RPG User Defined Functions (UDFs)

and Table Functions (UDTFs)

Presented by

Scott Klement

http://www.scottklement.com

© 2009-2011, Scott Klement

"There are 10 types of people in the world.
Those who understand binary, and those who don't."

Objectives Of This Session

- What is a User defined function (UDF)?
- Why would I consider using one?
- UDF info and examples
- What is a User defined table function (UDTF)?
- UDTF info and examples
- Links to more information

Although UDFs can be written in many languages, including SQL itself, this talk will focus on writing them in RPG.

2

SQL Routines

SQL Supports the ability to write routines, callable as an SQL statement, or as part of a larger SQL statement.

- Procedures ("Stored Procedures")
  - To be used via the CALL SQL command.
  - Input/Output parameters and result sets.
  - "Subroutine" for SQL.

- Triggers
  - Fire automatically when an event happens.

- Functions *(…this is the one I'll talk about …)*
  - Called as part of a select, insert, update, etc.
  - Take input as parameters.
  - Return output as a return value – or a table ("file").

What's a UDF (1 of 2)

A UDF is a function that takes input parameters, and returns an output (variable or table) – you can write them in SQL or in "external" languages like RPG, Cobol, C, CL, Java and more…

Think: Calling an RPG subprocedure from SQL!

```
Select empno, lastname, firstname, midinit, hiredate
  from EMPMAST
 order by hiredate
```

This query can read a file, but what about calculated data?

- File has date in YYYYMMDD format, but you need it in MM/DD/YYYY format
- File has the date hired, but you need to know the years of service.

What's a UDF (2 of 2)

```
Select empno, lastname, firstname, midinit,
       toMdy(hiredate), yearsActive(hiredate,termdate)
  from EMPMAST
 order by hiredate
```

toMdy() calls an RPG subprocedure, passes hiredate as a parameter.
➢ RPG routine outputs the date in MM/DD/YYYY format.

yearsActive() calls an RPG subprocedure, passes hiredate as a parameter
➢ RPG routine figures out how many years since that date, and returns it.

Output of select will contain the data from the RPG program

5

Why Use a UDF?

They are useful, because they:

• Simplify SQL statements

• Let you re-use existing RPG business logic

• Easy plumbing for app integration across a network

• There are things RPG can do that SQL cannot

6

SQL Can't Do Everything, Right?

SQL (by itself) can't:

- Check if a (non-file) object exists
- Read a data area
- Parse XML
- Work with the IFS
- Write HTML to a browser
- Send a text message to your cell phone (or for that matter, to QSYSOPR)

SQL and RPG make a great team. RPG is right for some things, and SQL is right for some things!

Like What?!

7

But, My #1 Reason Is….

UDFs let me write business logic in RPG, and use it anywhere.
any place I can run an SQL statement.

- Windows apps (.NET, VB, C++, etc)
- PHP apps (IBM i, Linux, Windows)
- Java apps (anywhere)
- ILE RPG, Cobol, C
- Even OPM
- Even Microsoft Office!

WIN

I'll take RPG UDFS for the win, please!

8

Start With an RPG Subprocedure

…it can also be a program, but I find that less intuitive.

You can have multiple procedures in the module if you like.

```
P toMdy          B                    export
D toMdy          PI          10a
D   ymd                       8p 0 const

D retval         s          10a
 /free
    monitor;
      retval = %char( %date(ymd:*iso) : *USA );
    on-error;
      retval = 'ERROR';
    endmon;

    return retval;
 /end-free
P                E
```

Compile & Bind Into a SRVPGM

Build a service program from the module:

```
CRTRPGMOD MODULE(mylib/UDFDEMO)        (or PDM #15)
          SRCFILE(*libl/QRPGLESRC)


CRTSRVPGM SRVPGM(mylib/UDFDEMO1)
```

You now have routines that are callable from ILE languages – but how do you call them from elsewhere?

Create An SQL Function

```
Create Function toMdy ( ymd Decimal(8,0) )
               returns char(10)
               language rpgle
               deterministic
               no sql
               external name 'mylib/UDFDEMO1(TOMDY)'
               parameter style general
```

Think of "Create Function" as SQL's version of a prototype. It gives SQL all of the details needed to call your subprocedure.

