in representation of Company and City



Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study...

---

### Refining the Inner Join

- Exclude **COMPNY** or **CITY** from join criteria
- Exclude blank **FNAME** and blank **LNAME**
- Shorten columns for readability
  - FNAME
  - LNAME
  - COMPNY
  - CITY



Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study...

---

### Exclude CITY from Join Criteria

```

SELECT a.idnbr,
       SUBSTR(a.fname,1,10) AS fname,
       SUBSTR(a.lname,1,10) AS lname,
       SUBSTR(a.compny,1,30) AS compny
  FROM maillist a INNER JOIN maillist b
    ON a.state = b.state
     AND SUBSTR(a.fname,1,3) = SUBSTR(b.fname,1,3)
     AND a.fname > ' '
     AND SUBSTR(a.lname,1,3) = SUBSTR(b.lname,1,3)
     AND a.lname > ' '
     AND SUBSTR(a.compny,1,3) = SUBSTR(b.compny,1,3)
     AND a.idnbr <> b.idnbr
 ORDER BY lname, compny, fname

```

- **366 potential duplicate rows returned**

Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study...

---

### Exclude COMPNY from Join Criteria

```

SELECT a.idnbr,
       SUBSTR(a.fname,1,10) AS fname,
       SUBSTR(a.lname,1,10) AS lname,
       SUBSTR(a.city,1,15) AS city
  FROM maillist a INNER JOIN maillist b
    ON a.state = b.state
     AND SUBSTR(a.fname,1,3) = SUBSTR(b.fname,1,3)
     AND a.fname > ' '
     AND SUBSTR(a.lname,1,3) = SUBSTR(b.lname,1,3)
     AND a.lname > ' '
     AND SUBSTR(a.city,1,3) = SUBSTR(b.city,1,3)
     AND a.idnbr <> b.idnbr
 ORDER BY lname, compny, fname

```

- **100 potential duplicate rows returned**

Copyright 2008 System i Developer, LLC

## Duplicate Row Case Study...

---

### Considerations

- **When result sets for previous two examples (exclude CITY and exclude COMPANY) were compared...**
  - Only 4 rows were in both result sets
  - Therefore multiple analysis of data with different criteria is a good idea
- **Two different tables can be compared/joined for analysis**
- **Technique will work for other than name & address tables**

Copyright 2008 System i Developer, LLC

---

## Search by Sound with SOUNDEX



Copyright 2008 System i Developer, LLC

## What is SOUNDEX?

---

### SOUNDS LIKE Function Introduced in V4R5

- **SOUNDEX is useful for finding character strings for which the sound is known but the precise spelling is not.**
  - IBM publication: *DB2 UDB for iSeries SQL Reference*
- ***Makes assumptions about the way that letters and combinations of letters sound that can help to search out words with similar sounds***
- **Exact spelling or case not required**
- **More flexibility than with LIKE**



Copyright 2008 System i Developer, LLC

## What is SOUNDEX...

---

### SOUNDEX Syntax

#### SOUNDEX(string-expression)

```
SELECT lastname FROM address_table
   WHERE SOUNDEX(lastname) =
SOUNDEX('markasonnee')
   ORDER BY lastname
```

- **Can be used anywhere in an SQL statement where a function is allowed**



Copyright 2008 System i Developer, LLC

## Working with SOUNDEX

---

### Searching for Marchesani

- **Phonetic spellings**

- mark-a-son-nee
- mark-us-annie
- march-ez-annie

```
SELECT lastname FROM address_table
WHERE SOUNDEX(lastname) =
SOUNDEX('markasonnee')
ORDER BY lastname
```

There were 25,000 rows in address\_table for these tests



Copyright 2008 System i Developer, LLC

## Working with SOUNDEX...

---

### Searching for Marchesani

- **Phonetic spellings**

- mark-a-son-nee mark-us-annie march-ez-annie

```
SELECT lastname FROM address_table
WHERE SOUNDEX(lastname) = SOUNDEX('markasonnee')
ORDER BY lastname
```

- **23 Rows returned in result set**

Marages	Marcoccia	Marquis
Maraschky	Marcoux	Marsico
<b>Marchesani</b>	Marcus	Marzouk
<b>Marchesani</b>	Marcus	Mergy
<b>Marchesani</b>	Markowich	Mierzejewski
<b>Marchesani</b>	Marquez	Mrazek
Marchese	Marquez	Mroczek
Marcocci	Marquis	



Copyright 2008 System i Developer, LLC

## Working with SOUNDEX...

---

### Searching for Marchesani

- **Other phonetic spellings - Take #1**
  - mark-us
    - Same 23 rows returned
    - Is this a joke - I'm starting to get suspicious
- **Other phonetic spellings - Take #2**
  - mark
  - marks
    - 115 different rows returned, none of which are Marchesani



Copyright 2008 System i Developer, LLC

## Working with SOUNDEX...

---

### Testing More Complex Names

- **Other names with phonetic spellings**
  - Zborovskiy: za-bo-rov-ski, za-brov-ski
  - Zuehlsdorf: zuls-dorf
  - Barszczak: bar-sak
  - Abaunza: ab-an-sa
- **Result sets returned**
  - Zborovskiy: single row exact match
  - Zuehlsdorf: single row exact match
  - Barszczak: 26 rows
  - Abaunza: 3 rows
- **How many rows will be returned?**

```
SELECT COUNT(*) FROM address_table
WHERE SOUNDEX(lastname) =
SOUNDEX('markus')
```

Copyright 2008 System i Developer, LLC

## How Does SOUNDEX Work?

---

### SOUNDEX Uses an Algorithm

- **Generates a four character code in format ANNN**
  - A = first (alpha) character in string expression
  - NNN = 3 digit number generated by the algorithm
- **To view 4 character SOUNDEX code**

```
SELECT SOUNDEX(character-string), COUNT(*)
FROM any-table-name
```

- **SOUNDEX code = M622**
  - markasonnee, markusannie, marchezannie, marcus
- **SOUNDEX code = M620**
  - mark, marks



Copyright 2008 System i Developer, LLC

## How Does SOUNDEX Work...

---

### SOUNDEX Considerations

- **First letter of phonetic spelling must be correct**
  - Will differentiate between **S & Z**, and **F & PH**
- **Sensitivity to the number of syllables**
  - If target string is multiple syllables phonetic spelling should also be multiple syllables
  - Best if target and phonetic spelling have equal number of syllables
  - Single syllable phonetic spelling can return multiple syllable result



Copyright 2008 System i Developer, LLC

## How Does SOUNDEX Work...

---

### SOUNDEX Performance Comparison with LIKE

- **Environment**

- 25,000 rows in address\_table
- Lightly loaded Model 820 with 3 gigs of memory
- Lots of disk arms
- Interactive SQL (STRSQL command)
- Response time measure with digital stop watch
- Ran several iterations of each test

- **SOUNDEX**

- marcus to find Marchesani - 2.2 seconds

- **LIKE**

- Marc% to find Marchesani - less than 0.5 seconds



Copyright 2008 System i Developer, LLC

## Summary

---

### I Didn't Know You Could Do that with SQL!

- **Working with Edit Characters**

- **Subselect**

- Basic Subselect
- Scalar Subselect
- Subselect and INSERT, UPDATE, & DELETE
- Subselect and CREATE TABLE
- Subselect and Derived Table

- **Summarizing Data with SELECT**

- **Identifying Potential Duplicate Rows**

- **Searching by Sound with SOUNDEX**



Copyright 2008 System i Developer, LLC

## Summary...

---

### I Didn't Know You Could Do that with SQL...

- From this presentation you should have a better understanding of SQL as a programming language.
- The more you work and play with it, the more you realize the power of SQL and what it can do for you as an application development or database manipulation tool.
- With a little thought and creativity you will find you can use SQL for things that at first glance you did not think possible.



Copyright 2008 System i Developer, LLC

## V5 SQL Information Sources

---

- **iSeries Information Center Publications - Web or CD**
  - SQL Reference
  - SQL Programming
  - Embedded SQL Programming
  - Query Manager Use
  - SQL Messages and Codes
- **To access Info Center on the Web**
  - <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp>
  - <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>
    - In left scroll bar
      - Click on iSeries Information Center ...
      - Click on Database
      - Click on Printable PDFs
      - Use right scroll bar to scroll down to above SQL publication
- **DB2 for iSeries on the Web**
  - <http://www.ibm.com/servers/eserver/iseries/db2/>

Copyright 2008 System i Developer, LLC

---

# Part 3

Copyright 2008 System i Developer, LLC

## Overview

---

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5

Copyright 2008 System i Developer, LLC

## Overview Part 3

---

### Embedding SQL in RPG (and Cobol) Programs

- Executing SQL on the System i, iSeries, & AS/400
- V5 SQL Support
- Basic SQL Statements
- SQL Precompiler
- RPG and Cobol Interface
- Cursor Operations
- Error Detection and Handling
- Dynamic SQL
- Editing and Compiling Embedded SQL
- Performance Tips
- Summary
- V5 SQL Information Sources



Copyright 2008 System i Developer, LLC

## What is SQL?

---

### SQL - Structured Query Language

- Data language for manipulation of a Relational database
- English keyword-oriented
- Excellent tool for application development environment
  - Query
  - Data definition
  - Data manipulation
  - Data control
- Powerful language for Set-At-A-Time processing

**SQL is NOT an end user query product!**

Copyright 2008 System i Developer, LLC

## **What is SQL on the System i & iSeries?**

---

### **An Alternative Database Interface Language**

- **NOT a database management system**
- **High level, simple statement formats**
- **A language used for:**
  - Data Definition (DDL)
  - Data Manipulation (DML)
- **Completely interchangeable data methods**
  - SQL tables may be accessed with native language
  - DDS created files can be access with SQL

### **SQL skill is transferable across platforms**

Copyright 2008 System i Developer, LLC

## **Executing SQL on System i & iSeries**

---

### **Different Ways to Execute SQL Statements**

- **Interactive SQL**
- **Embedded or compiled in application programs**
- **Query Manager**
- **SQL Statement Processor**
- **Operations Navigator**
- **Dynamic SQL**
- **SQL Procedure Language**
- **Extended Dynamic SQL**
- **ODBC, JDBC**
- **JSQL or SQLJ**
- **X/Open SQL Call Level Interface**
- **Query Management (different from Query Manager)**

Copyright 2008 System i Developer, LLC

# V5 SQL Support

---

## Part 2: DB2 UDB Query Manager & SQL Development Kit

- **Program number 5722-ST1**
- **Query Manager**
- **Interactive SQL**
- **SQL precompilers**
  - Required for embedding SQL in HLL programs
  - V5R4 RPG IV precompiler for SQL supports Free-form RPG
  - More later
- **SQL REXX interface**



Copyright 2008 System i Developer, LLC

# V5 SQL Support...

---

## Part 1: DB2 UDB Database Manager

- **Included with OS/400 (operating system)**
  - V5: 5722-SS1
- **SQL parser and run time support**
  - SQL license (5722-ST1) not required to run SQL applications
  - Support for compiled programs using embedded SQL
- **SQL APIs**
  - QSQPRCED - Provides extended dynamic SQL capability
  - QSQCHKS - Provides syntax checking for SQL statements
- **X/Open SQL Call Level Interface**
- **V5R1 - SQL Statement Processor**
  - RUNSQLSTM Command

Copyright 2008 System i Developer, LLC

# Embedding SQL In a HLL Program

---

## High Level Languages Supported

- RPG
- COBOL
- C
- Java
- AS/400 PL/I
- FORTRAN/400



Copyright 2008 System i Developer, LLC

# Embedding SQL In a HLL Program...

---

## Basic SQL Statements

- **Data Manipulation statements (DML)**
  - SELECT - retrieves data; one row or multiple
  - UPDATE - updates one row or multiple
  - DELETE - deletes one row or multiple
  - INSERT - adds one row or multiple
- **Data Definition statements (DDL)**
  - CREATE and DROP
    - Collection (library)
    - Table (physical file)
    - View (logical file)
    - Index (logical file)



Copyright 2008 System i Developer, LLC

## **Embedding SQL In a HLL Program...**

### **SQL Statements for Program Control**

- **Provides Functions for Complex Processing**
  - DECLARE CURSOR - builds temp result table
  - OPEN and CLOSE - open/close result table
  - FETCH - row retrieval
  - COMMIT and ROLLBACK - journaling functions
  - GRANT and REVOKE - security functions

Copyright 2008 System i Developer, LLC

## **Embedding SQL In a HLL Program...**

### **Why embed SQL in programs?**

- **Perform dynamic selection functions**
  - ala OPNQRYF, except more flexible
- **Perform set-at-a-time functions under program control**
- **Even to replace HLL I/O operations**
  - e.g., READ, WRITE, UPDATE, CHAIN

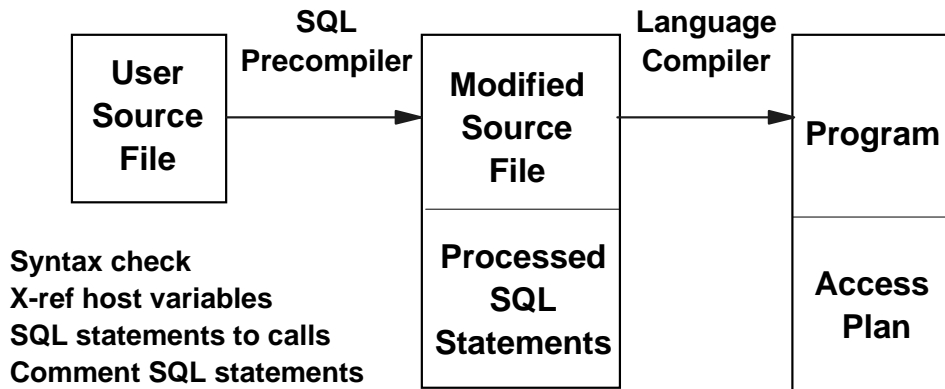
### **What can be embedded?**

- **Basic SQL DDL and DML statements**
  - e.g., SELECT, UPDATE, INSERT, CREATE TABLE, etc.
- **Program control statements**
  - e.g., DECLARE CURSOR, OPEN, CLOSE, FETCH, COMMIT, ROLLBACK

Copyright 2008 System i Developer, LLC

# SQL Precompiler

## HLLs with Embedded SQL Use a Precompiler



Copyright 2008 System i Developer, LLC

# RPG Interface - Fixed Format

## SELECT A Single Row From A Table - RPG

```

I      DS
I      1  50 EMPNBR
I      6  30  NAM
I      31 32  DEPT
C*
C/EXEC SQL
C+      SELECT  nbr, nam, dpt
C+      INTO   :empnbr, :nam, :dept
C+      FROM   emp
C+      WHERE  nbr = :empnbr
C/END-EXEC
  
```

- **RPG and other HLLs more efficient for doing single record random retrievals**
- **SQL more efficient for multiple record sets or groups**

Copyright 2008 System i Developer, LLC

## RPG Interface - Fixed Format ...

---

### Rules for Embedding SQL in RPG

- All SQL statements must be coded on a C spec
- SQL statements begin with /EXEC SQL in positions 7-15
  - with the slash in position 7
- and end with /END-EXEC in positions 7-15
- You can enter SQL statements on same line as /EXEC SQL
  - However, /END-EXEC must be on a separate line
- Between beginning and ending delimiters, all SQL statements must have + in position 7
- Use Upper Case for C, /EXEC, and /END-EXEC in RPG IV
- SQL statements **CANNOT**
  - go past position 80
  - be included via a /COPY statement
- RPG IV precompiler for SQL now supports embedded SQL in Free-form RPG

Copyright 2008 System i Developer, LLC

## RPG Interface - Free Format

---

### V5R4 Supports Free Fromat Embedded SQL

```

d getOrderCount pi      10i 0
d inCustNumber  9b 0 const
d outSqlState   5a
d outSqlMsg     256a

d ordercount   s      10i 0

/free
outSqlMsg = *blanks;

exec sql

SELECT count(*) into :orderCount FROM orders
WHERE cust_no = :inCustNumber;

outSqlState = sqlState;
if %subst(sqlState:1:2) <> '00';
  exec sql GET DIAGNOSTICS CONDITION 1
  :outSqlMsg = MESSAGE_TEXT;
  orderCount = 0;
endif;
return orderCount;
/end-free

```

Copyright 2008 System i Developer, LLC

## Cobol Interface - Source


---

### SELECT A Single Row From A Table - Cobol

```

WORKING-STORAGE SECTION.
  77  EMPNBR          PIC  S9(5)  COMP-3.
  77  DEPT            PIC  S9(3)  COMP-3.
  77  NAM             PIC  X(25).
  .
  .
PROCEDURE DIVISION.
  EXEC SQL
    SELECT  nbr, nam, dpt
    INTO   :empnbr, :nam, :dept
    FROM   emp
    WHERE  nbr = :empnbr
  END-EXEC.

```



Copyright 2008 System i Developer, LLC

## RPG and Cobol Interface

---


### Using Structures in SQL

- **Host structures are groups of variables**
  - Data structures in RPG
  - Group items in COBOL
- **Structures can be used in SQL statements**
  - Replaces list of variables

```

I empds DS
I          1 50 EMPNBR
I          6 30 NAM
I          31 32 DEPT
C*
C/EXEC SQL
C+  SELECT  nbr, nam, dpt
C+  INTO   :empds
C+  FROM   emp
C+  WHERE  nbr = :empnbr
C/END-EXEC

```



Copyright 2008 System i Developer, LLC

## RPG and Cobol Interface...

---

### SELECT... INTO Expects Only a Single Row/Record

- Retrieve column/field values into program variables
- One-to-one correspondence between SELECT list and INTO list

```
C/EXEC SQL
C+  SELECT nbr, nam, dpt
C+    INTO :empnbr, :nam, :dept
C+    FROM emp
C+    WHERE nbr = :empnbr
C/END-EXEC
```



Copyright 2008 System i Developer, LLC

## RPG and Cobol Interface...

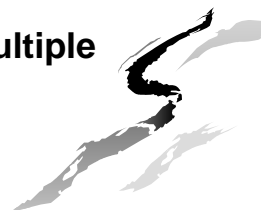
---

### Multiple Rows/Records Can Be Processed

- Single SQL statement can process multiple rows/records
  - Processed rows/records **NOT** returned to program

```
C/EXEC SQL
C+  UPDATE emp
C+    SET dpt = :newdept
C+    WHERE dpt = :olddept
C/END-EXEC
```

- SQL cursor operations used to return multiple rows/records to program



Copyright 2008 System i Developer, LLC

# Cursor Operations

---

## SELECT & Process Multiple Rows/Records from a Table

1. Declare SQL cursor
2. Open cursor
3. Fetch next record
4. Process record (UPDATE/INSERT/etc)
5. If last record: go to Step 6,  
    ▶ else: go to Step 3
6. Close cursor



Copyright 2008 System i Developer, LLC

## Cursor Operations...

---

### Defining and Using an SQL Cursor

#### + PROBLEM:

Update SALARY for all records in a specified department

```

C*
C/EXEC SQL
C+
C+   DECLARE empcsr CURSOR FOR
C+   SELECT nbr, nam, sal
C+   FROM emp
C+   WHERE dpt = :dept
C+   FOR UPDATE OF sal
C+
C/END-EXEC
C*
C   EXFMT PROMPT
C*
```

Example continued on next page

Copyright 2008 System i Developer, LLC

## Cursor Operations...

---

### SQL Cursor Example - Continued

```

C*
C/EXEC SQL
C+
C+   OPEN   empcsr
C+
C/END-EXEC
C*
C   SQLCOD   DOWNE   0
C*
C/EXEC SQL
C+
C+   FETCH  NEXT  FROM  empcsr
C+   INTO   :number, :name, :salary
C+
C/END-EXEC
C*

```

Example continued on next page

Copyright 2008 System i Developer, LLC

## Cursor Operations...

---

### SQL Cursor Example - Continued

```

C*
C   SQLCOD   IFEQ   0
C*
C/EXEC SQL
C+
C+   UPDATE  emp
C+       SET  sal = sal + :raise
C+       WHERE CURRENT OF empcsr
C+
C/END-EXEC
C   END
C   END
C*
C/EXEC SQL
C+
C+   CLOSE  empcsr
C+
C/END-EXEC

```

Copyright 2008 System i Developer, LLC

# DECLARE CURSOR...

---

## DECLARE CURSOR Statement

- **Similar in function to HLL file declarations**
  - F-specs or FD's
  - No processing actually takes place - just definition
- **Host variables may be included in the statement**
- **Created using an embedded SELECT command**
  - most SELECT clauses may be used - ORDER BY, GROUP BY, etc
- **Must be declared before being referenced**

Copyright 2008 System i Developer, LLC

# DECLARE CURSOR...

---

## DECLARE CURSOR Statement

- **By default, all columns may be updated or deleted**
  - FOR UPDATE OF - lists the columns that are to be updated
    - columns listed in an ORDER BY clause may not be listed in FOR UPDATE OF clause also
  - FOR READ ONLY - specifies no updating/deleting allowed
- **Considerations:**
  - FOR READ ONLY - may improve performance; better documentation
  - FOR UPDATE OF - security; may improve performance

Copyright 2008 System i Developer, LLC

# DECLARE CURSOR...

---

## DECLARE CURSOR Statement

- **With Hold clause useful with Commitment Control**
  - By default, cursors are closed when Commit/Rollback commands execute
  - With Hold - keeps cursor open
  - With Hold also an optional clause on the Commit/Rollback commands

Copyright 2008 System i Developer, LLC

# DECLARE CURSOR...

---

## DECLARE CURSOR statement prompting

```

Columns . . . :   1  71           Edit           SKIP/QRPGSRC
SEU==> _____ AASQLTST
FMT **   ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+.
          ***** Beginning of data *****
0011.00      C/EXEC SQL
0012.00 ==>> C+                               <<=== Prompt from
0013.00 ==>> C+   DECLARE empcsr CURSOR FOR   <<=== any of
0014.00 ==>> C+                               <<=== these lines
0015.00      C/END-EXEC
          ***** End of data *****

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor
F16=Repeat find   F17=Repeat change   F24=More keys

```

**Any SQL statement can be prompted from within a source program**

Copyright 2008 System i Developer, LLC

## DECLARE CURSOR

### DECLARE CURSOR statement prompting - continued

```

Specify DECLARE CURSOR Statement

Type choices, press Enter.

Cursor name . . . . . empcsr_____

Cursor type . . . . . 1          1=Not scrollable
                                   2=SCROLL
                                   3=DYNAMIC SCROLL

Hold . . . . . N          Y=Yes, N=No

Cursor definition . . . . . 1      1=SELECT statement
                                   2=Statement name

F3=Exit   F5=Refresh   F12=Cancel   F21=Display statement

