



RPG CLI:

**Using SQL (ODBC)
with RPG**

Craig Pelkie

Bits & Bytes Programming, Inc.

craig@web400.com

What We'll Cover ...

- **What is CLI and why use it with RPG?**
- **The basics of CLI**
- **Using CLI APIs with RPG**
- **Resources for working with CLI**

What is CLI and why use it with RPG?

- **CLI = Call Level Interface**
- **SQL interface, alternative to embedded SQL**
- **Open Database Connectivity (ODBC)**
 - **Connect to database**
 - **Get information about the database and its objects**
 - **Run dynamic SQL statements, work with result sets**

What is CLI and why use it with RPG?

- **Can use CLI instead of embedded SQL**
- **Does not require DB2 Query Manager and SQL Developers Kit (5722-ST1)**
- **Does not require CRTSQLRPG / CRTSQLRPGI commands**
- **Can use all RPG techniques (free-format, includes, etc.)**

The Basics of CLI Programming

- **Basic flow:**
 - **Initialize**
 - **Connect to database(s)**
 - **Run SQL statement(s), work with results**
 - **Close**
- **Each activity is supported by one or more API calls**
- **Most API calls are simple, but are dependent upon previous calls**

CLI APIs

- **81 APIs (function calls) in 6 main categories**
 - **Handles (identifiers used with APIs)**
 - **Connect**
 - **Attributes (get/set)**
 - **Process SQL statements**
 - **Metadata ("data about data")**
 - **Diagnostics**

Where are these APIs located?

- **Service program QSYS/QSQCLI (DSPSRVPGM QSQCLI)**
- **Prototypes are in QSYSINC/QRPGLESRC(SQLCLI)**
 - **Only problem:** nothing in it
- **Required prototypes are in QSYSINC/H(SQLCLI)**
 - **Only problem:** all in C
- **RPG prototypes:**

<http://www.web400.com/Penton/rpgcli.html>

What's needed to run CLI in RPG?

- **Example: program to run an SQL statement**
- **11 APIs (not 80)**
 - **3 APIs – to make connection ***
 - **4 APIs – to run statement, get results**
 - **4 APIs – to disconnect, close ***
- *** can be easily put into initialization / termination procedures**

Example Program – EX01, make connection

```
//*****
// allocate environment, connection
//*****
rc = SQLAllocEnv(henv);                (Handle to Environment)

rc = SQLAllocConnect(henv : hdbc);     (Handle to Database Connection)

//*****
// connect using default user ID / password
//*****
szDSN = 'M270';                        (String, null-terminated)
cbDSN = SQL_NTS;                       (Count-of-bytes, Null-terminated string)

szUID = '';                             (Use job user ID)
cbUID = SQL_NTS;

szAuthStr = '';                         (Use job password)
cbAuthStr = SQL_NTS;

rc = SQLConnect(hdbc :                (rc – return code)
                szDSN :
                cbDSN :
                szUID :
                cbUID :
                szAuthStr :
                cbAuthStr);
```

Example Program – EX01, run SQL statement

```

//*****
// allocate / execute statement
//*****
rc = SQLAllocStmt(hdbc : hstmt);

szSqlStr = 'SELECT * FROM QIWS.QCUSTCDT ORDER BY BALDUE DESC';
cbSqlStr = SQL_NTS;

rc = SQLExecDirect(hstmt      :
                  szSqlStr    :
                  cbSqlStr);

//*****
// bind columns
//*****
rc = SQLBindCol(hstmt      :
               1          :
               SQL_DECIMAL :
               ptrCusnum   :
               1536       :
               pcbValue);

rc = SQLBindCol(hstmt      :
               2          :
               SQL_CHAR    :
               ptrLstnam   :
               8           :
               pcbValue);

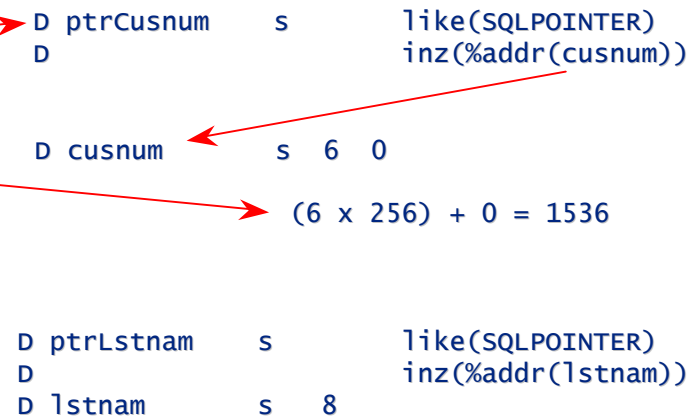
//*****
// process all rows in result set
//*****
dow (SQLFetch(hstmt) = SQL_SUCCESS);
    except exdetail;
enddo;

```

(Handle to Statement)

(Execute the statement)

(data from result set to RPG fields)



(read rows, now have data in RPG fields)

Example Program – EX01, disconnect/close

```
//*****  
// end of program processing  
//*****  
rc = SQLFreeStmt(hstmt : SQL_DROP);  
  
rc = SQLDisconnect(hdbc);  
  
rc = SQLFreeConnect(hdbc);  
  
rc = SQLFreeEnv(henv);
```