This is an SQL statement.
You can run it from any place that SQL is available…   STRSQL, IBM i Navigator, RUNSQLSTM, from your code, etc. It doesn't matter.

Personally, I like putting them into a source member, and running with RUNSQLSTM. Then you can repeat it when you need to.

11

What Does Create Function Do?

* The "prototype" information needed to call your routine is saved into the database.

* No disk object is produced. (It's not like a compiler.)

* Info about parameters and how to call is saved into files in QSYS2 library

    File **SYSROUTINES** = info about SQL routines
    File **SYSFUNCS** = info about SQL functions

  The **Drop Function** SQL statement removes these definitions from the files.

12

Quick Create Function Syntax Overview

```
Create Function toMdy ( ymd Decimal(8,0) )
                 returns char(10)
    . . .
```

✓toMdy = function name (name you use in SQL statements)
✓ymd = parameter name. Decimal(8,0) is the data type
✓Returns char(10) describes the return value

```
    . . .
              language rpgle
              deterministic
              no sql
              external name 'mylib/UDFDEMO1(TOMDY)'
              parameter style general
```

Programming language to call -- controls how parms are passed.

Routine to call – name of *SRVPGM object and subprocedure.

✓Options that control how your routine gets called.
   (more info coming up…)

Contains SQL?

Remember: Your UDF is run from an SQL statement.
➢ *If your RPG uses SQL, it's a statement inside another statement!*

SQL can handle this, but it needs to know how you plan to use SQL

- **NO SQL** *(fastest)*
     Doesn't use any SQL whatsoever.   If it tries to use SQL, an error will occur.

- **CONTAINS SQL**
     Can use only a very restrictive number of SQL statements that neither read nor update files. (such as COMMIT/ROLLBACK or SET with various variables.)

- **READS SQL DATA**
     Reads data, such as a SELECT or FETCH statements. No updates allowed.

- **MODIFIES SQL DATA** *(slowest)*
     All SQL statements allowed, including INSERT, UPDATE, CREATE TABLE, etc.

If you specify too little access for what you're doing, you'll get an error. If you're not sure what level is required, see Appendix B of the SQL Reference manual, it lists all SQL statements and which one of the above is required.

14

Deterministic?

In SQL, "deterministic" means that if a function is called with the same parameter values, it'll always return the same result.

`toMdy(20091231)` returns 12/31/2009

Every time you call `toMdy()` and pass 20091231, it'll always return 12/31/2009.

That means SQL doesn't have to call it repeatedly if the parameter value doesn't change. It can remember the last answer – and not call your function. (Improves performance.)

By default, functions are `NOT DETERMINISTIC` so specify the `DETERMINISTIC` option if your routine would benefit from it…

Democritus was one of the first philosophers to anticipate determinism, and many consider him the father of modern science.

But has nothing to do with RPG or SQL.

15

Parameter Styles (1 of 2)

The previous example used the "GENERAL" parameter style
- Sometimes called "SIMPLE CALL"
- The parameters passed from the SQL statement match what's passed to your program.
- The return value from your program matches what's returned back to the SQL statement.

Other parameter styles are a little different.
- The SQL statement looks the same. Same parameter(s) are passed when calling the UDF.
- There will be additional parameters passed from the database engine to your RPG code, however.
- Null indicators, Error handling fields, and more.
- In some styles, the return value is in the parameter list. (So *PGMs will work.)
- The exact parameters passed will depend on which parameter style is used.

16

Parameter Styles (2 of 2)

- **GENERAL** (only works with *SRVPGM calls) (SIMPLE CALL is an alias)
  - What you see is what you get.
  - There are no extra parameters passed from SQL to your *srvpgm, just the ones given on the SQL statement.
  - The return value of the subprocedure becomes the return value of the UDF.

- **GENERAL WITH NULLS** (only with *SRVPGM calls)
  - Same as GENERAL except that extra parameters are passed from SQL to RPG containing null indicators for all parameters and return values..