```

Copyright 2008 System i Developer, LLC

## DECLARE CURSOR...

### DECLARE CURSOR statement prompting - continued

```

Specify SELECT Statement

Type information for SELECT statement. Press F4 for a list.

FROM files . . . . . emp_____
SELECT fields . . . . . nbr, nam, sal_____
WHERE conditions . . . . . dpt=_:dept_____
GROUP BY fields . . . . . _____
HAVING conditions . . . . . _____
ORDER BY fields . . . . . _____
FOR UPDATE OF fields . . . . . sal_____

Bottom

Type choices, press Enter.

Number of records to optimize . . . . . _____
DISTINCT records in result file . . . . . N   Y=Yes, N=No
FOR FETCH ONLY . . . . . N   Y=Yes, N=No
UNION with another SELECT . . . . . N   Y=Yes, N=No

F3=Exit   F4=Prompt   F5=Refresh   F6=Insert line   F9=Subquery
F10=Copy  F12=Cancel  F14=Delete  F15=Split line  F24=More keys

```

Copyright 2008 System i Developer, LLC

## DECLARE CURSOR...

### DECLARE CURSOR statement prompting - continued

```

Columns . . . :   1  71                Edit                SKIP/QRPGSRC
SEU==> _____ AASQLTST
FMT **   ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+.
          ***** Beginning of data *****
0011.00      C/EXEC SQL
0012.00      C+
0013.00      C+   DECLARE empcsr CURSOR FOR
0013.01      C+           SELECT nbr, nam, sal
0013.02      C+           FROM emp
0013.03      C+           WHERE dpt = :dept
0013.04      C+           FOR UPDATE OF sal
0014.00      C+
0015.00      C/END-EXEC
          ***** End of data *****

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor
F16=Repeat find   F17=Repeat change   F24=More keys

```

Copyright 2008 System i Developer, LLC

## OPEN Statement

### Executes SELECT Statement For a Declared Cursor

- Builds the access path if necessary
- Successful Open places the file cursor before the first row of the result table
- Cursor must be closed before it can be opened

```

C*
C/EXEC SQL
C+
C+   OPEN   empcsr
C+
C/END-EXEC
C*

```

Copyright 2008 System i Developer, LLC

# FETCH Statement

---

## Two Functions

- Position the cursor for the next operation

```
C*
C/EXEC SQL
C+
C+   FETCH NEXT FROM empcsr
C+
C/END-EXEC
```

- Bring rows/records into the program

```
C*
C/EXEC SQL
C+
C+   FETCH NEXT FROM empcsr
C+   INTO :number, :name, :salary
C+
C/END-EXEC
```

Copyright 2008 System i Developer, LLC

# FETCH Statement...

---

## Alternatives to NEXT Processing

- Cursor must be defined as a Scrollable

```
C/EXEC SQL
C+
C+   DECLARE empcsr SCROLL CURSOR FOR
C+   SELECT nbr, nam, sal
C+   FROM emp
C+   ORDER BY empid
C+
C/END-EXEC
C*
C/EXEC SQL
C+
C+   FETCH PRIOR FROM empcsr
C+   INTO :number, :name, :salary
C+
C/END-EXEC
```

Copyright 2008 System i Developer, LLC

# FETCH Statement...

## Alternatives to NEXT Processing

Keyword	Positions Cursor
Next	On the next row after the current row
Prior	On the row before the current row
First	On the first row
Last	On the last row
Before	Before the first row - must not use INTO
After	After the last row - must not use INTO
Current	On the current row (no change in position)
Relative <i>n</i>	<i>n</i> < -1 Positions to <i>n</i> th row before current <i>n</i> = -1 Same as Prior keyword <i>n</i> = 0 Same as Current keyword <i>n</i> = 1 Same as Next keyword <i>n</i> > 1 Positions to <i>n</i> th row after current

Copyright 2008 System i Developer, LLC

# FETCH Statement...

## FETCH statement prompting

```

Columns . . . :   1  71                Edit                SKIP/QRPGSRC
SEU==>                                     AASQLTST
FMT **   ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+.
0023.00      C/EXEC SQL
0024.00 ==>> C+                               <<=== Prompt from
0025.00 ==>> C+      FETCH                     <<===  any of
0026.00 ==>> C+                               <<===  these lines
0027.00      C/END-EXEC
          ***** End of data *****

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor
F16=Repeat find   F17=Repeat change   F24=More keys

```

Copyright 2008 System i Developer, LLC

**FETCH Statement...****FETCH statement prompting - continued**

Specify FETCH Statement			
Type choices, press Enter.			
Cursor name . . . . .	empcsr	_____	
Positioning . . . . .	1	1=NEXT, 2=PRIOR, 3=FIRST 4=LAST, 5=BEFORE, 6=AFTER 7=CURRENT, 8=RELATIVE	
Fetch type . . . . .	1	1=Single record 2=Blocked 3=Position only	
F3=Exit    F5=Refresh    F12=Cancel    F21=Display statement			

Copyright 2008 System i Developer, LLC

**Embedded SQL...****BLOCKED FETCH statement prompting**

Specify FETCH Statement			
Type choices, press Enter.			
Cursor name . . . . .	empcsr	_____	
Positioning . . . . .	1	1=NEXT, 2=PRIOR, 3=FIRST 4=LAST, 5=BEFORE, 6=AFTER 7=CURRENT, 8=RELATIVE	
Fetch type . . . . .	2	1=Single record 2=Blocked 3=Position only	
F3=Exit    F5=Refresh    F12=Cancel    F21=Display statement			

Copyright 2008 System i Developer, LLC

**Embedded SQL...****BLOCKED FETCH statement prompting**

```

                                Specify FETCH Statement

Type choices, press Enter.

FOR records . . . . . 10____ 1-32767
-OR-
FOR host variable . . . . . _____

Type choices, press Enter.

INTO . . . . . 1          1=Host structure array
                                2=Record storage area

Host structure array . . . . :array_____

F3=Exit   F5=Refresh   F12=Cancel   F21=Display statement

```

Copyright 2008 System i Developer, LLC

**FETCH Statement...****FETCH statement prompting - continued**

```

Columns . . . : 1 71          Edit          SKIP/QRPGSRC
SEU==>                AASQLTST
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+.
0023.00      C/EXEC SQL
0024.00      C+
0025.00      C+   FETCH NEXT FROM empcsr
0025.01      C+           INTO :number, :name, :salary
0026.00      C+
0027.00      C/END-EXEC
                ***** End of data *****

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor
F16=Repeat find   F17=Repeat change   F24=More keys

```

Copyright 2008 System i Developer, LLC

# Updating and Deleting Data

---

## Positioned Update and Delete Statements

- Updates or deletes the current row of an updatable cursor
- Can only be done after successful Fetch operation
- Adds a "Where Current of" clause to the Update and Delete statements



Copyright 2008 System i Developer, LLC

# Updating and Deleting Data...

---

## Positioned Update and Delete Statements

```

C/EXEC SQL
C+   DECLARE empcsr CURSOR FOR
C+       SELECT nbr, nam, sal
C+           FROM emp
C+           ORDER BY empid
C+           FOR UPDATE OF sal
C/END-EXEC
C*
C/EXEC SQL
C+   FETCH NEXT FROM empcsr
C+   INTO :number, :name, :salary
C/END-EXEC
C*
C/EXEC SQL
C+   UPDATE emp
C+       SET sal = sal + :raise
C+       WHERE CURRENT OF empcsr
C/END-EXEC

```



Copyright 2008 System i Developer, LLC

## CLOSE Statement

---

### Closes the cursor

- **Cursor must be opened in order to be closed**
- **DB2/400 closes cursors for other reasons also:**
  - job end
  - activation group ends
  - program ends
  - modules ends
  - commit or rollback without a 'with hold' clause
  - error handling.....
- **Rule of thumb**
  - close the cursor as soon as you're done with it

```

C*
C/EXEC  SQL
C+
C+      CLOSE  empcsr
C+
C/END-EXEC

```



Copyright 2008 System i Developer, LLC

## Error Detection and Handling

---

### SQLCODE in SQL Communications Area

- **Status always returned in SQLCODE**
  - both successful and unsuccessful statements
- **Programmer must check return codes within program**
- **SQL Communications Area (SQLCA)**
  - contains feedback information
  - must be included in all SQL programs
  - RPG includes SQLCA automatically
  - other languages must have specific include:

```

EXEC  SQL

      INCLUDE SQLCA

END-EXEC

```



Copyright 2008 System i Developer, LLC

## Error Detection and Handling

### SQL Communications Area (SQLCA)

SQLCAID	Char(8)	Structure identifying literal: "SQLCA"
SQLCABC	Integer	Length of SQLCA
SQLCode	Integer	Return code
SQLErrML	SmallInt	Length of SQLErrMC
SQLErrMC	Char(70)	Message Replacement text
SQLErrP	Char(8)	Product ID literal: "QSQ" for DB2/400
SQLErrD	Array of Integers	<p>SQLErrD(1) - treated as Char(4); last 4 characters of CPF or other escape message</p> <p>SQLErrD(2) - treated as Char(4); last 4 characters of CPF or other diagnostic message</p> <p>SQLErrD(3) - for Fetch, Insert, Update or Delete, number of rows retrieved or updated</p> <p>SQLErrD(4) - for Prepare, relative number indicating resources required for execution</p> <p>SQLErrD(5) - for multiple-row Fetch, contains 100 if last available row is fetched; for Delete, number of rows affected by referential constraints; for Connect or Set Connection, contains t-1 if unconnected, 0 if local and 1 if connection is remote</p> <p>SQLErrD(6) - when SQLCode is 0, contains SQL completion message id</p>

Copyright 2008 System i Developer, LLC

## Error Detection and Handling...

### SQL Communications Area (SQLCA) - continued

SQLWarn	Char(11)	Set of 11 warning indicators; each is blank, W, or N
SQLWarn0	Char(1)	Blank if all other SQLWARNx warning indicators are blank W if any warning indicator contains W or N
SQLWarn1	Char(1)	W if a string column was truncated when assigned to host variable
SQLWarn2	Char(1)	W if null values were eliminated from a function
SQLWarn3	Char(1)	W if number of columns is larger than number of host variables
SQLWarn4	Char(1)	W if prepared Update or Delete statement doesn't include a Where clause
SQLWarn5	Char(1)	Reserved
SQLWarn6	Char(1)	W if date arithmetic results in end-of-month adjustment
SQLWarn7	Char(1)	Reserved
SQLWarn8	Char(1)	W if result of character conversion contains the substitution character
SQLWarn9	Char(1)	Reserved
SQLWarnA	Char(1)	Reserved
SQLState	Char(5)	Return code; '00000' if no error or warning

Copyright 2008 System i Developer, LLC

# SQLCODE Error Handling

---

## SQLCODE Values

- **SQLCODE (SQLCOD) contains return code**
  - ▶ = 0 Successful statement execution
  - ▶ > 0 Successful, with warning condition
  - ▶ < 0 Unsuccessful - statement failed
- **SQLCODE value indicates exact error or condition**
  - ▶ e.g.. 100 = Row not found (or end of file)
  - ▶ e.g.. -552 = Not authorized to object
- **SQLCODE values have corresponding messages**
  - ▶ e.g.. SQL0100 = Row not found
  - ▶ e.g.. SQL0552 = Not authorized to &1.

Copyright 2008 System i Developer, LLC

## Error Checking Within a HLL Program

---

```

RPG      C/EXEC SQL
          C+  SELECT name INTO  :nam
          C+   WHERE emp = 100
          C/END-EXEC
          C           SQLCOD  IFLT           0
          C           EXSR    ERR
          C           ENDIF
          C           SQLCOD  IFGT           0
          C           EXSR    NF
          C           ENDIF

```

```

COBOL    EXEC SQL
          SELECT name INTO  :lastname
          WHERE  emp = 100
          END-EXEC.
          IF SQLCODE < 0 PERFORM ERROR-ROUTINE.
          IF SQLCODE > 0 PERFORM
          NOT-FOUND-ROUTINE.

```

Copyright 2008 System i Developer, LLC

# WHENEVER Error Handling

---

## WHENEVER statement checks SQLCA

- Can branch to a location based on condition
- Three conditions:
  - SQLWARNING (SQLCODE > 0 except 100)
    - OR (SQLWARN0 = 'W')
  - SQLERROR (SQLCODE < 0)
  - NOT FOUND (SQLCODE = 100)
- Two possible actions
  - CONTINUE
  - GO TO label

```

C*
C/EXEC SQL
C+
C+      WHENEVER SQLERROR GO TO error
C+
C/END-EXEC

```

Copyright 2008 System i Developer, LLC

## Better Error Handling Technique

---

Directly following each SQL statement - code an 'error catcher'

- Easier to determine source of the error
- More discrete method of determining action to be taken

```

C/EXEC SQL
C+  DECLARE empcsr CURSOR FOR
C+  SELECT nbr, nam, sal
C+  FROM emp
C+  WHERE dpt = :dept
C+  FOR UPDATE OF sal
C/END-EXEC
C*
C      EXFMT PROMPT
C*
C/EXEC SQL
C+  OPEN empcsr
C/END-EXEC
C*
C      SQLCOD DOUNE 0

```

Copyright 2008 System i Developer, LLC

## Better Error Handling Technique...

---

Directly following each SQL statement - continued

```

C*
C/EXEC SQL
C+
C+    FETCH NEXT FROM empcsr
C+    INTO  :number, :name, :salary
C+
C/END-EXEC
C*
C    SQLCOD  IFEQ  0
C*
C/EXEC SQL
C+    UPDATE emp
C+        SET sal = sal + :raise
C+        WHERE CURRENT OF empcsr
C/END-EXEC
C        END
C        END
C/EXEC SQL
C+    CLOSE empcsr
C/END-EXEC

```

Copyright 2008 System i Developer, LLC

## Dynamic SQL

---

What is Dynamic SQL?

- **Dynamic SQL is a different way to use SQL**
- **SQL statements are not pre-defined in program**
  - Dynamically created on the fly as part of program logic
- **SQL pre-compiler cannot fully process dynamically created SQL statements**
- **PREPARE statement used in program logic to compile dynamically created SQL statements**
  - Can be used with SQL EXECUTE statement



Copyright 2008 System i Developer, LLC

## Dynamic SQL...

---

### Example 1

**PROBLEM: Dynamically select records from the Employee Master File - Any fields, records, or sequence**

```

C*
C/EXEC SQL
C+
C+   PREPARE  search  FROM  :sqlstm
C+
C/END-EXEC
C*
C/EXEC SQL
C+
C+   DECLARE  empcsr  CURSOR  FOR  search
C+
C/END-EXEC
C*
C/EXEC SQL
C+
C+   OPEN  empcsr
C+
C/END-EXEC

```

Copyright 2008 System i Developer, LLC

## Dynamic SQL...

---

### Example 2

**User prompted to enter the delete condition (InpCond variable)**

```

C*
C   Eval  SQLStmtStr = 'Delete From Customer
C           Where '
C   Eval  SQLStmt = SQLStmtStr + InpCond
C/EXEC SQL
C+
C+   PREPARE  DynSQLStmt
C+   FROM  :SQLStmt
C+
C/END-EXEC
C*
C   If    (SQLCod = 0) And (SQLWn0 = *Blank)
C/EXEC SQL
C+
C+   EXECUTE  DynSQLStmt
C+
C/END-EXEC
C*

```

Copyright 2008 System i Developer, LLC

## Dynamic SQL...

---

### Where to use Dynamic SQL

- **Report programs with user run time selection**
  - Files
  - Fields
  - Record selection criteria
  - Sorting
  - SQL built in functions
- **Whenever the exact syntax of an SQL statement cannot be determined beforehand**



Copyright 2008 System i Developer, LLC

## Dynamic SQL...

---

### Where to use Dynamic SQL

- **Offers a high degree of application flexibility**
- **Can create/build SQL statement**
  - Based on parameters received from
    - Interactive user interface
    - List selection techniques
    - Application control file
  - Use any programming language



Copyright 2008 System i Developer, LLC

# Dynamic SQL...

---

## Performance considerations

- **Dynamic SQL can be resource intensive**
- **Remember that a dynamic SQL statement has to be parsed (interpreted) and executed within the application program it is part of**
  - Negative impact on performance

**Use it ... Don't abuse it!**



Copyright 2008 System i Developer, LLC

# Editing & Compiling Embedded SQL

---

## Embedded SQL Tips

- **Use SQL source member types**
  - e.g., SQLRPG, SQLRPGLE, SQLCBL, SQLCBLLE
  - Prompting won't work without SQL member type
- **You can prompt SQL statements in SEU only**
  - You MUST key both EXEC SQL and END-EXEC statements first
    - Then you can prompt (F4) for statements in between
    - Same prompter as interactive SQL
- **Compile commands**
  - Pre-ILE compilers
    - CRTSQLRPG, CRTSQLCBL
  - ILE compilers
    - CRTSQLRPGI, CRTSQLCBLI
    - Creates either \*PGM, \*SRVPGM or \*MODULE depending on parameter value specified for "Compile type" or OBJTYPE

Copyright 2008 System i Developer, LLC

# Editing & Compiling Embedded SQL...

---

## Embedded SQL Tips

- **Test statements in Interactive SQL before embedding them**
  - When exiting Interactive SQL session
    - You can save session statements to a source member
    - Copy from this member into your program source
- **Default for SQL is to use Commitment Control**
  - Requires journaling
    - Program execution fails if updated files are not journaled
  - To request no commitment control
    - COMMIT(\*NONE) on compile
- **SQL precompile step happens before RPG/COBOL compile**
  - Therefore, if SQL syntax or semantic error occurs, no "typical" compile source listing available
  - Can be very difficult to work through problems at this stage

Copyright 2008 System i Developer, LLC

# Performance Tips

---

## Designing and Writing Efficient SQL Queries

- **SQL uses two basic ways to retrieve data**
  - Dataspace scan or arrival sequence Generally used when MORE than 20% of records will be selected
  - Index based or keyed access Generally used when LESS than 20% of records will be selected
- **If SQL can use an index, performance can improve significantly!**



Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Designing and Writing Efficient SQL Queries

- **Create indexes for fields frequently referenced in**
  - WHERE clause
  - GROUP BY clause
  - ORDER BY clause
- **Create indexes for fields that are frequently used to join files**

### Performance Analysis Tools

- **STRDBMON**
- **Predictive Query Governor**
- **TRCJOB**
- **Visual Explain (V4R5)**



Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Multiple Row FETCH

- **Based on host structure array**
  - RPG - Multiple Occurrence Data Structure
  - COBOL - Occurs clause on declaration of the group item
- **Clause on FETCH**
  - FOR n ROWS
  - n = number of rows to be returned
- **Specifies the number of rows to be retrieved to fill the array structure**

Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Multiple Row FETCH - continued

```

D  EMP  DS          Occurs(10)
D      NBR          5  0
D      NAME         25
D      JOB          1
C*
C          Z-ADD    5  JOB
C/EXEC SQL
C+      FETCH Next
C+          FROM CustomerCursor
C+          FOR 10 ROWS
C+      INTO :Emp
C/END-EXEC
C          Eval    ErrCond = SQLErrD(5)
C*
```

Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Multiple Row FETCH - continued

- **First row fetched-** placed into first element of host structure
- **Fetching stops** when n rows are returned - or no more records
- **Program variable** may be used to specify number of records to be fetched (NOTE: RPG IV - %Elem built in function)
- **Be Careful - Fetching** is always forward from position set by **positioning keyword**
  - FETCH RELATIVE -3 .... FOR 3 ROWS
    - is not the same as
  - FETCH PRIOR .... FOR 3 ROWS
- **Results:**
  - SQLCA updates SQLErrCode(3) to reflect # rows retrieved
  - If no rows returned and no other exceptions, SQLCode is 100
  - If row(s) returned contain last available row, SQLErrCode(5) set to 100

Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Blocked INSERT

- Multiple elements from a host structure are inserted into a table
- Program variable may be used to set the number of records to be inserted
- Executes as if n INSERT statements had been issued
  - but with improved performance

```

D EMP          DS          Occurs(10)
D   NBR        5 0
D   NAME       25
D   JOB        1
C*
C/ EXEC SQL
C+   INSERT INTO Customer
C+   10 ROWS
C+   VALUES   :Emp
C/ END-EXEC

```

Copyright 2008 System i Developer, LLC

## Performance Tips...

---

### Invest time to review and become familiar with

- Performance and Optimization
  - Click on Database in iSeries Information Center
- You need to understand SQL performance before making extensive use of it!
  - Small changes in statements, indexes can significantly impact performance

**SQL is another programming language and you have to learn the tricks**

Copyright 2008 System i Developer, LLC

## Summary...

---

- Executing SQL on the Systemi, iSeries, & AS/400
- V5 SQL Support
- Basic SQL Statements
- SQL Precompiler
- RPG and Cobol Interface
- Cursor Operations
- Error Detection and Handling
- Dynamic SQL
- Editing and Compiling Embedded SQL
- Performance Tips
- Summary
- V5R3 and V5R4 SQL Information Sources



Copyright 2008 System i Developer, LLC

## V5 SQL Information Sources

---

- **iSeries Information Center Publications - Web or CD**
  - SQL Reference
  - SQL Programming
  - Embedded SQL Programming
  - Query Manager Use
  - SQL Messages and Codes
- **To access Info Center on the Web**
  - ▶ <http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp>
  - ▶ <http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp>
    - In left scroll bar
      - Click on iSeries Information Center ...
      - Click on Database
      - Click on Printable PDFs
      - Use right scroll bar to scroll down to above SQL publication
- **DB2 for iSeries on the Web**
  - ▶ <http://www.ibm.com/servers/eserver/iseries/db2/>

Copyright 2008 System i Developer, LLC

---

# Part 4

Copyright 2008 System i Developer, LLC

## Overview

---

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5

Copyright 2008 System i Developer, LLC

# Overview Part 4

---

## Objectives

- This session will introduce you to and discuss the basic theory and concepts needed to understand and begin designing, coding, and creating (System i, iSeries, and AS/400) external and SQL stored procedures. It assumes you have some understanding of and experience with application programming, and SQL.
- Thank you to Kent Milligan from the IBM Lab in Rochester, MN for providing much of the information for and assisting with the development of this presentation



Copyright 2008 System i Developer, LLC

# Overview Part 4...

---

- What is a Stored Procedure
- External Procedure
- SQL Procedure
- SQL Procedure Body
- SQL Procedure Language
- Getting Feedback Information
- Creating an SQL Procedure
- Error Handling
- CREATE PROCEDURE Statement
- Considerations
- Summary
- Appendix
  - CREATE PROCEDURE
  - SET OPTION



Copyright 2008 System i Developer, LLC

## Part 3 Overview...

---

### Syntax Used in SQL Examples

- **All upper case**
  - REQUIRED SQL SYNTAX
- **Lower case or combination of upper and lower case**
  - Parameter data supplied by user



Copyright 2008 System i Developer, LLC

## What is a Stored Procedure?

---

### Just a Called Program

- **Called from any SQL interface via SQL CALL statement**
- **Supports input and output parameters**
- **Created with CREATE PROCEDURE statement**
  - One time operation creates persistent object
- **Result sets can be returned on some interfaces**
  - Defined in Option List of CREATE PROCEDURE statement
- **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**

Copyright 2008 System i Developer, LLC

## What is a Stored Procedure...

---

### SQL Interfaces on the iSeries and AS/400

- Interactive SQL
- Imbedded or compiled in application programs
- Query Manager
- SQL Statement Processor
- Operations Navigator
- Dynamic SQL
- SQL Procedure Language
- Extended Dynamic SQL
- ODBC, JDBC
- JSQL
- X/Open SQL Call Level Interface
- Query Management (different from Query Manager)