**Free / Disconnect in reverse order of
allocate / connect**

More About the APIs

- **What if a column value is null?**
 - Value returned in pcbValue (last parameter of **SQLBindCol**) is set to **SQL_NULL_DATA**
- **What if there are errors?**
 - rc (return code) is set to other than **SQL_SUCCESS (0)**
 - Call **SQLGetDiagRec** to get diagnostic information
 - **SQL State (5 char)**
 - **Native error code**
 - **Error message**

Another Option: Prepared Statement

- **SQL SELECT statements are frequently run multiple times**
- **What varies are WHERE clause values**
- **(also INSERT, UPDATE statement values)**
- **Statement can be "prepared" in advance, parameter markers indicate variable data**

```
D sql...
```

```
D          c          'SELECT * FROM QIWS.QCUSTCDT -  
D          WHERE BALDUE >= ? -  
D          ORDER BY BALDUE DESC'
```

? used as parameter marker, value for BALDUE provided at run-time

A statement can have as many ? parameter markers as needed

Example Program – EX02, prepare statement

```
D sql...
D          c          'SELECT * FROM QIWS.QCUSTCDT -
D          WHERE BALDUE >= ? -
D          ORDER BY BALDUE DESC'
```

```
D baldueParm...
D          s          6 2
D ptrBaldueParm...
D          s          like(SQLPOINTER)
D          inz(%addr(baldueParm))
```

```
/**
// prepare the statement (do this once)
**
```

```
rc = SQLAllocStmt(hdbc : hstmt);
```

```
szSqlStr = sql;
cbSqlStr = SQL_NTS;
```

```
rc = SQLPrepare(hstmt :
                szSqlStr :
                cbSqlStr);
```

At SQLPrepare, the statement is not executed. It is sent to the database to be prepared (an access plan can be computed).

Example Program – EX02, prepare statement

```
/**
// bind parameter (WHERE clause, BALDUE), execute
// (do each time parameter value changes)
**
baldueParm = 100.00;
```

```
rc = SQLBindParameter(hstmt :
                        1      :
                        SQL_PARAM_INPUT :
                        SQL_DECIMAL  :
                        SQL_DECIMAL  :
                        6              :
                        2              :
                        ptrBaldueParm :
                        0              :
                        strlen);
```

Value is bound to the parameter marker identified by the number (1).

```
rc = SQLExecute(hstmt);
```

SQLExecute is used to execute a prepared statement.

SQLExecDirect is used to execute non-prepared statements.

7 Key Points to Take Home

- **For simple RPG/SQL tasks, use embedded SQL (if SQL Developer Kit is available)**
- **Become familiar with C-style API and parameter notation**
- **Become familiar with constants and defines in the SQLCLI include member**
- **Create subprocedures or subroutines to encapsulate connect and disconnect tasks**

7 Key Points to Take Home

- **Become familiar with pointers and %addr function**
- **Use prepared statements where possible**
- **Test the return code after each API, call SQLGetDiagRec if return code is not zero**

Resources

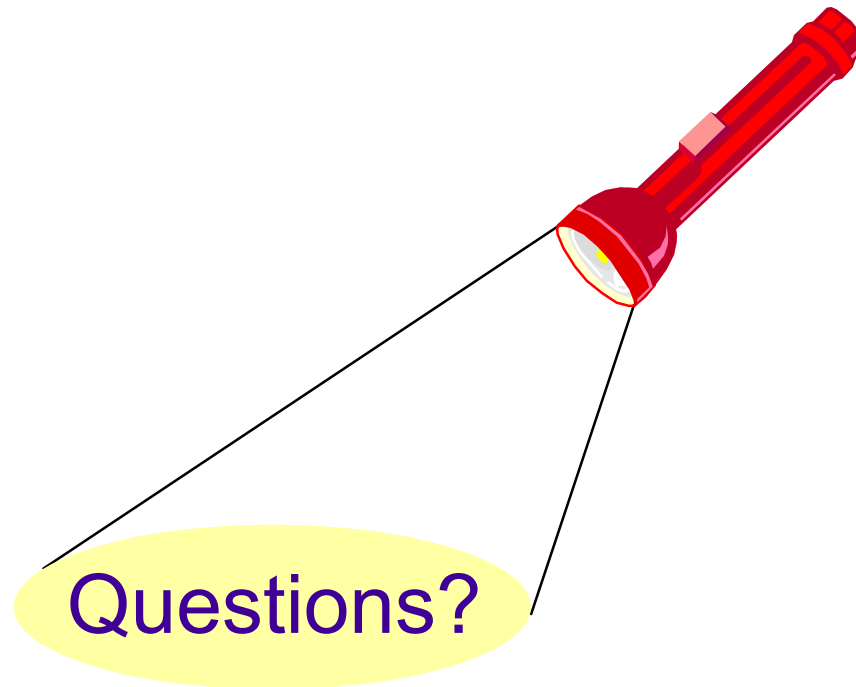
IBM manual on iSeries InfoCenter site:

**iSeries DB2 Universal Database for iSeries SQL
Call Level Interface (ODBC) V5R3**

**DB2 Universal Database Call Level Interface
Developer's Guide (Roger E. Sanders, McGraw-Hill,
ISBN 0-07-134572-8)**

**Microsoft ODBC 3.0 Software Development Kit and
Programmer's Reference (Microsoft Press)**

Contact Info



**How to Contact Me:
Craig Pelkie
craig@web400.com**