- **SQL** *(or DB2SQL which is the same thing!)*
  - Subprocedure return values aren't used, instead return value is passed in the parameter list.
  - Null indicators are passed in the parameter list for all parameters and return values.
  - Various additional parms for error handling, and other stuff. *(more later!)*
  - Supports calling programs as well as subprocedures

There are others, but these are the ones that are useful from an RPG program

YearsActive in GENERAL Parameter Style

```
Create Function yearsActive (
        hiredate Decimal(8, 0),
        termdate Decimal(8, 0)
        )
        returns Integer
        language rpgle
        not deterministic
        no sql
        external name 'mylib/UDFDEMO1(COUNTYEARS)'
        parameter style general;
```

Two parameters, separated by commas, just like you'd have with CREATE TABLE (etc)

Parameter style is GENERAL

The RPG subprocedure name doesn't have to match the SQL name. (It can, but it doesn't have to.) In this case, the RPG name is CountYears, but the SQL name is yearsActive

CountYears w/GENERAL style

```
P countYears      B                        export
D countYears      PI              10i 0
D   hireDate                       8p 0 const
D   termDate                       8p 0 const

D myTerm          s                 d   inz(*sys)
D retval          s              10i 0
 /free

    monitor;
      if (termDate <> 0);
          myTerm = %date(termDate:*iso);
      endif;
      retval = %diff( myTerm
                    : %date( hireDate : *iso)
                    : *YEARS );
    on-error;
      retval = -1;
    endmon;

    return retval;
 /end-free
P                 E
```

> **Date defaults to the current system date.**

> **To specify no date, 0 must be passed. That's okay in this case, but not in every UDF!**
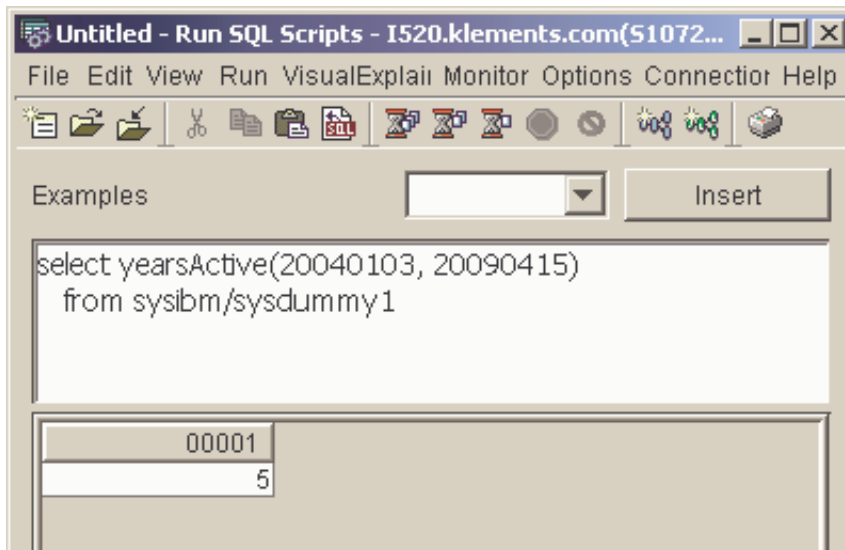
> **Any invalid date (in either parameter) will return an error of -1.**
>
> **Not the best way of handling errors!!**

0 is a special value in our employee master file that means "never terminated"

19

Testing your UDFs from iNav

iNav's Run SQL Scripts is an easy way to test the UDF.

If employee worked from 2004-2009, it's 5 years.

I can easily change the numbers to test different dates.

Tip: SYSIBM/SYSDUMMY1 is an IBM-supplied file that's intended for testing UDFs. It makes it easy to make ad-hoc calls to your UDF.

20

Limitations of the GENERAL Style

- General cannot work with programs because programs cannot return a value.

- Can't report errors in a standard way. Have to roll-your-own error handling.

- Doesn't provide support for null indicators on the parameters – so special values must be used. What if your database is using nulls?

