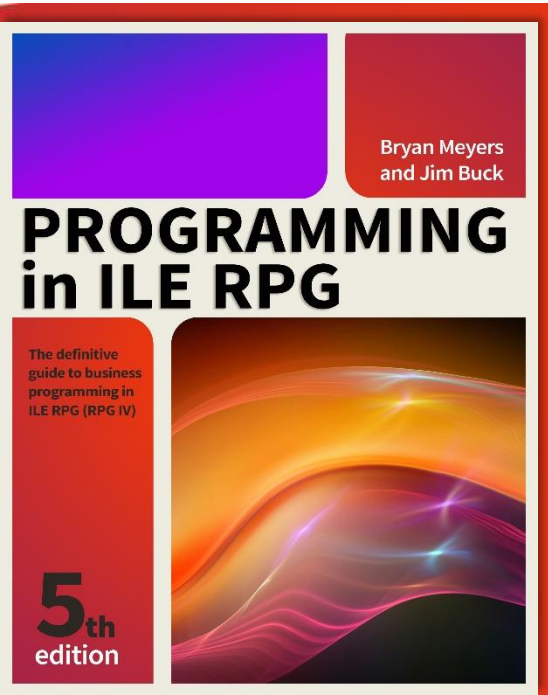
Cursored Again!
Using SQL & CURSORS In Your Programs



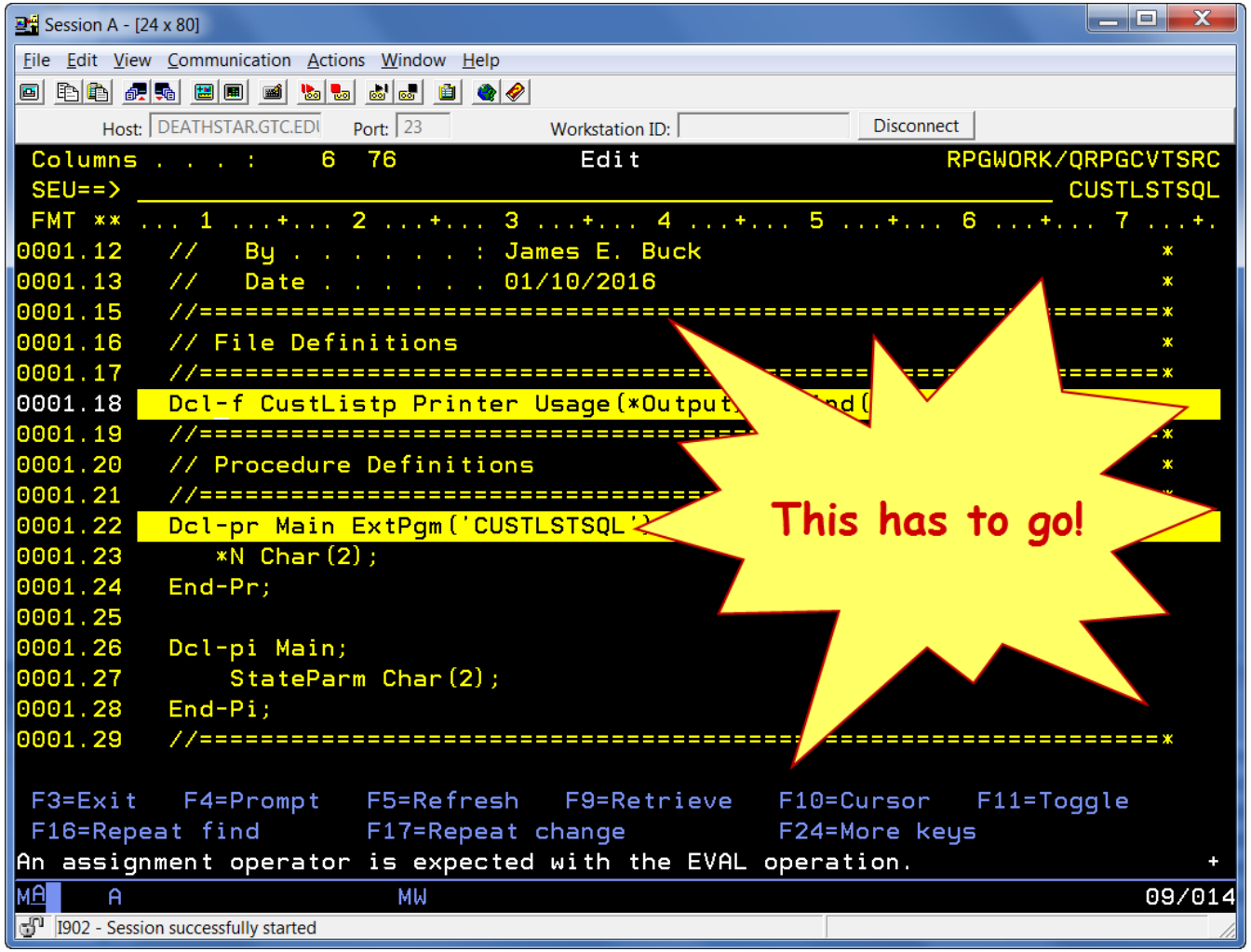
Jim Buck
Phone 262-705-2832
jbuck@impowertechnologies.com
Twitter - @j_buck51

5250 & SEU – Doesn't work anymore!

