Embedded SQL in RPG - Beyond the Basics

Paul Tuohy
ComCon
System *i* Developer
5, Oakton Court, Ballybrack
Co. Dublin
Ireland

Phone: +353 1 282 6230
e-Mail: paul@systemideveloper.com
Web: www.systemideveloper.com
www.ComConAdvisor.com

System *i* Developer
The Summit of Education Excellence
Producers of the RPG & DB2 Summit

ComCon

Paul Tuohy

Paul Tuohy has worked in the development of IBM Midrange applications since the '70s. He has been IT manager for Kodak Ireland Ltd. and Technical Director of Precision Software Ltd. and is currently CEO of ComCon, a midrange consultancy company based in Dublin, Ireland. He has been teaching and lecturing since the mid-'80s.

Paul is the author of "Re-engineering RPG Legacy Applications", "The Programmers Guide to iSeries Navigator" and the self teach course "iSeries Navigator for Programmers". He writes regular articles for many publications and is one of the quoted industry experts in the IBM Redbook "Who knew you could do that with RPG IV?".

Paul is one of the co-founders of System i Developer and is also an award winning speaker who speaks regularly at US Common and other conferences throughout the world.

Disclaimer

This presentation may contain small code examples that are furnished as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. We therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All code examples contained herein are provided to you "as is". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Have you visited SystemiDeveloper.com?

Check it out for the latest and greatest in System i RPG and database education

Agenda

- Keys to Embedded SQL
- Writing SQL to Embed
  - Tools
  - Methodology
- Controlling CRTSQLRPGI
- SQLCA and SQLCODE considerations
- Getting data without SELECT
  - GET DIAGNOSTICS
  - VALUES INTO
- Mastering Multi Row Fetch
  - Page at a time processing
- Handling NULL
- Dates, Times and Timestamps

The Keys to Embedded SQL

Master SQL

- Beyond the scope of this presentation. But a few pointers...
- Learn to use CASE
  - Not just for column values
  - Can be used on inserts
  - Can be used in a WHERE clause
  - Can even be used on a join condition
- Learn to use Common Table Expressions (CTE)
  - Easier to use than sub queries
- Embrace views
  - A view is a stored SELECT statement
    - without the ORDER BY clause
  - No maintenance overhead
  - Can simplify often repeated select conditions, joins etc.
  - Can have a view of a view
  - And, of course, you can select from a view

Choose carefully between Static and Dynamic embedded SQL

- Always use Static where possible
- Only use Dynamic when there is no other choice

Coding Process

Tools

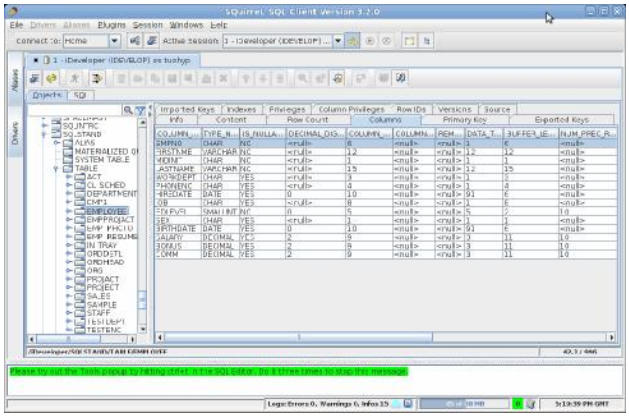
- RSE (Remotes System Explorer Perspective)
  - In WDSC/RDi/RD for Power
- Run SQL Scripts
  - Open Source Alternatives
    - SQuirrel
    - FROG
- Visual Explain
  - Do you have all your indexes?

Does the SQL statement work?

