Embedding SQL the IBM i Way

Dan Cruikshank
Agenda

- Getting Started with Embedded SQL
- Writing Flexible and Reusable SQL
- Dynamic SQL
- RPG Open Access
Getting Started with Embedded SQL

- Is There a Difference?
Reasons for Using Embedded SQL in HLL Programs

- Leverage HLL skills with SQL set based processing when accessing large amounts of data
  - Blocked FETCH and INSERT.
- Take advantage of HLL programming constructs not available to DB2 SQL Procedure Language (PSM)
  - Externally described data structures
  - SQL can use descriptors but does support blocked FETCH
- Use ILE concepts
  - Prototyped Procedures
  - Modules
- Rdi Visual Application Diagramming
Rdi Visual Application Diagram (VAD)

You want this!!!
Rules for Embedding SQL in HLL programs

- Each SQL statement must begin with EXEC SQL and end with a semicolon (;)
- The EXEC SQL keywords must be on the same line
  - The SQL statement can be on the same line or multiple lines
- Only 1 SQL statement for each EXEC SQL group
- Refer to the Embedded SQL Programming manual for other language specific restrictions/conventions
Types of SQL

- Embedded SQL is a general term for SQL in an application program and can be either Static or Dynamic
  - Static SQL validated during compilation
  - Dynamic SQL validated during preparation
  - Both produce plans that can be shared and reused
- Both Static and Dynamic can be used in the same program
  - SQL EXEC SELECT COUNT(*) INTO...
  - SQL EXEC EXECUTE IMMEDIATE ‘CREATE ALIAS…’
WRITING FLEXIBLE AND REUSABLE SQL
Writing Flexible and Reusable SQL

A reusable SQL procedure is one that can be called from multiple procedures.

A flexible SQL procedure is a reusable procedure that:
- Imports varying input parameters
- Produces varying output based on that input
- And does it without program modification

The following fall within this definition:
- Integrated Language Environment (ILE)
  - Modules and Service Programs
- Dynamic SQL with simplified SQL descriptor usage
- Flexible views (use of global variable)

There are several new SQL and High Level Language (HLL) enhancements on IBM i that make this possible.
VAD for Integrated Language Environment (ILE)

Flexible Service Program
Prototyped procedures are coded once and reused.

Custom SQL code modules. Bound to program module to create a single program object Easy to clone

Embedded SQL referencing cursors cannot be re-used.
VAD for SQL Modules and Service Programs

- Many SQL statements support host variables
- These statements need only be coded once in a service program
  - Parameters can be used to provide variables
SQL and HLL Enhancements

- Many new enhancements have been made in the last decade that makes coding in SQL more attractive
  - SQL Enhancements
    - Result Set Consumption by HLL
    - Global Variables
    - Simplified SQL Descriptor Usage
    - Named And Default Parameters For SQL Procedures
    - Plus more
  - HLL Enhancements
    - CL RUNSQL command
    - RPG total free format coding
    - RPG Open Access

- Keeping up to date
  - DB2 for i Web Page
    - [http://www-03.ibm.com/systems/power/software/i/db2/index.html](http://www-03.ibm.com/systems/power/software/i/db2/index.html)
  - DB2 for i Technology Updates
SQL Enhancement: Result Set Consumption

- New SQL statements allow simplification of result set consumption by host centric programs using embedded SQL
  - ASSOCIATE LOCATORS and ALLOCATE CURSOR
- Allows common stored procedures for both host centric and external applications
  - Result set can be returned to caller (RPG/COBOL) or client (Java, .NET, PHP, etc.)
- Conceptual use of result set consumption

CALL SQLLE2 ()

- CALL SP1
- ASSOCIATE/ALLOCATE cursor
- FETCH from allocated cursor
- CLOSE cursor when complete

SP1

- DECLARE CURSOR
- OPEN CURSOR WITH RETURN TO CALLER

Data Flow

Result Set
VAD for Result Set Consumption

Procedure names and result set locators can be variables.