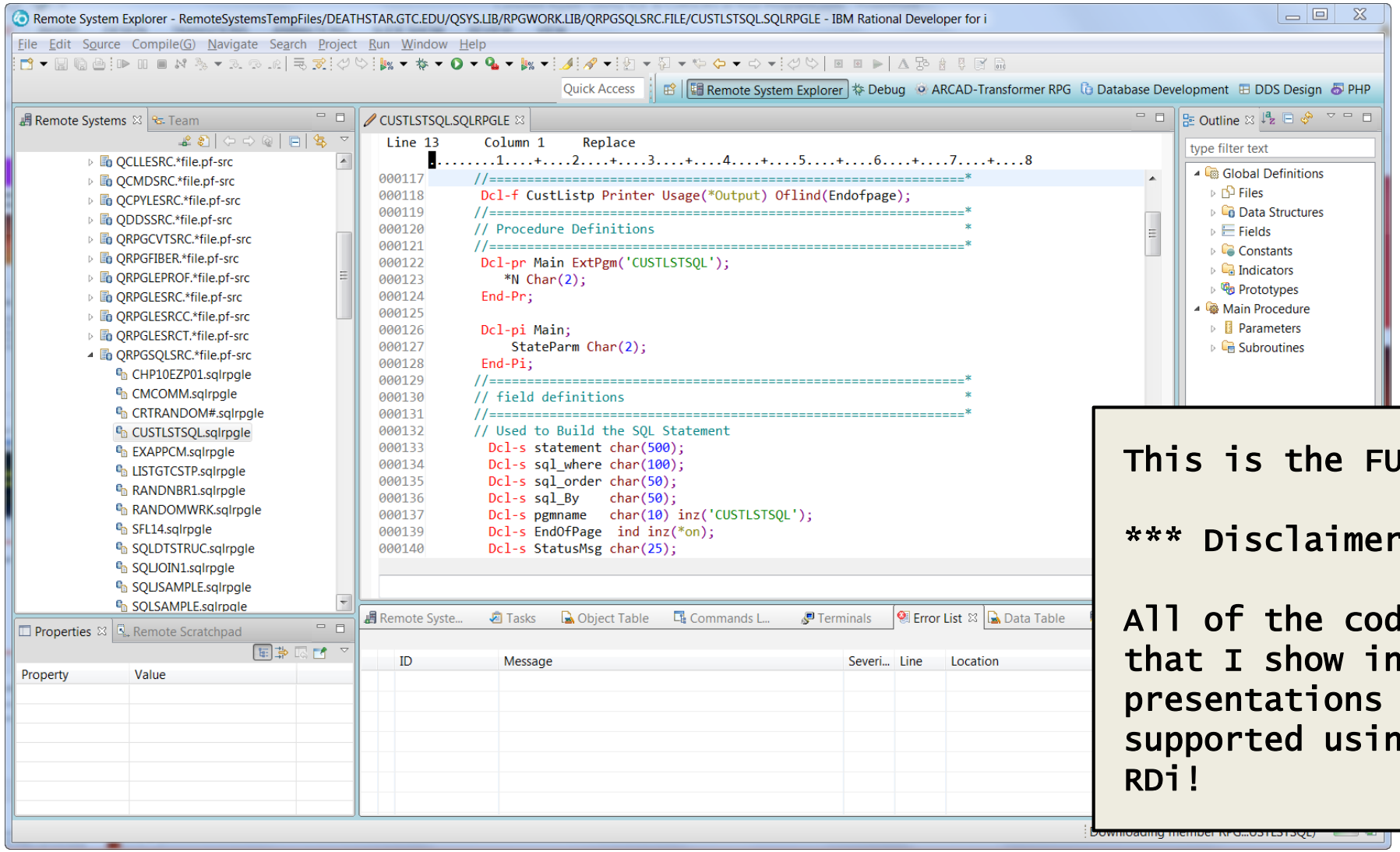
SEU doesn't support the latest version of RPG.

Well I guess, you could turnoff Syntax Checking!

My students have a short introduction… in case of emergencies!



Rational Developer for i – 9.5.1.2



SQL Data Manipulation Language (DML)

- DML statements retrieve and manage data within tables (physical files)
- DML statements appropriate for embedded SQL
  - Select
  - Insert
  - Update
  - Delete

Legacy Code: /Exec SQL or

We have come a long way!

- Fixed format /Exec SQL compiler directive
  - Must have “C/” in positions 6-7
- SQL statement may continue onto subsequent lines
  - Must have “C+” in positions 6-7
- /End-exec compiler directive ends statement
  - No semicolon required

```
C/Exec SQL
C+  Delete From Customers
C+   Where Czip = '60606'
C/End-exec
```

Select

- SQL's primary query statement
  - Select clause lists columns to retrieve
    - Or specifies with an asterisk (*) all columns, in order, from the record layout
    - From clause indicates file(s) from which columns are retrieved
    - Where clause (optional) can identify a search condition
    - Order by clause (optional) sequences result set

```
Select Custno, Cfname, Clname From Customers Where Custno = 'AB0097531'
Select * From Customers Where Custno = 'AB0097531'
Select * From Customers Where Czip = '10010' Order by Custno
```

Insert

- Insert statement adds record (row) to table or view
  - Approximately equivalent to RPG Write
- Insert Into clause names table
- Column list (optional) in parentheses specifies columns for which values will be provided
- Values clause supplies values for each column in column list
  - If column list is omitted, Insert assumes all columns in layout have corresponding values in Values clause

```
Insert Into Customers
  (Custno, Cfname, Clname, Caddr, Czip, Cphone, Cemail, Cdob, Cgender)
  Values ('AB0097532', 'John', 'Smith', '123 Main St', '60606',
          '3125555432', 'johnsmith431@mail.com', 19750701, 'M')
```