SQL (or DB2SQL) Parameter Style

- First X parameters are the parameters you specified on Create Function.
- A parameter for the return value.
- One parameter for each input parameter's null indicator
- One parameter for the return value's null indicator
- The SQLSTATE (SQLSTT) value, CHAR(5)
- Fully qualified function name VARCHAR(517)
- Specific Name VARCHAR(128)
- Error Message Text – VARCHAR(70)
- There are 3 optional parameters, not discussed here…
    1. Scratchpad
    2. Call type
    3. DBINFO structure

YearsActive in SQL Parameter Style

```
Create Function yearsActiveSql (
        hiredate Decimal(8, 0),
        termdate Decimal(8, 0)
        )
     returns Integer
     language rpgle
     not deterministic
     no sql
     external name 'mylib/UDFDEMO1(YEARSACTIVESQL)'
     parameter style sql;
```

Notice....
2 parameters

1 return value

Parameter style is now SQL

```
.
.
External name 'mylib/UDFPROGRAM'
```

SQL parameter style can call programs instead of procedures. To do that, simply leave off the parenthesis and procedure name. d

RPG Code w/SQL Parameter Style (1 of 2)

```
    P yearsActiveSql  B                    export
    D yearsActiveSql  PI
    D   hireDate                  8p 0 const
    D   termDate                  8p 0 const
    D   yearCount                10i 0
    D   n_hireDate                5i 0 const
    D   n_termDate                5i 0 const
    D   n_yearCount               5i 0
    D   sqlstt                     5a
    D   function                 517a   varying
    D   specific                 128a   varying
    D   errorMsg                  70a   varying

    D myTerm          s             d   inz(*sys)
    D PARM_NULL       C                 CONST(-1)
     /free

        monitor;
          if (n_termDate<>PARM_NULL and termDate <> 0);
            myTerm = %date(termDate:*iso);
          endif;
        on-error;
          sqlstt   = '38999';
          errorMsg = 'Invalid termination date!';
          return;
        endmon;
```

Input parameters

Returned value

Null indicators for each input parameter

Null indicator for return value

SQL State and error message let you return your own SQL errors

```
      if n_hireDate = PARM_NULL;
        sqlstt = '38998';
        errorMsg = 'Hire date cannot be null!';
        return;
      endif;


      monitor;
        yearCount = %diff( myTerm
                         : %date( hireDate : *iso)
                         : *YEARS );
      on-error;
        sqlstt   = '38997';
        errorMsg = 'Invalid hire date!';
      endmon;


      return;
   /end-free
  P                 E
```
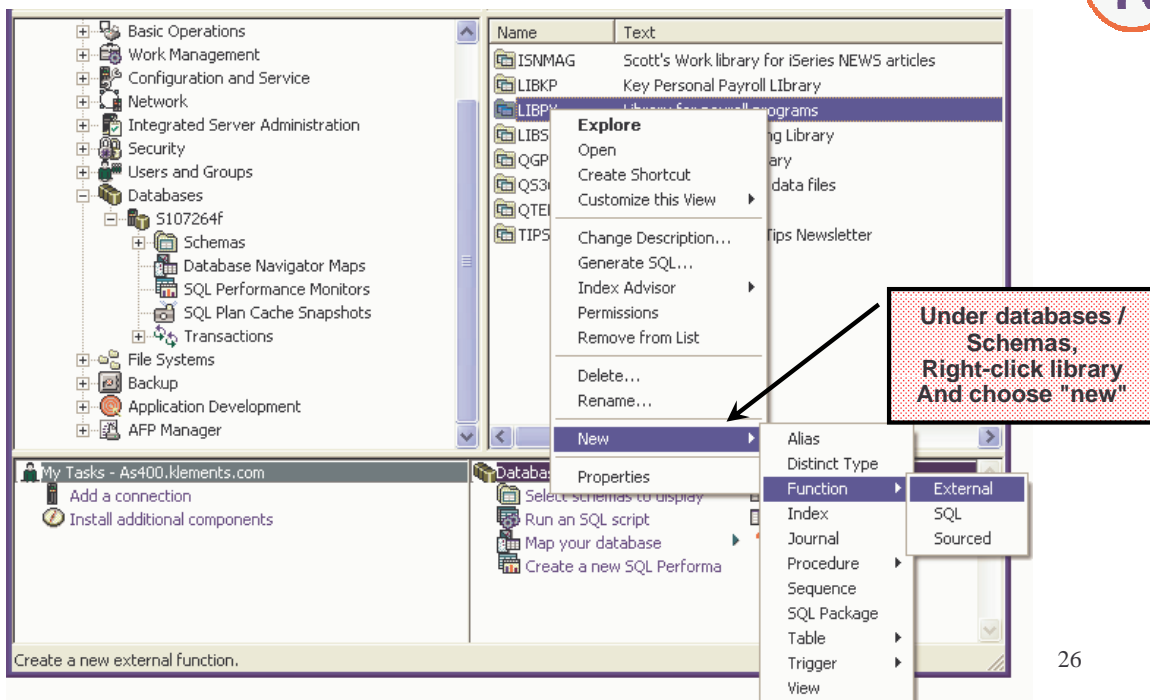
Even though parms to the RPG have changed, the SQL call is the same…

```
Select yearsActiveSql( hireDate, termDate ) from EMPMAST
```