Copyright 2008 System i Developer, LLC

## What is a Stored Procedure...

---

### Client Invocation of Predefined Function on Server

- **Two Types**
  - External Procedure - Implemented in V3R1
    - Register high-level language program as a stored procedure
      - RPG, COBOL, C, etc
    - External program for procedure may use embedded SQL
      - DB2 SQL Development Kit required
  - SQL Procedure - Implemented in V4R2
    - Entire procedure including business logic is coded with SQL
    - System Openness Includes (5722-SS1) required for creation
      - DB2 SQL Development Kit requirement eliminated in V5R2
      - C compiler requirement eliminated in V5R1
- **iSeries can be BOTH the client and server**
- **CREATE PROCEDURE (SQL) statement used for creation**

Copyright 2008 System i Developer, LLC

## What is a Stored Procedure...

---

### Can Provide Significant Performance Improvement

- Client server applications
- Database serving
- Network centric computing
- Web based computing
- Any environment where two or more computers are connected to a network structure and one or more of these computers requests information or data from a primary or host computer on the same network structure



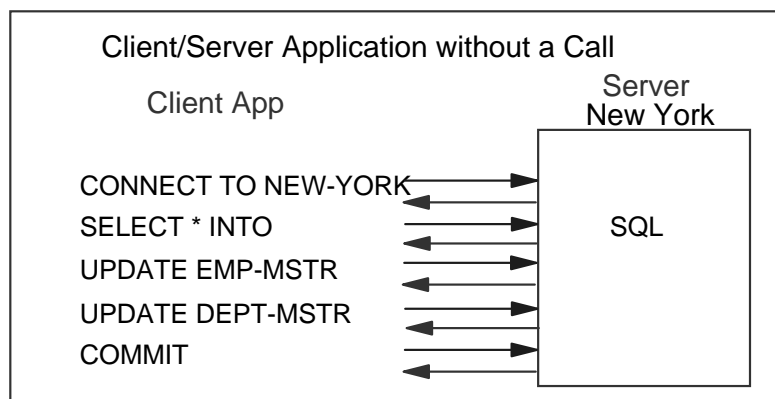
Copyright 2008 System i Developer, LLC

## What is a Stored Procedure...

---

### Improving Performance - Network Traffic Flow

- Without CALL of stored procedure every executable SQL statement causes a network trip from client application to server and back to the client
- Each SQL statement is passed to server to perform SQL function
- Resource intensive

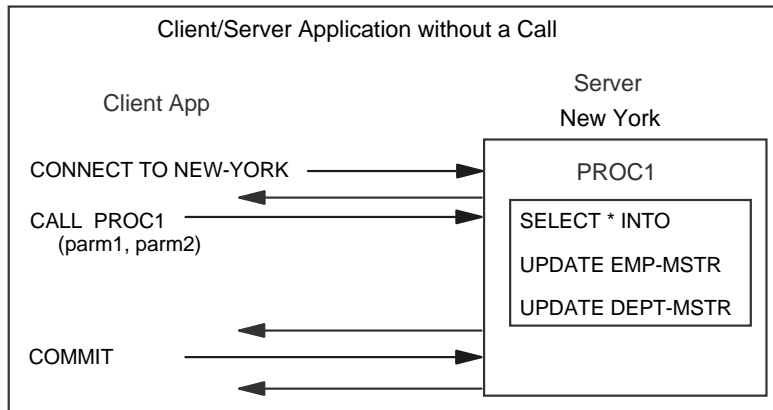


Copyright 2008 System i Developer, LLC

## What is a Stored Procedure...

### Improving Performance - Minimize Network Traffic Flow

- **CALL** of stored procedure results in a significant reduction in the number of network trips in the conversation between client and server
- **SQL statements** are packaged as a stored procedure on server resulting in very efficient SQL performance



Copyright 2008 System i Developer, LLC

## External Procedure

### \*PGM Object Performs Function on Server

- **Languages supported**
  - Command Language (CL)
  - RPG
  - COBOL
  - C
  - Java
  - Others
- **Can contain embedded SQL statements**
  - Static or dynamic
- **Creation**
  - CREATE PROCEDURE (SQL) statement
  - iSeries Navigator graphical database interface
- **\*PGM object must exist on server when procedure is created**



Copyright 2008 System i Developer, LLC

# SQL Procedure

---

## SQL Procedure Body Performs Function on Server

- **Stored Procedure portability across platforms**
  - Supported across DB2 UDB Family
  - New (V4R2) SQL Procedure Language
    - Similar to procedure languages available from other DBMSs (PL/SQL, T-SQL, etc)
- **Follows SQL PSM Standard**
- **Makes it easier for SQL programmers to be productive more quickly on the iSeries**
- **Language also used for iSeries**
  - SQL User Defined Functions
  - SQL Triggers
- **Invoked with SQL CALL statement**



Copyright 2008 System i Developer, LLC

# SQL Procedure...

---

## SQL Procedure Body

- **aka SQL Routine Body**
  - BEGIN
    - Local variable declarations
    - Local cursor declarations
    - Local condition and/or handler declarations
    - one or more simple, compound, and/or nested SQL Procedure Language statements that define procedure logic to perform desired function
  - END
- **Coded using SQL Procedure Language**
  - SQL Control Statements
    - Procedural programming constructs
    - Error handling
    - Standard DDL and DML statements



Copyright 2008 System i Developer, LLC

# SQL Procedure Language

---

## SQL Control Statements

- **SQL statements that allow SQL to be used in a manner similar to writing a program in a structured programming environment**
- **Provide the capability to**
  - Control logic flow
  - Declare and set variables
  - Handle warnings and exceptions.
- **Some SQL control statements include other nested SQL statements**
- **For detailed Control Statement information, including statement syntax see Chapter 6: *SQL Control Statements* in the SQL Reference manual**



Copyright 2008 System i Developer, LLC

# SQL Procedure Language...

---

## SQL Control Statements...

- **Compound statement**
  - BEGIN and END
  - DECLARE (local variables)
- **Procedure statements**
  - SELECT, UPDATE, INSERT, DELETE, etc
- **Assignment statement**
  - SET (local variables)
- **Comments**
- **CALL**
  - SQL procedure
  - External procedure to access
    - HLL programs
    - OS/400 APIs



Copyright 2008 System i Developer, LLC

## **SQL Procedure Language...**

---

### **SQL Control Statements...**

- **CASE (two forms), END CASE**
- **IF, THEN, ELSE, END IF**
- **FOR, END FOR**
- **LOOP, END LOOP**
- **LEAVE (loop or block)**
- **REPEAT, END REPEAT**
- **WHILE, END WHILE**
- **ITERATE**
- **GOTO**



Copyright 2008 System i Developer, LLC

## **SQL Procedure Language...**

---

### **SQL Control Statements for Error Handling**

- **GET DIAGNOSTICS**
- **DECLARE CONDITION**
- **DECLARE HANDLER**
- **SIGNAL**
- **RESIGNAL**



Copyright 2008 System i Developer, LLC

# Compound Statement

---

## Defines the Procedure or Routine Body

- **Groups SQL statements together within the SQL procedure body and allows for the declaration of SQL variables, cursors, conditions, and handlers.**
- **Compound (SQL) statement must be structured and ordered as follows**
  - **BEGIN**
    - Local variable declarations
    - Local cursor declarations
    - Local condition and/or handler declarations
    - one or more simple, compound, and/or nested SQL Procedure Language statements that define procedure logic to perform desired function
  - **END**

Copyright 2008 System i Developer, LLC

# Compound Statement...

---

## Defines the Procedure or Routine Body...

- **Each SQL statement must end with a ; (semi colon)**

```

BEGIN
    DECLARE CurLvl INT;
    SELECT edlevel FROM emptbl INTO CurLvl
        WHERE empno=Emp#;
    IF NwLvl > CurLvl THEN
        UPDATE emptbl SET edlevel=NwLvl,
            salary=salary + (salary*0.05) WHERE empno=Emp#;
    END IF;
END

```

Copyright 2008 System i Developer, LLC

# Compound Statement...

---

## BEGIN and END

- **Delineates the compound (SQL) statement**

```
BEGIN NOT ATOMIC (the default) or ATOMIC
  One or more SQL procedure statements;
END
```

- Compound statements can be nested within each other - V5R2
- For details see *compound-statement* in SQL Reference manual

- **NOT ATOMIC**

- If an error occurs, it does NOT cause statements to be rolled back

- **ATOMIC**

- If an error occurs, all SQL statements in the compound statement group will be rolled back
- If ATOMIC specified
  - COMMIT or ROLLBACK not allowed in stored procedure
  - Procedure must be created with COMMIT ON RETURN YES - V5R2

Copyright 2008 System i Developer, LLC

# Compound Statement...

---

## Variable Declaration

```
+---<-,-----<---+           +-DEFAULT NULL---+
DECLARE+-SQL variable name+-data type---+-----+
                                           +-DEFAULT--value-+
```

- **Variable initialized when the SQL procedure is called**

```
DECLARE midinit, edlevel CHAR(1);
DECLARE ordquantity INT DEFAULT 0;
DECLARE enddate DATE DEFAULT NULL;
```

- **If default value not defined for a variable, it is initialized as null**

Copyright 2008 System i Developer, LLC

# SQL Procedure Statements

---

## DDL and DML Statements

- Used in conjunction with SQL Control Statements
- Standard DDL and DML Statements listed under SQL procedure statements
  - SELECT, UPDATE, INSERT, DELETE
  - CREATE, DROP, ALTER
  - OPEN, CLOSE, FETCH
  - PREPARE, EXECUTE, DESCRIBE
  - GRANT, REVOKE
  - COMMIT, ROLLBACK - **SQL procedures only**
  - CONNECT, DISCONNECT - **SQL procedures only**
  - SAVEPOINT
  - LABEL
  - RENAME
  - and more



Copyright 2008 System i Developer, LLC

# Assignment Statement and Comments

---

## SET and Comments

- **SET**
  - Use for assigning a value to SQL parameter or SQL variable
 

```
SET total_salary = emp_salary + emp_commission;
SET total_salary = NULL;
SET loc_avg_salary = (SELECT AVG(salary) FROM employees);
```
  - For details see *assignment-statement* in SQL Reference manual
- **Two options for Comments**
  - Two consecutive hyphens (--)
  - Bracketed comments (/\* ... \*/)



Copyright 2008 System i Developer, LLC

# CALL Statement

---

## For Invoking Stored Procedures

- Provides a mechanism for accessing system functions and APIs from an SQL Stored Procedure

```
CALL ProcedureName(Parmeter1, Parmeter2, etc);
```

- Parameters or arguments allowed on CALL statement
  - 1024 - V5R4
  - 254 - V5R3
- A parameter can be
  - SQL parameter name
  - SQL variable name
  - Constant
  - Special register
  - NULL



Copyright 2008 System i Developer, LLC

# Conditional Constructs

---

## CASE Expression - Two Forms

- First form

```
CASE workdept
  WHEN 'A00' THEN UPDATE department
    SET deptname = 'ACCOUNTING';
  WHEN 'B01' THEN UPDATE department
    SET deptname = 'SHIPPING';
  WHEN 'A01' THEN UPDATE department
    SET deptname = 'MARKETING';
  ELSE UPDATE department
    SET deptname = 'UNKNOWN';
END CASE
```



Copyright 2008 System i Developer, LLC

# Conditional Constructs...

---

## CASE Expression - Two Forms

- **Second form**

```
CASE
  WHEN workdept = 'A00' THEN UPDATE department
    SET deptname = 'ACCOUNTING';
  WHEN workdept = 'B01' THEN UPDATE department
    SET deptname = 'SHIPPING';
  WHEN workdept = 'A01' THEN UPDATE department
    SET deptname = 'MARKETING';
  ELSE UPDATE department
    SET deptname = 'UNKNOWN';
END CASE
```



Copyright 2008 System i Developer, LLC

# Conditional Constructs...

---

## IF Statement

- **Traditional IF-THEN-ELSE logic**

```
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;
```



Copyright 2008 System i Developer, LLC

# Looping Constructs

---

## FOR Statement

- **Execute an SQL 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 with WHERE CURRENT OF clause**

Copyright 2008 System i Developer, LLC

# Looping Constructs...

---

## LOOP and LEAVE Statement

- **Repeat the execution of an SQL statement**

```
fetch_loop:
LOOP
  FETCH cursor1 INTO  firstname, midinit, lastnm;
  IF SQLCODE < > 0
    THEN LEAVE fetch_loop;
  END IF;
  SET fullname = firstname || ' ' || midinit || ' ' || lastnm;
END LOOP;
```

- **LEAVE statement**
  - Exit the specified loop or block and continue execution at the statement immediately following the END LOOP statement
  - Can be used with all looping constructs except FOR statement

Copyright 2008 System i Developer, LLC

## Looping Constructs...

---

### REPEAT Statement

- Same as LOOP but with loop exit condition

```
REPEAT
  FETCH cursor1 INTO firstname, midinit, lastnm;
  SET fullname = firstname || ' ' || midinit || ' ' || lastnm;
  UNTIL SQLCODE <> 0
  END REPEAT;
```

- Exit condition evaluated after each execution of the loop

Copyright 2008 System i Developer, LLC

## Looping Constructs...

---

### WHILE Statement

- Same as REPEAT but exit condition evaluated before each execution of the loop

```
WHILE at_end = 0 DO
  FETCH cursor1 INTO firstname, midinit, lastnm;
  IF SQLCODE <> 0
    THEN SET at_end = 1;
  ELSE
    SET fullname = firstname || ' ' || midinit || ' ' || lastnm;
  END IF;
END WHILE;
```

Copyright 2008 System i Developer, LLC

# Looping Constructs...

---

## ITERATE Statement

- **Causes flow of control to return to the beginning of labeled loop**
  - Can be used with any looping construct

```

ins_loop: LOOP
  FETCH c1 INTO dept, deptname, admdept;
  IF at_end = 1
    THEN LEAVE ins_loop;
  ELSEIF dept = 'D11'
    THEN ITERATE ins_loop;
  END IF;
  INSERT INTO department VALUES ('NEW', deptname, admdept);
END LOOP;

```

Copyright 2008 System i Developer, LLC

# Branching Construct

---

## GOTO Statement

- **Branches to a user defined label**
- **Should be used sparingly**
  - Support provided for application porting comparability
  - Other databases use it primarily for error handling and status reporting
  - Cannot jump into or out of a FOR loop or handler with GOTO

```

IF P1 = 1
  THEN GOTO whileloop2;
END IF;
...

whileloop2: WHILE at_end = 0 DO
  FETCH cursor1 INTO firstname, midinit, lastnm;
  SET fullname = firstname || ' ' || midinit || ' ' || lastnm;
END WHILE;

```

Copyright 2008 System i Developer, LLC

## Getting Feedback Information

---

### No Direct Access to SQLCA Provided

- **Lack of data structure support in SQL procedure language results in no SQLCA access from an SQL Procedure**
- **Access feedback information by declaring SQLSTATE or SQLCODE variables**
  - Automatically updated by DB2 UDB

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

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

- **Each procedure language statement is an SQL statement**
  - May need to check SQLSTATE or SQLCODE after each statement

Copyright 2008 System i Developer, LLC

## Getting Feedback Information...

---

### GET DIAGNOSTICS

- **Provides information about the previous SQL statement that was executed**
- **Function available when used in an SQL procedure, SQL function, or SQL trigger is a subset of what is available when used in other contexts**
- **Current support only provides the count of the number of rows processed by an INSERT, UPDATE, or DELETE statement**

```
DECLARE update_counter INTEGER;
...
```

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

Copyright 2008 System i Developer, LLC

## Error Handling - Conditions and Handlers

### Condition

- **CONDITION declaration**

- Associates user condition name with specific SQLSTATE value

```
DECLARE row_not_fnd CONDITION FOR SQLSTATE '02000';
```

- Condition name must be unique within the compound-statement
  - excludes any declarations in compound statements nested within the compound statement
- Condition name can only be referenced within the compound statement in which it is declared
  - Includes any compound statements nested within the compound statement

Copyright 2008 System i Developer, LLC

## Error Handling - Conditions and Handlers...

### Handler

- **HANDLER declaration**

- Associates a handler option with a user condition name

```
DECLARE CONTINUE HANDLER FOR row_not_fnd
```

- Specifies **one of three options** to execute as the handler for an exception or completion condition occurring in the compound statement

```
CONTINUE
EXIT
UNDO
```

- A handler is active for the set of SQL procedure statements that follow the handler declaration within the compound statement in which it is declared
- A handler for a condition may exist at several levels of nested compound statements

Copyright 2008 System i Developer, LLC

## **Error Handling - Conditions and Handlers...**

---

### **Conditions and Handlers**

- **Example**

```
DECLARE row_not_fnd CONDITION FOR SQLSTATE '02000';  
DECLARE CONTINUE HANDLER FOR row_not_fnd  
SET at_end='Y';
```

- **User condition name row\_not\_found associated with SQLSTATE '02000'**
- **When condition row\_not\_found occurs the CONTINUE option in the handler for row\_not\_found causes the SET statement after the handler declaration to be executed**

Copyright 2008 System i Developer, LLC

## **Error Handling - SIGNAL and RESIGNAL**

---

### **SIGNAL and RESIGNAL**

- **SIGNAL**
  - Causes an error or warning to be returned with optional message text for the specified SQLSTATE or condition
- **RESIGNAL**
  - Same as SIGNAL but it is used within a handler to return an error or warning with optional message text.
- **ODBC and JDBC Drivers make SIGNAL and RESIGNAL error information available to the client application**

Copyright 2008 System i Developer, LLC

## Error Handling - SIGNAL and RESIGNAL...

---

### SIGNAL and RESIGNAL Example

```

DECLARE row_not_found CONDITION FOR SQLSTATE '02000';

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

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

-- check if raise within the limit

IF (raise > 2500)
    THEN SIGNAL SQLSTATE '38S01';
END IF;

UPDATE employee SET salary = salary + raise
    WHERE empno_db = empno_input;

```

Copyright 2008 System i Developer, LLC

## Creating an SQL Procedure

---

### Options for Creating an SQL Procedure

- **CREATE PROCEDURE statement**

```

CREATE PROCEDURE...
    Required and optional parameters
BEGIN
    SQL Procedure body;
    One or more SQL statements;
END

```

- **iSeries Navigator graphical database interface**

- **DB2 Store Procedure Builder in**

- DB2 Development Center
  - Download DB2 Personal Developer's Edition from web

Copyright 2008 System i Developer, LLC

# CREATE PROCEDURE Statement

## Create External Procedure - Simplified Syntax

```

>>--CREATE PROCEDURE----procedure-name----->
>-----+-----+----->
>   +- (-----) +-
>     | +- ,-----+
>     | |-----|
>     +-+---parameter-declaration---+
>-----+-----+-----option-list-----><
>   +-LANGUAGE---programming-language-+

```

Copyright 2008 System i Developer, LLC

# CREATE PROCEDURE Statement...

## Create SQL Procedure - Simplified Syntax

```

>>--CREATE PROCEDURE----procedure-name----->
>-----+-----+----->
>   +- (-----) +-
>     | +- ,-----<-----+
>     | |-----|
>     +-+---parameter-declaration---+
>-----+-----+-----option-list-----><
>   --LANGUAGE--SQL-----
>-----+-----+-----BEGIN-----SQL-routine-body-----END--><
>   +-SET OPTION--option-+          aka SQL procedure body

```

Copyright 2008 System i Developer, LLC







## Option List Parameters...

---

### SAVEPOINT LEVEL and COMMIT ON RETURN

```

+-OLD SAVEPOINT LEVEL--+      +-COMMIT ON RETURN-----+
>-----+-----+-----+-----+-----+-----+----->
+-NEW SAVEPOINT LEVEL--+      +-COMMIT ON RETURN YES--+

```

- **SAVEPOINT LEVEL**
  - Savepoint is a commitment control synchronization point for a unit of work
  - OLD - use same savepoint level as the caller of the procedure
  - NEW - create new savepoint level on entry to the procedure, one level down from caller
- **COMMIT ON RETURN NO**
  - No commit is issued for the transaction when the procedure returns or ends
  - No is the default
- **COMMIT ON RETURN YES**
  - Commit is issued for the transaction when the procedure returns or ends successfully
  - No commit is issued if procedure returns or ends with an error

Copyright 2008 System i Developer, LLC

## SET OPTION Statement

---

### SQL Procedures Only

```

>-----+-----+-----+-----+-----+-----+----->
|                                     +-,-<--<--+ |
+-SET OPTION---option---+

```

- **SET OPTION statement**
  - Specifies the processing options to be used when creating the SQL procedure
  - Many options available - ALWCPYDTA, COMMIT, DATFMT, DBGVIEW, to name a few
  - For details see SET OPTION Statement in appendix following CREATE PROCEDURE
  - SET OPTION DBGVIEW = \*SOURCE

Copyright 2008 System i Developer, LLC

# Creating the Procedure

---

## CREATE PROCEDURE Example - SQL Procedure

```
CREATE PROCEDURE CREDITPGM
  ( IN perinc_in DECIMAL(3,2), IN group_in CHAR(1),
    INOUT numrec_inout 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 crsr1 CURSOR FOR
    SELECT cusnbr, cuscrd FROM ordapplib/customer
      WHERE cusgroup = group_in;
```

... continued on next slide ...