Update

- Update statement modifies row(s) in table
  - Corresponds to RPG's Update operation combined with %Fields function
- Set clause assigns value(s) to column(s)
  - Or use column list with Values clause
- Where clause identifies rows to modify

```
Update Customers Set Czip = '61443' Where Custno = 'AB0097532'

Update Customers (Cphone, Cemail) Values ('3095559753', 'jsmith75@mail.com')
  Where Custno = 'AB0097532'
```

Delete

- Delete statement removes row(s) from table
  - Analogous to RPG Delete operation
- Where clause selects row(s) to delete
  - Without Where clause, deletes all rows in table

```
Delete From Customers Where Custno = 'AB0097533'

Delete From Customers Where Czip = '60606'
```

Dynamic vs Static SQL

Static SQL

- Basic structure of each SQL statement is when program is compiled
- Can use host variables to substitute values at runtime
- General purpose and construction don’t change after program is created
- Faster than dynamic SQL

Dynamic SQL

- Program can build SQL statement as character string using program data values
- SQL statement is prepared at runtime
- Does not use host variables
  - Substitutes parameter values
- More flexible than static SQL

Introduction to Embedded SQL

- SQL statements can complement or entirely replace native file operations in RPG program
- Useful for *set-at-a-time* processing – **The Real Power of SQL!**
  - Program acts upon many records using single SQL statement
- Database features unavailable with DDS or through native RPG file operations
  - Example: security at field, or column, level

Introduction to Embedded SQL – cont.

- Functions not directly available to RPG
  - Examples: Avg, Count, Sum
  - Date calculations
- Flexibility in database processing
  - Example: Dynamic SQL statement execution
- Cross platform consistency
- ~~Possible~~ Performance improvements

Exec SQL

- Exec SQL directive signals rest of the line is embedded SQL statement
  - Precompiler automatically converts SQL statement into appropriate RPG operations before program is created
- If program uses SQL to process file, Dcl-f instruction is not necessary
  - Dcl-f required only for native file I/O operations (Read, Write, etc.)
- Not all SQL statements are allowed
  - Some require modification
  - Some are allowed only as embedded statements in a program

```
Exec SQL Delete From Customers Where Czip = '60606';

Exec SQL Update Customers (Cphone, Cemail)
                          Values ('3095559753', 'jsmith75@mail.com')
                          Where Custno = 'AB0097532';
```

Using Host Variables

- Host variable is data item that SQL statement uses to synchronize RPG program variables with data from tables and views
  - Can substitute host variables for explicit values in SQL statement
- Host variable name is preceded by colon (:)
  - Name corresponds to declared RPG program variable
  - RPG variable should have same data type and size as associated database column
  - Name cannot begin with SQ, SQL, RDI, or DSN

```
Exec SQL Update Customers (Cphone, Cemail) Values (:Cphone, :Cemail)
                          Where Custno = :Custno;
```

Select Into

- Modified form of Select statement to retrieve result set into host variables
- Retrieves single row and places result set into RPG host variables
  - If result set includes more than one row, SQL returns exception code

```
Dcl-s Custno Char(9);
Dcl-s Cfname Char(15);
Dcl-s Clname Char(20);
…
                          // Program will provide a value for Custno.
Exec SQL Select  Cfname, Clname
           Into  :Cfname, :Clname
           From  Customers
           Where Custno = :Custno;
                          // Cfname and Clname will contain result set values.
```

Using Host Structures

- Host structure is data structure used with Select Into

```
Dcl-s Custno Char(9);

Dcl-ds CustdataDS;
  Cfname Char(15);
  Clname Char(20);
End-ds;
…
                          // Program will provide a value for Custno.
Exec SQL Select  Cfname, Clname
           Into  :CustdataDS
           From  Customers
           Where Custno = :Custno;
                          // Custdata subfields will contain result set values.
```

Using Host Structures

- Useful for retrieving all columns from record layout
  - Externally described data structure is appropriate here
  - If an externally data structure is ***NOT*** used, care should be taken to match the retrieved fields with the data structure (order, number and type)!
  - **BE CAREFUL** using an Asterisk (*) in Select statements. What happens to you program if the file changes?

```
Dcl-s Custno Char(9);

Dcl-ds CustomersDS Ext ExtName('CUSTOMER') Qualified End-ds;
…
                          // Program will provide a value for Custno.
Exec SQL Select * Into  :CustomersDS From  Customers
          Where Custno = :Custno;
                          // Customers subfields will contain result set values.
```

Handling Null Values

- Ctl-Opt Option(*NoDebugIO) alwnull(*usrctl);
- Indicator variable detects null column values
  - Defined as signed integer, five digits
  - Value -1 indicates null value
- To detect null value, include indicator variable following host variable

```
Dcl-s Custno     Char(9);
Dcl-s Cfname     Char(15);
Dcl-s Clname     Char(20);
Dcl-s NullCfname Int(5);
Dcl-s NullClname Int(5);
…
Exec SQL Select  Cfname, Clname
           Into  :Cfname :NullCfname,
                 :Clname :NullClname
           From  Customers
           Where Custno = :Custno;
```

Handling Null Values

- To set column to null value, use SQL Update statement, setting indicator variable to -1

```
Cfname = *Blanks
Clname = *Blanks
NullCfname = -1;
NullClname = -1;
Exec SQL Update Customers
           Set   Cfname = :Cfname :NullCfname,
                 Clname = :Clname :NullClname
           Where Custno = :Custno;
```

