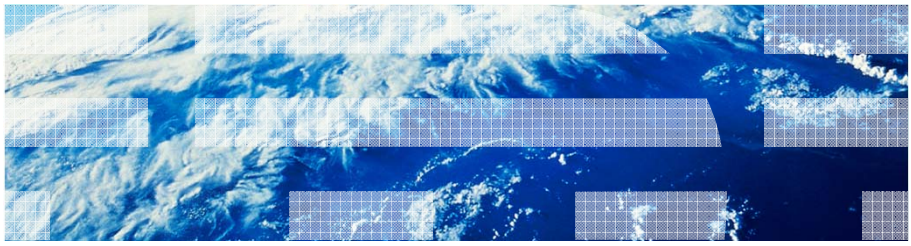
Accessing SQL Functions using Rational Open Access: RPG Edition

Dan Cruikshank



stgls@us.ibm.com
ibm.com/systems/services/labservices

Agenda

- Bridge Concepts
- Rational Open Access Overview
- Types of Handlers
- Handler Scenarios

Traditional IO To SQL

- So, how do we get are legacy programs from here to there?

Here:	Answer:	There:
Traditional IO	A bridge	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

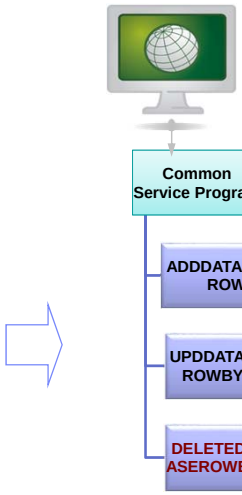
Legacy Program is modified to call service program instead of performing IO