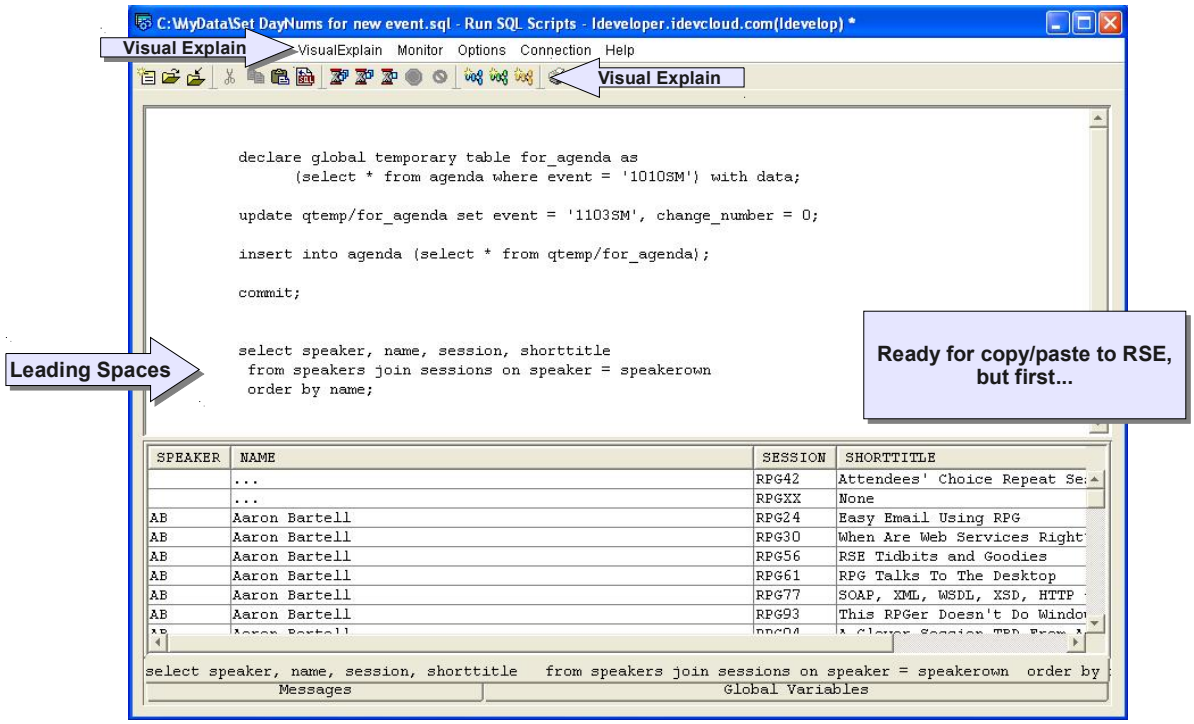
What is the Query Engine doing?

- Avoid the CQE if possible

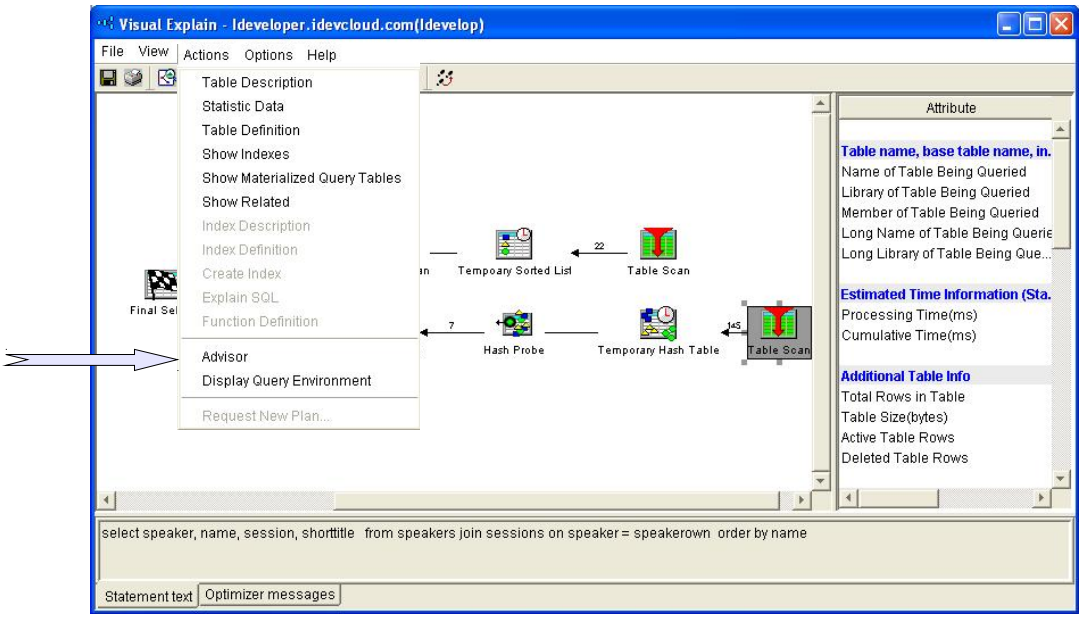
Does Index Advisor have advise?



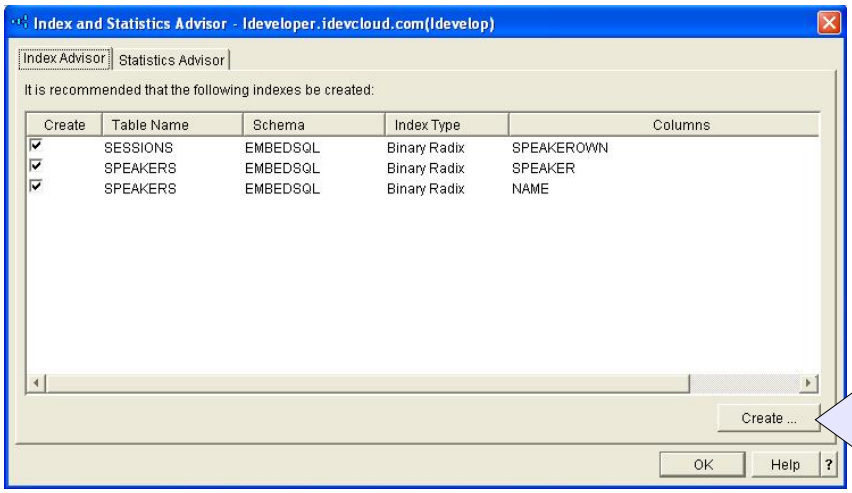
Check in Run SQL Scripts



Check Statements in Visual Explain



What Does Advisor Advise?