Copyright 2008 System i Developer, LLC

# Creating the Procedure...

---

## CREATE PROCEDURE Example - SQL Procedure

... continued from previous slide ...

```
  SET numrec = 0;
  OPEN crsr1;
  FETCH crsr1 INTO proc_cusnbr, proc_cuscrd;

  WHILE at_end = 0 DO
    SET proc_cuscrd = proc_cuscrd +(proc_cuscrd * perinc_in);
    UPDATE ordapplib/customer SET cuscrd = proc_cuscrd
      WHERE CURRENT OF crsr1;
    SET numrec = numrec + 1;
    FETCH crsr1 INTO proc_cusnbr, proc_cuscrd;
  END WHILE;

  SET numrec_inout = numrec;
  CLOSE crsr1;
```

```
END
```

- **Under the covers DB2 UDB creates a C program object based on the procedure body, using the CRTSQLCI and CRTPGM commands**
  - ILE C compiler no longer required as of V5R1
  - SQL licensed product no longer required as of V5R2

Copyright 2008 System i Developer, LLC

# Creating a Procedure with iSeries Navigator

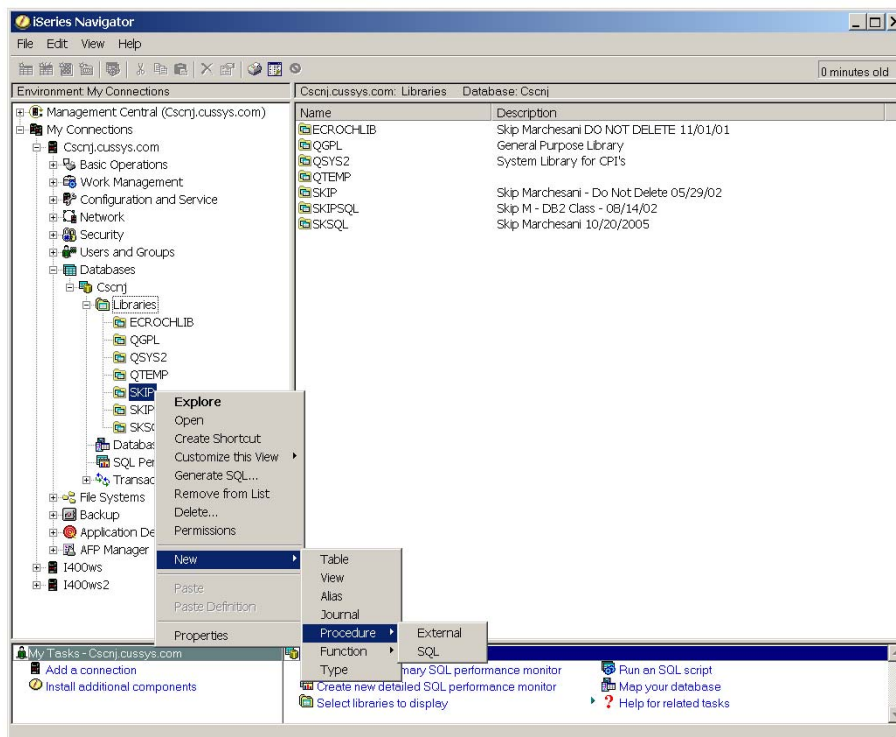
## Database Function Required

- **Left click on**
  - Databases
  - Database name
  - Libraries
- **Right click on target library name**
- **Move cursor over**
  - New
  - Procedures
- **Left click on External or SQL**



Copyright 2008 System i Developer, LLC

## Creating a Procedure with iSeries Navigator...



Copyright 2008 System i Developer, LLC

## Creating a Procedure with iSeries Navigator...

New SQL Procedure in SKIP - Cscnj.cussys.com(Cscnj)

General | Parameters | SQL Statements

Procedure:

Description:

Maximum number of result sets:

Same result returned from successive calls with identical input (Deterministic)

Commit changes when control returns to caller

Initiate new savepoint level when invoked

Data access:

Specific name:

OK Cancel Help

Copyright 2008 System i Developer, LLC

## Moving Procedures Into Production

### Saving and Restoring a Procedure

- **Need to save & restore the C program object that gets created by DB2 UDB for iSeries**
  - Program object is tagged so that DB2 UDB can recognize it as an SQL Stored Procedure (in most cases - see next chart)
  - Procedure is registered/restored into the target library specified on the Restore command regardless of the CREATE PROCEDURE source
  - If SPECIFIC specified on CREATE PROCEDURE statement, then SPECIFIC name must be unique
    - System only generates unique specific name when SPECIFIC not specified

Copyright 2008 System i Developer, LLC

# **Moving Procedures Into Production...**

---

## **Restoring a Procedure**

- **DB2 UDB will try to recognize the C program as an SQL Stored Procedure when restoring**
  - If DB2 UDB does not find a matching procedure in the catalogs, then the C program is registered as an SQL Stored Procedure
  - If DB2 UDB finds one procedure with the same name (same parameters - 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 UDB finds one or more procedure 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

Copyright 2008 System i Developer, LLC

# **Procedures and Adopted Authority**

---

## **Use Adopted Authority**

- **Do not have to give each ODBC & JDBC application user authority/privileges to every database table and file**
- **Remote data access can be restricted by requiring stored procedures to be used**
- **How?**
  - SET OPTION USRPRF = ... DYNUSRPRF...
  - or
  - CHGPGM ... USRPRF(\*OWNER) ...

Copyright 2008 System i Developer, LLC

## Result Sets Processing

---

### Can Offer Significant Performance Improvement

- **When used with SQL procedures can drastically reduce network trips by returning blocks of data**
- **SQL interfaces that support returning result sets**
  - iSeries Access ODBC driver
  - CLI - Call Level Interface
  - JDBC driver
    - Toolbox
    - Native
  - DB2 Connect
  - iSeries DRDA Connections (V5R2)
    - No support for embedded SQL to handle returned result set
- **SQL cursor used to return result set**

Copyright 2008 System i Developer, LLC

## Result Sets Processing...

---

### Result Set Processing Can Be Complex

- **For detailed information on Result Set Processing see:**
  - **SET RESULTS SET** statement
  - **DYNAMIC RESULTS SETS** parameter of the CREATE PROCEDURE statement
- **Chapter 6: Statements in the *SQL Reference* manual**



Copyright 2008 System i Developer, LLC

## Performance Considerations

---

### Do Multiple Operations with Single Procedure Call

- **User written code may be more efficient than generated C code with embedded SQL**
- **No support for blocked FETCH and INSERT**
- **Minimize the number of CALLs to other SQL Procedures**
  - SQL CALL is not bound
- **Local variable suggestions**
  - Declare as NOT NULL
  - Use integer in place of decimal with precision of 0 (zero)
  - Minimize the usage of character and date variables
  - Use the same data type, length, and scale for numeric variables that are used together in the same assignment

Copyright 2008 System i Developer, LLC

## General Considerations

---

### Transactions

- **ILE C program object for procedure created with Activation Group \*CALLER**
- **If SQL procedure created as ATOMIC then the invoker has to be at a commit boundary before invoking the ATOMIC SQL Stored Procedure**
- **COMMIT and ROLLBACK not allowed in a procedure with the ATOMIC attribute**



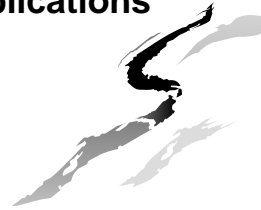
Copyright 2008 System i Developer, LLC

## Summary Part 3

---

### Stored Procedures Offer Flexibility

- **Function Implemented On Server - Not On the Client**
- **Client application can invoke**
  - Common and/or complex application function
  - **ANY** iSeries program
  - SQL procedure
- **Improved client server application performance by minimizing the number of communication flows between the client and server**
- **Simplifies the design of client server applications**



Copyright 2008 System i Developer, LLC

## Summary Part 3...

---

- **What is a Stored Procedure**
- **External Procedure**
- **SQL Procedure**
- **SQL Procedure Body**
- **SQL Procedure Language**
- **Getting Feedback Information**
- **Creating an SQL Procedure**
- **Error Handling**
- **CREATE PROCEDURE Statement**
- **Considerations**



Copyright 2008 System i Developer, LLC

## Appendix Part 3

---

- **CREATE PROCEDURE Statement**
- **SET OPTION Statement**



Copyright 2008 System i Developer, LLC

---

## CREATE PROCEDURE

The CREATE PROCEDURE statement defines a procedure at the current server.

The following types of procedures can be defined:

- External

The procedure program or service program is written in a programming language such as C, COBOL, or Java. The external executable is referenced by a procedure defined at the current server along with various attributes of the procedure. See “CREATE PROCEDURE (External)” on page 639.

- SQL

The procedure is written exclusively in SQL. The procedure body is defined at the current server along with various attributes of the procedure. See “CREATE PROCEDURE (SQL)” on page 653.

### Notes

**Choosing data types for parameters:** For portability of procedures across platforms that are not DB2 UDB for iSeries, do not use the following data types, which might have different representations on different platforms:

- FLOAT. Use DOUBLE or REAL instead.
- NUMERIC. Use DECIMAL instead.

**Specifying AS LOCATOR for a parameter:** Passing a locator instead of a value can result in fewer bytes being passed in or out of the procedure. This can be useful when the value of the parameter is very large. The AS LOCATOR clause specifies that a locator to the value of the parameter is passed instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type.

AS LOCATOR cannot be specified for SQL procedures.

**Determining the uniqueness of procedures in a schema:** At the current server, each procedure signature must be unique. The signature of a procedure is the qualified procedure name combined with the number of the parameters (the data types of the parameters are not part of a procedure’s signature). This means that two different schemas can each contain a procedure with the same name that have the same number of parameters. However, a schema must not contain two procedures with the same name that have the same number of parameters.

**The specific name for a procedure:** When defining multiple procedures with the same name and schema (with different number of parameters), it is recommended that a specific name also be specified. The specific name can be used to uniquely identify the procedure when dropping, granting to, revoking from, or commenting on the procedure.

If the SPECIFIC clause is not specified, a specific name is generated.

**Special registers in procedures:** The settings of the special registers of the invoker are inherited by the procedure when called and restored upon return to the invoker.

---

## CREATE PROCEDURE (External)

The CREATE PROCEDURE (External) statement defines an external procedure at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- For the SYSPROCS catalog view and SYSPARMS catalog table:
  - The INSERT privilege on the table, and
  - The system authority \*EXECUTE on library QSYS2
- Administrative authority

If the external program or service program exists, the privileges held by the authorization ID of the statement must include at least one of the following:

- For the external program or service program that is referenced in the SQL statement:
  - The system authority \*EXECUTE on the library that contains the external program or service program.
  - The system authority \*EXECUTE on the external program or service program, and
  - The system authority \*CHANGE on the program or service program. The system needs this authority to update the program or service program object to contain the information necessary to save/restore the procedure to another system. If user does not have this authority, the procedure is still created, but the program or service program object is not updated.
- Administrative Authority

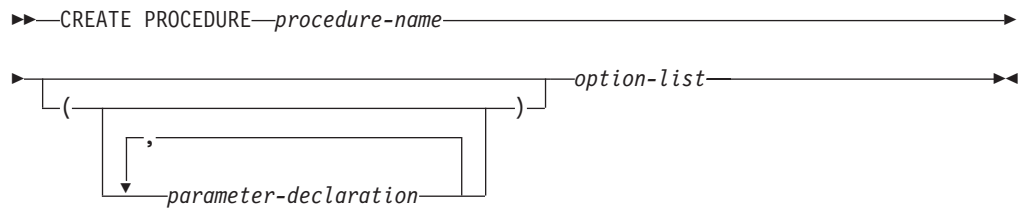
If a distinct type is referenced, the privileges held by the authorization ID of the statement must include at least one of the following:

- For each distinct type identified in the statement:
  - The USAGE privilege on the distinct type, and
  - The system authority \*EXECUTE on the library containing the distinct type
- Administrative authority

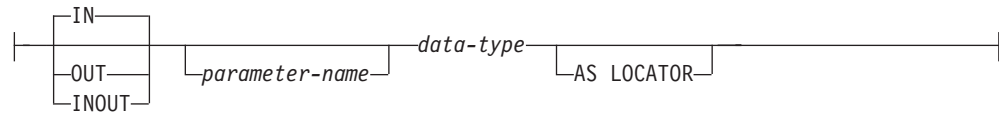
For information on the system authorities corresponding to SQL privileges, see “Corresponding System Authorities When Checking Privileges to a Function or Procedure” on page 864 and “Corresponding System Authorities When Checking Privileges to a Distinct Type” on page 856.

### Syntax

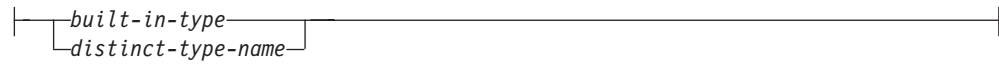
## CREATE PROCEDURE (External)



### parameter-declaration:

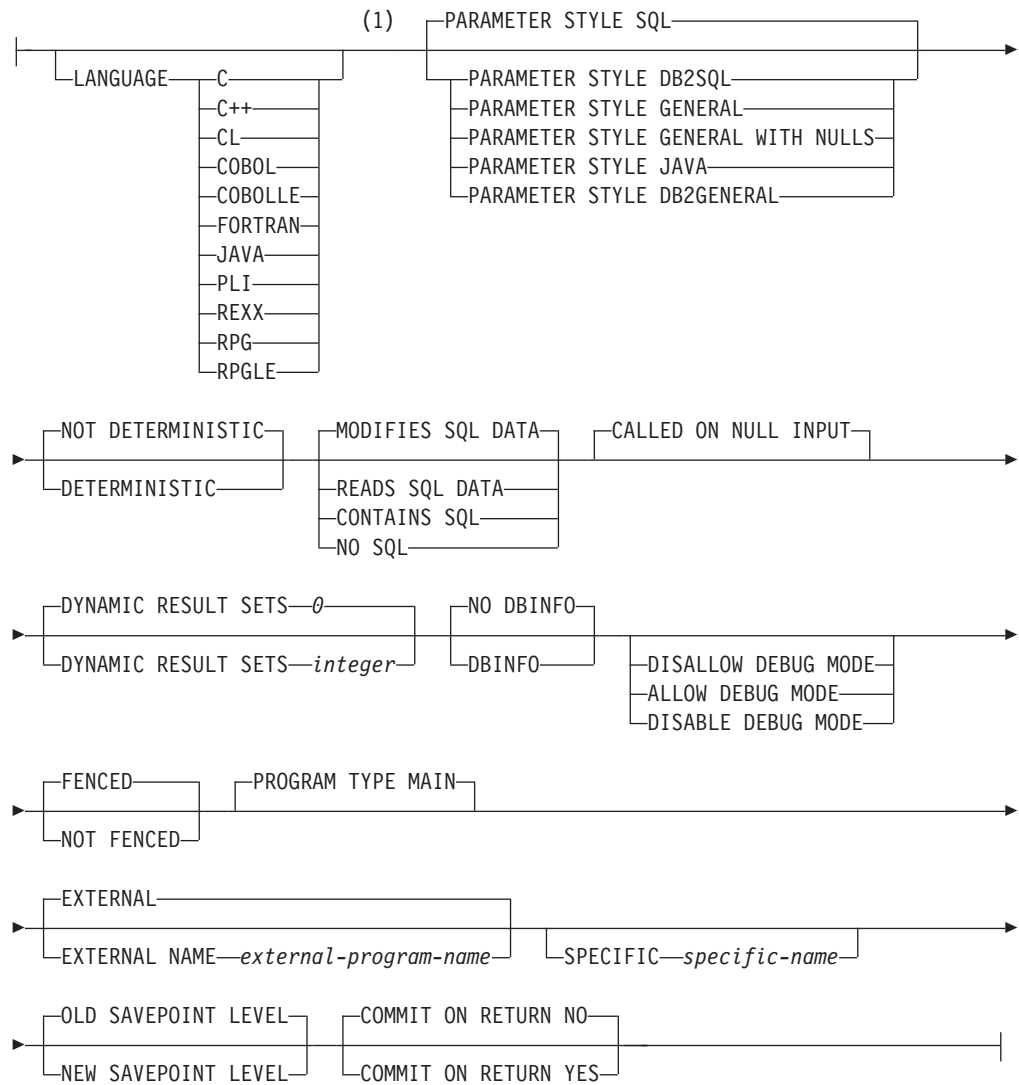


### data-type:



## CREATE PROCEDURE (External)

### option-list:

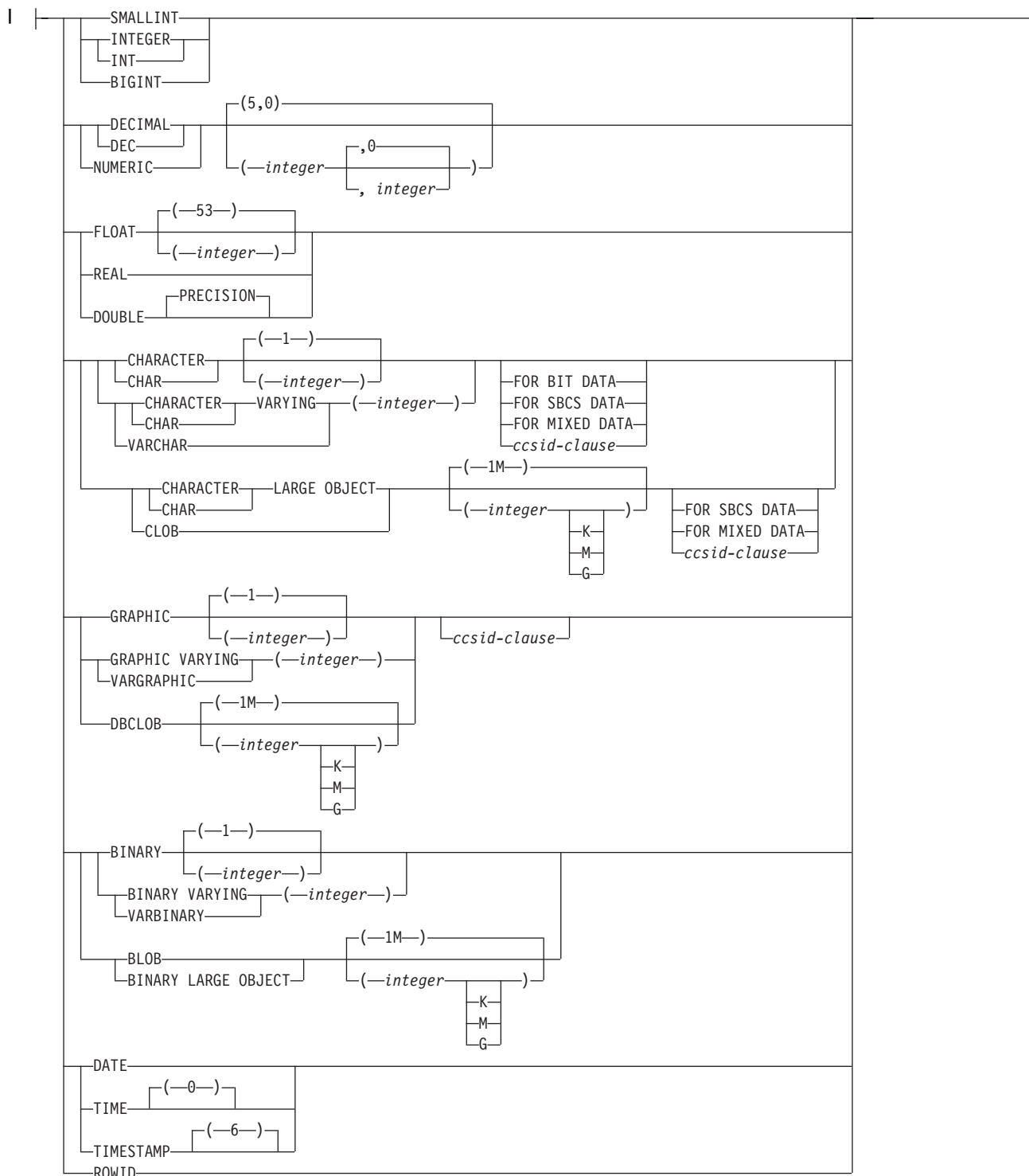


### Notes:

- 1 The optional clauses can be specified in a different order.

# CREATE PROCEDURE (External)

## built-in-type:



## ccsid-clause:



## Description

### *procedure-name*

Names the procedure. The combination of name, schema name, the number of parameters must not identify a procedure that exists at the current server.

For SQL naming, the procedure will be created in the schema specified by the implicit or explicit qualifier.

For system naming, the procedure will be created in the schema specified by the qualifier. If no qualifier is specified:

- If the value of the CURRENT SCHEMA special register is \*LIBL, the procedure will be created in the current library (\*CURLIB).
- Otherwise, the procedure will be created in the current schema.

### *(parameter-declaration,...)*

Specifies the number of parameters of the procedure and the data type of each parameter. A parameter for a procedure can be used only for input, only for output, or for both input and output. Although not required, you can give each parameter a name.

The maximum number of parameters allowed in CREATE PROCEDURE depends on the language and the parameter style:

- If PARAMETER STYLE GENERAL is specified, in C and C++, the maximum is 1024. Otherwise, the maximum is 255.
- If PARAMETER STYLE GENERAL WITH NULLS is specified, in C and C++, the maximum is 1023. Otherwise, the maximum is 254.
- If PARAMETER STYLE SQL or PARAMETER STYLE DB2SQL is specified, in C and C++, the maximum is 508. Otherwise, the maximum is 90.
- If PARAMETER STYLE JAVA or PARAMETER STYLE DB2GENERAL is specified, the maximum is 90.

The maximum number of parameters is also limited by the maximum number of parameters allowed by the licensed program used to compile the external program or service program.

**IN** Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned.<sup>66</sup>

### **OUT**

Identifies the parameter as an output parameter that is returned by the procedure.

A DataLink or a distinct type based on a DataLink may not be specified as an output parameter.

### **INOUT**

Identifies the parameter as both an input and output parameter for the procedure.

A DataLink or a distinct type based on a DataLink may not be specified as an input and output parameter.

### *parameter-name*

Names the parameter. The name cannot be the same as any other *parameter-name* for the procedure.

66. When the language type is REXX, all parameters must be input parameters.

## CREATE PROCEDURE (External)

### *data-type*

Specifies the data type of the parameter. The data type can be a built-in data type or a distinct type.

### *built-in-type*

Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE" on page 675. Some data types are not supported in all languages. For details on the mapping between the SQL data types and host language data types, see Embedded SQL Programming book. Built-in data type specifications can be specified if they correspond to the language that is used to write the procedure.

### *distinct-type-name*

Specifies a user-defined distinct type. The length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE DISTINCT TYPE). For more information on creating a distinct type, see "CREATE DISTINCT TYPE" on page 563.

If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

If a CCSID is specified, the parameter will be converted to that CCSID prior to passing it to the procedure. If a CCSID is not specified, the CCSID is determined by the default CCSID at the current server at the time the procedure is invoked.

### **AS LOCATOR**

Specifies that the parameter is a locator to the value rather than the actual value. You can specify AS LOCATOR only if the parameter has a LOB data type or a distinct type based on a LOB data type. If AS LOCATOR is specified, FOR SBCS DATA or FOR MIXED DATA must not be specified.

### **LANGUAGE**

Specifies the language that the external program or service program is written in. The language clause is required if the external program is a REXX procedure.

If LANGUAGE is not specified, the LANGUAGE is determined from the attribute information associated with the external program or service program at the time the procedure is created. If the attribute information associated with the program or service program does not identify a recognizable language or the program or service program cannot be found, then the language is assumed to be C.

**C** The external program is written in C.

### **C++**

The external program is written in C++.

### **CL**

The external program is written in CL.

### **COBOL**

The external program is written in COBOL.

### **COBOLLE**

The external program is written in ILE COBOL.

### **FORTRAN**

The external program is written in FORTRAN.

### JAVA

The external program is written in JAVA.

### PLI

The external program is written in PL/I.

### REXX

The external program is a REXX procedure.

### RPG

The external program is written in RPG.

### RPGLE

The external program is written in ILE RPG.

## PARAMETER STYLE

Specifies the conventions used for passing parameters to and returning the values from procedures:

### SQL

Specifies that in addition to the parameters on the CALL statement, several additional parameters are passed to the procedure. The parameters are defined to be in the following order:

- The first N parameters are the parameters that are specified on the CREATE PROCEDURE statement.
- N parameters for indicator variables for the parameters.
- A CHAR(5) output parameter for SQLSTATE. The SQLSTATE returned indicates the success or failure of the procedure. The SQLSTATE returned is assigned by the external program.