RPG

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

COBOL

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



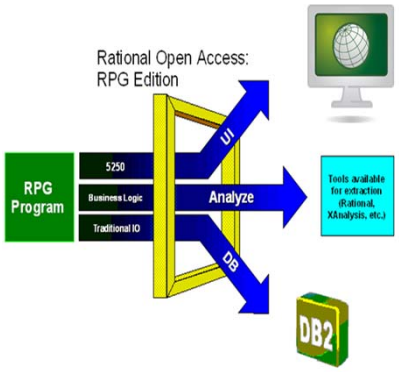
Any Language

No SQL Here

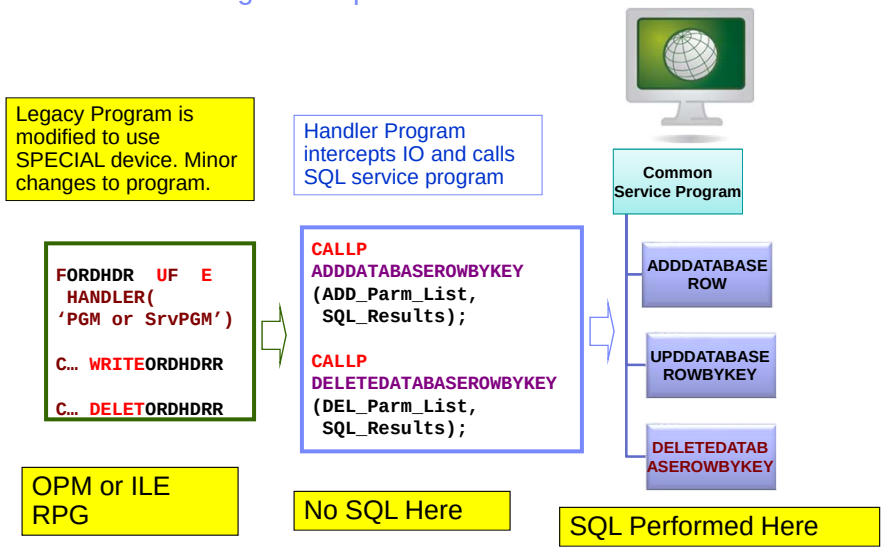
SQL Performed Here

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



Rational Open Access Overview

Rational Open Access: RPG Edition

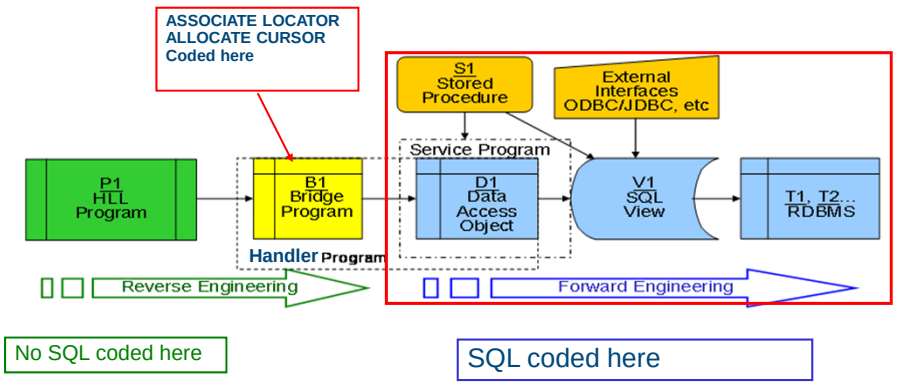
- Provides a way for RPG programmers to use the simple and well-understood RPG I/O model to access SQL procedures used by SQL based languages.
  - Everyone is now playing by the same rules
- Open Access opens up RPG's file I/O capabilities allowing HLL programmers to write innovative I/O handlers that:
  - 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 parts1and 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



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
```

1 Line of code per file

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
D HANDLER_W1      PI
D Inp_Ds...
D                      LIKEDS(QrnOpenAccess_T)

P Declare_SQL_VIEW_INT1...
P                 E
P OpenCursor_SQL_VIEW_INT1...
P                 E
P FetchFirstFrom_SQL_VIEW_INT1...
P                 B           EXPORT
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...
P                 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

Subfield	Type	Set by	Used by
structLen	UBIN4	RPG	Handler
parameterFormat	CHAR(8)	RPG	Handler
userArea	Pointer 2	RPG	Handler and RPG programmer
stateInfo	Pointer 4	Handler	Handler
recordLevels 1	Pointer 2	RPG	Handler
inputBuffer	Pointer 2,3	Handler	RPG
inputNullMap 1	Pointer 2,3	Handler	RPG
outputBuffer	Pointer 2,3	RPG	Handler
outputNullMap 1	Pointer 2,3	RPG	Handler
namesValues 1	Pointer 2,3	RPG and Handler RPG and Handler	

Passed automatically by IBM i
Provided as /COPY member

Subfield	Type	Set by	Used by
keyNullMap 1	Pointer 2,3	RPG	Handler
keyNamesValues 1	Pointer 2,3	RPG	Handler
Indara	Pointer 2,3	RPG and Handler	RPG and Handler
Prtctl	Pointer 2,3	RPG and Handler	RPG and Handler
openFeedback	Pointer 4	Handler	RPG
ioFeedback	Pointer 4	Handler	RPG
deviceFeedback	Pointer 4	Handler	RPG
externalFile	QrnObject_T	RPG	Handler
externalMember	CHAR(10)	RPG	Handler
compileFile 1	QrnObject_T	RPG	Handler
recordName 1	CHAR(10)	RPG and Handler	RPG and Handler
rpgOperation	UINT(4)	RPG	Handler
rpgStatus	INT(4)	Handler	RPG
inputBufferLen	UINT(4)	RPG	Handler
inputNullMapLen 1	UINT(4)	RPG	Handler
outputBufferLen	UINT(4)	RPG	Handler
outputNullMapLen 1	UINT(4)	RPG	Handler
keyLen	UINT(4)	RPG	Handler
keyNullMapLen 1	UINT(4)	RPG	Handler
inputDataLen	UINT(4)	Handler	RPG
openFeedbackLen	UINT(4)	Handler	RPG
ioFeedbackLen	UINT(4)	Handler	RPG
deviceFeedbackLen	UINT(4)	Handler	RPG
numKeys 1	UINT(4)	RPG	Handler
Rrn	UINT(4)	RPG and Handler	RPG and Handler
formLen	UINT(4)	RPG	Handler
formOfl	UINT(4)	RPG	Handler
functionKey	UINT(1)	Handler	RPG

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

Constant Example