Handling Null Values

- If SQL statement uses host structure, it can also organize indicator variables into indicator structure
  - Data structure that uses indicator variables as subfields
- Indicator structure must contain corresponding subfield for each column in result set
  - Even if column is not null capable

Handling Null Values

```
Dcl-s Custno Char(9);

Dcl-ds CustdataDS;
  Cfname Char(15);
  Clname Char(20);
End-ds;

Dcl-ds CustnullsDS;
  NullCfname Int(5);
  NullClname Int(5);
End-ds;
…
          // Program will provide a value for Custno.
Exec SQL Select  Cfname, Clname
           Into  :CustdataDS :CustnullsDS
           From  Customers
           Where Custno = :Custno;
                    // CustdataDS subfields will contain result set values.
                    // CustnullsDS subfields will contain indicator variables.
```

Handling SQL Results – *RPG Doesn't Care!*

- Important for the RPG programmer to:
  - Check the results of an SQL statement – Error or Success
  - Check results of "Set Processing"
    - Example *Number of Rows* Returned used to load subfile
- Two ways to check the results:
  - SQL Communication Area (SQLCA) data structure – **Okay**
    - Limited amount of information can be returned
  - GET DIAGNOSTICS Statement - **Best**
    - More than a 100 possible values

Handling SQL Return Codes

- SQLCA Data Structure is updated every time program executes an SQL statement
- Subfields Sqlcode and Sqlstate signal success or failure
  - Sqlcode values are specific to IBM i
  - Sqlstate values are industry standards
- Subfield Sqlerrd contains additional diagnostic messages

Handling SQL Return Codes

- If SQL statement ends in error, RPG program does not stop!
- Diagnostic return codes are in SQL Communication Area (SQLCA) data structure
  - Automatically created for SQLRPGLE programs

```
Dcl-ds Sqlca;
  Sqlcaid  Char(8);          // (Null)
  Sqlcabc  Int(10);          // Length of Sqlca: 136
  Sqlcode  Int(10);          // SQL return code
  Sqlerrml Int(5);           // Length of Sqlerrmc
  Sqlerrmc Char(70);         // Message replacement text
  Sqlerrp  Char(8);          // Product identifier: "QSQ…"
  Sqlerrd  Int(10) Dim(6);   // Diagnostic information
  Sqlwarn  Char(1) Dim(11);  // Warning flags
  Sqlstate Char(5);          // SQL standard return code
End-ds;
```

Handling SQL Return Codes – IBM i STD.

- Sqlcode = 0
  - SQL statement was successful (may be warnings)
- Sqlcode = 100
  - No row was found (end-of-file)
- Sqlcode > 0 (but not 100)
  - Sql statement was successful, but warnings were issued
- Sqlcode < 0
  - SQL statement was unsuccessful
- Sqlcode value corresponds to IBM i message
  - Sqlcode 100 = message SQL0100
  - Sqlcode -0313 = message SQL0313

Handling SQL Return Codes – Industry STD.

- Sqlstate = 00xxx
  - SQL statement was successful (no errors or warnings)
- Sqlstate = 02xxx
  - No row was found (end-of-file)
- Sqlstate = 01xxx
  - SQL statement was successful, but warnings were issued
- Any other Sqlstate
  - SQL statement was unsuccessful

Handling SQL Return Codes

```
Exec SQL Select  Cfname, Clname
           Into  :Cfname :NullCfname,
                 :Clname :NullClname
           From  Customers
           Where Custno = :Custno;

Select;
  When Sqlstate = '00000';
    // Select was successful
  When Sqlstate = '02000';
    // End-of-file
  When %Subst(Sqlstate:1:2) = '01';
    // Select generated warnings
  Other;
    // Select was unsuccessful
Endsl;
```

Handling SQL Return Codes

```
Dcl-c Endoffile    '02000';
Dcl-c Success      '00000';
Dcl-c Warning      '01';
…
Exec SQL Select  Cfname, Clname
           Into  :Cfname :NullCfname, :Clname :NullClname
           From  Customers
           Where Custno = :Custno;
Select;
  When Sqlstate = Success;                  // Sqlstate 00000
      // Select was successful
  When Sqlstate = Endoffile;                // Sqlstate 02000
      // End-of-file
  When %Subst(Sqlstate:1:2) = Warning;      // Sqlstate 01xxx
      // Select generated warnings
  Other;
      // Select was unsuccessful
Endsl;
```