The user may set the SQLSTATE to any valid value in the external program to return an error or warning from the procedure.

- A VARCHAR(517) input parameter for the fully qualified procedure name.
- A VARCHAR(128) input parameter for the specific name.
- A VARCHAR(70) output parameter for the message text.

For more information about the parameters passed, see the include sqludf in the appropriate source file in library QSYSINC. For example, for C, sqludf can be found in QSYSINC/H.

PARAMETER STYLE SQL cannot be used with LANGUAGE JAVA.

### DB2GENERAL

Specifies that the procedure will use a parameter passing convention that is defined for use with Java methods.

PARAMETER STYLE DB2GENERAL can only be specified with LANGUAGE JAVA. For details on passing parameters in JAVA, see the IBM Developer Kit for Java.

### DB2SQL

Specifies that in addition to the parameters on the CALL statement, several additional parameters are passed to the procedure. DB2SQL is identical to the SQL parameter style, except that the following additional parameter may be passed as the last parameter:

- A parameter for the dbinfo structure, if DBINFO was specified on the CREATE PROCEDURE statement.

## CREATE PROCEDURE (External)

For more information about the parameters passed, see the include `sqludf` in the appropriate source file in library `QSYSINC`. For example, for `C`, `sqludf` can be found in `QSYSINC/H`.

PARAMETER STYLE `DB2SQL` cannot be used with `LANGUAGE JAVA`.

### GENERAL

Specifies that the procedure will use a parameter passing mechanism where the procedure receives the parameters specified on the `CALL`. Additional arguments are not passed for indicator variables.

PARAMETER STYLE `GENERAL` cannot be used with `LANGUAGE JAVA`.

### GENERAL WITH NULLS

Specifies that in addition to the parameters on the `CALL` statement as specified in `GENERAL`, another argument is passed to the procedure. This additional argument contains an indicator array with an element for each of the parameters of the `CALL` statement. In `C`, this would be an array of short `INTs`. For more information about how the indicators are handled, see the `SQL Programming` book.

PARAMETER STYLE `GENERAL WITH NULLS` cannot be used with `LANGUAGE JAVA`.

### JAVA

Specifies that the procedure will use a parameter passing convention that conforms to the Java language and `ISO/IEC FCD 9075-13:2003, Information technology - Database languages - SQL - Part 13: Java Routines and Types (SQL/JRT)` specification. `INOUT` and `OUT` parameters will be passed as single entry arrays to facilitate returning values.

PARAMETER STYLE `JAVA` can only be specified with `LANGUAGE JAVA`. For increased portability, you should write Java procedures that use the `PARAMETER STYLE JAVA` conventions. For details on passing parameters in `JAVA`, see the `IBM Developer Kit for Java` book.

Note that the language of the external procedure determines how the parameters are passed. For example, in `C`, any `VARCHAR` or `CHAR` parameters are passed as `NUL`-terminated strings. For more information, see the `SQL Programming` book. For Java routines, see the `IBM Developer Kit for Java`.

### EXTERNAL NAME *external-program-name*

Specifies the program or service program that will be executed when the procedure is called by the `CALL` statement. The program name must identify a program or service program that exists at the application server at the time the procedure is called. If the naming option is `*SYS` and the name is not qualified:

- The current path will be used to search for the program or service program at the time the procedure is called.
- `*LIBL` will be used to search for the program or service program at the time grants or revokes are performed on the procedure.

The validity of the name is checked at the application server. If the format of the name is not correct, an error is returned.

If *external-program-name* is not specified, the external program name is assumed to be the same as the procedure name.

The external program or service program need not exist at the time the procedure is created, but it must exist at the time the procedure is called.

CONNECT, SET CONNECTION, RELEASE, DISCONNECT, and SET TRANSACTION statements are not allowed in a procedure that is running on a remote application server. COMMIT and ROLLBACK statements are not allowed in an ATOMIC SQL procedure or in a procedure that is running on a connection to a remote application server.

### **DYNAMIC RESULT SETS** *integer*

Specifies the maximum number of result sets that can be returned from the procedure. *integer* must be greater than or equal to zero and less than 32768. If zero is specified, no result sets are returned. If the SET RESULT SETS statement is issued, the number of results returned is the minimum of the number of result sets specified on this keyword and the SET RESULT SETS statement. If the SET RESULT SETS statement specifies a number larger than the maximum number of result sets, a warning is returned. Note that any result sets from cursors that have a RETURN TO CLIENT attribute are included in the number of result sets of the outermost procedure.

The result sets are scrollable if a cursor is used to return a result set and the cursor is scrollable. If a cursor is used to return a result set, the result set starts with the current position. Thus, if 5 FETCH NEXT operations have been performed prior to returning from the procedure, the result set will start with the 6th row of the result set.

Result sets are only returned if both the following are true:

- the procedure is directly called or if the procedure is a RETURN TO CLIENT procedure and is indirectly called from ODBC, JDBC, OLE DB, .NET, the SQL Call Level Interface, or the iSeries Access Family Optimized SQL API, and
- the external program does not have an attribute of ACTGRP(\*NEW).

For more information about result sets see “SET RESULT SETS” on page 980.

### **SPECIFIC** *specific-name*

Provides a unique name for the procedure. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another procedure or procedure that exists at the current server. If unqualified, the implicit qualifier is the same as the qualifier of the procedure name. If qualified, the qualifier must be the same as the qualifier of the procedure name.

If *specific-name* is not specified, it is the same as the procedure name. If a function or procedure with that specific name already exists, a unique name is generated similar to the rules used to generate unique table names.

### **DETERMINISTIC** or **NOT DETERMINISTIC**

Specifies whether the procedure returns the same results each time the procedure is called with the same IN and INOUT arguments.

#### **NOT DETERMINISTIC**

The procedure may not return the same result each time the procedure is called with the same IN and INOUT arguments, even when the referenced data in the database has not changed.

#### **DETERMINISTIC**

The procedure always returns the same results each time the procedure is called with the same IN and INOUT arguments, provided the referenced data in the database has not changed.

### **CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA, or NO SQL**

Specifies which SQL statements, if any, may be executed in the procedure or

## CREATE PROCEDURE (External)

any routine called from this procedure. See Appendix B, “Characteristics of SQL statements,” on page 1075 for a detailed list of the SQL statements that can be executed under each data access indication.

### CONTAINS SQL

Specifies that SQL statements that neither read nor modify SQL data can be executed by the procedure.

### NO SQL

Specifies that the procedure cannot execute any SQL statements.

### READS SQL DATA

Specifies that SQL statements that do not modify SQL data can be included in the procedure.

### MODIFIES SQL DATA

Specifies that the procedure can execute any SQL statement except statements that are not supported in procedures.

### CALLED ON NULL INPUT

Specifies that the procedure is to be invoked, if any, or all, argument values are null, making the procedure responsible for testing for null argument values. The procedure can return a null or nonnull value.

### DISALLOW DEBUG MODE, ALLOW DEBUG MODE, or DISABLE DEBUG MODE

Indicates whether the procedure is created so it can be debugged by the Unified Debugger. If DEBUG MODE is not specified, the procedure will be created with the debug mode specified by the CURRENT DEBUG MODE special register.

DEBUG MODE can only be specified with LANGUAGE JAVA.

#### DISALLOW DEBUG MODE

The procedure cannot be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is DISALLOW, the procedure can be subsequently altered to change the debug mode attribute.

#### ALLOW DEBUG MODE

The procedure can be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is ALLOW, the procedure can be subsequently altered to change the debug mode attribute.

#### DISABLE DEBUG MODE

The procedure cannot be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is DISABLE, the procedure cannot be subsequently altered to change the debug mode attribute.

### FENCED or NOT FENCED

This parameter is allowed for compatibility with other products and is not used by DB2 UDB for iSeries.

### PROGRAM TYPE MAIN

Specifies that the procedure executes as a main routine.

### DBINFO

Specifies whether or not the procedure requires the database information be passed.

#### DBINFO

Specifies that the database manager should pass a structure containing status information to the procedure. Table 51 on page 649 contains a description of the DBINFO structure. Detailed information about the

DBINFO structure can be found in include sqludf in the appropriate source file in library QSYSINC. For example, for C, sqludf can be found in QSYSINC/H.

DBINFO is only allowed with PARAMETER STYLE DB2SQL.

Table 51. DBINFO fields

Field	Data Type	Description
Relational database	VARCHAR(128)	The name of the current server.
Authorization ID	VARCHAR(128)	The run-time authorization ID.
CCSID Information	INTEGER INTEGER INTEGER  INTEGER INTEGER INTEGER  INTEGER  CHAR(8)	The CCSID information of the job. Three sets of three CCSIDs are returned. The following information identifies the three CCSIDs in each set: <ul style="list-style-type: none"> <li>• SBCS CCSID</li> <li>• DBCS CCSID</li> <li>• Mixed CCSID</li> </ul> Following the three sets of CCSIDs is an integer that indicates which set of three sets of CCSIDs is applicable and eight bytes of reserved space. <p>If a CCSID is not explicitly specified for a parameter on the CREATE PROCEDURE statement, the input string is assumed to be encoded in the CCSID of the job at the time the procedure is executed. If the CCSID of the input string is not the same as the CCSID of the parameter, the input string passed to the external procedure will be converted before calling the external program.</p>
Target Column	VARCHAR(128)  VARCHAR(128) VARCHAR(128)	Not applicable for a call to a procedure.
Version and release	CHAR(8)	The version, release, and modification level of the database manager.
Platform	INTEGER	The server's platform type.

**NO DBINFO**

Specifies that the procedure does not require the database information to be passed.

**OLD SAVEPOINT LEVEL or NEW SAVEPOINT LEVEL**

Specifies whether a new savepoint level is to be created on entry to the procedure.

**OLD SAVEPOINT LEVEL**

A new savepoint level is not created. Any SAVEPOINT statements issued within the procedure with OLD SAVEPOINT LEVEL implicitly or explicitly specified on the SAVEPOINT statement are created at the same savepoint level as the caller of the procedure. This is the default.

**NEW SAVEPOINT LEVEL**

A new savepoint level is created on entry to the procedure. Any savepoints set within the procedure are created at a savepoint level that is nested deeper than the level at which this procedure was invoked. Therefore, the name of any new savepoint set within the procedure will not conflict with any existing savepoints set in higher savepoint levels (such as the savepoint level of the calling program or service program) with the same name.

## CREATE PROCEDURE (External)

### COMMIT ON RETURN

Specifies whether the database manager commits the transaction immediately on return from the procedure.

### NO

The database manager does not issue a commit when the procedure returns. NO is the default.

### YES

The database manager issues a commit if the procedure returns successfully. If the procedure returns with an error, a commit is not issued.

The commit operation includes the work that is performed by the calling application process and the procedure.<sup>67</sup>

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

## Notes

**General considerations for defining procedures:** See "CREATE PROCEDURE" on page 638 for general information on defining procedures.

**Language considerations:** For information needed to create the programs for a procedure, see the Embedded SQL Programming book.

**Owner privileges:** The owner is authorized to call the procedure (EXECUTE) and grant others the privilege to call the procedure. See "GRANT (Function or Procedure Privileges)" on page 858. For more information on ownership of the object, see "Authorization, privileges and object ownership" on page 17.

**Error handling considerations:** Values of arguments passed to a procedure which correspond to OUT parameters are undefined and those which correspond to INOUT parameters are unchanged when an error is returned by the procedure.

**Creating the procedure:** When an external procedure associated with an ILE external program or service program is created, an attempt is made to save the procedure's attributes in the associated program or service program object. If the \*PGM object is saved and then restored to this or another system, the catalogs are automatically updated with those attributes.

The attributes can be saved for external procedures subject to the following restrictions:

- The external program library must not be QSYS.
- The external program must exist when the CREATE PROCEDURE statement is issued.
- The external program must be an ILE \*PGM or \*SRVPGM object.

If the object cannot be updated, the procedure will still be created.

During restore of the procedure:

- If the specific name was specified when the procedure was originally created and it is not unique, an error is issued.

---

67. If the external program or service program was created with ACTGRP(\*NEW) and the job commitment definition is not used, the work that is performed in the procedure will be committed or rolled back as a result of the activation group ending.

## CREATE PROCEDURE (External)

- If the specific name was not specified, a unique name is generated if necessary.
- If the same procedure name and number of parameters already exists,
  - If the external program name or service program name is the same as the one registered in the catalog, the procedure information in the catalog will be replaced.
  - Otherwise, the procedure cannot be registered, and an error is issued.

**Invoking the procedure:** If a DECLARE PROCEDURE statement defines a procedure with the same name as a created procedure, and a static CALL statement where the procedure name is not identified by a variable is executed from the same source program, the attributes from the DECLARE PROCEDURE statement will be used rather than the attributes from the CREATE PROCEDURE statement.

The CREATE PROCEDURE statement applies to static and dynamic CALL statements as well as to a CALL statement where the procedure name is identified by a variable.

When an external procedure is invoked, it runs in whatever activation group was specified when the external program or service program was created. However, ACTGRP(\*CALLER) should normally be used so that the procedure runs in the same activation group as the calling program.

**Notes for Java procedures:** To be able to run Java procedures, you must have the IBM Developer Kit for Java installed on your system. Otherwise, an SQLCODE of -443 will be returned and a CPDB521 message will be placed in the job log.

If an error occurs while running a Java procedure, an SQLCODE of -443 will be returned. Depending on the error, other messages may exist in the job log of the job where the procedure was run.

**Syntax alternatives:** The following keywords are synonyms supported for compatibility to prior releases. These keywords are non-standard and should not be used:

- The keywords VARIANT and NOT VARIANT can be used as synonyms for NOT DETERMINISTIC and DETERMINISTIC.
- The keywords NULL CALL and NOT NULL CALL can be used as synonyms for CALLED ON NULL INPUT and RETURNS NULL ON NULL INPUT.
- The keywords SIMPLE CALL can be used as a synonym for GENERAL.
- The value DB2GENRL may be used as a synonym for DB2GENERAL.
- DYNAMIC RESULT SET, RESULT SETS, and RESULT SET may be used as synonyms for DYNAMIC RESULT SETS.
- The keywords PARAMETER STYLE in the PARAMETER STYLE clause are optional.

### Example

*Example 1:* Create the procedure definition for a procedure, written in Java, that is passed a part number and returns the cost of the part and the quantity that are currently available.

## CREATE PROCEDURE (External)

```
CREATE PROCEDURE PARTS_ON_HAND (IN PARTNUM INTEGER,  
                                OUT COST    DECIMAL(7,2),  
                                OUT QUANTITY INTEGER)  
  
LANGUAGE JAVA  
PARAMETER STYLE JAVA  
EXTERNAL NAME 'parts.onhand'
```

*Example 2:* Create the procedure definition for a procedure, written in C, that is passed an assembly number and returns the number of parts that make up the assembly, total part cost and a result set that lists the part numbers, quantity and unit cost of each part.

```
CREATE PROCEDURE ASSEMBLY_PARTS (IN ASSEMBLY_NUM INTEGER,  
                                 OUT NUM_PARTS   INTEGER,  
                                 OUT COST        DOUBLE)  
  
LANGUAGE C  
PARAMETER STYLE GENERAL  
DYNAMIC RESULT SETS 1  
FENCED  
EXTERNAL NAME ASSEMBLY
```

---

## CREATE PROCEDURE (SQL)

The CREATE PROCEDURE (SQL) statement creates an SQL procedure at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

The privileges held by the authorization ID of the statement must include at least one of the following:

- The privilege to create in the schema. For more information, see “Privileges necessary to create in a schema” on page 18.
- Administrative authority

The privileges held by the authorization ID of the statement must include at least one of the following:

- For the SYSPROCS catalog view and SYSPARMS catalog table:
  - The INSERT privilege on the table, and
  - The system authority \*EXECUTE on library QSYS2
- Administrative authority

The privileges held by the authorization ID of the statement must include at least one of the following:

- The following system authorities:
  - \*USE on the Create Program (CRTPGM) command, and
- Administrative authority

If SQL names are specified and a user profile exists that has the same name as the library into which the procedure is created, and that name is different from the authorization ID of the statement, then the privileges held by the authorization ID of the statement must include at least one of the following:

- The system authority \*ADD to the user profile with that name
- Administrative authority

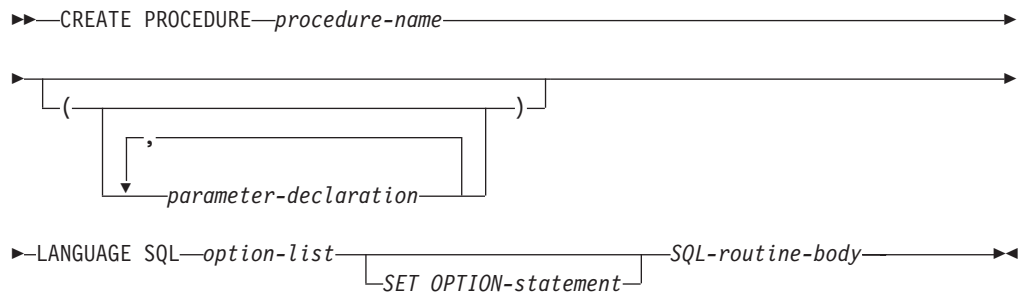
If a distinct type is referenced, the privileges held by the authorization ID of the statement must include at least one of the following:

- For each distinct type identified in the statement:
  - The USAGE privilege on the distinct type, and
  - The system authority \*EXECUTE on the library containing the distinct type
- Administrative authority

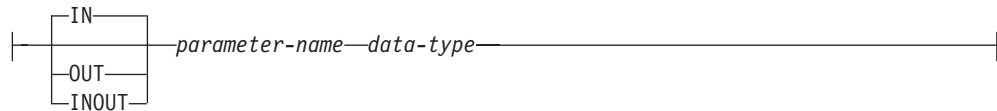
For information on the system authorities corresponding to SQL privileges, see “Corresponding System Authorities When Checking Privileges to a Function or Procedure” on page 864 and “Corresponding System Authorities When Checking Privileges to a Distinct Type” on page 856.

### Syntax

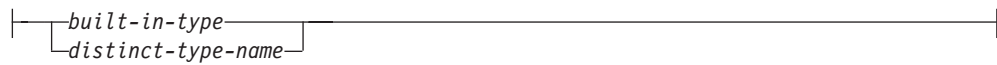
## CREATE PROCEDURE (SQL)



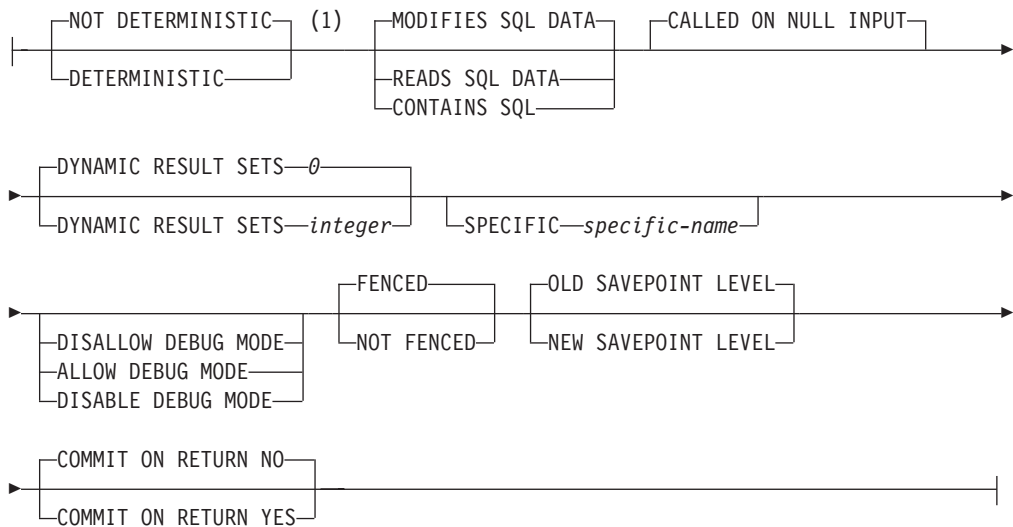
### parameter-declaration:



### data-type:



### option-list:



### Notes:

- 1 The optional clauses can be specified in a different order.