```
D QrnOperation_OPEN...
D                 C                   1
D QrnOperation_READ...
D                 C                   4
D QrnOperation_READE...
D                 C                   6
D QrnOperation_CHAIN...
D                 C                   9
D QrnOperation_SETLL...
D                 C                   12
D QrnOperation_UPDATE...
D                 C                   14
D QrnOperation_WRITE...
D                 C                   15
D QrnOperation_DELETE...
D                 C                   16
D QrnOperation_CLOSE...
D                 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

```
QOA_Ds.rpgStatus = *Zero;

select;
when QOA_Ds.rpgOperation = QrnOperation_OPEN;
      QOA_Ds.rpgStatus = Handle_Open();
when QOA_Ds.rpgOperation = QrnOperation_CHAIN;
when QOA_Ds.rpgOperation = QrnOperation_READ;
when QOA_Ds.rpgOperation = QrnOperation_READE;
when QOA_Ds.rpgOperation = QrnOperation_UPDATE;
when QOA_Ds.rpgOperation = QrnOperation_DELETE;
when QOA_Ds.rpgOperation = QrnOperation_WRITE;
when QOA_Ds.rpgOperation = QrnOperation_CLOSE;
      DeAlloc QOA_Ds.stateInfo;
Other;
ENDSL;
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.

userArea

```
//userArea Data structure
  template
D Optional_Parms...
D              Ds
D  Open_attr...
D  Read_attr...
D  Update_attr...
D  Delete_attr...
```

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

stateInfo

```
//stateInfo data structure
  template
D stateInfo_parms...
D              Ds
D  Last_Op...
D                      25   Varying
```

Handling RPG Open

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

RPG Implicit OPEN


```
select;
 when
rpgIO.rpgOperation =
QrnOperation_OPEN;
  rpgIO.rpgStatus =
Handle_Open(rpgIO);
```

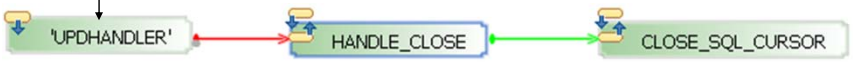
```
If rpgIO.externalFile.library = '*LIBL';
 TableReference = rpgIO.externalFile.name;
Else;
 TableReference =
%TRIMR(rpgIO.externalFile.library) + '/' +
%TRIMR(rpgIO.externalFile.name);
ENDIF;
SQLString =
'SELECT * FROM ' +
%TRIMR(TableReference)              + '
WHERE EMPNO = ? FOR FETCH ONLY';
IF Prepare_SQL_Statement(SQLString) =
*On; retField = 1299;
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

```
 *INLR = *On;
 Return;
```

RPG Implicit CLOSE




```
when
 rpgIO.rpgOperation
 =
 QrnOperation_CLOSE;

 rpgIO.rpgStatus =
 Handle_Close
 (rpgIO);
```

```
Dealloc
 rpgIO.stateInfo;

 Close_SQL_Cursor(
 );
```

```
EXEC SQL CLOSE
  rpgIOInp_C1;
```

Types of Handlers

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

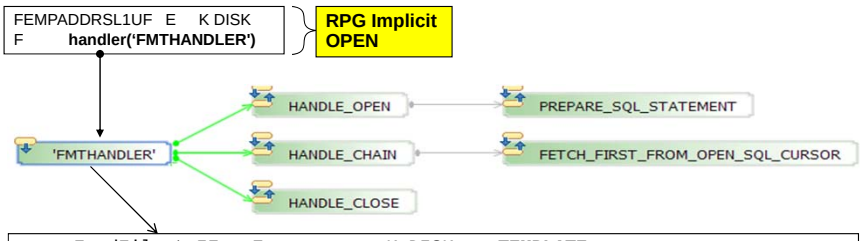
Generic Database Handlers

- 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




```
     FrcdFile_t IF   E           K DISK    TEMPLATE
     F                                     EXTDESC('EMPADDRESS')
     F                                     RENAME(empAddr:rcdFormat)

     D rcdFormat_t...
     D                 DS                  LIKEREC(rcdFormat)
     D                                     TEMPLATE

     D keys_t          DS                  LIKEREC(rcdFormat:*KEY)
     D                                     TEMPLATE

     D Ind_Array_t...
     D                 S              5i 0 DIM(7)
     D                                     TEMPLATE
```