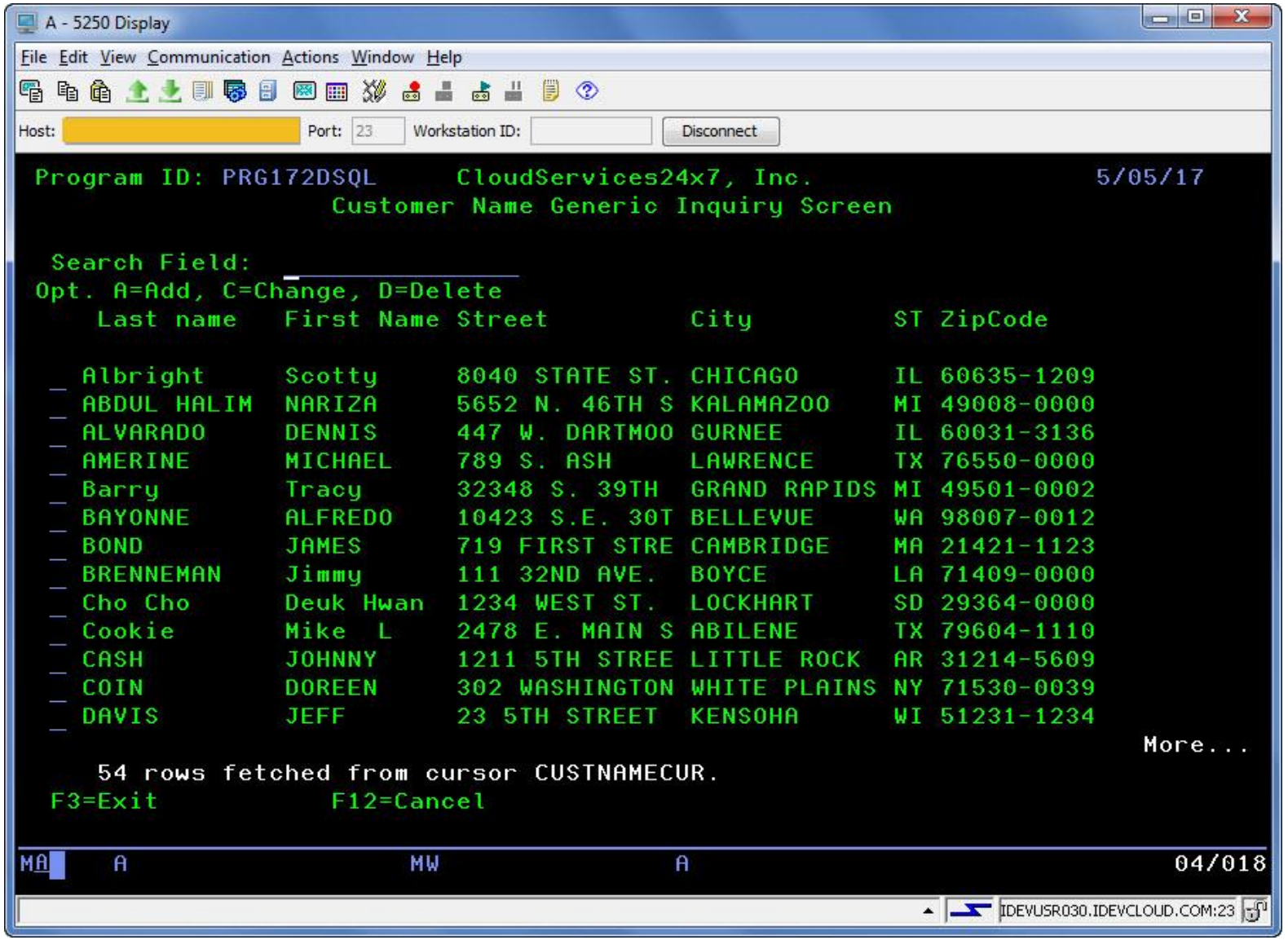
Handling SQL Return Codes

SQLERR(3) is a subfield of SQLCA Data Structure

- For INSERT, MERGE, UPDATE, REFRESH, and DELETE, shows the number of rows affected.
- For a TRUNCATE statement, the value will be -1.
- For a FETCH statement, SQLERRD(3) contains the number of rows fetched.

```
295 D  SQLERR                    24A
296 D    SQLER1                   9B 0 OVERLAY(SQLERR:*NEXT)
297 D    SQLER2                   9B 0 OVERLAY(SQLERR:*NEXT)
298 D    SQLER3                   9B 0 OVERLAY(SQLERR:*NEXT)
299 D    SQLER4                   9B 0 OVERLAY(SQLERR:*NEXT)
300 D    SQLER5                   9B 0 OVERLAY(SQLERR:*NEXT)
301 D    SQLER6                   9B 0 OVERLAY(SQLERR:*NEXT)
302 D    SQLERRD                 10I 0 DIM(6)  OVERLAY(SQLERR)
```

Subfile Application Example



Subfile Application Example

- Comprised of three programs
  - **GETSQLDIAG – Service program that:**
    - Processes the GET DIAGNOSTICS command
    - Puts the results into a data structure
    - Returns this data structure to the calling program
  - CUSTSRVPGM – Service program that handles SQL I/O
    - SQL INSERT, UPDATE, SELECT and DELETE Code
    - Returns data Structures (Customer and SQL Status)
  - PRG175DSQL – Main Driver Program
    - Runs the 5250 screens
    - Handles the CREATE, READ, UPDATE and DELETE Logic

Using GET DIAGNOSTICS Statement

Start using GET DIAGNOSTICS

- Returns additional information on the last SQL statement
- Superset of SQLCA
- Over 100 values can be returned
- SQLCA is limited in size (136 Byte)
  - Many of the return values are truncated

Using GET DIAGNOSTICS Statement

Sample Service Program

```
Ctl-opt nomain Option(*NoDebugIO:*SrcStmt:*NoUnRef);
 //-------------------------------------------------//
 //   Service Program: GETSQLLDIAG                  //
 //   Procedure:  Getdiagnostics                    //
 //   Service program to capture and return the results //
 //   of an SQL Statement                           //
 //-------------------------------------------------//
 Dcl-Pr GetDiagnostics;
   *n LikeDS(UtilDSSQL);
 End-Pr ;
```

Using GET DIAGNOSTICS Statement

Data Structure

Returns information on the last SQL statement

```
Dcl-Ds UtilDSSQL Qualified inz;
   MessageId Char(10);
   MessageId1 VarChar(7);
   MessageId2 VarChar(7);
   MessageLength int(5);
   MessageText varchar(32740);
   ReturnedSQLCode int(5);
   ReturnedSQLState char(5);
   RowsCount int(10);
   RowsReturned Zoned(31:0);
   ResultSetNoOfRows Zoned(31:0);
End-Ds;
```

Using GET DIAGNOSTICS Statement

DB2_MESSAGE_ID - Message ID corresponding to the MESSAGE_TEXT.
DB2_MESSAGE_ID1 - CPF escape message that originally caused this error or an empty string is returned.

