Accessing SQL Functions using Rational Open Access: RPG Edition

Dan Cruikshank

Agenda

- Bridge Concepts
- Rational Open Access Overview
- Types of Handlers
- Handler Scenarios
So, how do we get are legacy programs from here to there?

- Here: Traditional IO
- Answer: A bridge
- There: SQL Data Access
Legacy Program Bridge Concepts

- A technique used to transform a traditional IO operation into an SQL function call
  - A function call is a prototyped procedure within an ILE Service Program

- Benefits
  - Existing programs do not have to use SQL
  - One service program can service many legacy programs
  - One legacy program can access multiple service programs
  - Can be written in any language
  - Can utilize Call Level Interface (CLI) to access SQL Stored Procedure result set
  - Can easily be modified to use future enhancements
    - For example, ALLOCATE CURSOR, RESULT SET LOCATOR, etc.

ILE Bridge Examples

RPG

```rpg
//DELETE(e) LEGACYFILE;
CALLP DELETEDATABASEASEROWBYKEY
(DEL_Parm_List,
SQL_Results);
```

COBOL

```cobol
//DELETE LEGACYFILE
CALL PROCEDURE "DELETEDATABASEASEROWBYKEY"
USING DEL-Parm-List,
SQL-Results.
```

Legacy Program is modified
to call service program
instead of performing IO

No SQL Here

SQL Performed Here

Common Service Program

Any Language

ADDDATABASE ROW

UPDDATABASE ROWBYKEY

DELETEDATABASEASEROWBYKEY
De-coupling UI and DB

- Rational Open Access
  - Handlers intercept traditional IO operations
  - Minimal change to existing RPG programs

- Handlers:
  - Transform 5250 to any UI device
    - 3rd party tools available
  - Transform data access to SQL
    - Utilize advanced database feature and function

- Other tools available for mining source code
  - Identify and extract business rules

RPG Handler Bridge Examples

Legacy Program is modified to use SPECIAL device. Minor changes to program.

Handler Program intercepts IO and calls SQL service program

CALLP ADDDATABASEROWKEY
(ADD_Parm_List, SQL_Results);

CALLP DELETEDATABASEROWKEY
(DEL_Parm_List, SQL_Results);

Common Service Program

OPM or ILE RPG

No SQL Here

SQL Performed Here
Rational Open Access: RPG Edition

- Provides a way for RPG programmers to use the simple and well-understood RPG I/O model to access SQL procedures used by SQL based languages.
  - Everyone is now playing by the same rules
- Open Access opens up RPG’s file I/O capabilities allowing HLL programmers to write innovative I/O handlers that:
  - Transform traditional record at a time I/O operations to SQL set based operations
  - Take advantage of data centric programming techniques
    - RI
    - Auto-generated values
    - Advanced embedded SQL programming techniques
Open Access Structure

- An Open Access application has three parts:
  - An RPG program that uses normal RPG coding to define an Open Access file and use I/O operations against the file.
  - A handler procedure or program that is called by Open Access to handle the I/O operations for the file.
  - The data access service program that the handler is using or communicating with.

- Open Access is the linkage between parts 1 and 2.

- Licensed program 5733-OAR is required to use Open Access at runtime.
  - Fee based
  - Announced for 7.1, PTF’ed to 6.1
    - 6.1 PTFs: SI39480, SI39914

Where the handler fits in

- Data Centric Programming Cornerstones
  - Common service program used by both traditional and new development
  - Handler program can use 7.1 result set access capability
IBM Systems Lab Services and Training

RPG HANDLER Keyword

**FSURROGATE IF E DISK**
- F handler("HANDLER_W1")
- F ExtDesc("EMPLOYEE")
- F USROPN

/Free

Open(e) Surrogate;
READ(e) Surrogate;
Close(e) Surrogate;
Return;

/End-Free

RPG IO operations coded as usual

The HANDLER Program

**D/COPY QOAR/QRPGLESRC,QRNOPENACC**

**D HANDLER_W1 PR**
**D Inp_Ds...**
**D LIKEDS(QrnOpenAccess_T)**

**P Declare_SQL_VIEW_INT1...**
**P OpenCursor_SQL_VIEW_INT1...**
**P FetchFirstFrom_SQL_VIEW_INT1...**
**P Export B**
**P FetchNextFrom_SQL_VIEW_INT1...**
**P E**
**P CloseCursor_SQL_VIEW_INT1...**
**P E**
**P Prepare_SQL_VIEW_INT1...**
**P E**
**P Return_A_Row_From_An_Array...**
**E**