25

---

iNav Has a Wizard for UDFs
helps you write the 'create function' statement

Under databases / Schemas, Right-click library And choose "new"

26

That Was an "External Scalar" UDF

What I've shown you so far is an external scalar UDF.

• External – means it's written in an HLL (not in SQL)

• Scalar – returns a single field value.

• Typically takes the place of a field value on a SELECT staement

• Can also be used in WHERE clause on SELECT, UPDATE, DELETE, etc. (but beware performance – forces a table scan…)

• Basically anyplace you'd have put a "field value"

What if you wanted to return many values?

• An "array" of values?

• Like a result set – like you'd have with a stored procedure?

• Or perhaps it's easier to visualize as a "temporary file".

To do that, you need an "External Table" function…

User Defined Table Function

Although the technical name is "External table function", people, articles and books frequently refer to table functions as UDTF:

User Defined Table Function

Table functions are:

• Also defined with the Create Function SQL Statement

• Return the contents of a (temporary) table (SQL name for "file")

• Works like an array, multiple rows (records) with the same fields.

• Called from a SELECT statement (takes the place of a FILE)

• Usually used like a stored procedure's with a result set – but can do more.

Table Function vs. Stored Procedure

Much has been written about using stored procedures as a means to call RPG business logic in an MVC environment (or other environment where a GUI front-end wants to call existing business logic.)

Advantages of stored procedures:
- Easy to code
- Can return data in parameter list (not so useful for arrays)
- Can return data in a result set (Prior to IBM i 7.1, it was only to ODBC, JDBC or CLI based code. In 7.1, embedded SQL, also.)

Thoughts on using Table Functions, instead:
- Slightly more difficult to code (at first, anyway!)
- Parameter list is input only.
- Can return data in result set to all languages
- Can use SELECT statement to do stuff to result set (more info later…)

When Would You Use UDTF?

Any time you would've previously created a subfile.

(Or equivalent web page)
- Natural way to return "rows" of data that repeat.
- Built-in capability to sort (ORDER BY) the rows
- Built-in capability to filter (WHERE) the rows
- Built-in capability to total up (SUM) the rows.

Since they're callable from embedded SQL, they can even be used to add this sort of capability to green-screen applications!

Existing Report

```
Customer: 4760
Date: 04/09/2009

   ItemNo   Description                      Qty    UOM      Weight        Cost
----------   -------------------------    ---------   ---   -----------   ----------
     7895    CHESY-GRIL GM SKLS 5-1 6"        11    BXS       110.00        21.89
     1891    TURKEY BREAST ROLL 26#BOX         4    BXS       107.53         8.76
     2050    CHICKEN BREAST 10# CW BX         12    BXS        12.66        29.88
     1894    SMK TURKY BRST LOG 26#CW          9    BXS       213.63        19.71
     6970    SMK 25% HAM ROUND 35#CW           4    BXS       154.25         7.92
     3261    KLEMENT FS COTTO 25# EW           3    BXS        75.00         4.77
     2393    GAR PEP BF C-OFF IN 13#BX         4    BXS        54.66        11.96
     8063    CKD BR SLDR PATY 1.5OZ10#         1    BXS        10.00         0.00
     2053    CHICKEN ROLL 5#PC-10#BOX          2    BXS        20.00         4.38
```

Code the business logic as a UDTF, and I can re-use this report as part of any program!