DB2_MESSAGE_ID2 - CPD diagnostic message that originally caused this error or empty string is returned.

MESSAGE_LENGTH - Length of the message text of the error, warning, or successful completion.
MESSAGE_TEXT - - Message text of the error, warning, or successful completion. If the SQLCODE is 0 an empty string is returned.

DB2_RETURNED_SQLCODE - Contains the SQLCODE.
RETURNED_SQLSTATE - Contains the SQLSTATE.
DB2_SQLERRD3 - Value of SQLERRD(3) from the SQLCA.

Using GET DIAGNOSTICS Statement

```
Dcl-proc GetDiagnostics Export;
   Dcl-Pi *N;
     DiagUtilDS LikeDS(UtilDSSQL);
   End-Pi ;

   Clear DiagUtilDS;

   Exec sql GET DIAGNOSTICS CONDITION 1
     :DiagUtilDS.MessageId          = DB2_MESSAGE_ID,
     :DiagUtilDS.MessageId1         = DB2_MESSAGE_ID1,
     :DiagUtilDS.MessageId2         = DB2_MESSAGE_ID2,
     :DiagUtilDS.MessageLength      = MESSAGE_LENGTH,
     :DiagUtilDS.MessageText        = MESSAGE_TEXT,
     :DiagUtilDS.ReturnedSqlCode    = DB2_RETURNED_SQLCODE,
     :DiagUtilDS.ReturnedSQLState   = RETURNED_SQLSTATE,
     :DiagUtilDS.RowsCount          = DB2_SQLERRD3;
     Return ;
 End-Proc;
```

Using SQL Cursors

- Four requirements for using SQL Cursors
  - Declare Cursor
  - Open
  - Fetch
  - Close
- SQL cursor is named entity that SQL uses to point to and process row from multiple row result set
  - Can loop through result set, fetching and processing records individually
- Program can retrieve multiple rows into result set and then process each rows individually… Very Fast! ☺

Declare Cursor

- Declare Cursor SQL statement is similar to RPG Dcl-f instruction
  - Must appear in source code before any statements that refer to cursor
  - Should be near beginning of program
- Assigns name to nested Select statement indicating rows to retrieve
  - May include host variables

```
Exec SQL Declare CustzipCUR Cursor For Select  Cfname, Clname
                                       From  Customers
                                       Where Czip = :Czip
                                       Order by Custno;
```

Declare Cursor

- Cursor is usually updateable
  - Unless Declare Cursor indicates otherwise
  - May explicitly specify updateable cursor
  - May optionally limit updates to certain columns

```
Exec SQL Declare CustzipCUR Cursor For Select  Cfname, Clname
                                       From  Customers
                                       Where Czip = :Czip
                                   For Update;
```

```
Exec SQL Declare CustzipCUR Cursor For Select  Cfname, Clname
                                       From  Customers
                                       Where Czip = :Czip
                                   For Update of Cfname, Clname;
```

Declare Cursor

- Specify Read Only to improve performance
- Specify Insensitive cursor to work with copy of result set
  - Will not recognize subsequent changes to result set
  - May improve performance

```
Exec SQL Declare CustzipCUR Cursor For Select  Cfname, Clname
                                       From  Customers
                                       Where Czip = :Czip
                                  For Read Only;
```

```
Exec SQL Declare CustzipCUR Insensitive Cursor For Select  Cfname, Clname
                                                  From  Customers
                                                  Where Czip = :Czip
                                             For Read Only;
```

Declare Cursor

- Serial cursor (default) navigates through result set sequentially forward
  - Program can only retrieve each row once
  - Must be closed and reopened to process again
- Scrollable cursor permits result set navigation in either direction
  - Program can retrieve row multiple times

```
Exec SQL Declare CustzipCUR Scroll Cursor For Select  Cfname, Clname
                                                From  Customers
                                                Where Czip = :Czip
                                           For Read Only;
```

Open

- Opening cursor processes Select statement associated with cursor
  - Makes rows in result set available to program
  - Host/indicator variable values are substituted into statement

```
Exec SQL Open CustzipCUR;
```

Fetch

- Fetch SQL statement retrieves row(s) from result set that contains multiple rows
  - Reads into host variables or host structure
  - Select statement associated with specified cursor dictates result set contents
- RPG program can then process rows in loop
  - Similar to native file operations
- Row being processed is called current row

```
Exec SQL Fetch CustzipCUR Into :Cfname :NullCfname,
                               :Clname :NullClname;
```

Fetch

- Fetch may use host structure

```
Dcl-ds Customers Ext Qualified End-ds;
…
Exec SQL Declare CustcursorCUR Cursor For
                                    Select  *
                                      From  Customers
                                      Where Czip = :Czip
                                      Order by Custno
                                      For   Read Only;
…
Exec SQL Fetch CustcursorCUR Into :CustomersDS;
```