/FREE

//Your code here
return;
/END-FREE

Data structure template provided in QOAR.
Data structure passed by IBM I to handler program

Determine SQL function based on I/O operation
### Open Access Data Structure (QRNOPENACC) Layout

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Type</th>
<th>Set by</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>structLen</td>
<td>UBIN4</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>parameterFormat</td>
<td>CHAR(8)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>userArea</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler and RPG programmer</td>
</tr>
<tr>
<td>stateInfo</td>
<td>Pointer 4</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>recordLevels</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>inputBuffer</td>
<td>Pointer 2</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>outputBuffer</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>nameValues</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>keyNullMap</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>keyNamesValues</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>Indara</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>Prtctl</td>
<td>Pointer 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>openFeedback</td>
<td>Pointer 4</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>ioFeedback</td>
<td>Pointer 4</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>deviceFeedback</td>
<td>Pointer 4</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>externalFile</td>
<td>QrnObject 2</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>externalMember</td>
<td>CHAR(10)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>compileFile</td>
<td>1</td>
<td>QrnObject 2</td>
<td>RPG</td>
</tr>
<tr>
<td>recordName</td>
<td>CHAR(10)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>rpgOperation</td>
<td>UINT(4)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>rpgStatus</td>
<td>INT(4)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>inputDataLen</td>
<td>UINT(4)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>openFeedbackLen</td>
<td>UINT(4)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>ioFeedbackLen</td>
<td>UINT(4)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>deviceFeedbackLen</td>
<td>UINT(4)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
<tr>
<td>numKeys</td>
<td>1</td>
<td>UINT(4)</td>
<td>RPG</td>
</tr>
<tr>
<td>formLen</td>
<td>UINT(4)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>formOf1</td>
<td>UINT(4)</td>
<td>RPG</td>
<td>Handler</td>
</tr>
<tr>
<td>functionKey</td>
<td>UINT(1)</td>
<td>Handler</td>
<td>RPG</td>
</tr>
</tbody>
</table>

**Passed automatically by IBM**

**Provided as /COPY member**

## Handler Considerations

- **Record and Key Data**
  - 2 Modes: Name values or Data structures
  - Name values only available for externally described data
  - Data structures map directly to IO buffers

- **Establish IO Feedback and stateInfo sizes**
  - IO Feedback for returning error
  - stateInfo for tracking where you are

- **Not all data is available for SQL statement**
  - May need to wait until specific IO operation occurs before constructing and executing statement
  - Utilize userArea and stateInfo parameters to build as you go

- **Handler can be program or service program**
Record and Key Data Handling

- **Name values**
  - Each data item is passed with attribute information
    - Field names, size, actual data

- **Data structures**
  - Data is passed as a record
    - Information about the data must be provided by handler

- **Keep it simple**
  - `useNameValues = '1'` can be very complex
    - Ideal for varying-list dynamic SQL programs
  - `useNameValues = '0'`
    - Ideal for mere mortals
    - Data structures can be mapped to external file definitions
    - Will require multiple handlers – 1 per file format

Handler Program vs Handler Service Program

- **Handler Program**
  - **Advantages**
    - Single program can be generic
    - Record format specific modules can be bound to generic handler
    - `*INZSR` can be utilized for one time only operations

  - **Disadvantages**
    - Requires extra coding for different open scenarios

- **Handler Service Program**
  - **Advantages**
    - Multiple entry points can be used for different open scenarios
    - Record format specific modules can be bound to generic handler

  - **Disadvantages**
    - `*INZSR` cannot be used
      - Requires separate module for one time only operations
Database Operation Type Constants

- The RPG IO operation is passed as an unsigned integer
- The RPG OA provided copy book (QRNOPENACC) contains constants mapped to the possible values

```
Const Example
D QmOperation.OPEN...
C 1
D QmOperation.READ...
C 4
D QmOperation.READE...
C 6
D QmOperation.CHAIN...
C 9
D QmOperation.SETLL...
C 12
D QmOperation.UPDATE...
C 14
D QmOperation.WRITE...
C 15
D QmOperation.DELETE...
C 16
D QmOperation.CLOSE...
C 18
```

Processing RPG IO Operations

- An rpgStatus value of zero indicates success
  - Any valid IO error status code can be used to signal an error