Using a Statement-based Handler

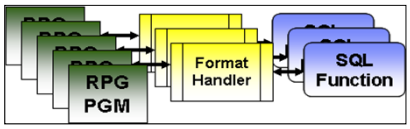
- Used with column-based method, record format varies
  - Allows single handler for multiple formats
- Handler determines ordering via passed key column list
  - System API not required
  - User area can be used to pass entire statements (update, delete, etc)
- An SQL Descriptor must be used in data 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

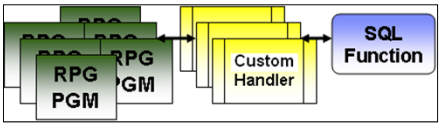
- This type of handler is used when SQL Stored Procedures are already in use by new or external applications
- The SQL Stored Procedure name is passed via the User Area.
  - Best approach for minimal number of handlers
- The handler consumes the result set created by the stored procedure and returns 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

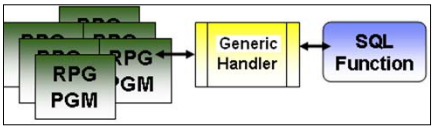
- Format based handlers and data access service programs
  - Most flexible, least manageable



- 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

HLL IO Operation	I5/OS Module
RPG SETxx or COBOL START	QDBGETKY (Position into an index)
RPG CHAIN or READE or COBOL READ (Dynamic)	QDBGETKY (Position into an index followed by random read by RRN)
RPG READ or COBOL READ (Sequential)	QDBGETSQ or QDBGETM (Sequential read of 1 or multiple records respectively. Applies to both keyed and non-keyed reads.)
RPG or COBOL WRITE	QDBPUT or QDBPUTM (insert 1 or multiple records respectively)
RPG or COBOL DELETE	QDBUDR (Delete a record)
RPG UPDATE or COBOL REWRITE	QDBUDR (Update a record)

QDB…

Search Index

Retrieve Record

RPG HLL IO
Open, Close
Set.., CHAIN, READE
READ
WRITE
FEOD
Others…

HANDLER

SQL Service

Possible Handler Scenarios

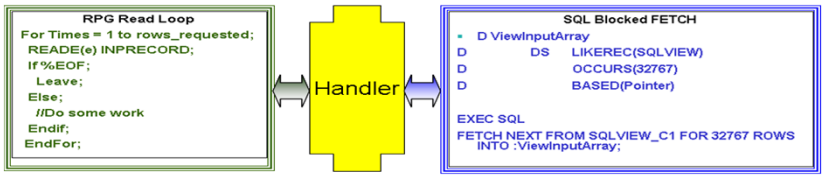
- Areas providing best return on reengineering effort
  - SQL blocked FETCH versus RPG READE loop
  - SQL blocked INSERT versus RPG WRITE loop
  - SQL searched UPDATE/DELETE versus RPG UPDATE/DELETE loop
  - SQL JOIN versus RPG READE then RPG CHAIN
  - SQL MERGE versus HLL Archive/Purge

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

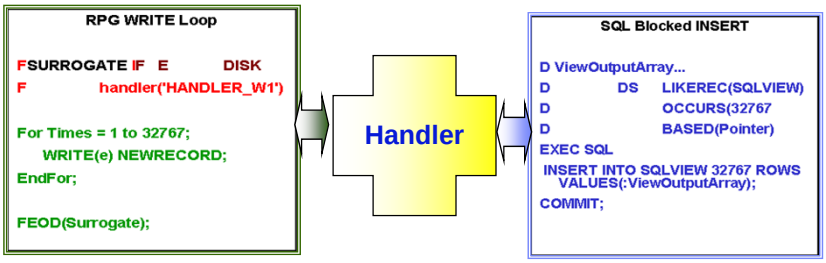
- The SQL format based service program defines the host array based on externalFile
- The host array is populated by the SQL FETCH function on first call
  - The first array occurrence is moved to the inputBuffer
  - Subsequent calls result in the occurrence being incremented and the next row is returned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



Handler performs 1 I/O and populates host array
Rows are returned 1 at a time per RPG READ
Data is CPU bound, not I/O bound