Fetch

- Fetch {Next} {From} cursor-name Into host-variables
  - Only option available for serial cursors
- Fetch Prior From cursor-name Into host-variables
  - Retrieves previous row from scrollable cursor
- Fetch First From cursor-name Into host-variables
  - Retrieves first row from scrollable cursor
- Fetch Last From cursor-name Into host-variables
  - Retrieves last row from scrollable cursor
- Fetch Current From cursor-name Into host-variables
  - Re-reads current row in scrollable cursor

Fetch

- Fetch Relative n From cursor-name Into host-variables
  - Reads row n number of rows before or after current row in scrollable cursor
- Fetch Before From cursor-name
  - Positions cursor before first row in scrollable cursor
  - Does not read data, requires Fetch Next
  - Similar to Setll *Loval
- Fetch After From cursor-name
  - Positions cursor after last row in scrollable cursor
  - Does not read data, requires Fetch Prior
  - Similar to Setgt *Hival

Close

- Close SQL statement closes open cursor
  - Discards result set and releases locks on tables or views
- After processing, program should explicitly close cursor
  - Must close serial cursor to reprocess it

```
Exec SQL Close CustzipCUR;
```

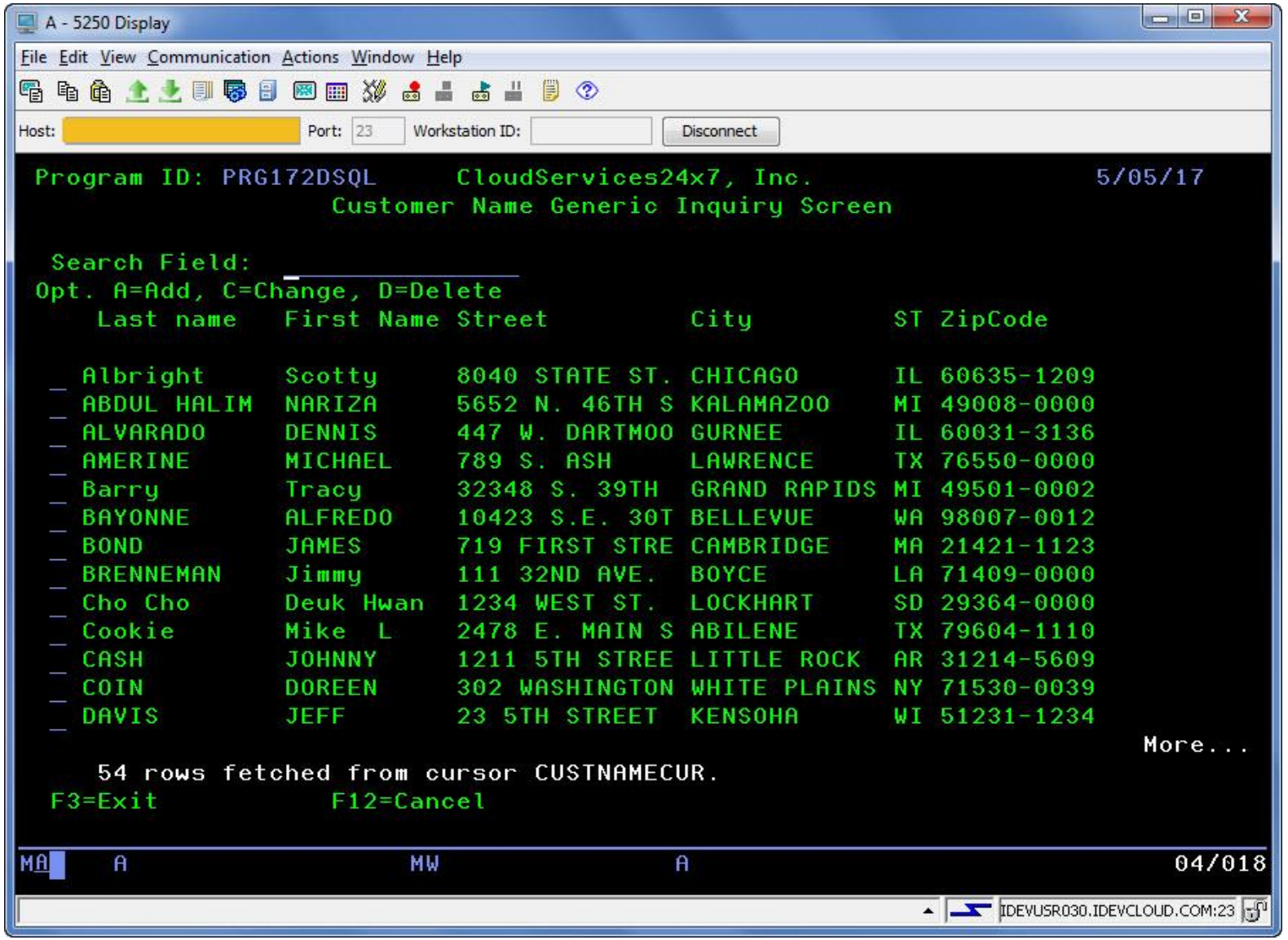
Cursor Updates and Deletes

- Update and Delete SQL statements support *Where Current Of* clause to update or delete current row
- Program must first position cursor on that row
  - Positioned update/delete
- After positioned delete, cursor is moved to next row
  - Or after last row if no more rows

```
Exec SQL Update  Customers
           Set   Cfname = :Cfname, Clname = :Clname
           Where Current of CustzipCUR;
```

```
Exec SQL Delete From Customers Where Current Of CustzipCUR;
```