SET OPTION – Embedded SQLs H Spec

SET OPTION overrides compile keywords

- In the same way as an RPG H spec
- And keywords that are not on the compile command
  - ALWBLK, ALWCPYDTA , CLOSQLCSR, COMMIT, COMPILEOPT, CONACC, DATFMT, DATSEP, DBGVIEW, DECFLTRND, DECMPT, DECRESULT, DFTRDBCOL, DLYPRP, DYNDFTCOL, DYNUSRPRF, EVENTF, EXTIND, LANGID, MONITOR, NAMING, OPTLOB, OUTPUT, RDBCNNMTH, SQLCA, SQLCURRULE, SQLPATH, SRTSEQ, TGTRLS, TIMFMT, TIMSEP, USRPRF
- Candidate for a copy member
- Must be first SQL in source
  - Remember, sequence SQL is coded in is important
    - Regardless of the sequence it is executed

```
/free
 exec SQL
    set option commit = *NONE,
               naming = *SYS,
               datFmt = *ISO,
               datSep = '-',
               srtSeq = *LANGIDSHR;
/end-Free
```

SET OPTION Keywords

COMMIT – determines use of commitment control within the program

- SQL assumes commitment control
  - default for COMMIT is *CHG
- Tables must be journalled for commitment control

DATFMT/DATSEP/TIMFMT/TIMSEP – determines the format for dates and times used in SQL statements

- Regardless of the format defined on the table
  - More later

SRTSEQ – determines the collating sequence to be used for string comparisons

- Use national language support (*LANGIDSHR) to ignore case in sequence and comparisons

DBGVIEW – determines the type of debug information to be provided by the compiler

- Keyword on CRTSQLRPGI command allows *NONE and *SOURCE
- But *SOURCE actually results in *SOURCE and *LIST
  - You can select the view you want in the debugger
- SET OPTION allows you to specify *NONE, *SOURCE, *STMT and *LIST

Changing CRTSQLRPGI Defaults

Consider changing a couple of the CRTSQLRPGI command defaults

- ```
  CHGCMDDFT CMD(CRTSQLRPGI)
            NEWDFT('dbgview(*source) rpgppopt(*lvl2)')
  ```

DBGVIEW – Debug View

- As noted on previous slide
- DBGVIEW(*SOURCE) actually results in *SOURCE and *LIST
  - You can select the view you want in the debugger
- Alternative to specifying option with SET OPTION

RPGPPOPT – RPG Pre Processing Option

- Determines whether or not the ILE RPG compiler is first called to process compiler directives
  - i.e Copy Members and Conditional Includes
- *LVL1 - /COPY directives are processed
  - but not /INCLUDE
- *LVL2 - /COPY and /INCLUDE directives are processed

DATFMT/DATSEP/TIMFMT/TIMSEP – Date Format and Date Separator

- Also candidates for consideration
  - More later
  ```

How Many Rows in a Result Set

Can be useful to know

- More than you intended to process in a multi row fetch?
- Display a meaningful message
  - 51 to 100 of 372 Employees

The GET DIAGNOSTICS statement

- Obtains information about the previous SQL statement that was executed
  - DB2_NUMBER_ROWS - Returns the number of rows in the result table if the previous SQL statement was an OPEN or a FETCH
  - See manual for full list of 96+ item names

```
exec SQL
   declare C1 scroll cursor for
     select deptNo, deptName from department order by deptNo
       for read only;
exec SQL
   open C1;
exec SQL
   get diagnostics :numRows1 = DB2_NUMBER_ROWS;
dsply ('Open C1 DB2_NUMBER_ROWS= ' + %char(numRows1));
```

```
Open C1 DB2_NUMBER_ROWS= 14
```

SQLCA Considerations

SQLCA sub-fields are re-set with EVERY SQL statement

- SQLCODE, SQLSTATE, SQLERRD often checked in wrong place
- In this example, GET DIAGNOSTICS resets SQLCA fields. Output is

```
DSPLY  Open C1 DB2_NUMBER_ROWS= 14
DSPLY  Fetch SQLERRD(3)= 3
DSPLY  Fetch DB2_NUMBER_ROWS= 0
DSPLY  SQLERRD(3) after Get Diagnostic= 0
```