Called procedure result set can be consumed by other applications.

Cursor names cannot be variables

Diagnostics and error handling

Service program can contain multiple modules
Using Host Variables

- **Host Variable**
  - Single field commonly used to:
    - specify a value in the predicate of a search condition
    - replace a literal value in an expression
    - Provide a target to receive a value from an INTO clause

- **Host Structure**
  - A group of host variables commonly used to:
    - Specify values for VALUES clause (INSERT), SET clause (UPDATE)
    - Provide a target to receive multiple values from a FETCH or SELECT INTO

- **Host Structure Array**
  - A multiple occurrence data structure or array
    - Commonly used for blocked FETCH and INSERT statements
    - Currently not supported in SQL PL

- **SQL Descriptor**
  - A flexible Host Structure
SQL Enhancements: Global Variables

- SQL Global variables can be used for inter-application communication
  - Similar to HLL Import/Export
- Instantiated in session (job) on first reference
  - Each session has its own global variables and values
- Example of creating a global variable
  - CREATE VARIABLE PROC_NAME VARCHAR(300) DEFAULT NULL;
- Conceptual use of global variable

CALL SQLLE1 ()
- SET PROC_NAME = 'SP1'
- CALL SQLLE2 ();

SQLLE2
- SET vSQL = CONCAT('CALL', PROC_NAME)
- PREPARE CALL_PROC FROM vSQL
- EXECUTE CALL_PROC

SP1
- DECLARE CURSOR
- OPEN CURSOR WITH RETURN TO CALLER

Result
Set
VAD for Global Variables

Application exports global variable.

Reusable service imports global variable.

GetGV may not be required in most SQL Statements as GV can be referenced directly.

Reusable service program module contains setters and getters.
DYNAMIC SQL
Advantages of Dynamic SQL

- Offers a high degree of application flexibility
- You can create/build SQL statement based on parameters received from:
  - Interactive user interface
  - List selection techniques
  - Application control file
  - RPG Open Access
Dynamic SQL Concepts

- Statement is defined and executed at program run time
  - PREPARE then DECLARE dynamic cursor for SELECT
  - PREPARE then EXECUTE for INSERT, UPDATE, DELETE
  - EXECUTE IMMEDIATE for DDL statements (ALIAS, CREATE INDEX, etc)

- Two types of dynamic SELECT statements
  - Fixed list – result columns are predictable and can be pre-defined
  - Varying list – result columns can vary, SQLDA required to define host variables

- Parameter Markers
  - A placeholder (?) for substitution variables used in dynamic SQL statements
SQL Descriptor Areas

What are they?

- An SQL descriptor area is used to contain information about a dynamic SQL statement
- A set of variables are used to communicate this information between SQL and your program
  - Think externally described data structure with a variable file name
- The meaning of the information in the descriptor area depends on the type of statement
  - SELECT or non-SELECT (UPDATE, INSERT, etc.)

Where could they be used?

- Column lists
- Data for VALUES on INSERT or UPDATE
- Data for WHERE clauses
- Example
  sqlString = 'DELETE FROM T1 WHERE EMPNO = ?';
  EXEC SQL PREPARE S1 USING SQL DESCRIPTOR :sqlDescriptor FROM :sqlString;
  EXEC SQL SET SQL DESCRIPTOR :sqlDescriptor DATA = :vEmpno;
  EXECUTE S1 USING :sqlDescriptor;
Simplified SQL Descriptor Usage

- New SQL statements available for passing and receiving variable information from the database
  - ALLOCATE/DEALLOCATE DESCRIPTOR GLOBAL ‘INPUTPARMS’
  - SET/GET DESCRIPTOR GLOBAL
- Global Descriptor similar in concept to Global Variable
- Simplified alternative method to using SQLDA
- Conceptual use of global SQL descriptors