Create Function for UDTF

```
Create Function CustSales( CustNo  Decimal(4, 0),
                           Date    Decimal(8, 0) )
          Returns Table
          (
               itemNo Char(10),
               desc   Char(25),
               Qty    Decimal(9, 0),
               Unit   Char(3),
               Weight Decimal(11, 2),
               Cost   Decimal(9, 2)
          )
          external name 'my-lib/UDTFDEMO1(CUSTSALES)'
          language rpgle
          parameter style db2sql
          no sql
          not deterministic
          disallow parallel;
```

"Returns Table" is how you tell SQL that your function returns a table instead of a scalar

You must list the columns (fields) in your returned table.

DB2SQL is the only allowed parameter style

Currently "disallow parallel" is required. May change in future release?
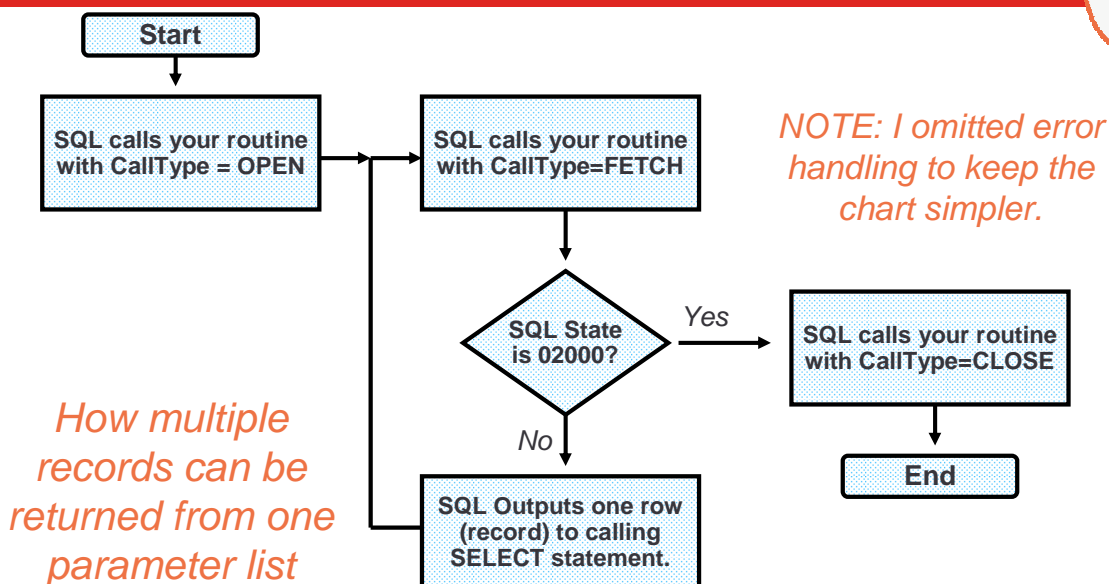
DB2SQL Parameter Style for Table Function

DB2SQL Parameter Style is the only one currently supported for UDTFs – and is very very similar to the SQL Parameter Style used with a scalar function.

- One RPG parameter for each input parameter (CustNo & Date)
- One RPG parameter for <u>each column</u> (field) returned in a single record.
- One null indicator parameter for each input parameter.
- One null indicator parameter for <u>each column</u> returned in a single record.
- SQL State for handing errors, CHAR(5)
- Fully-qualified function name VARCHAR(517)
- Specific Name VARCHAR(128)
- Error message VARCHAR(70)
- Call type – an integer (10I 0) to tell which "event".
  - -2=first call*, -1=open, 0=fetch, 1=close, 2=final call*
  - (* first call & final call are only used if you specify the "final call" option on the create function statement.)

33

How a UDTF is Called

Start

SQL calls your routine with CallType = OPEN

SQL calls your routine with CallType=FETCH

NOTE: I omitted error handling to keep the chart simpler.

SQL State is 02000?   *Yes* → **SQL calls your routine with CallType=CLOSE**

No

SQL Outputs one row (record) to calling SELECT statement.