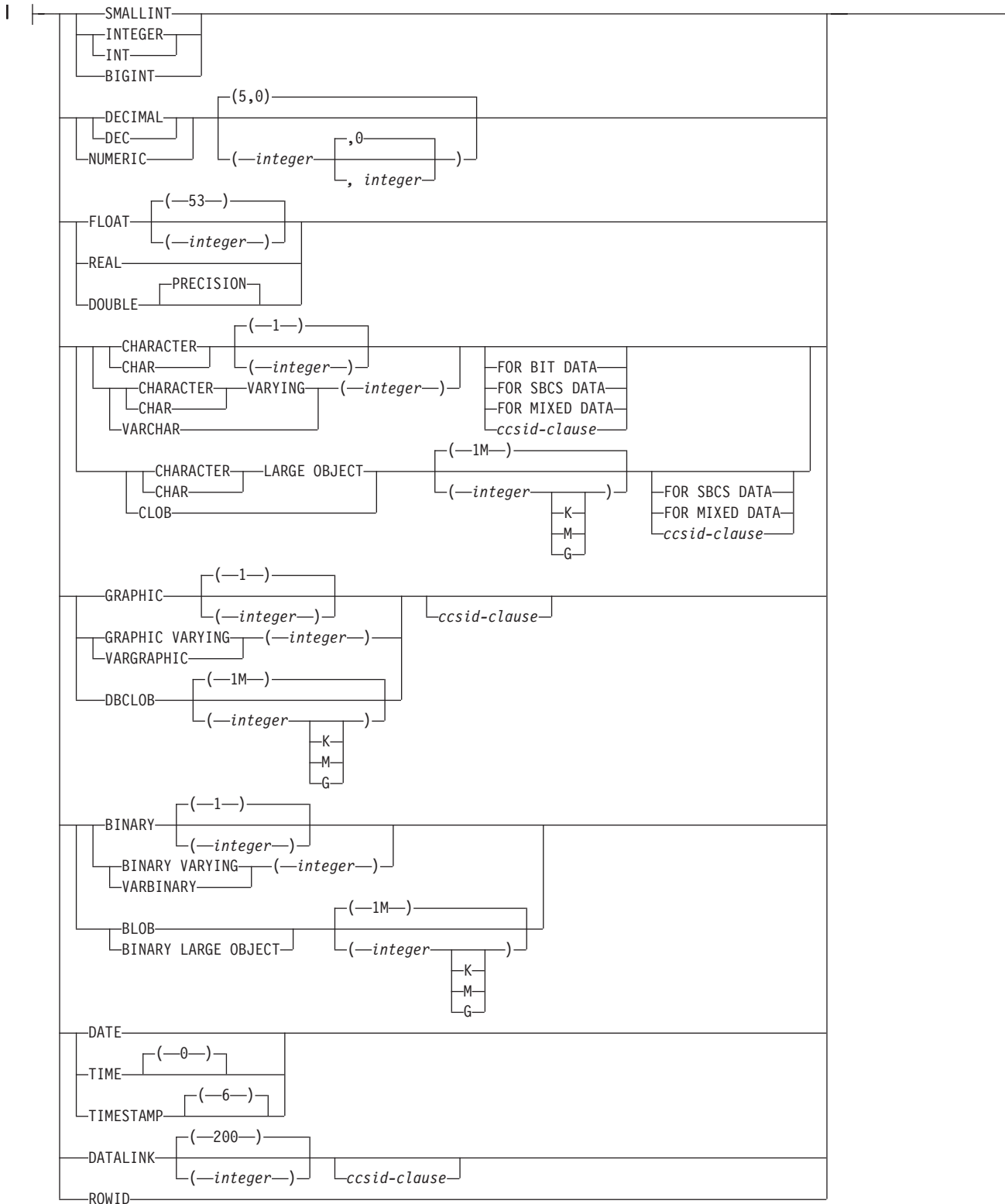
## SQL-routine-body:

I

<del>SQL-control-statement</del>
<del>ALLOCATE DESCRIPTOR-statement</del>
<del>ALTER PROCEDURE (External)-statement</del>
<del>ALTER SEQUENCE-statement</del>
<del>ALTER TABLE-statement</del>
<del>COMMENT-statement</del>
<del>COMMIT-statement</del>
<del>CONNECT-statement</del>
<del>CREATE ALIAS-statement</del>
<del>CREATE DISTINCT TYPE-statement</del>
<del>CREATE FUNCTION (External Scalar)-statement</del>
<del>CREATE FUNCTION (External Table)-statement</del>
<del>CREATE FUNCTION (Sourced)-statement</del>
<del>CREATE INDEX-statement</del>
<del>CREATE PROCEDURE (External)-statement</del>
<del>CREATE SCHEMA-statement</del>
<del>CREATE SEQUENCE-statement</del>
<del>CREATE TABLE-statement</del>
<del>CREATE VIEW-statement</del>
<del>DEALLOCATE DESCRIPTOR-statement</del>
<del>DECLARE GLOBAL TEMPORARY TABLE-statement</del>
<del>DELETE-statement</del>
<del>DESCRIBE-statement</del>
<del>DESCRIBE INPUT-statement</del>
<del>DESCRIBE TABLE-statement</del>
<del>DISCONNECT-statement</del>
<del>DROP-statement</del>
<del>EXECUTE IMMEDIATE-statement</del>
<del>GET DESCRIPTOR-statement</del>
<del>GRANT-statement</del>
<del>INSERT-statement</del>
<del>LABEL-statement</del>
<del>LOCK TABLE-statement</del>
<del>REFRESH TABLE-statement</del>
<del>RELEASE-statement</del>
<del>RELEASE SAVEPOINT-statement</del>
<del>RENAME-statement</del>
<del>REVOKE-statement</del>
<del>ROLLBACK-statement</del>
<del>SAVEPOINT-statement</del>
<del>SELECT INTO-statement</del>
<del>SET CONNECTION-statement</del>
<del>SET CURRENT DEBUG MODE-statement</del>
<del>SET CURRENT DEGREE-statement</del>
<del>SET DESCRIPTOR-statement</del>
<del>SET ENCRYPTION PASSWORD-statement</del>
<del>SET PATH-statement</del>
<del>SET RESULT SETS-statement</del>
<del>SET SCHEMA-statement</del>
<del>SET TRANSACTION-statement</del>
<del>UPDATE-statement</del>
<del>VALUES INTO-statement</del>

# CREATE PROCEDURE (SQL)

## built-in-type:



## ccsid-clause:



## Description

### *procedure-name*

Names the procedure. The combination of name, schema name, the number of parameters must not identify a procedure that exists at the current server.

For SQL naming, the procedure will be created in the schema specified by the implicit or explicit qualifier.

For system naming, the procedure will be created in the schema specified by the qualifier. If no qualifier is specified:

- If the value of the CURRENT SCHEMA special register is \*LIBL, the procedure will be created in the current library (\*CURLIB).
- Otherwise, the procedure will be created in the current schema.

### *(parameter-declaration,...)*

Specifies the number of parameters of the procedure and the data type of each parameter. A parameter for a procedure can be used only for input, only for output, or for both input and output. Although not required, you can give each parameter a name.

The maximum number of parameters allowed in an SQL procedure is 1024.

**IN** Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned.

### **OUT**

Identifies the parameter as an output parameter that is returned by the procedure. If the parameter is not set within the procedure, the null value is returned.

### **INOUT**

Identifies the parameter as both an input and output parameter for the procedure.

### *parameter-name*

Names the parameter. The name cannot be the same as any other *parameter-name* for the procedure.

### *data-type*

Specifies the data type of the parameter. The data type can be a built-in data type or a distinct data type.

### *built-in-type*

Specifies a built-in data type. For a more complete description of each built-in data type, see "CREATE TABLE" on page 675.

### *distinct-type-name*

Specifies a distinct type. The length, precision, or scale attributes for the parameter are those of the source type of the distinct type (those specified on CREATE DISTINCT TYPE). For more information on creating a distinct type, see "CREATE DISTINCT TYPE" on page 563.

If the name of the distinct type is unqualified, the database manager resolves the schema name by searching the schemas in the SQL path.

If a CCSID is specified, the parameter will be converted to that CCSID prior to passing it to the procedure. If a CCSID is not specified, the CCSID is determined by the default CCSID at the current server at the time the procedure is called.

## CREATE PROCEDURE (SQL)

### LANGUAGE SQL

Specifies that this is an SQL procedure.

### DYNAMIC RESULT SETS *integer*

Specifies the maximum number of result sets that can be returned from the procedure. *integer* must be greater than or equal to zero and less than 32768. If zero is specified, no result sets are returned. If the SET RESULT SETS statement is issued, the number of results returned is the minimum of the number of result sets specified on this keyword and the SET RESULT SETS statement. If the SET RESULT SETS statement specifies a number larger than the maximum number of result sets, a warning is returned. Note that any result sets from cursors that have a RETURN TO CLIENT attribute are included in the number of result sets of the outermost procedure.

The result sets are scrollable if the cursor is used to return a result set and the cursor is scrollable. If a cursor is used to return a result set, the result set starts with the current position. Thus, if 5 FETCH NEXT operations have been performed prior to returning from the procedure, the result set will start with the 6th row of the result set.

Result sets are only returned if the procedure is directly called or if the procedure is a RETURN TO CLIENT procedure and is indirectly called from ODBC, JDBC, OLE DB, .NET, the SQL Call Level Interface, or the iSeries Access Family Optimized SQL API. For more information about result sets, see "SET RESULT SETS" on page 980.

### SPECIFIC *specific-name*

Provides a unique name for the procedure. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another procedure or function that exists at the current server. If unqualified, the implicit qualifier is the same as the qualifier of the procedure name. If qualified, the qualifier must be the same as the qualifier of the procedure name.

If *specific-name* is not specified, it is the same as the procedure name. If a function or procedure with that specific name already exists, a unique name is generated similar to the rules used to generate unique table names.

### DETERMINISTIC or NOT DETERMINISTIC

Specifies whether the procedure returns the same results each time the procedure is called with the same IN and INOUT arguments.

#### NOT DETERMINISTIC

The procedure may not return the same result each time the procedure is called with the same IN and INOUT arguments, even when the referenced data in the database has not changed.

#### DETERMINISTIC

The procedure always returns the same results each time the procedure is called with the same IN and INOUT arguments, provided the referenced data in the database has not changed.

### CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA

Specifies which SQL statements may be executed in the procedure or any routine called from this procedure. See Appendix B, "Characteristics of SQL statements," on page 1075 for a detailed list of the SQL statements that can be executed under each data access indication.

### CONTAINS SQL

Specifies that SQL statements that neither read nor modify SQL data can be executed by the procedure.

**READS SQL DATA**

Specifies that SQL statements that do not modify SQL data can be included in the procedure.

**MODIFIES SQL DATA**

Specifies that the procedure can execute any SQL statement except statements that are not supported in procedures.

**CALLED ON NULL INPUT**

Specifies that the procedure is to be invoked, if any, or all, argument values are null, making the procedure responsible for testing for null argument values. The procedure can return a null or nonnull value.

**DISALLOW DEBUG MODE, ALLOW DEBUG MODE, or DISABLE DEBUG MODE**

Indicates whether the procedure is created so it can be debugged by the Unified Debugger. If DEBUG MODE is specified, a DBGVIEW option in the SET OPTION statement must not be specified.

**DISALLOW DEBUG MODE**

The procedure cannot be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is DISALLOW, the procedure can be subsequently altered to change the debug mode attribute.

**ALLOW DEBUG MODE**

The procedure can be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is ALLOW, the procedure can be subsequently altered to change the debug mode attribute.

**DISABLE DEBUG MODE**

The procedure cannot be debugged by the Unified Debugger. When the DEBUG MODE attribute of the procedure is DISABLE, the procedure cannot be subsequently altered to change the debug mode attribute.

If DEBUG MODE is not specified, but a DBGVIEW option in the SET OPTION statement is specified, the procedure cannot be debugged by the Unified Debugger, but may be debugged by the system debug facilities. If neither DEBUG MODE nor a DBGVIEW option is specified, the debug mode used is from the CURRENT DEBUG MODE special register.

**FENCED or NOT FENCED**

This parameter is allowed for compatibility with other products and is not used by DB2 UDB for iSeries.

**OLD SAVEPOINT LEVEL or NEW SAVEPOINT LEVEL**

Specifies whether a new savepoint level is to be created on entry to the procedure.

**OLD SAVEPOINT LEVEL**

A new savepoint level is not created. Any SAVEPOINT statements issued within the procedure with OLD SAVEPOINT LEVEL implicitly or explicitly specified on the SAVEPOINT statement are created at the same savepoint level as the caller of the procedure. This is the default.

**NEW SAVEPOINT LEVEL**

A new savepoint level is created on entry to the procedure. Any savepoints set within the procedure are created at a savepoint level that is nested deeper than the level at which this procedure was invoked. Therefore, the name of any new savepoint set within the procedure will not conflict with any existing savepoints set in higher savepoint levels (such as the savepoint level of the calling program) with the same name.

## CREATE PROCEDURE (SQL)

### COMMIT ON RETURN

Specifies whether the database manager commits the transaction immediately on return from the procedure.

#### NO

The database manager does not issue a commit when the procedure returns. NO is the default.

#### YES

The database manager issues a commit if the procedure returns successfully. If the procedure returns with an error, a commit is not issued.

The commit operation includes the work that is performed by the calling application process and the procedure.

If the procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

### SET OPTION-statement

Specifies the options that will be used to create the procedure. For example, to create a debuggable procedure, the following statement could be included:

```
SET OPTION DBGVIEW = *SOURCE
```

For more information, see "SET OPTION" on page 961.

The options CLOSQLCSR, CNULRQD, COMPILEOPT, NAMING, and SQLCA are not allowed in the CREATE PROCEDURE statement.

### SQL-routine-body

Specifies a single SQL statement, including a compound statement. See Chapter 6, "SQL control statements," on page 1013 for more information about defining SQL procedures.

CONNECT, SET CONNECTION, RELEASE, DISCONNECT, and SET TRANSACTION statements are not allowed in a procedure that is running on a remote application server. COMMIT and ROLLBACK statements are not allowed in an ATOMIC SQL procedure or in a procedure that is running on a connection to a remote application server.

## Notes

**General considerations for defining procedures:** See "CREATE PROCEDURE" on page 638 for general information on defining procedures.

**Procedure ownership:** If SQL names were specified:

- If a user profile with the same name as the schema into which the procedure is created exists, the *owner* of the procedure is that user profile.
- Otherwise, the *owner* of the procedure is the user profile or group user profile of the job executing the statement.

If system names were specified, the *owner* of the procedure is the user profile or group user profile of the job executing the statement.

**Procedure authority:** If SQL names are used, procedures are created with the system authority of \*EXCLUDE on \*PUBLIC. If system names are used, procedures are created with the authority to \*PUBLIC as determined by the create authority (CRTAUT) parameter of the schema.

If the owner of the procedure is a member of a group profile (GRPPRF keyword) and group authority is specified (GRPAUT keyword), that group profile will also have authority to the procedure.

**Error handling in procedures:** Consideration should be given to possible exceptions that can occur for each SQL statement in the body of a procedure. Any exception SQLSTATE that is not handled within the procedure using a handler within a compound statement, results in the exception SQLSTATE being returned to the caller of the procedure.

**Creating the procedure:** When an SQL procedure is created, SQL creates a temporary source file that will contain C source code with embedded SQL statements. A program object is then created using the CRTPGM command. The SQL options used to create the program are the options that are in effect at the time the CREATE PROCEDURE statement is executed. The program is created with ACTGRP(\*CALLER).

When an SQL procedure is created, the procedure's attributes are stored in the created program object. If the \*PGM object is saved and then restored to this or another system, the catalogs are automatically updated with those attributes.

During restore of the procedure:

- If the specific name was specified when the procedure was originally created and it is not unique, an error is issued.
- If the specific name was not specified, a unique name is generated if necessary.
- If the same procedure name and number of parameters already exists,
  - If the name of the created program is the same as the one registered in the catalog, the procedure information in the catalog will be replaced.
  - Otherwise, the procedure cannot be registered, and an error is issued.

| The specific procedure name is used as the name of the member in the source file  
| and the name of the program object, if it is a valid system name. If the procedure  
| name is not a valid system name, a unique name is generated. If a source file  
| member with the same name already exists, the member is overlaid. If a module or  
| a program with the same name already exists, the objects are not overlaid, and a  
| unique name is generated. The unique names are generated according to the rules  
| for generating system table names.

**Invoking the procedure:** If a DECLARE PROCEDURE statement defines a procedure with the same name as a created procedure, and a static CALL statement where the procedure name is not identified by a variable is executed from the same source program, the attributes from the DECLARE PROCEDURE statement will be used rather than the attributes from the CREATE PROCEDURE statement.

The CREATE PROCEDURE statement applies to static and dynamic CALL statements as well as to a CALL statement where the procedure name is identified by a variable.

SQL procedures must be called using the SQL CALL statement. When called, the SQL procedure runs in the activation group of the calling program.

**Syntax alternatives:** The following keywords are synonyms supported for compatibility to prior releases. These keywords are non-standard and should not be used:

## CREATE PROCEDURE (SQL)

- The keywords VARIANT and NOT VARIANT can be used as synonyms for NOT DETERMINISTIC and DETERMINISTIC.
- The keywords NULL CALL and NOT NULL CALL can be used as synonyms for CALLED ON NULL INPUT and RETURNS NULL ON NULL INPUT.
- DYNAMIC RESULT SET, RESULT SETS, and RESULT SET may be used as synonyms for DYNAMIC RESULT SETS.

### Example

Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

```
CREATE PROCEDURE MEDIAN_RESULT_SET (OUT medianSalary DECIMAL(7,2))
LANGUAGE SQL
DYNAMIC RESULT SETS 1
BEGIN
  DECLARE v_numRecords INTEGER DEFAULT 1;
  DECLARE v_counter INTEGER DEFAULT 0;
  DECLARE c1 CURSOR FOR
    SELECT salary
    FROM staff
    ORDER BY salary;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, salary
    FROM staff
    WHERE salary > medianSalary
    ORDER BY salary;
  DECLARE EXIT HANDLER FOR NOT FOUND
    SET medianSalary = 6666;
  SET medianSalary = 0;
  SELECT COUNT(*) INTO v_numRecords FROM STAFF;
  OPEN c1;
  WHILE v_counter < (v_numRecords / 2 + 1)
    DO FETCH c1 INTO medianSalary;
    SET v_counter = v_counter + 1;
  END WHILE;
  CLOSE c1;
  OPEN c2;
END
```

## SET OPTION

The SET OPTION statement establishes the processing options to be used for SQL statements.

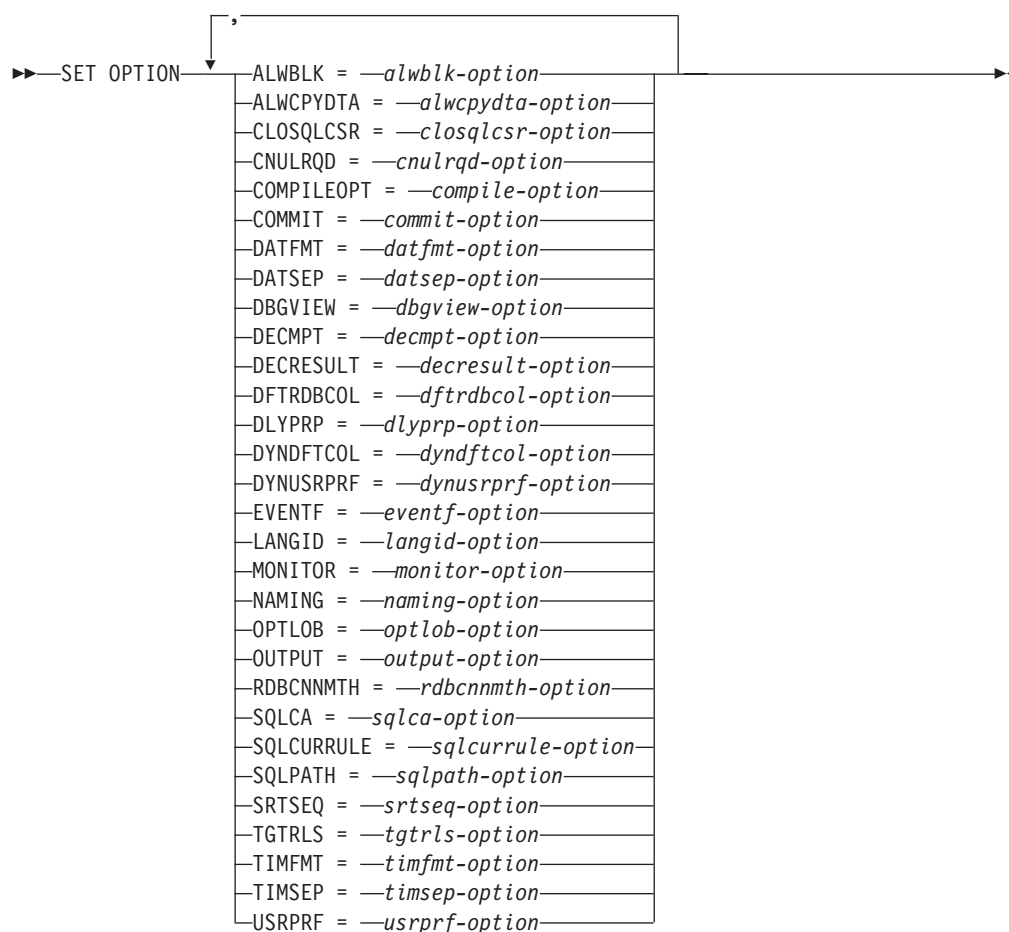
### Invocation

This statement can be used in a REXX procedure or embedded in an application program. If used in a REXX procedure, it is an executable statement. If embedded in an application program, it is not executable and must precede any other SQL statements. This statement cannot be dynamically prepared.

### Authorization

None required.

### Syntax



#### alwblk-option:



## SET OPTION

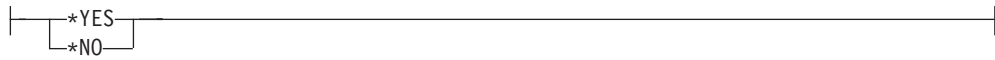
### alwcpydta-option:



### closqlcsr-option:



### cnulrqd-option:



### commit-option:



### compile-option:



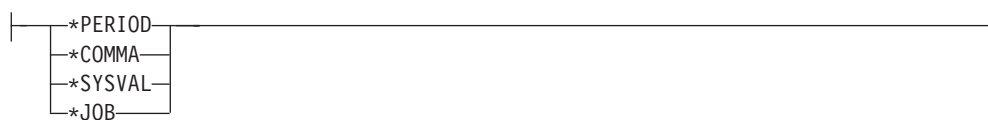
### datfmt-option:



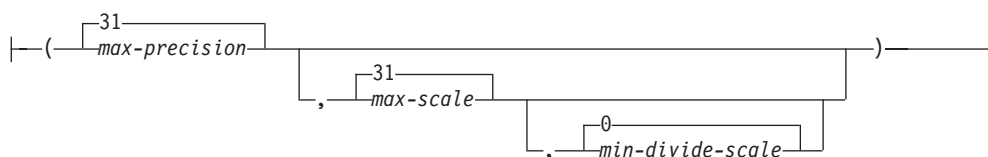
### datsep-option:



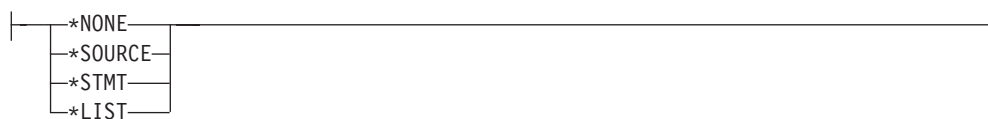
**decmnt-option:**



**decresult-option:**



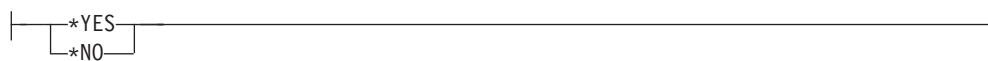
**dbgview-option:**



**dftrdbcol-option:**



**dlyprp-option:**



**dyndftcol-option:**



## SET OPTION

### **dynusrprf-option:**



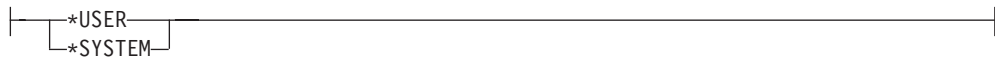
### **eventf-option:**



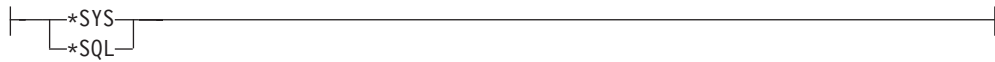
### **langid-option:**



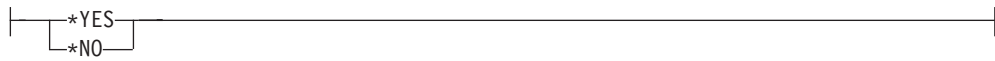
### **monitor-option:**



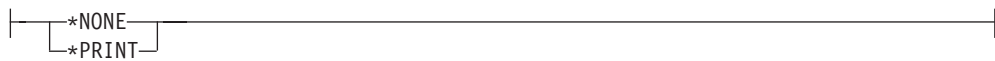
### **naming-option:**



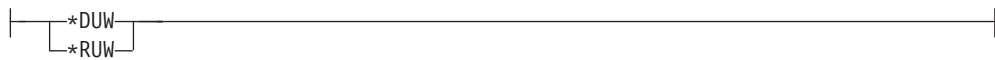
### **optlob-option:**



### **output-option:**



### **rdbcnmth-option:**



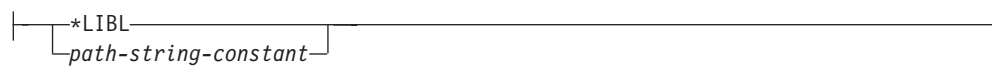
### **sqlca-option:**



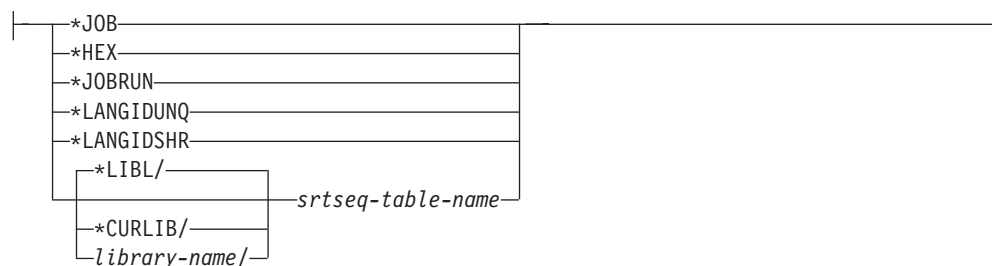
**sqlcurrule-option:**



**sqlpath-option:**



**srtseq-option:**



**tgtrls-option:**



**timfmt-option:**



**timsep-option:**



**usrprf-option:**



## Description

### ALWBLK

Specifies whether the database manager can use row blocking and the extent to which blocking can be used for read-only cursors. This option will be ignored in REXX.