Job/Thread Init
- EXEC SQL ALLOCATE DESCRIPTOR GLOBAL ‘INPUTPARMS’
- EXEC SQL CALL SP1 ()

SP1
- EXEC SQL SET DESCRIPTOR GLOBAL ‘INPUTPARMS’ COUNT
- EXEC SQL SET DESCRIPTOR GLOBAL ‘INPUTPARMS’ VALUE 1
- EXEC SQL CALL SP2 ()

SP2
- EXEC SQL GET DESCRIPTOR GLOBAL ‘INPUTPARMS’ COUNT
- EXEC SQL PREPARE S1
- EXEC SQL DECLARE C1 FROM S1
  - IF count > 0
    - EXEC SQL OPEN C1 USING DESCRIPTOR GLOBAL ‘INPUTPARMS’
  - Else
    - EXEC SQL OPEN CI
  - Endif
The SQL DESCRIBE INPUT statement is used to populate an INPUT descriptor prior to using an SQL EXECUTE statement.
VAD Dynamic SQL Application Overview

Application module

- Custom SQL
- Dynamic SQL
- Setters/ Getters
- Cursors/ Result Sets
- Diagnostics and Error handling
VAD Dynamic SQL – Setters and Getters

1. *INZSR is used to allocate the global descriptor and set the name in a global variable
2. The SQL string is set as part of the transaction process
   INSERT INTO t1 (c1,c2) VALUES(?,?,?)
3. SQL string and descriptor name (4) are imported by service program
VAD Dynamic SQL Global Descriptors and Variables

4. Transaction is Add, Change or Delete

1. *INZSR
2. Number of parameter markers
3. SET DATA and INDICATOR for each parameter

5. Prepare the imported SQL string
6. Execute the string using the global descriptor values
Named And Default Parameters For SQL Procedures

- Extend procedures without breaking existing callers
- Simplify calling requirements by adding default values
  - Parameters may be omitted or specified in any order
  - Default can be expressions or global variables
- Examples of using named and default parameters
  - CALL SQLLE1 ('D21'); CALL SQLLE1 (PARM2 => 16);
- Conceptual usage:

```plaintext
CALL SQLLE1 (PARM2 => 16);

• IN PARM_COUNT INTEGER
• IN PARM1 DEFAULT NULL
• ALLOCATE DESCRIPTOR GLOBAL
• SET DESCRIPTOR PARM_COUNT
  • If parm_count > 0
    • SET DESCRIPTOR VALUE(1) DATA(PARM1)
  • End if
• SET PROC_NAME = 'SP1'
• CALL SQLLE2 ()

SQLLE2

• SET vSQL = CONCAT('CALL', PROC_NAME)
• PREPARE CALL_PROC FROM vSQL
• EXECUTE CALL_PROC

SP1

• GET DESCRIPTOR GLOBAL count
• PREPARE S1
• DECLARE C1 FROM S1
  • IF count > 0
    • OPEN C1 USING descriptor
  • Else
  • OPEN CI
  • Endif

Result Set
```
RPG OPEN ACCESS
Open Access Structure

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

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

- Open Access is part of the RPG compiler
  - Runtime PTFs:
    - V6R1: SI45906
    - V7R1: SI45905
  - Compiler PTFs:
    - V6R1: SI45904
    - V7R1 TGTRLS(*CURRENT): SI45903
    - V7R1 TGTRLS(*PRV): SI45902
RPG HANDLER Keyword

New file level keyword HANDLER
no additional syntax related to using an Open Access file.