Coding tip:
- Create local handler procedures for each RPG IO Operation
  - Use rpgStatus as the return value
  - For handler programs use return with *LR off or the MAIN keyword (6.1)
  - Not an issue if using service programs

```
Example Code
00A_6s.rpgStatus = 'Zero;
select
  when 00A_6s.rpgOperation = QmOperation.OPEN;
  00A_6s.rpgStatus = Handle_Open;
  when 00A_6s.rpgOperation = QmOperation.CHAIN;
  when 00A_6s.rpgOperation = QmOperation.READ;
  when 00A_6s.rpgOperation = QmOperation.READE;
  when 00A_6s.rpgOperation = QmOperation.UPDATE;
  when 00A_6s.rpgOperation = QmOperation.WRITE;
  when 00A_6s.rpgOperation = QmOperation.DELETE;
  when 00A_6s.rpgOperation = QmOperation.CLOSE;
  EndSelect 00A_6s.rstatinfoq
Other;
ENDS ;
return;
```
User Defined Parameters

- **userArea and stateInfo**
  - Pointers to user defined variables
  - Typically data structures

- **userArea**
  - Use to pass additional information to handler from RPG program
  - For example, dynamic SQL statement

- **stateInfo**
  - Use to track program state information
  - For example, last operation successfully executed

Sample User Defined Structures

- Use this data structure to pass additional parameters to further enhance RPG ops.
  - For example
    - Open type, blocking factor, indicator array, etc.

- Use this data structure to pass the state information about a given file or operation
  - For example
    - File name, last successful op, etc
Handling RPG Open

```sql
select;
when
rpgIO.rpgOperation = QrnOperation_OPEN;
  rpgIO.rpgStatus = Handle_Open(rpgIO);
ENDIF;
SQLString = 'SELECT * FROM ' + TableReference + 'WHERE EMPNO = ? FOR FETCH ONLY' + SQLString);
ENDIF;
EXEC SQL Prepare S1 FROM :v_SQL_String;
If SqlState = '00000'; EXEC SQL
Declare C1 SCROLLCURSOR FOR S1; If SqlState <> '00000';
RetField = *ON; EndIf;
Else;
RetField = *ON; EndIf;
```

Handling RPG CLOSE

```sql
when
rpgIO.rpgOperation = QrnOperation_CLOSE;
  rpgIO.rpgStatus = Handle_Close (rpgIO);
```

EXEC SQL Close S1
```sql
EXEC SQL CLOSE rpgIOInp_C1;
```
Methods for Passing Data To and From a Handler

- Two methods available
  - Structure-based
  - Column-based

- Structure-based
  - Data and Key values passed using buffers
  - Buffers can be externally described
  - Good for getting feet wet

- Column-based
  - Data and key values passed as individual items
  - Each item contains data and key attributes
  - More advanced capabilities

- Dynamic SQL required in either case to minimize number of handlers
Generic Database Handlers

- Various methods can be used to build handlers
  - Format-based
    - Best for reengineering startup and structure-based method
    - Handler program uses template based on physical file format
    - Data access module utilizes fixed-list dynamic SQL
  - Statement-based
    - Best for:
      - Column-based method
      - Reengineering multiple distinct format LFs to use a single handler
    - Data access service module utilizes varying-list dynamic SQL
  - Procedure-based
    - Strategic handler intended for 7.1 and beyond
    - Best for result set consumption and “one and done” type functions
      - E.g. SQL MERGE, ROLLUP, CUBE, etc

Using a Fixed-List Handler

- Used with externally described data structures or templates
  - Record format in program is fixed
- Row selection and ordering are dynamic
  - Handler implicitly determines WHERE and ORDER BY from external file attributes
    - DDS Select/Omit criteria is implicitly converted to WHERE
    - System API required to retrieve key columns
      - E.g. file uses K8, K1, K2. K8 is for SETLL and K1 and K2 for ordering
  - User area provided for passing explicit WHERE or ORDER BY clauses
    - Must code logic in handler to close existing cursor then prepare, declare and open new cursor
- An SQL Descriptor is not required for fixed list dynamic SQL
  - Data is written to in one operation
Handling RPG Using Fixed-List Handler

```
FEMPADDRSL1UF E K DISK
F handler('FMTHANDLER')

RPG Implicit
OPEN
```

```
FrcdFile_t IF E K DISK TEMPLATE
F EXTDESC('EMPADDRESS')
F RENAME(empAddr:rcdFormat)
D rcdFormat_t...
D D DS LIKEREC(rcdFormat)
D
D keys_t D DS LIKEREC(rcdFormat:*KEY)
D
D Ind_Array_t...
D S 5i 0 DIM(7)
D
```