```
exec SQL
   open C1;
exec SQL
   get diagnostics :numRows1 = DB2_NUMBER_ROWS;
dsply ('Open C1 DB2_NUMBER_ROWS= ' + %char(numRows1));
exec SQL
   fetch first from C1 for :getRows rows into :data ;
numRows2 = SQLERRD(3);
exec SQL
   get diagnostics :numRows3 = DB2_NUMBER_ROWS;
dsply ('Fetch SQLERRD(3)= ' + %char(numRows2));
dsply ('Fetch DB2_NUMBER_ROWS= ' + %char(numRows3));
dsply ('SQLERRD(3) after Get Diagnostic= ' + %char(SQLERRD(3)));
```

Access to SQL Functions and Registers (1 of 2)

RPG has lots of Built in Functions

- Many them the same as SQL Scalar Functions

RPG can access all of SQL Scalar Function and Registers

- Simply use VALUES INTO

```
d system          s              10a
d myName          s               5a   inz('Paul')
d aDate           s                d   datFmt(*ISO) inz(D'2010-12-07')
d dayNumber       s              10i 0
 /free
  exec SQL values current server into :system ;
     dsply ('System name is ' + system);
  exec SQL values upper(:myName) into :myName;
     dsply ('My name in upper case is ' + myName);
  exec SQL values dayOfWeek(:aDate) into :dayNumber ;
     dsply ('Day of week is ' + %char(dayNumber));
  exec SQL values dayOfWeek_ISO(:aDate) into :dayNumber ;
     dsply ('ISO Day of week is ' + %char(dayNumber));
  exec SQL values dayOfYear(:aDate) into :dayNumber ;
     dsply ('Day of year is ' + %char(dayNumber));
```

Access to SQL Functions and Registers (2 of 2)

This is the output from the previous program

```
DSPLY  System name is IDEVELOP
DSPLY  My name in upper case is PAUL
DSPLY  Day of week is 3
DSPLY  ISO Day of week is 2
DSPLY  Day of year is 341
```

Temporary Tables

System i Developer The Summit of Education Excellence ComCon

Temporary Tables can be useful

- Created using DECLARE GLOBAL TEMPORARY
  - Temporary table created in QTEMP
- A trivial example below
  - Creates a temporary table with a row per day of the week
  - Retrieves the day of the week for a date
  - Retrieves the day name from the temporary table

```
d aDate           s               d   datFmt(*ISO) inz(D'2010-12-07')
d dayNumber       s              10i 0
d dayName         s              10a
 /free
  exec SQL Set Option Commit=*NONE, Naming=*SYS;
  exec SQL declare global temporary table temp_daynames
          (day_number integer, day_Name varchar (9)) ;
  exec SQL insert into temp_daynames
              values(1, 'Monday'), (2, 'Tuesday'), (3, 'Wednesday'),
                    (4, 'Thursday'), (5, 'Friday'), ('6', 'Saturday'),
                    (7, 'Sunday');
  exec SQL values dayOfWeek_ISO(:aDate) into :dayNumber ;
  exec SQL select day_Name into :dayname
                 from temp_daynames where day_number = :dayNumber;
  dsply ('Day is ' + dayName);
  exec SQL drop table qtemp/temp_daynames;
```

Temporary Tables – Practical Example

System i Developer The Summit of Education Excellence ComCon

A not so trivial example from an application

- Table contains n rows for an Event
- Copy these n row to another Event

Method

- Copy the required rows to a temporary table
- Update the key value (and any other columns) in the temporary table
- Insert the rows from the temporary table into the main table

```
d copyEvent       PI
d  fromEvent                    5a    const
d  toEvent                      5a    const
 /free
  exec SQL declare global temporary table for_agenda as
          (select * from agenda where event = :fromEvent) with data;
  exec SQL update for_agenda
                 set event = :newEvent, change_number = 0;
  exec SQL insert into agenda (select * from for_agenda);
  exec SQL commit;
```

Reminder – Using a Cursor

Sequential read of a file – Fetch row at a time

```
H option(*srcStmt : *noDebugIO))
d data            Ds                    qualified
d  deptNo                         3a
d  deptName                      36a    varying
 /include STANDARD
 /free
  exec SQL
     declare C1 cursor for
       select deptNo, deptName from department order by deptNo
         for read only;
  exec SQL
     open C1;
  exec SQL
     fetch next from C1 into :data ;
  doW (SQLCODE >= 0 and SQLCODE <> 100);
     dsply ('Fetch Loop ' + data.deptNo + ' ' + data.deptName);
     exec SQL
        fetch next from C1 into :data ;
  endDo;
  exec SQL
     close C1;
  *inLR = *on;
```

Multi Row Fetch

A Multi Row Fetch is a much more efficient way of retrieving rows