### \*ALLREAD

Rows are blocked for read-only cursors if COMMIT is \*NONE or \*CHG. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program.

Specifying \*ALLREAD:

- Allows row blocking under commitment control level \*CHG in addition to the blocking allowed for \*READ.
- Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways:
  - The Rollback (ROLLBACK) command, a ROLLBACK statement in host languages, or the ROLLBACK HOLD SQL statement does not reposition a read-only cursor when:
    - ALWBLK(\*ALLREAD) was specified when the program or routine that contains the cursor was created
    - ALWBLK(\*READ) and ALWCPYDTA(\*OPTIMIZE) were specified when the program or routine that contains the cursor was created
  - Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE), cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

### \*NONE

Rows are not blocked for retrieval of data for cursors.

Specifying \*NONE:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

### \*READ

Rows are blocked for read-only retrieval of data for cursors when:

- \*NONE is specified on the COMMIT parameter, which indicates that commitment control is not used.
- The cursor is declared with a FOR READ ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

Specifying \*READ can improve the overall performance of queries that meet the above conditions and retrieve a large number of rows.

### ALWCPYDTA

Specifies whether a copy of the data can be used in a SELECT statement. This option will be ignored in REXX.

**\*OPTIMIZE**

The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If COMMIT is \*CHG or \*CS and ALWBLK is not \*ALLREAD, or if COMMIT is \*ALL or \*RR, then a copy of the data is used only when it is necessary to run a query.

**\*YES**

A copy of the data is used only when necessary.

**\*NO**

A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

**CLOSQLCSR**

Specifies when SQL cursors are implicitly closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released. SQL cursors are explicitly closed when you issue the CLOSE, COMMIT, or ROLLBACK (without HOLD) SQL statements. This option will be ignored in REXX. \*ENDACTGRP and \*ENDMOD are for use by ILE programs and modules. \*ENDPGM, \*ENDSQL, and \*ENDJOB are for use by non-ILE programs.

This option is not allowed in an SQL function, SQL procedure, or SQL trigger.

**\*ENDACTGRP**

SQL cursors are closed, SQL prepared statements are implicitly discarded, and LOCK TABLE locks are released when the activation group ends.

**\*ENDMOD**

SQL cursors are closed and SQL prepared statements are implicitly discarded when the module is exited. LOCK TABLE locks are released when the first SQL program on the call stack ends.

**\*ENDPGM**

SQL cursors are closed and SQL prepared statements are discarded when the program ends. LOCK TABLE locks are released when the first SQL program on the call stack ends.

**\*ENDSQL**

SQL cursors remain open between calls and can be fetched without running another SQL OPEN. One of the programs higher on the call stack must have run at least one SQL statement. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the first SQL program on the call stack ends. If \*ENDSQL is specified for a program that is the first SQL program called (the first SQL program on the call stack), the program is treated as if \*ENDPGM was specified.

**\*ENDJOB**

SQL cursors remain open between calls and can be fetched without running another SQL OPEN. The programs higher on the call stack do not need to have run SQL statements. SQL cursors are left open, SQL prepared statements are preserved, and LOCK TABLE locks are held when the first SQL program on the call stack ends. SQL cursors are closed, SQL prepared statements are discarded, and LOCK TABLE locks are released when the job ends.

**CNULRQD**

Specifies whether a NUL-terminator is returned for character and graphic host variables. This option will only be used for SQL statements in C and C++ programs.

## SET OPTION

This option is not allowed in an SQL function, SQL procedure, or SQL trigger.

### **\*YES**

Output character and graphic host variables always contain the NUL-terminator. If there is not enough space for the NUL-terminator, the data is truncated and the NUL-terminator is added. Input character and graphic host variables require a NUL-terminator.

### **\*NO**

For output character and graphic host variables, the NUL-terminator is not returned when the host variable is exactly the same length as the data. Input character and graphic host variables do not require a NUL-terminator.

## **COMMIT**

Specifies the isolation level to be used. In REXX, files that are referred to in the source are not affected by this option. Only tables, views, and packages referred to in SQL statements are affected. For more information about isolation levels, see "Isolation level" on page 25

### **\*CHG**

Specifies the isolation level of Uncommitted Read.

### **\*NONE**

Specifies the isolation level of No Commit. If the DROP SCHEMA statement is included in a REXX procedure, \*NONE must be used.

### **\*CS**

Specifies the isolation level of Cursor Stability.

### **\*ALL**

Specifies the isolation level of Read Stability.

### **\*RR**

Specifies the isolation level of Repeatable Read.

## **COMPILEOPT**

Specifies additional parameters to be used on the compiler command. The COMPILEOPT string is added to the compiler command built by the precompiler. If 'INCDIR(' is anywhere in the string, the precompiler will call the compiler using the SRCSTMF parameter. The contents of the string is not validated. The compiler command will issue an error if any parameter is incorrect. Using any of the keywords that the precompiler passes to the compiler will cause the compiler command to fail because of duplicate parameters. Refer to the Embedded SQL Programming information for a list of parameters that the precompiler generates for the compiler command. This option will be ignored in REXX.

This option is not allowed in an SQL function, SQL procedure, or SQL trigger.

### **\*NONE**

No additional parameters will be used on the compiler command.

### *character-string*

A character constant of no more than 5000 characters containing the compiler options.

## **DATFMT**

Specifies the format used when accessing date result columns. All output date fields are returned in the specified format. For input date strings, the specified value is used to determine whether the date is specified in a valid format.

**Note:** An input date string that uses the format \*USA, \*ISO, \*EUR, or \*JIS is always valid.

**\*JOB:**

The format specified for the job is used. Use the Display Job (DSPJOB) command to determine the current date format for the job.

**\*ISO**

The International Organization for Standardization (ISO) date format (yyyy-mm-dd) is used.

**\*EUR**

The European date format (dd.mm.yyyy) is used.

**\*USA**

The United States date format (mm/dd/yyyy) is used.

**\*JIS**

The Japanese Industrial Standard date format (yyyy-mm-dd) is used.

**\*MDY**

The date format (mm/dd/yy) is used.

**\*DMY**

The date format (dd/mm/yy) is used.

**\*YMD**

The date format (yy/mm/dd) is used.

**\*JUL**

The Julian date format (yy/ddd) is used.

**DATSEP**

Specifies the separator used when accessing date result columns.

**Note:** This parameter applies only when \*JOB, \*MDY, \*DMY, \*YMD, or \*JUL is specified on the DATFMT parameter.

**\*JOB**

The date separator specified for the job is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

**\*SLASH** or '/'

A slash (/) is used.

**\*PERIOD** or '.'

A period (.) is used.

**\*COMMA** or ','

A comma (,) is used.

**\*DASH** or '-'

A dash (-) is used.

**\*BLANK** or ' '

A blank ( ) is used.

**DBGVIEW**

Specifies whether the object can be debugged by the system debug facilities and the type of debug information to be provided by the compiler. The DBGVIEW parameter can only be specified in the body of SQL functions, procedures, and triggers.

|  
|  
|  
|

## SET OPTION

If `DEBUG MODE` in a `CREATE PROCEDURE` or `ALTER PROCEDURE` statement is specified, a `DBGVIEW` option in the `SET OPTION` statement must not be specified.

The possible choices are:

### **\*NONE**

A debug view will not be generated.

### **\*SOURCE**

Allows the compiled module object to be debugged using SQL statement source. If `*SOURCE` is specified, the modified source is stored in source file `QSQDSRC` in the same schema as the created function, procedure, or trigger.

### **\*STMT**

Allows the compiled module object to be debugged using program statement numbers and symbolic identifiers.

### **\*LIST**

Generates the listing view for debugging the compiled module object.

If `DEBUG MODE` is not specified, but a `DBGVIEW` option in the `SET OPTION` statement is specified, the procedure cannot be debugged by the Unified Debugger, but can be debugged by the system debug facilities. If neither `DEBUG MODE` nor a `DBGVIEW` option is specified, the debug mode used is from the `CURRENT DEBUG MODE` special register.

## **DECMPT**

Specifies the symbol that you want to represent the decimal point. The possible choices are:

### **\*PERIOD**

The representation for the decimal point is a period.

### **\*COMMA**

The representation for the decimal point is a comma.

### **\*SYSVAL**

The representation for the decimal point is the system value (`QDECFMT`).

### **\*JOB**

The representation for the decimal point is the job value (`DECFMT`).

## **DECRESULT**

Specifies the maximum precision, maximum scale, and minimum divide scale that should be used during decimal operations, such as decimal arithmetic. The specified limits only apply to `NUMERIC` and `DECIMAL` data types.

### *max-precision*

An integer constant that is the maximum precision that should be returned from decimal operations. The value can be 31 or 63. The default is 31.

### *max-scale*

An integer constant that is the maximum scale that should be returned from decimal operations. The value can range from 0 to the maximum precision. The default is 31.

### *min-divide-scale*

An integer constant that is the minimum scale that should be returned from division operations. The value can range from 0 to the maximum scale. The default is 0.

**DFTRDBCOL**

Specifies the schema name used for the unqualified names of tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements. This option will be ignored in REXX.

**\*NONE**

The naming convention specified on the OPTION precompile parameter or by the SET OPTION NAMING option will be used.

*schema-name*

Specify the name of the schema. This value is used instead of the naming convention specified on the OPTION precompile parameter or by the SET OPTION NAMING option.

**DLYPRP**

Specifies whether the dynamic statement validation for a PREPARE statement is delayed until an OPEN, EXECUTE, or DESCRIBE statement is run. Delaying validation improves performance by eliminating redundant validation. This option will be ignored in REXX.

**\*NO**

Dynamic statement validation is not delayed. When the dynamic statement is prepared, the access plan is validated. When the dynamic statement is used in an OPEN or EXECUTE statement, the access plan is revalidated. Because the authority or the existence of objects referred to by the dynamic statement may change, you must still check the SQLCODE or SQLSTATE after issuing the OPEN or EXECUTE statement to ensure that the dynamic statement is still valid.

**\*YES**

Dynamic statement validation is delayed until the dynamic statement is used in an OPEN, EXECUTE, or DESCRIBE SQL statement. When the dynamic statement is used, the validation is completed and an access plan is built. If you specify \*YES, you should check the SQLCODE and SQLSTATE after running an OPEN, EXECUTE, or DESCRIBE statement to ensure that the dynamic statement is valid.

**Note:** If you specify \*YES, performance is not improved if the INTO clause is used on the PREPARE statement or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

**DYNDFTCOL**

Specifies the schema name specified for the DFTRDBCOL parameter is also used for dynamic statements. This option will be ignored in REXX.

**\*NO**

Do not use the value specified for DFTRDBCOL for unqualified names of tables, views, indexes, and SQL packages for dynamic SQL statements. The naming convention specified on the OPTION precompile parameter or by the SET OPTION NAMING option will be used.

**\*YES**

The schema name specified for DFTRDBCOL will be used for the unqualified names of the tables, views, indexes, and SQL packages in dynamic SQL statements.

**DYNUSRPRF**

Specifies the user profile to be used for dynamic SQL statements. This option will be ignored in REXX.

## SET OPTION

### **\*USER**

Local dynamic SQL statements are run under the user profile of the job. Distributed dynamic SQL statements are run under the user profile of the application server job.

### **\*OWNER**

Local dynamic SQL statements are run under the user profile of the program's owner. Distributed dynamic SQL statements are run under the user profile of the SQL package's owner.

### **EVENTF**

Specifies whether an event file will be generated. CoOperative Development Environment/400 (CODE/400) uses the event file to provide error feedback integrated with the CODE/400 editor.

### **\*YES**

The compiler produces an event file for use by CoOperative Development Environment/400 (CODE/400).

### **\*NO**

The compiler will not produce an event file for use by CoOperative Development Environment/400 (CODE/400).

### **LANGID**

Specifies the language identifier to be used when SRTSEQ(\*LANGIDUNQ) or SRTSEQ(\*LANGIDSHR) is specified.

### **\*JOB** or **\*JOBRUN**

The LANGID value for the job is used.

For distributed applications, LANGID(\*JOBRUN) is valid only when SRTSEQ(\*JOBRUN) is also specified.

### *language-id*

Specify a language identifier to be used. For information on the values that can be used for the language identifier, see the Language identifier topic in the iSeries Information Center.

### **MONITOR**

Specifies whether the statements should be identified as user or system statements when a database monitor is run.

### **\*USER**

The SQL statements are identified as user statements. This is the default.

### **\*SYSTEM**

The SQL statements are identified as system statements.

### **NAMING**

Specifies whether the SQL naming convention or the system naming convention is to be used. This option is not allowed in an SQL function, SQL procedure, or SQL trigger.

The possible choices are:

### **\*SYS**

The system naming convention will be used.

### **\*SQL**

The SQL naming convention will be used.

### **OPTLOB**

Specifies whether accesses to LOBs can be optimized when accessing through DRDA. The possible choices are:

**\*YES**

LOB accesses should be optimized. The first FETCH for a cursor determines how the cursor will be used for LOBs on all subsequent FETCHes. This option remains in effect until the cursor is closed.

If the first FETCH uses a LOB locator to access a LOB column, no subsequent FETCH for that cursor can fetch that LOB column into a LOB variable.

If the first FETCH places the LOB column into a LOB variable, no subsequent FETCH for that cursor can use a LOB locator for that column.

**\*NO**

LOB accesses should not be optimized. There is no restriction on whether a column is retrieved into a LOB locator or into a LOB variable. This option can cause performance to degrade.

**OUTPUT**

Specifies whether the precompiler and compiler listings are generated. The OUTPUT parameter can only be specified in the body of SQL functions, procedures, and triggers. The possible choices are:

**\*NONE**

The precompiler and compiler listings are not generated.

**\*PRINT**

The precompiler and compiler listings are generated.

**RDBCNNMTH**

Specifies the semantics used for CONNECT statements. This option will be ignored in REXX.

**\*DUW**

CONNECT (Type 2) semantics are used to support distributed unit of work. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections.

**\*RUW**

CONNECT (Type 1) semantics are used to support remote unit of work. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.

**SQLCA**

Specifies whether the fields in an SQLCA will be set after each SQL statement. The SQLCA option is only allowed for ILE C, ILE C++, ILE COBOL, and ILE RPG. This option is not allowed in an SQL function, SQL procedure, or SQL trigger.

The possible choices are:

**\*YES**

The fields in an SQLCA will be set after each SQL statement. The user program can reference all the values in the SQLCA following the execution of an SQL statement.

**\*NO**

The fields in an SQLCA will not be set after each SQL statement. The user program should use the GET DIAGNOSTICS statement to retrieve information about the execution of the SQL statement.

SQLCA(\*NO) will typically perform better than SQLCA(\*YES).

## SET OPTION

| In other host languages, an SQLCA is required and fields in the SQLCA will be  
| set after each SQL statement.

### SQLCURRULE

Specifies the semantics used for SQL statements.

#### \*DB2

The semantics of all SQL statements will default to the rules established for DB2. The following semantics are controlled by this option:

- Hexadecimal constants are treated as character data.

#### \*STD

The semantics of all SQL statements will default to the rules established by the ISO and ANSI SQL standards. The following semantics are controlled by this option:

- Hexadecimal constants are treated as binary data.

### SQLPATH

Specifies the path to be used to find procedures, functions, and user defined types in static SQL statements. This option will be ignored in REXX.

#### \*LIBL

The path used is the library list at runtime.

*character-string*

A character constant with one or more schema names that are separated by commas.

### SRTSEQ

Specifies the sort sequence table to be used for string comparisons in SQL statements.

**Note:** \*HEX must be specified if a REXX procedure connects to an application server that is not a DB2 UDB for iSeries or an iSeries system whose release level is prior to V2R3M0.

#### \*JOB or \*JOB RUN

The SRTSEQ value for the job is used.

#### \*HEX

A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.

#### \*LANGIDUNQ

The sort sequence table must contain a unique weight for each character in the code page.

#### \*LANGIDSHR

The shared-weight sort table for the LANGID specified is used.

*srtseq-table-name*

Specify the name of the sort sequence table to be used with this program. The name of the sort sequence table can be qualified by one of the following library values:

#### \*LIBL

All libraries in the user and system portions of the job's library list are searched until the first match is found.

#### \*CURLIB

The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.

*library-name*

Specify the name of the library to be searched.

### TGTRLS

Specifies the release of the operating system on which the user intends to use the object being created. The TGTRLS parameter can only be specified in the body of SQL functions, procedures, and triggers. The possible choices are:

#### VxRxMx

Specify the release in the format VxRxMx, where Vx is the version, Rx is the release, and Mx is the modification level. For example, V5R4M0 is version 5, release 4, modification level 0. The object can be used on a system with the specified release or with any subsequent release of the operating system installed.

Valid values depend on the current version, release, and modification level, and they change with each new release. If you specify a release-level which is earlier than the earliest release level supported by the database manager, an error message is sent indicating the earliest supported release.

The TGTRLS option can only be specified for SQL functions, SQL procedures, and triggers.

### TIMFMT

Specifies the format used when accessing time result columns. All output time fields are returned in the specified format. For input time strings, the specified value is used to determine whether the time is specified in a valid format.

**Note:** An input time string that uses the format \*USA, \*ISO, \*EUR, or \*JIS is always valid.

#### \*HMS

The (hh:mm:ss) format is used.

#### \*ISO

The International Organization for Standardization (ISO) time format (hh.mm.ss) is used.

#### \*EUR

The European time format (hh.mm.ss) is used.

#### \*USA

The United States time format (hh:mm xx) is used, where xx is AM or PM.

#### \*JIS

The Japanese Industrial Standard time format (hh:mm:ss) is used.

### TIMSEP

Specifies the separator used when accessing time result columns.

**Note:** This parameter applies only when \*HMS is specified on the TIMFMT parameter.

#### \*JOB

The time separator specified for the job is used. Use the Display Job (DSPJOB) command to determine the current value for the job.

#### \*COLON or ':'

A colon (:) is used.

#### \*PERIOD or '.'

A period (.) is used.

## SET OPTION

**\*COMMA** or **'**  
A comma (,) is used.

**\*BLANK** or **' '**  
A blank ( ) is used.

### USRPRF

Specifies the user profile that is used when the compiled program object is run, including the authority that the program object has for each object in static SQL statements. The profile of either the program owner or the program user is used to control which objects can be used by the program object. This option will be ignored in REXX.

### \*NAMING

The user profile is determined by the naming convention. If the naming convention is \*SQL, USRPRF(\*OWNER) is used. If the naming convention is \*SYS, USRPRF(\*USER) is used.

### \*USER

The profile of the user running the program object is used.

### \*OWNER

The user profiles of both the program owner and the program user are used when the program is run.

## Notes

**Default values:** The default option values depend on the language, object type, and the options in effect at create time:

- When an SQL procedure, SQL function, or SQL trigger is created, the default options are those in effect at the time the object is created. For example, if an SQL procedure is created and the current COMMIT option is \*CS, \*CS is the default COMMIT option. Each default option is then updated as it is encountered within the SET OPTION statement.
- For application programs other than REXX, the default options values specified on the CRTSQLxxx command. Each option is updated as it is encountered within a SET OPTION statement. All SET OPTION statements must precede any other embedded SQL statements.
- At the start of a REXX procedure the options are set to their default value. The default value for each option is the first value listed in the syntax diagram. When an option is changed by a SET OPTION statement, the new value will stay in effect until the option is changed again or the REXX procedure ends.

**Syntax alternatives:** The following keywords are synonyms supported for compatibility to prior releases. These keywords are non-standard and should not be used:

- \*UR can be used as a synonym for \*CHG.
- \*NC can be used as a synonym for \*NONE.
- \*RS can be used as a synonym for \*ALL.