SQL Blocked Insert

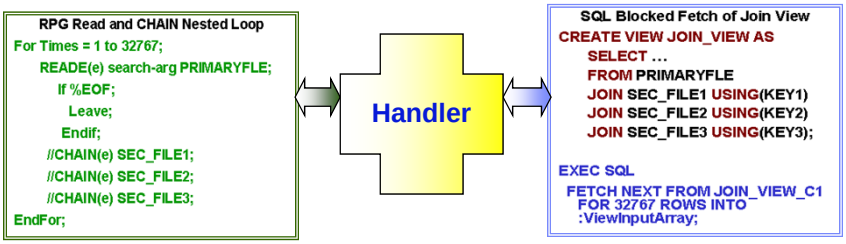
- Opposite of Blocked Fetch
- Huge performance benefits when combined with
  - SMP for parallel index maintenance
  - OVRDBF REUSEDLT (*NO) for bulk inserts



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
FEOD indicates end of logical unit of work

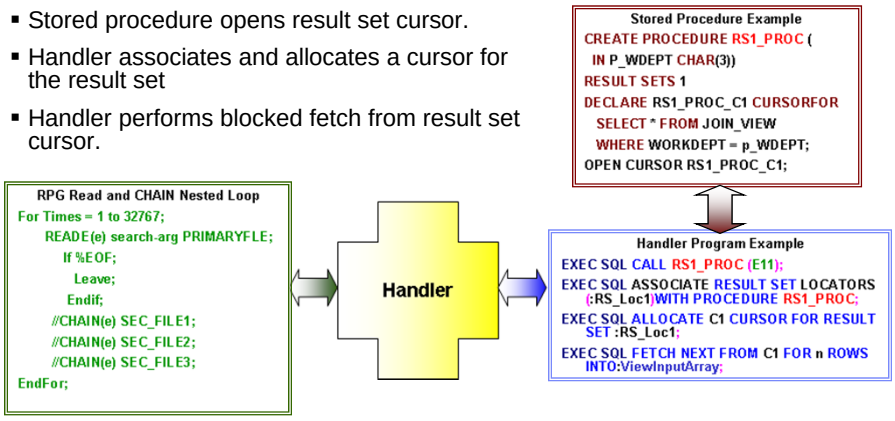
Eliminating Program Joins

- 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

```
Update(E) EMPADDR inpRecord;
If %Error; MESSAGETXT = 'Error-Review Joblog';
Else;
  MESSAGETXT = 'Address Changed Successfully';
ENDIF;
```


```
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

```
        RPG UPDATE/DELETE Loop
FSURROGATE IF  E           DISK
F                handler('HANDLER_W1')
SETLL search-arg UDR_RECORD;
If %EQUAL
  DOU %EOF
    READE(e) search-arg UDR_RECORD;
      If %EOF;
        Leave;
      Endif;
      DELETE(e) UDR_RECORD;
      // OR
      UPDATE(e) UDR_RECORD;
  EndDo;
  FEOD(Surrogate);
EndIf;
```

Handler

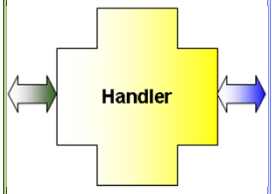
```
        SQL searched UPDATE or DELETE
EXEC SQ
  UPDATE SQL_VIEW SET(COL1) = (value)
  WHERE COL2 = search-arg;
  // OR
  DELETE SQL_VIEW
  WHERE COL2 = search-arg;
  COMMIT;
```

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

```
//Summarize transaction
For 1 to *HIVAL;
  id READE(e) INPRECORD;
  If %EOF;
    Leave;
  Else;
    sum_amount += amount;
  Endif;
 EndFor;
//Check for existence
 id SETLL search-arg UDR_RECORD;
  If %EQUAL;
     UPDATE(e) UDR_RECORD;
  Else;
     WRITE(e) NEWRECORD;
EndIf;
COMMIT;
sum_amount = *Zero;
```



SQL MERGE performed by handler

```
--Summarize 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
--Check for existence
 WHEN MATCHED THEN
   UPDATE SET balance =
                a.balance + t.sum_amount
 WHEN NOT MATCHED THEN
    INSERT (id, balance)
   VALUES (t.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

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