Using a Statement-based Handler

- Used with column-based method, record format varies
  - Allows single handler for multiple formats
- Handler determines ordering via passed key column list
  - System API not required
  - User area can be used to pass entire statements (update, delete, etc)
- An SQL Descriptor must be used in data access service modules
  - SQL ALLOCATE DESCRIPTOR or SQLDA can be used
  - SQL PREPARE INTO can be used to describe table columns and parameters
Using a Procedure-Based Handler

- This type of handler is used when SQL Stored Procedures are already in use by new or external applications.
- The SQL Stored Procedure name is passed via the User Area.
  - Best approach for minimal number of handlers.
- The handler consumes the result set created by the stored procedure and returns data using column based methods.
  - Consuming result sets via embedded SQL available in 7.1.
  - Can be done using Call Level Interface APIs on prior releases.
- Ideal for replacing long running IO intensive processes such as:
  - archive and purge (use MERGE 7.1).
  - Mass aggregations (use ROLLUP or CUBE 6.1).
  - Mass inserts, copies, duplication, etc.

Handler Scenarios

- Format based handlers and data access service programs
  - Most flexible, least manageable.
- Custom handlers, single data access service program
  - Flexible and manageable.
- Generic handler, single data access service program
  - Least flexible, most manageable.
### Handler Scenarios

Out with the OLD, In with the New

<table>
<thead>
<tr>
<th>HLL I/O Operation</th>
<th>ISOS Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPG SETxx or COBOL START</td>
<td>QDBRETVK (Position into an index)</td>
</tr>
<tr>
<td>RPG CHAIN or READE or COBOL READ (Dynamic)</td>
<td>QDBRETVK (Position into an index followed by random read by RRN)</td>
</tr>
<tr>
<td>RPG READ or COBOL READ (Sequential)</td>
<td>QDBQROW or QDBQRI M (Sequential read of 1 or multiple records respectively. Applies to both keyed and non-keyed)</td>
</tr>
<tr>
<td>RPG or COBOL WRITE</td>
<td>QDBPUTU or QDBPUTUM (insert 1 or multiple records respectively)</td>
</tr>
<tr>
<td>RPG or COBOL DELETE</td>
<td>QDBUDR (Delete a record)</td>
</tr>
<tr>
<td>RPG UPDATE or COBOL REWRITE</td>
<td>QDBUDR (Update a record)</td>
</tr>
</tbody>
</table>

**QDB...**
- Search Index
- Retrieve Record

**HANDLER**

**SQL Service**

**RPG HLL I/O**
- Open, Close
- Set... CHAIN, READE
- READ
- WRITE
- FEOD
- Others...
Possible Handler Scenarios

- Areas providing best return on reengineering effort
  - SQL blocked FETCH versus RPG READE loop
  - SQL blocked INSERT versus RPG WRITE loop
  - SQL searched UPDATE/DELETE versus RPG UPDATE/DELETE loop
  - SQL JOIN versus RPG READE then RPG CHAIN
  - SQL MERGE versus HLL Archive/Purge

RPG BLOCK(*YES *NO *IT DEPENDS)

- BLOCK(*YES) Allows compiler generated blocking for input only files
  - Positioned by SETGT, SETLL or CHAIN
  - Processed via RPG READ operations
  - OVRDBF SEQONLY(*YES n) required to increase blocking factor

- If any READE or READPE operations are used, NO record blocking will occur for the input file (as stated in the RPG Manual)
  - The BLOCK(*YES) is specified the RPG compiler will issue a severity 10 error indicating blocking is not allowed

- The compiler will generate blocking code for output only files, however:
  - IBM i OS may turn off blocking at file OPEN
  - A message will be logged stating the output only ODP has been changed to SEQONLY(*YES)

- SQL Read/Write only cursors are always blocked at most efficient blocking factor per release
SQL FETCH Processing

- The SQL format based service program defines the host array based on `externalFile`.
- The host array is populated by the SQL FETCH function on first call:
  - The first array occurrence is moved to the `inputBuffer`.
  - Subsequent calls result in the occurrence being incremented and the next row is returned from the array.
  - This continues until all occurrences have been returned.
- The next call results in another FETCH.