End

How multiple records can be returned from one parameter list

34

```
P CustSales        b                      export
D CustSales        pi
D    CustNo                    4p 0 const
D    Date                      8p 0 const
D    itemNo                    10a
D    desc                      25a
D    qty                       9p 0
D    unit                      3a
D    weight                    11p 2
D    cost                      9p 2
D    n_CustNo                  5i 0 const
D    n_date                    5i 0 const
D    n_ItemNo                  5i 0
D    n_Desc                    5i 0
D    n_Qty                     5i 0
D    n_Unit                    5i 0
D    n_Weight                  5i 0
D    n_Cost                    5i 0
D    SQLSTT                    5a
D    Function                  517a    varying const
D    Specific                  128a    varying const
D    errorMsg                  70a     varying
D    CallType                  10i 0 const
```

Two input parms are first

The columns of the returned file.

Null indicators for input parms and output columns

Call Type lets SQL notify if it's the Open, Fetch or Close event.

35

```
D CALL_OPEN        C                      CONST(-1)
D CALL_FETCH       C                      CONST(0)
D CALL_CLOSE       C                      CONST(1)
D PARM_NULL        C                      CONST(-1)
D PARM_NOTNULL     C                      CONST(0)

 /free

      if n_Date=PARM_NULL or n_CustNo=PARM_NULL;
         SQLSTT = '38999';
         errorMsg = 'Both CUSTNO and DATE are mandatory';
         return;
      endif;

      select;
      when CallType = CALL_OPEN;
         exsr doOpen;
      when CallType = CALL_FETCH;
         exsr doFetch;
      when CallType = CALL_CLOSE;
         exsr doClose;
         *INLR = *ON;
      endsl;
```

This routine requires non-null parameters.

This routine will be called with OPEN first, then with FETCH repeated for every row, and finally CLOSE.

36

RPG Code for CustSales (3 of 6)

```
        begsr doOpen;
            if not %open(CUSTMAS);
                open CUSTMAS;
            endif;
            if not %open(ITMMAST);
                open ITMMAST;
            endif;
            if not %open(ORDBYCUS);
                open ORDBYCUS;
            endif;

            chain (CustNo) CUSTMAS;
            if not %found;
                SQLSTT='38998';
                errorMsg = 'Unknown customer';
                return;
            endif;

            setll (CustNo:Date) ORDBYCUS;
        endsr;
```

Move to the start of the list of "rows" that this UDTF will return.

37

RPG Code for CustSales (4 of 6)

```
    begsr doFetch;
        reade (CustNo:Date) ORDBYCUS;
        if %eof;
            SQLSTT='02000';
            return;
        endif;

        ItemNo = ocItem;
        Qty    = ocQty;
        cost   = ocPric * ocQty;

        chain (%dec(ItemNo:5:0)) ITMMAST;
        if not %found;
            SQLSTT='38998';
            errorMsg = 'Unknown item found in list';
            return;
        endif;

        Desc = imDesc;
```

Set SQLSTT to 02000 to tell SQL when you're done returning rows.

If you forget this your program will be called over and over again in an endless loop!

38

```
        select;
        when ocUnit = 'L';
           Unit = 'Lbs';
           Weight = Qty;

        when ocUnit = 'B';
           Unit = 'Bxs';
           Weight = Qty * imLbBx;

        when ocUnit = 'P';
           Unit = 'Pcs';
           Weight = Qty * imLbPc;

        when ocUnit = 'Z';
           Unit = 'Plt';
           Weight = Qty * imLbPl;

        endsl;

        Cost = Cost * cuDPct;
      endsr;
```

Your business logic can be anything you like. In this case, I'm calculating the weight, unit of measure and cost.

I can do any sort of RPG Calculations I need here….

39

```
      begsr doClose;
         close *ALL;
      endsr;

   /end-free
  P                 E
```

You might think of these the way you think of the RPG Cycle.

…The fetch subrotuine is called once for each record…

- Called once with CallType=OPEN.   You do whatever you'd traditionally do in *INZSR.

- Called many times with CallType=FETCH – each time you use the parameters to return one record. Return '02000' to signal the end.