```
H option(*srcStmt : *noDebugIO)
d MAX_ROWS        C                     10
d i               s               10i 0
d getRows         s               10i 0 inz(MAX_ROWS)

d data            Ds                    dim(MAX_ROWS) qualified
d  deptNo                         3a
d  deptName                      36a    varying
 /include STANDARD
 /free
  exec SQL declare C1 scroll cursor for
       select deptNo, deptName from department order by deptNo
         for read only;
  exec SQL open C1;

  exec SQL fetch first from C1 for :getRows rows into :data ;

  for i = 1 to SQLERRD(3);
     dsply ('Normal ' + data(i).deptNo + ' ' + data(i).deptName);
  endFor;

  exec SQL close C1;
  *inLR = *on;
```

Multi Row Fetch Considerations

Much faster than a FETCH Loop
- That alone is reason enough to use it

An easy way of generating a result set
- When using embedded SQL for stored procedures

DS Array can be passed as a parameter
- Provides an easy means of using result sets in RPG applications

Data Structure Array or Multiple Occurrence Data Structure (MODS)
- MODS is the older (and more cumbersome) technique
- DS Arrays are much easier

Only a finite number of rows may be retrieved
- Pre-V6R1 – 64K of data
- Post V6R1 – 16M of data

What if the result set exceeds the size of the DS array?
- Does “subfile paging” ring a bell?

Sequential Multi Row Fetch

Sequential read of a “page” at a time

Ahem!

```
H option(*srcStmt : *noDebugIO)
d MAX_ROWS        C                   3
d i               s              10i 0
d getRows         s              10i 0 inz(MAX_ROWS)

d data            Ds                   dim(MAX_ROWS) qualified
d  deptNo                         3a
d  deptName                      36a   varying
 /include STANDARD
 /free
  exec SQL declare C1 scroll cursor for
       select deptNo, deptName from department order by deptNo
         for read only;
  exec SQL open C1;
  doU SQLCODE <> 0;
     exec SQL fetch relative 1 from C1 for :getRows rows into :data ;
     for i = 1 to SQLERRD(3);
        dsply ('Sequential ' + data(i).deptNo + ' ' + data(i).deptName);
     endFor;
  endDo;
  exec SQL close C1;
  *inLR = *on;
```

FETCH RELATIVE

FETCH RELATIVE is relative to the current cursor position in the result set

- 0 is the current position of the cursor
- 1 is the next row
  - i.e. ***Fetch relative 1*** is the same as ***Fetch Next***
- -1 is the previous row
  - i.e. ***Fetch relative -1*** is the same as ***Fetch Prior***

As rows are fetched, cursor is placed on last row read

Paging Multi Row Fetch – A Sample Program

To page forward/back through a result set

- Using a multi row fetch
- A simple example
  - *declareAndOpen()* contains the same Declare Cursor and Open Cursor as previous
  - *closeCursor()* contains the same Close Cursor as previous example
  - Remember *getRows()* must be physically coded between *declareAndOpen()* and *closeCursor()*
  - Complete listing in notes

```
H option(*srcStmt : *noDebugIO)
d MAX_ROWS        C                   11
d pageSIze        s             10i 0 inz(MAX_ROWS)

 /include STANDARD
 /free
  dsply 'Number of rows per page: ' ' ' pageSize;
  if (pageSize > (MAX_ROWS-1));
     pageSize = (MAX_ROWS-1);
  endIf;
  declareAndOpen();
  getRows(pageSize);
  closeCursor();
  *inLR = *on;
 /end-Free
```

Paging Considerations

Paging considerations:-

- SQLCODE not set if rows read < page size
  - Use GET DIAGNOSTICS to determine if EOF reached
- EOF not set if last row of page is last row of result set
  - i.e. EOF not set if 10 rows in result set, 10 rows in page
- Read one more row than page size
  - To detect EOF

Factors

- The size of a page
- The number of rows just read
- EOF

Controlling the relative position

- For first page, set relative position to 1
- If Page Back requested, set relative position to (1 - (rows on this page + page size))
  - i.e. Next Page starts with the first row of the previous page
- Read page size + 1
- If not EOF – set relative position to 0
  - i.e. Next Page starts with the last row read
- If EOF – set relative position to (1 – rows just read)
  - i.e. Next Page starts with the first row of this page

Paging Multi Row Fetch – getRows() (1 of 3)

These are the D Specs for the getRows() subprocedure

- ***direction*** - F = Forward, B = Back, E = End
- ***getPageSize*** - set to pageSize + 1
- ***relativeRow*** Initialized to 1 for the first page read

```
p getRows...
p                 b
d                 PI
d pageSize                      10i 0 const

d data            Ds                  dim(MAX_ROWS)
d                                     qualified
d  deptNo                        3a
d  deptName                     36a   varying

d i               s             10i 0
d direction       s              1a   inz('F')
d getPageSize     s             10i 0
d relativeRow     s             10i 0 inz(1)
d backRows        s             10i 0
d lastRow         s             10i 0
```

```
the requested Page Size

DS array for the fetch




paging direction
rows to retrieve on the fetch
relative offset for next read
number of rows fetched
status for EOF
```

Paging Multi Row Fetch – getRows() (2 of 3)

The basic logic is (continued on next slide)

- Set the no. of rows to retrieve on the fetch
- If page back requested – set relative offset to start of previous page
- Fetch the page
- Store the no of rows retrieved
- Check for EOF
- Assume next relative offset is from last row just read
- If EOF - set relative offset to start of this page