<table>
<thead>
<tr>
<th>FILE1</th>
<th>IF</th>
<th>E</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>handler('HANDLERPGM')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILE2</th>
<th>IF</th>
<th>E</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>handler('HANDLERSVP(Handle_OPNQRYF)')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Handler can be program or service program

/Free
Setll(e) FILE1;
READE(e) FILE1;
CHAIN(e) FILE2;
Update FILE1;
/End-Free

RPG IO operations coded as usual

1 Line of code per file
The HANDLER Program

/COPY QOAR/QRPGLESRC,QRNOPENACC

DCL-PR Handle_Main ;
   rpgIO LIKE D(QrnOpenAccess_T);
   userArea LIKE D(userArea_t)
      OPTIONS(*NOPASS) ;
END-PR ;

DCL-PR Prepare_Then_Execute_CALL IND;
DCL-PR Call_Stored_Procedure IND;
DCL-PR Allocate_Cursor ;
DCL-PR Associate_Result_Set_Locator IND
DCL-PR Fetch_From_Allocated_Cursor IND;
DCL-PR Return_Row_From_Host_Array IND;
DCL-PR Close_SQL_Cursor END-PR ;

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

Determine SQL function based on I/O operation
So What's Stopping You?

- Simply change the HLL read operation to an SQL blocked FETCH
- Nobody wants to touch the program.
- Why???
- Now what?

*Rdi Visual Application Diagram 24k line RPG Module Using OPNQRYF*
Handler’s to the Rescue

- **CL Pgm**
  - RUNSQL
  - SET PROCEDURE NAME global variable
  - CALL HLL_PROGRAM

- **HLL Pgm**
  - HANDLER keyword assigned to file

- **Handler**
  - OPEN Operation
    - GET PROCEDURE NAME from global variable
    - First IO Operation
    - CALL SQL Stored Procedure
    - ASSOCIATE/ALLOCATE cursor

- **SQL Stored Proc**
  - DECLARE CURSOR
  - OPEN CURSOR WITH RETURN TO CALLER

- **Handler**
  - FETCH result set from allocated cursor into host array
  - CLOSE cursor when complete
  - Return Row from host array to HLL Program

- **HLL Pgm**
  - Read rows until complete

- **SET Global Variables**

- **GET Global Variables**

- **DB2 for i Table**

- **Result Set**
HLL Enhancements: CL RUNSQL command

- RUNSQL CL Command allows executing SQL strings without requiring a source file
  - SQL string is limited to 5000 bytes
  - SQL string cannot be used with SELECT statements
  - CL program must be ILE to use advanced SQL (e.g. global variables)

- Example of RUNSQL in CL Program
  - \texttt{RUNSQL SQL('SET SCHEMA USER') ;}

- Conceptual use of RUNSQL
Open Access provides a unique opportunity for RPG programmers to access SQL procedures with minimal change to existing programs.

- Not available for COBOL

Opens up RPG’s file I/O capabilities allowing programmers to code their own programs (in any host language) to handle RPG operations.

I/O handlers can be used to:

- Transform traditional record at a time I/O operations to SQL set based operations
- Take advantage of SQL enhancements

Conceptual use of Open Access

CALL CLPLE ()
• RUNSQL (SET PROC_NAME = &PROCNAME)
• CALL RPGLE1

RPGLE
• HANDLER (SQLLE)
• 1 Line of code change

HANDLER/SQLLE
• Provides bridge between RPG and SQL
• SET vSQL = CONCAT('CALL', PROC_NAME)
• PREPARE CALL_PROC FROM vSQL
• EXECUTE CALL_PROC
• ASSOCIATE/ALLOCATE Cursor
• FETCH/CLOSE

SP1
• DECLARE CURSOR
• OPEN CURSOR WITH RETURN TO CALLER

Result Set

Data Flow
RPG Visual Diagram for a Procedure-based Handler

It all starts with
The addition of
one line of code!
Summary

- Now is the time to move to SQL
- New enhancements in SQL development has made Data Centric Programming very attractive
- RPG Open Access combined with result set consumption have simplified the process of bridging to SQL
  - Only 1 line of code needs to be added to existing RPG program
  - COBOL programmers can use these same techniques but will have to change more lines of code
- Modern development tools and methods have dramatically improved ease of creating and maintaining SQL
  - IBM Data Studio
  - Rational Developer for i
Questions?
Thank you!