- Finally called with CallType=CLOSE.

40

Use SQL's Built-In Functions To "Do More"

Other "Do More" Ideas

Only show orders over 500 lbs:

```
select item, desc, qty from table(CustSales( :CustNo, :Date )) as x
      where weight > 500.00
```

Loading data to be displayed in a table (subfile), and want user to be able to sort by clicking the column?

```
Select item, desc, qty from table(CustSales( :CustNo, :Date )) as y
      order by case
         when :ColNo = '1' then item
         when :ColNo = '2' then desc
         when :ColNo = '3' then qty
      end
```

Search the Product Description:

```
Select item, desc, qty from table(CustSales( :CustNo, :Date )) as z
      where desc like '%ITAL%'
```

Example: Calling UDTF from RPG

```
D C1              ds
D   Itemno                  10a
D   desc                    25a
D   qty                      9p 0
D   Unit                     3a
D   weight                  11p 2
D   cost                     9p 2

 /free
      exec SQL declare C1 cursor for
         select itemno, desc, qty, unit, weight, cost
           from table(CustSales( :CustNo, :Date )) as c
           order by ItemNo;

      exec SQL open C1;
      exec SQL fetch next from C1 into :C1;
      dow sqlstt='00000';

   ... Do something with fetched data here...

         exec SQL fetch next from C1 into :C1;
      enddo;

      exec SQL close C1;
```

It's not different from any other SELECT statement in embedded SQL!

You can take advantage of stuff like WHERE and ORDER BY to filter the output…

Example: Calling UDTF from PHP

```php
$conn = db2_connect( "", "", "" );

$sql = "select itemno, desc, qty, unit, weight, cost "
       . "from table(CustSales("
       . db2_escape_string($_GET['custno']) . ", "
       . db2_escape_string($_GET['date']) . ")) as c  "
       . "order by ItemNo";

$stmt = db2_exec($conn, $sql);
while (($row = db2_fetch_assoc($stmt))) {
   echo '<tr>'
       . '<td>' . $row['ITEMNO'] . '</td>'
       . '<td>' . $row['DESC'] . '</td>'
       . '<td>' . $row['QTY'] . '</td>'
       . '<td>' . $row['UNIT'] . '</td>'
       . '<td>' . $row['WEIGHT'] . '</td>'
       . '<td>' . $row['COST'] . '</td>'
       . '</tr>';
}
```

More Information – UDFs

In System iNEWS magazine (ProVIP membership, or print copy):

March 2006, Sharon L. Hoffman, "Unleash the Power of SQL Functions"
http://systeminetwork.com/article/unleash-power-sql-functions

August 2007, Lynne Noll, "Calling RPG from SQL: External UDFs to Extend SQL"
http://systeminetwork.com/article/calling-rpg-sql-external-udfs-extend-sql

In System iNetwork Programming Tips newsletter (free):

July 29, 2004, Scott Klement, "SQL User Defined Functions in RPG"
http://systeminetwork.com/node/60942

More Information – UDTFs

In System iNEWS magazine (ProVIP membership, or print copy):

January, 2007, Kent Milligan & David Andruchuk, "UDTFs: the Unsung Function"
http://systeminetwork.com/article/udtfs-unsung-db2-function

January, 2009, Jagannath Lenka, "Use a UDTF and SQL to Query XML Files"
http://systeminetwork.com/article/use-udtf-and-sql-query-xml-files

In System iNetwork Programming Tips newsletter (free):

February 8, 2007, Jean-Paul LaMontre,
"Build a New Interface to Legacy Applications with SQL UDTFs"
http://systeminetwork.com/article/build-new-interface-legacy-applications-sql-udtfs

May 24, 2007, Scott Klement, "A UDTF For IFS Directories"
http://systeminetwork.com/article/udtf-ifs-directories
http://systeminetwork.com/article/udtf-recursive-ifs-directories

June 11, 2009, Scott Klement "UDTFs and Subfiles Make a Great Pair
http://systeminetwork.com/article/udtfs-and-subfiles-make-great-pair

47

This Presentation

You can download a PDF copy of this presentation from:

http://www.scottklement.com/presentations/

Thank you!

48