```
/free
 doU (direction = 'E');
    getPageSize = pageSize + 1;
    if (direction = 'B');
       relativeRow = (1 - (pageSize + backRows));
    endIf;
    exec SQL fetch relative :relativeRow from C1
                for :getPageSize rows into :data;
    backRows =  SQLERRD(3);
    exec SQL get diagnostics
              :lastRow = DB2_LAST_ROW;
    relativeRow = 0;
    if (lastRow = 100);
       dsply ('Reached EOF');
       relativeRow = (1 - backRows);
    endIf;
```

```
no. of rows to retrieve
Page back?
offset to start of previous page

Fetch page

Store rows retrieved
Check for EOF

Assume next relative offset
EOF?

offset to start of this page
```

Paging Multi Row Fetch – getRows() (3 of 3)

The basic logic is (continued from previous slide)

- If no rows retrieved, load first page
  - Usually caused by paging beyond start of result set
- Display page
  - This example display all rows retrieved
  - Usually display backRows or pageSize
    - Whichever is less
- Prompt for next paging option

```
    if (backRows = 0);
       exec SQL fetch first from C1 for :getPageSize rows into :data;
       backRows =  SQLERRD(3);
    endIf;

    for i = 1 to backRows;
       dsply ('Paging ' + data(i).deptNo + ' ' + data(i).deptName);
    endFor;
    dsply 'Direction (F/B/E) ' ' ' direction ;
 endDo;
/end-Free
p                e
```

```
H option(*srcStmt : *noDebugIO)
d MAX_ROWS        C                   11
d pageSIze        s             10i 0 inz(MAX_ROWS)

 /include STANDARD
 /free
  dsply 'Number of rows per page: ' ' ' pageSize;
  if (pageSize > (MAX_ROWS-1));
     pageSize = (MAX_ROWS-1);
  endIf;
  declareAndOpen();
  getRows(pageSize);
  closeCursor();
  *inLR = *on;
 /end-Free

p declareAndOpen...
p                 b
d                 PI

 /free
  exec SQL declare C1 scroll cursor for
       select deptNo, deptName from department order by deptNo
         for read only;
  exec SQL open C1;
 /end-Free
p                 e
```



```
p getRows...
p                 b
d                 PI
d pageSIze                      10i 0 const

d data            Ds                  dim(MAX_ROWS) qualified
d  deptNo                        3a
d  deptName                     36a   varying

d i               s             10i 0
d direction       s              1a   inz('F')
d getPageSize     s             10i 0
d relativeRow     s             10i 0 inz(1)
d backRows        s             10i 0
d lastRow         s             10i 0

 /free
  doU (direction = 'E');
     getPageSize = pageSize + 1;
     if (direction = 'B');
        relativeRow = (1 - (pageSize + backRows)) ;
     endIf;
     exec SQL fetch relative :relativeRow from C1
                 for :getPageSize rows into :data;
     backRows =  SQLERRD(3);
     exec SQL get diagnostics :lastRow = DB2_LAST_ROW;
```



```
      relativeRow = 0;
      if (lastRow = 100);
         dsply ('Reached EOF');
         relativeRow = (1 - backRows);
      endIf;

      if (backRows = 0);
         exec SQL fetch first from C1 for :getPageSize rows into :data;
         backRows =  SQLERRD(3);
      endIf;

      for i = 1 to backRows;
         dsply ('Paging ' + data(i).deptNo + ' ' + data(i).deptName);
      endFor;
      dsply 'Direction (F/B/E) ' ' ' direction ;
   endDo;
  /end-Free
p                 e

p closeCursor...
p                 b
d                 PI

  /free
   exec SQL close C1;
  /end-Free
p                 e
```


A Multi Row Insert

System i Developer The Summit of Education Excellence ComCon

Insert multiple rows using a DS Array

- Specify the number of rows on the INSERT statement
- Should really be using commitment control

```
d MAX_ROWS        C                   100
d numOrderDetails...
d                 s             10i 0

d orderHeader   e ds                  extName(ORDHEAD) qualified

d orderDetail   e ds                  extName(ORDDETL) qualified
d                                     dim(MAX_ROWS)
 /free
  exec SQL set option naming = *SYS, datFmt = *ISO, datSep = '-';
  exec SQL insert into ORDHEAD values( :orderHeader);
  if (SQLCODE = 0);
     exec SQL insert into ORDDETL :numOrderDetails rows
                  values (:orderDetail);
  endIf;
  if (SQLCODE = 0);
     exec SQL commit;
  else;
     exec SQL rollBack;
  endIf;
```


SQLCODE (and SQLSTATE) Misuse

SQL errors will not cause a program to fail

- Standard exception/error handling does apply
- As if there is an E extender or Monitor for each SQL statement

SQLCODE reflects what happened on the last SQL statement

- Much like %Status() in RPG
- Imagine if RPG only had %Status() as opposed to %EOF() and E Extender + %Error()

SQLCODE should be checked if there is doubt about statement success

SQLCODE should be checked properly

- e.g `doU (SQLCODE <> 0);` - to control a Fetch loop
  - Will fail for ANY warning message – not just EOF (SQLCODE=100)
  - It may work now, but a new warning message can cause later errors
    - New warning statuses may be introduced on new releases of OS
- Better to use `doW (SQLCODE >= 0 and SQLCODE <> 100);`

Remember that SQLCODE is reset for EVERY SQL statement

ALWAYS check SQLCODE after dynamics SQL PREPARE