## Examples

*Example 1:* Set the isolation level to \*ALL and the naming mode to SQL names.

```
EXEC SQL SET OPTION COMMIT =*ALL, NAMING =*SQL
```

*Example 2:* Set the date format to European, the isolation level to \*CS, and the decimal point to the comma.

```
EXEC SQL SET OPTION DATFMT = *EUR, COMMIT = *CS, DECMPT = *COMMA
```



---

# Part 5

Copyright 2008 System i Developer, LLC

## Overview

---

- **Part 1**
  - SQL Functions for Data manipulation and Analysis
  - Summarizing Data with the SELECT Statement
  - Other Interesting Stuff
- **Part 2**
  - Working with Edit Characters
  - Subselect - Basic to Advanced
  - Identifying Potential Duplicate Rows
  - Searching by Sound
- **Part 3**
  - Embedding SQL in RPG (and Cobol) Programs
- **Part 4**
  - Stored Procedures
  - SQL Procedure Language
- **Part 5**
  - SQL Triggers and Other Trigger Enhancements in V5

Copyright 2008 System i Developer, LLC

## Overview Part 5

---

### SQL Triggers and Other Trigger Enhancements in V5

- V5R1 Trigger Enhancements
- Read Triggers - Good or Bad News?
- More Than One Trigger per Database Event
- Named Triggers
- CHGPFTRG - Change Physical File Trigger
- System Catalog Trigger Enhancements
- SQL Triggers
- SQL Triggers Summary



Copyright 2008 System i Developer, LLC

## What is a Trigger?

---

### Trigger Definition

- **An action or set of actions that can be automatically invoked whenever a change is made to a physical file or table**
  - Insert of a new record
  - Update of an existing record
  - Delete of an existing record
  - Read of an existing record (New for V5R1)
    - You really don't want to use Read Triggers!
- **An attribute added to a physical file or table**
  - Can be displayed with DSPFD
  - DSPFD FILE( DEPT-MSTR ) TYPE( \*TRG )



Copyright 2008 System i Developer, LLC

# Uses for Triggers

---

## Where to Use triggers

- **Global enforcement of business rules**
  - Business rules defined at database level and NOT coded in every application program
  - Faster application development and more efficient program maintenance
- **Validation of data**
- **Generation of audit trail**
- **Invocation of special functions**



Copyright 2008 System i Developer, LLC

# OS/400 Support for Triggers

---

## V5R1 - Two Types of Triggers

- **External Triggers**
  - First available with DB2/400 in V3R1
  - Originally referred to as Native Triggers
  - Called System Triggers in V5R1
    - Poor name - conveys incorrect meaning
- **SQL Triggers**
  - New in V5R1



Copyright 2008 System i Developer, LLC

## OS/400 Support for Triggers...

---

### V4R5 and Prior Releases of OS/400

- **External triggers only (aka Native or System triggers)**
- **Two commands**
  - ADDPFTRG
  - RMVPFTRG
- **External triggers can be added or removed from a file using the Database function in Operations Navigator**
- **A file can have a total of 6 triggers**
  - BEFORE trigger event:
    - 1 Insert, 1 Update, 1 Delete ( 3 total )
  - AFTER trigger event:
    - 1 Insert, 1 Update, 1 Delete ( 3 total )
- **No triggers on catalog files or tables**



Copyright 2008 System i Developer, LLC

## OS/400 Support for Triggers...

---

### V5R1 and Subsequent Releases of OS/400

- **External and SQL Triggers**
- **Commands for External Triggers**
  - ADDPFTRG
  - RMVPFTRG
- **SQL Statements for SQL Triggers**
  - CREATE TRIGGER
  - DROP TRIGGER
- **External and SQL Triggers can be added or removed from a file using the Database function in Operations Navigator**
- **Maximum of 300 triggers per physical file or table**
- **No triggers on catalog files or tables**

Copyright 2008 System i Developer, LLC

# V5R1 Trigger Enhancements

---

## New Trigger Capabilities for DB2 UDB for iSeries

- **More than 1 trigger per database event**
  - Maximum of 300 triggers per physical file or table
  - Triggers for the same event fired in order created
  - Identified or qualified by Trigger Name
- **CHGPFTRG command**
  - Disable an active or enabled trigger
  - Enable an inactive or disabled trigger
- **New files for triggers in System Catalog**
- **SQL Triggers**
  - Column Level
  - Row Level
  - Statement Level



Copyright 2008 System i Developer, LLC

# V5R1 Trigger Enhancements...

---

## Read Triggers

- **Read Trigger fires EVERY time a record is read from a file opened for Read only**
  - Assumes Read Trigger is defined for the file

This is **NOT** good news!

- **\*READ trigger event**
  - Only available with External Triggers
  - **NOT** available with SQL Triggers



Copyright 2008 System i Developer, LLC

## Read Triggers

---

Improper use of Read Triggers will have a **SEVERE** detrimental effect on your job and/or career!

- **Translation: Read Triggers have a very specific purpose**
  - They are *NOT intended for general use*
  - Do not use them without a detailed database design and performance review by a person having an in-depth knowledge of DB2 UDB for iSeries and AS/400!



Copyright 2008 System i Developer, LLC

## Read Triggers...

---

**Do I Have Your Undivided  
Attention?!**



Copyright 2008 System i Developer, LLC

## Read Triggers...

---

### Why Read Triggers and Why You Should NOT Use Them!

- **Read Triggers have been implemented in OS/400 to satisfy a US Federal statute (HIPPA) requiring that any and all access to patient records in the medical profession be logged and auditable**
- **Use of Read Triggers will alter iSeries and AS/400 I/O and performance characteristics, severely degrading system performance and throughput**
- **Any read access to a file with a Read Trigger - either user program or system function - will cause trigger to fire**
  - CPYF - Copy file
  - DSPPFM - Display physical file member
  - Use of query products
  - User written display or report programs

Copyright 2008 System i Developer, LLC

## Read Triggers...

---

### Read Trigger Impact on Performance and Thruput

- **Read Triggers disable some system functions that optimize performance and throughput**
- **Sequential processing (arrival sequence)**
  - No adaptive blocking and double buffering of I/O
  - Records read and written one at a time
- **Keyed sequential process (sequence by key or index)**
  - No adaptive asynchronous fetch of next record in key sequence
  - Records read and written one at a time
- **Batch programs using above file processing will run orders of magnitude longer!**

Copyright 2008 System i Developer, LLC

## Read Triggers...

---

### Read Triggers Require Substantial System Resource!

- **If you like playing 'You Bet Your Job' and living dangerously, and want to use Read Triggers, you must increase system resources**
  - ▶ Upgrade processor power (by at least a factor of 2?)
    - Offset increase in CPU cycles required by millions of trigger program invocations caused by Read Triggers
  - ▶ Add more memory
    - Offset increase in page fault rates
  - ▶ Add more disk arms
    - Offset increase in disk I/Os



Copyright 2008 System i Developer, LLC

## Read Triggers...

---

***Do not use Read Triggers*** without a detailed database design and performance review by a person having an in-depth knowledge of and experience with DB2 on iSeries and AS/400!

**Any Questions?**



Copyright 2008 System i Developer, LLC

## **More Than 1 Trigger per Database Event**

---

### **Problem: Limit of 6 triggers per file**

- **Software providers now including triggers in their application packages**
- **iSeries and AS/400 users may also have trigger requirements on the same files**
- **Considerations with limit of 6 triggers per file**
  - **Conflicting requirements for triggers of same time and event**
  - **Difficulties/problems combining trigger programs**

### **Solution: Increase Limit to 300 triggers per file in V5R1**

- **Multiple triggers with same trigger time and trigger event now allowed on a file**

Copyright 2008 System i Developer, LLC

## **More Than 1 Trigger per Database...**

---

### **Triggers for the Same Event Fired in Order Created**

- **Considerations when deleting and recreating a trigger**
  - **Remove first trigger on execution list**
  - **Recreate trigger**
  - **Trigger now at bottom of execution list**
  - **Potential problem??**
- **To maintain trigger firing order for a specific file**
  - **Two SQL scripts or CL programs to:**
    - **Remove all triggers for a file**
    - **Recreate all triggers for a file in desired execution sequence**

Copyright 2008 System i Developer, LLC

# Named Triggers

---

## Pre V5R1 Trigger Identification or Qualification

- **Unique identification per *File* provided via combination of:**
  - Trigger time
  - Trigger event
- **No longer provides unique identification with increase to 300 triggers per file**
- **New mechanism needed for unique trigger identification**



Copyright 2008 System i Developer, LLC

# Named Triggers...

---

## V5R1 - Trigger Name Used to Identify Triggers

- **Provides distinction between triggers with the same trigger time and trigger event**
- **Trigger Name and Library parameters provided on**
  - **CL commands**
    - ADDPFTRG
    - RMVPFTRG
  - **SQL statements (more on these later)**
    - CREATE TRIGGER
    - DROP TRIGGER



Copyright 2008 System i Developer, LLC

## Named Triggers...

---

### Trigger Name

- **Must be unique within a *Library***
- **Maximum length is 128 characters**
- **If not provided at trigger creation time, default name will be created by database**
  - You won't like the default name!



Copyright 2008 System i Developer, LLC

## Named Triggers...

---

### Trigger Name

- **External Triggers**
  - Trigger name and library only referenced in system catalog
  - Database will assign a default trigger name when migrating V4 and prior triggers to V5R1
    - Default trigger name obtuse and not meaningful
    - Manually recreate triggers in V5R1 to assign meaningful trigger name
- **SQL Triggers**
  - Trigger name and library become the name and location of the trigger program generated from the SQL statements in the trigger body
  - More on trigger body later



Copyright 2008 System i Developer, LLC

# CHGPFTRG Command

---

## Disabling and Enabling a Trigger

- **CHGPFTRG command changes the state of one or all triggers defined for a physical file or table**
- **Trigger states**
  - Disabled
    - Trigger program will *NOT* be invoked when trigger event is satisfied
  - Enabled
    - Trigger program will be invoked when trigger event is satisfied
- **Eliminates the hassle of removing a trigger from a file to disable it, and then recreating the same trigger again to enable it**

Copyright 2008 System i Developer, LLC

# CHGPFTRG Command...

---

## CHGPFTRG - Change Physical File Trigger

```

>>--CHGPFTRG----->
      +-*LIBL/-----+
>---FILE(--+-----+---physical-file-name-)----->
      |-*CURLIB/-----|
      +-library-name/--+
>---TRG(--+*ALL-----+)->
      +-trigger-name--+

```

Continued on next page

Copyright 2008 System i Developer, LLC

## CHGPFTRG Command...

### CHGPFTRG - Change Physical File Trigger

```

>>+-----+-----+
|          +-*ALL-----+          |
+-TRGLIB( -+-*CURLIB-----+-)---+
|          +-library-name--+
|
>-----+-----+-----+-----<
|          +-*SAME-----+          |
+-STATE( -+-*ENABLED-----+-)---+
|          +-*DISABLED--+

```

Copyright 2008 System i Developer, LLC

## System Catalog Trigger Enhancements

### Trigger Files Available in System Catalog

- **SYSTRIGGERS**
  - Contains one row for each trigger in a library
- **SYSTRIGCOL**
  - Contains one row for each column either implicitly or explicitly referenced in the WHEN clause or the SQL statements for an SQL Trigger
- **SYSTRIGDEP**
  - Contains one row for each object referenced in the WHEN clause or the SQL statements for an SQL Trigger
- **SYSTRIGUPD**
  - Contains one row for each column identified in the UPDATE column list, if any

Copyright 2008 System i Developer, LLC

# SQL Triggers

---

## Superset of DB2 UDB V7.1 SQL Triggers

- **Provides an industry standard method for defining and managing triggers that has a high degree of portability**
- **Adds SQL procedural logic to trigger implementation**
  - CASE
  - IF test
  - WHILE looping
  - etc
- **More granularity and function than external triggers**
  - Column (field) level triggers
  - Row (record) level triggers
  - (SQL) Statement level triggers
- **Installation of DB2 UDB SQL development kit required on development system**

Copyright 2008 System i Developer, LLC

# Trigger Components

---

## External Trigger Components - V5R1

1. **Base file or table**
2. **Trigger name**
3. **Trigger event**
4. **Trigger time**
5. **Trigger program**

Copyright 2008 System i Developer, LLC



# Trigger Components...

---

## SQL Trigger Components - V5R1

1. **Base file or table**
2. **Trigger name**
3. **Trigger event**
4. **Trigger time**
5. **Trigger granularity**
6. **Transition variables**
7. **Transition tables**
8. **Trigger mode**
9. **Triggered action**



Copyright 2008 System i Developer, LLC

# SQL Trigger Components

---

## Parts that Make Up an SQL Trigger

1. **Base file or table**
  - Physical file or table which the trigger is added to
2. **Trigger name**
  - Provides unique trigger identification within a library
3. **Trigger event**
  - The condition that causes the trigger to fire
    - Insert of a new row
    - Update of existing
      - Row
      - Column (Column level triggers)
    - Delete of an existing record
4. **Trigger time**
  - When trigger program is to be run
  - Before or after the trigger event



Copyright 2008 System i Developer, LLC

# SQL Trigger Components...

---

## 5. Trigger Granularity

- **Column level triggers**

- ▶ Extension of UPDATE trigger event
- ▶ Columns listed as part of UPDATE trigger event
- ▶ UPDATE OF column\_name\_1, column\_name\_2,...
  - Only update of a listed column causes trigger to fire
  - If no columns listed, update to any column in the row causes trigger to fire



Copyright 2008 System i Developer, LLC

# SQL Trigger Components...

---

## 5. Trigger Granularity

- **Row level triggers**

- ▶ FOR EACH ROW
  - Triggered action executed for each row satisfying trigger condition
  - If trigger condition never satisfied, triggered action never executed

- **Statement level triggers**

- ▶ FOR EACH STATEMENT
  - Triggered action executed only once for the event causing the trigger to fire irregardless of the number of rows processed
  - If trigger condition never satisfied, triggered action executed once at end of statement processing
- ▶ Not valid with Before triggers or Trigger Mode of DB2ROW

Copyright 2008 System i Developer, LLC

## SQL Trigger Components...

---

### 6. Transition Variables

- **aka Correlation Variables**
- **Provides function similar to before and after images in trigger buffer for external triggers**
- **Qualification of column names for the single row image before and/or after the trigger event has completed**
  - OLD ROW - Before image of row
  - NEW ROW - After image of row
  - REFERENCING OLD ROW AS oldrow REFERENCING NEW ROW AS newrow
  - ...newrow.salary > oldrow.salary + 10000...
- **Not valid with Statement level triggers**

Copyright 2008 System i Developer, LLC

## SQL Trigger Components...

---

### 7. Transition Tables

- **Provides function similar to before and after images in trigger buffer for external triggers**
- **A single SQL statement can process multiple rows**
- **Temporary tables that contain the image of all affected rows before and/or after the trigger event completes**
  - OLD TABLE - Before image of all affected rows
  - NEW TABLE - After image of all affected rows
  - REFERENCING OLD TABLE AS oldtbl
  - ...(SELECT COUNT(\*) FROM oldtbl)...
- **Not valid with Before triggers or Trigger Mode of DB2ROW**

Copyright 2008 System i Developer, LLC

## SQL Trigger Components...

---

### 8. Trigger Mode

- **MODE DB2ROW**

- ▶ Trigger fires after each row operation
- ▶ Only valid with Row level triggers
- ▶ Exclusive function of DB2 UDB for iSeries and AS/400
  - Not available in other DB2 UDB implementations

Copyright 2008 System i Developer, LLC

## SQL Trigger Components...

---

### 8. Trigger Mode

- **MODE DB2SQL**

- ▶ Trigger fires after all row operations are complete
- ▶ If specified on a row level trigger, triggered action executed N times after all row operations completed
  - N = number of rows processed
- ▶ Not as efficient as DB2ROW since each row is processed twice
- ▶ Only valid with After triggers

Copyright 2008 System i Developer, LLC

# SQL Trigger Components...

---

## 9. Triggered Action

- Analogous to trigger program in external triggers
- Three parts
  1. SET OPTION
    - Specifies the options that will be used to create the trigger
  2. WHEN
    - Search condition or execution criteria for Trigger Body
    - Specifies when the SQL statements in Trigger Body will be executed
  3. SQL Trigger Body
    - Single SQL statement
    - Multiple SQL statements delineated with BEGIN and END

Copyright 2008 System i Developer, LLC

## Trigger Body

---

### SQL Statements Available for Trigger Body

- BEGIN and END
- DECLARE (local variables)
- SET (local variables)
- Comments
- CASE (two forms), END CASE
- IF, THEN, ELSE, END IF
- FOR, END FOR, LOOP, END LOOP
- LEAVE (loop or block)
- REPEAT, END REPEAT
- WHILE, END WHILE
- DECLARE CONDITION
- DECLARE HANDLER
- SIGNAL, RESIGNAL



Copyright 2008 System i Developer, LLC

## Trigger Body...

---

### SQL Statements Available for Trigger Body

- **GET DIAGNOSTICS**
  - Provide access to SQLCA-like information
- **CALL**
  - SQL procedure
  - External procedure to access HLL programs or OS/400 APIs
- **Normal DDL and DML**
  - CREATE
  - INSERT
  - DELETE
  - etc



Copyright 2008 System i Developer, LLC

## Trigger Body

---

### SQL Control Statements

- **BEGIN and END**
- **DECLARE (local variables)**
- **SET (local variables)**
- **Comments**
- **CASE (two forms), END CASE**
- **IF, THEN, ELSE, END IF**
- **FOR, END FOR, LOOP, END LOOP**
- **LEAVE (loop or block)**
- **REPEAT, END REPEAT**
- **WHILE, END WHILE**
- **CALL**
  - SQL procedure
  - External procedure to access
    - HLL programs
    - OS/400 APIs



Copyright 2008 System i Developer, LLC

## Trigger Body...

---

### SQL Control Statements for Error Handling

- **DECLARE CONDITION**
- **DECLARE HANDLER**
- **SIGNAL, RESIGNAL**
- **GET DIAGNOSTICS**
  - Provide access to SQLCA-like information



Copyright 2008 System i Developer, LLC

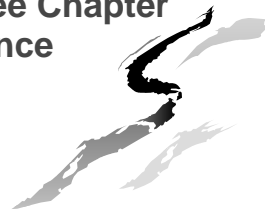
## Trigger Body...

---

### Other SQL Statements

- **Used in conjunction with SQL Control Statements**
- **Normal DDL and DML Statements listed under SQL procedure statements**
  - SELECT
  - UPDATE
  - INSERT
  - DELETE
  - CREATE
  - etc

**For detailed Control Statement information see Chapter 6: *SQL Control Statements* in the SQL Reference manual**



Copyright 2008 System i Developer, LLC

# CREATE TRIGGER Statement

---

## CREATE TRIGGER - Simplified Syntax

```

>>--CREATE TRIGGER---trigger-name---trigger-time----->
>---trigger-event---ON---base-table----->
>---transition-variables-and/or-transition-tables----->
>---trigger-granularity---trigger-mode---triggered-action--<

```

Copyright 2008 System i Developer, LLC

# CREATE TRIGGER Statement...

---

## Syntax for Trigger Name, Time, Event & Base Table

```

>>--CREATE TRIGGER--trigger-name--++-----+BEFORE+-->
                                     +-AFTER-----+
>--+--INSERT-----+--ON--table-name----->
|  |--DELETE-----|
|  +-UPDATE-----|
|          +- , -<-----|
|          +-OF--column-name-+--+

```

Continued on next page

Copyright 2008 System i Developer, LLC





# CREATE and DROP TRIGGER

---

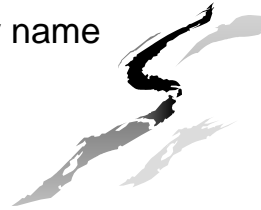
## Consideration for CREATE and DROP TRIGGER

- **CREATE TRIGGER**

- ▶ Base table can be qualified with library name
- ▶ Trigger name can be qualified with library name
- ▶ Triggers created with implicit attribute of
  - ALWREPCHG(\*YES)

- **DROP TRIGGER**

- ▶ Trigger name can be qualified with library name



Copyright 2008 System i Developer, LLC

# SQL Trigger Examples

---

## Row Level Trigger with Simple Trigger Body

```
CREATE TRIGGER audit_spending
  AFTER UPDATE ON expenses
  REFERENCING NEW ROW AS nw
  FOR EACH ROW MODE DB2ROW
  WHEN (nw.total_amount > 10000)
  INSERT INTO travel_audit
    VALUES(nw.empno, nw.deptno, nw.total_amount, nw.end_date);
```



Copyright 2008 System i Developer, LLC

# SQL Trigger Examples

---

## Column Level Trigger with Simple Trigger Body

```
CREATE TRIGGER empsal
  BEFORE UPDATE OF salary ON emp
  REFERENCING NEW AS nw OLD AS od
  FOR EACH ROW MODE DB2ROW
  WHEN (nw.salary > 1.5 * od.salary)
    SET nw.salary = 1.5 * od.salary;
```



Copyright 2008 System i Developer, LLC

# SQL Trigger Examples...

---

## Row Level Trigger with Complex Trigger Body

```
CREATE TRIGGER big_spenders
  AFTER INSERT ON expenses
  REFERENCING NEW ROW AS n
  FOR EACH ROW
  MODE DB2ROW
  WHEN (n.totalamount > 10000)
  BEGIN
    DECLARE empname CHAR(30);
    SET empname = (SELECT lname FROM employee
                  WHERE empid = n.empno);
    INSERT INTO travel_audit
      VALUES(n.empno, empname, n.deptno, n.totalamount, n.enddate);
  END
```

Copyright 2008 System i Developer, LLC

# SQL Trigger Examples

---

## Statement Level Trigger with Complex Trigger Body

```
CREATE TRIGGER account_trigger
  AFTER INSERT ON accounts
  REFERENCING NEW TABLE AS newtable
  FOR EACH STATEMENT MODE DB2SQL
BEGIN
  DECLARE chgcnt INT;
  SET chgcnt = (SELECT count(*) FROM newtable);
  INSERT INTO account_changes
    VALUES('I', CURRENT TIMESTAMP, CURRENT USER, chgcnt);
END
```



Copyright 2008 System i Developer, LLC

# SQL Trigger Considerations

---

## Requirements for Transition Variables & Tables

- **Transition Variables**
  - Trigger Granularity must be FOR EACH ROW
  - Trigger Mode can be either DB2ROW or DB2SQL
  - Trigger Time can be either BEFORE or AFTER
- **Transition Tables**
  - Trigger Granularity can be either FOR EACH ROW or FOR EACH STATEMENT
  - Trigger Mode must be DB2SQL
  - Trigger Time must be AFTER



Copyright 2008 System i Developer, LLC

## SQL Trigger Considerations...

---

### Availability of Transition Variables & Tables

- **Delete trigger**
  - Only OLD transition variables and tables
- **Insert trigger**
  - Only NEW transition variables and tables
- **Update trigger**
  - Both OLD and NEW transition variables and tables



Copyright 2008 System i Developer, LLC

## SQL Trigger Considerations...

---

### Updating Data Before It is Written to a Table

- **Trigger Time must be BEFORE**
  - AFTER not allowed
- **Use SET statement to alter appropriate NEW ROW transition variables**
  - Generate missing values
  - Data cleansing
  - Special functions
- **SQL statements INSERT, UPDATE, and DELETE not allowed to operate on base table**
- **Other ways to change NEW ROW transition variables**
  - SELECT INTO
  - Stored procedure output variable (OUT or INOUT)

Copyright 2008 System i Developer, LLC

## SQL Trigger Considerations...

---

### Inoperative Triggers

- **A trigger becomes inoperative if**
  - CRTDUPOBJ is used to duplicate the base table and
    - SQL statements in the triggered action reference the base table and/or
    - SQL statements in the triggered action reference objects in the FROM library and these objects do not exist in the TO library
  - Base table is restored to a new library and
    - SQL statements in the triggered action reference the base table and/or
    - SQL statements in the triggered action reference objects in the SAVE library and these objects do not exist in the RESTORE library
- **When trigger is inoperative**
  - Triggered action can no longer be executed
  - INSERT, UPDATE, or DELETE not allowed on base table
- **Drop trigger and then recreate to resolve**

Copyright 2008 System i Developer, LLC

## SQL Trigger Considerations...

---

### Other Considerations

- **Not allowed to rename or move a table referenced in a triggered action**
  - Drop trigger and recreate to resolve
- **A new column added to base table, or table referenced in a triggered action, is not available to the trigger until the trigger is recreated**
  - A new column will cause an UPDATE trigger to fire if the column is modified and there is no column list (column level trigger) specified for the trigger
- **Installation of DB2 UDB SQL development kit required on development system**



Copyright 2008 System i Developer, LLC

# SQL Triggers Summary

---

## Significant New Trigger Functions in V5R1

- **Read Triggers - Approach with Extreme Caution!**
- **300 Triggers per Physical File or Table**
- **A Trigger Now Has a Name**
- **Disable or Enable a Trigger with CHGPFTRG**
- **Files with Trigger information Added to System Catalog**
- **SQL Triggers Provide Enhanced Capability**



Copyright 2008 System i Developer, LLC