SQL Blocked Insert

- Opposite of Blocked Fetch.
- Huge performance benefits when combined with:
  - `SMP` for parallel index maintenance.
  - `OVRDBF REUSEDLT (*NO)` for bulk inserts.

- Rows are passed to handler 1 at a time per RPG WRITE.
- Handler stores passed rows in array.
- When array is full 1 I/O to insert up to 32767 rows into database.
- FEOID indicates end of logical unit of work.
Eliminating Program Joins

- **Scenario 1**
  - Primary file contains HANDLER keyword.
  - Secondary files become templates
  - First READE results in single IO to SQL Join View.
  - Handler returns 1 joined row.
  - CHAINs are no longer needed.

- **Scenario 2**
  - All files contain the HANDLER keyword.
  - First READE results in single IO to SQL Join View.
  - Handler returns primary file row
  - For subsequent CHAINs handler returns secondary file row data using EVAL-CORR

---

Result Set Consumption

- HLL can now access a result set from an SQL stored procedure
- First READE is transformed into stored procedure call.
- Stored procedure opens result set cursor.
- Handler associates and allocates a cursor for the result set
- Handler performs blocked fetch from result set cursor.
Handling RPG UPDATE using SQL Extended Indicator Support

```sql
select
  when rpgIO.rpgOperation = QrnOperation.UPDATE
  rpgIO.rpgStatus = Handle_Update(rpgIO)
  updIndAry = -7;
If updRcd.ADDRLINE1 <> OLD_ROW.ADDRLINE1;
  updIndAry(1) = *Zero;
ENDIF;
// Repeat the above for each updateable column
If Update_Columns_Using_Extended_Indicators (updRcd, updIndAry);
  RetField = 1299;
ENDIF;
EXEC SQL
  UPDATE empAddress
  SET ADDRLINE1 = :updRcd.ADDRLINE1:Ind_Ary_1,
  ADDRLINE2 = :updRcd.ADDRLINE2:Ind_Ary_2,
  ADDRLINE3 = :updRcd.ADDRLINE3:Ind_Ary_3,
  CITY = :updRcd.CITY:Ind_Ary_4,
  STATE = :updRcd.STATE:Ind_Ary_5,
  ZIPCODE = :updRcd.ZIPCODE:Ind_Ary_6
  WHERE EMPNO = :updRcd.EMPNO;
If SqlState = '00000';
  EXEC SQL COMMIT;
Else;
  RetField = *ON;
ENDIF;
```

Minimizing Mass Updates

- First READE results in SQL mass update or delete
- Handler performs 1 I/O and returns end-of-file if successful
- FEOD commits transaction
SQL MERGE – Combining UPDATE and/or INSERT

- MERGE statement added in 7.1
  - Combines summary, existence check, update or insert all in 1 SQL statement
- First READE results in single MERGE execution
- Handler performs 1 SQL statement and returns end-of-file if successful
- FEOD commits transaction

```
RPG UPDATE or INSERT Loop
//Summary transaction
For 1 to "NEWVAL"
  If READ w IMAGE RECORD
    For EACH
      If NEW
        sum_amount += amount
      EndIf
      Endfor
  EndIf
//Check for existence
If SQL search-arg UDR_RECORD
  Use:
  WRITE u NEWRECORD
Endif
COMMIT;
sum_amount = "Zero;
```

```
SQL MERGE performed by handler
--Summary transaction
MERGE INTO table AS a USING ( 
  SELECT id, SUM(amount) sum_amount 
  FROM trans
GROUP BY id) AS t 
ON a.id = t.id 
WHEN MATCHED THEN 
  UPDATE SET balance = a.balance + t.sum_amount 
WHEN NOT MATCHED THEN 
  INSERT (id, balance) VALUES (a.id, t.sum_amount)
COMMIT;
```

Rational Open Access: RPG Edition Recap

- Simplifies bridging RPG programs to new SQL services
- Start out using data structure approach
- Once comfortable, move up to names values
- Use generic handler if bridging is temporary
Questions?

DB2 for IBM i Consulting and Services

✓ Database modernization
✓ DB2 Web Query
✓ Database design, features and functions
✓ DB2 SQL performance analysis and tuning
✓ Data warehousing review and assessment
✓ DB2 for IBM i education and training

Contact: Dan Cruikshank
dcrank@us.ibm.com
IBM Systems and Technology Group
Rochester, MN USA