```
exec SQL prepare dynamicSQL from :mySQLStatement;
if (SQLCODE = 0);
   exec SQL execute dynamicSQL  ;
endIf;
```

The Dreaded NULL!

NULL is a reality with SQL – it is difficult to avoid

- NULL capable is the default for SQL defined tables
- NULL can result from outer joins

i5 NULL support for the database is unlike other platforms

- NULL is not a value
  - On other platforms it is usually hex '00'
- Each record has a null byte map
- Null byte map is an array of integers
  - One element for each column
    - 0 = NOT NULL, -1 = NULL
  - You may have come across null byte maps in trigger buffers

In RPG IV, for files defined on F Specs

- use the %NULLIND BIF to set or examine the null indicator for a variable

But F spec rules do not apply to embedded SQL!

- You have to handle it yourself – 2 choices
  - Handle NULL in your programs
  - Code SQL to ensure no resulting NULL values

DDS and DDL Defaults

DDS and DDL (SQL) have different defaults for null capable

```
A                                     UNIQUE
A          R EMPLOYEER
A            EMPID           7A
A            NAME           30A
A            GRADE           2A
A            BIRTHDATE        L          ALWNULL
A            JOINEDDATE       L          ALWNULL
A            SALARY        13P 2         ALWNULL
```

```
CREATE TABLE EMPLOYEE (
   EMPID CHAR(7) CCSID 37 NOT NULL DEFAULT '' ,
   NAME CHAR(30) CCSID 37 NOT NULL DEFAULT '' ,
   GRADE CHAR(2) CCSID 37 NOT NULL DEFAULT '' ,
   BIRTHDATE DATE DEFAULT NULL ,
   JOINEDDATE DATE DEFAULT NULL ,
   SALARY DECIMAL(13, 2) DEFAULT NULL ,
   PRIMARY KEY( EMPID ) ) ;
```

NULL Integers by Column

Fetching NULL integers by column

- Nothing changes on the SELECT statement
- Define required integers as 5 digit integers
- Place integer field after host variable on INTO clause
  - No comma between host variable and integer

```
exec SQL declare C1 scroll cursor for
     select deptno, deptname, mgrno from department order by deptno
       for read only;
exec SQL open C1;
```

SELECT for all NULL examples

```
d deptNo          s              3a
d deptName        s             20a   varying
d mgrNo           s              6a
d mgrNoNull       s              5i 0

  exec SQL fetch next from C1
           into :deptNo, :deptName, :mgrNo :mgrNoNull;
  dsply ('Simple ' + deptNo + ' ' + deptName + ' ' + mgrno +
         ' ' + %char(mgrNoNull));
```

```
DSPLY  Simple A00 SPIFFY COMPUTER SERV 000010 0
DSPLY  Simple D01 DEVELOPMENT CENTER       -1
```

Output for all NULL examples

NULL Integers with Host Structure

Fetching NULL integers using a Host Structure

- Integers for ALL columns must be defined as an array
  - One element per column in the host structure
- Place integer array after host structure on INTO clause
  - No comma between host structure and integer array
- Determine NULL value using element of array

```
d data            ds                  qualified
d  deptNo                        3a
d  deptName                     20a   varying
d  mgrNo                         6a

d dataNull        s                  5i 0 dim(3)

  exec SQL fetch next from C1 into :data :dataNull;
  dsply ('DS ' + data.deptNo + ' ' + data.deptName + ' ' +
         data.mgrno + ' ' + %char(dataNull(3)));
```

NULL Integers with DS Array

Fetching NULL integers using a DS Array Host Structure

- Define a corresponding DS Array for NULL integers
  - Each element must contain an array of integer s
    - One element per column in the host structure
- Place DS NULL integer array after DS host structure array on INTO clause
  - No comma between host structure and integer array
- Determine NULL value using element of array for element of DS

```
d data            Ds                  dim(MAX_ROWS) qualified
d  deptNo                        3a
d  deptName                     20a   varying
d  mgrNo                         6a

d dataNull        Ds                  dim(MAX_ROWS) qualified
d  nullInds                      5i 0 dim(3)

  exec SQL fetch first from C1 for :pageSIze rows into :data :dataNull;
  dsply ('Multi ' + data(i).deptNo + ' ' + data(i).deptName + ' ' +
         data(i).mgrno + ' ' + %char(dataNull(i).nullInds(3)));
```

Meaningful Names for Integer Array Elements

Integer array must be used with a host structure: but...

- Define a data structure
  - Based on a pointer
  - Containing a meaningful name for the corresponding column
    - I use a qualified DS and the same column name
- St pointer to the address of the required element of the DS NULL integer array
  - Or the NULL integer array if using a single host structure

```
d data            Ds                    dim(MAX_ROWS) qualified
d  deptNo                          3a
d  deptName                       20a   varying
d  mgrNo                           6a
d dataNull        Ds                    dim(MAX_ROWS) qualified
d  nullInds                        5i 0 dim(3)

d nullNames       ds                    based(nullPtr) qualified
d  deptNo                          5i 0
d  deptName                        5i 0
d  mgrNo                           5i 0

  exec SQL fetch first from C1 for :pageSIze rows into :data :dataNull;
  nullPtr = %addr(dataNull(i));
  dsply ('Meaningful ' + data(i).deptNo + ' ' + data(i).deptName + ' ' +
         data(i).mgrno + ' ' +  %char(nullNames.mgrno));
```

Ignore NULL Using Coalesce()

Coalesce returns the first non null value

- Null capable column as first parameter
- Default value as second parameter

No Null integers required!

Excellent candidate for views

```
exec SQL declare C1 scroll cursor for
           select deptno, deptname, coalesce( mgrno, '*NONE*')
             from department order by deptno for read only;
```

```
Coalesce A00 SPIFFY COMPUTER SERV 000010
Coalesce D01 DEVELOPMENT CENTER *NONE*
```

Dates and Times

You specify the formats you want

- Format on the table is cast to the format in the program
  - But watch out for errors
  - Code below will fail with an SQLCODE of -181
    - Value in date, time, or timestamp string not valid
    - Dates in table are outside 1940/2039 window
- *ISO is such a good idea!

```
d  empNo          s              6a
d  hireDate       s               d   datFmt(*MDY/)
d  birthDate      s               d   datFmt(*MDY/)
 /free
  exec SQL Set Option Commit=*NONE, Naming=*SYS, DatFmt=*MDY, DatSep='/' ;
  exec SQL declare C1 scroll cursor for
       select empno, hireDate, birthDate from employee order by empno
        for read only;
  exec SQL Open C1;

  exec SQL fetch next from C1 into :empNo, :hireDate: birthDate ;
```

Date and Time Formats

Formats and separators must agree

- Formats in the RPG program must correspond with formats on CRTSQLRPGI
  - CRTSQLRPGI defaults to *JOB
  - RPG defaults to *ISO
  - Format on table is cast to format in program – if possible
- Can set formats with SET OPTION
  - But there is a “bug” out there!

```
exec SQL Set option commit=*NONE, naming=*SYS, datFmt=*ISO, datSep='-' ;
```

```
d  empNo          s              6a
d  hireDate       s               d
d  birthDate      s               d

   exec SQL fetch next from C1 into :empNo, :hireDate: birthDate ;
```

```
d data            Ds                   qualified
d  empNo                          6a
d  hireDate                        d
d  birthDate                       d

   exec SQL fetch next from C1 into :data ;
```

Date and Times with Host Structure Arrays

There is a problem with Dates/Times and Host Structure Arrays

- You must specify the format on the subfield definitions
  - Which isn't such a bad idea anyway
  - Default is not taken from H Spec
- SET OPTION does not work
- You must specify the formats/separators as parameters on CRTSQLRPGI
- Consider changing command defaults

```
CHGCMDDFT CMD(CRTSQLRPGI) NEWDFT('datfmt(*ISO) datsep(-)')
```

```
d MAX_ROWS        C                   100
d pageSize        s             10i 0 inz(MAX_ROWS)

d data            Ds                  dim(MAX_ROWS) qualified
d  empNo                         6a
d  hireDate                       d  datFmt(*ISO)
d  birthDate                      d  datFmt(*ISO)

  exec SQL Set Option Commit=*NONE, Naming=*SYS;

  exec SQL fetch first from C1 for :pageSize rows into :data;

          CRTSQLRPGI OBJ(PGM) DATFMT(*ISO) DATSEP(-)
```

DATFMT/DATSEP must NOT be specified

Summary

Ground covered

- Keys to Embedded SQL
- Writing SQL to Embed
  - Tools
  - Methodology
- Controlling CRTSQLRPGI
- SQLCA and SQLCODE considerations
- Getting data without SELECT
  - GET DIAGNOSTICS
  - VALUES INTO
- Mastering Multi Row Fetch
  - Page at a time processing
- Handling NULL
- Dates, Times and Timestamps

By the Speaker

"Re-Engineering RPG Legacy Applications"
- ISBN 1-58347-006-9

"The Programmers Guide to iSeries Navigator"
- ISBN 1-58347-047-6
- www.mcpressonline.com
- www.midrange.com
- www.amazon.com
- etc.

iSeries Navigator for Programmers
- A self teach course
- www.lab400.com

Article links at
- www.comconadvisor.com