Subfile Application Example



Subfile Application Example

- Comprised of three programs
  - GETSQLDIAG – Service program that:
    - Processes the GET DIAGNOSTICS command
    - Puts the results into a data structure
    - Returns this data structure to the calling program
  - ***CUSTSRVPGM – Service program that handles SQL I/O***
    - SQL INSERT, UPDATE, SELECT and DELETE Code
    - Returns data Structures (Customer and SQL Status)
  - PRG175DSQL – Main Driver Program
    - Runs the 5250 screens
    - Handles the CREATE, READ, UPDATE and DELETE Logic

Cursor Adds, Updates and Deletes

Procedure to do **ADD** a record in service program

```
//*************************************************
// * Adds New DB2 Data For CUSTOMER
//*************************************************
Dcl-Proc WriteCUSTOMER_Data Export;
   Dcl-Pi *N IND;
      CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS);
      WrkCustNbr Zoned(6:0);
      WrkUtilDS LikeDS(UtilDSSQL);
   End-Pi ;

   Dcl-s InsertSuccess Ind inz(*off);
...
```

Cursor Adds, Updates and Deletes

Procedure to do **ADD** a record in service program

```
...
EXEC SQL
   INSERT INTO CUSTOMER
      (CUSTNO, CFNAME, CLNAME, CSTREET, CCITY, CSTATE,
       CZIP, CPHONE, CALPHONE, CEMAIL, ORDDAT, BALDUE)
       VALUES(:CUSTOMERDataDS);

     GetDiagnostics(WrkUtilDS);

   If ReturnedSQLCode = 000;
      InsertSuccess = *on;
      COMMIT;
   Else;
      InsertSuccess = *off;
   EndIf;

   Return InsertSuccess;

End-Proc;
```

Cursor Adds, Updates and Deletes

Procedure to do **DELETE** in service program

```
//*************************************************************
// * Delete DB2 Data For CUSTOMER
//*************************************************************

Dcl-Proc DeleteCUSTOMER_Data Export;

    Dcl-Pi *N IND;
      CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS);
      WrkCustNbr Zoned(6:0);
      WrkUtilDS LikeDS(UtilDSSQL);
    End-Pi ;

    Dcl-s DeleteError Ind inz(*off);

...
```

Cursor Adds, Updates and Deletes

Procedure to do **DELETE** in service program

```
...
 EXEC SQL
      Delete from CUSTOMER
            where CUSTNO = :WrkCustNbr;
    GetDiagnostics(UtilDSSQL);

    If ReturnedSQLCode = 000;
       DeleteError = *off;
       COMMIT;
    Else;
       DeleteError = *on;
    EndIf;

    return DeleteError;
  End-Proc;
```

Cursor Adds, Updates and Deletes

Procedure to do **UPDATE** in service program

```
//*************************************************************
// * Updates DB2 Data For CUSTOMER
//*************************************************************

Dcl-Proc UpDateCUSTOMER_Data Export;
   Dcl-Pi *N IND;
       CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS);
       WrkCustNbr Zoned(6:0);
       WrkUtilDS LikeDS(UtilDSSQL);
   End-Pi ;
      Dcl-s UpdateSuccess Ind inz(*off);


...
```

Cursor Adds, Updates and Deletes

Procedure to do **UPDATE** in service program

```
...
   EXEC SQL UPDATE CUSTOMER
        SET ROW = :CUSTOMERDataDS
                WHERE CUSTNO = :WrkCustNbr;
        GetDiagnostics(WrkUtilDS);

        If ReturnedSQLCode = 000;
           UpdateSuccess = *on;
           COMMIT;
        else;
           UpdateSuccess = *off;
        EndIf;

        Return UpdateSuccess;

End-Proc ;
```

Execute Immediate

- Execute Immediate SQL statement combines Prepare and Execute
  - More efficient than separate statements

```
Dcl-s SQLstring Varchar(256);
…
// Custno value provided by user input
SQLstring = 'Delete From Customers Where Custno = ' + Custno;

Exec Sql Execute Immediate :SQLstring;
```

Set Option

- Set Option SQL statement establishes processing options to use in program
- Must appear in program before any other SQL statements

```
Exec SQL Set Option Alwcpydta = *Yes,
                    Closqlcsr = *Endpgm,
                    Commit    = *None;
```

Alwcpydta

- Alwcpydta (Allow Copy of Data) option indicates whether Select statement can use temporary copy of
  - Improves performance
  - Risks using obsolete result set
- Alwcpydta(*Optimize) lets system decide
- Alwcpydta(*Yes) and Alwcpydta(*No) allow or prohibit using copy

```
Exec SQL Set Option Alwcpydta = *Yes,
                    Closqlcsr = *Endpgm,
                    Commit    = *None;
```

Closqlcsr

- Closqlcsr (Close SQL Cursor) specifies when to close SQL cursors
  - If program doesn't explicitly close them with Close statement
  - Also determines scope of prepared SQL statements and file locks
- Closqlcsr(*Endpgm) closes cursors when program ends
- Closqlcsr(*Endmod) closes cursors when module ends

```
Exec SQL Set Option Alwcpydta = *Yes,
                    Closqlcsr = *Endpgm,
                    Commit    = *None;
```

Closqlcsr

- Closqlcsr(*Endsql) allows cursors to remain open between program instances in job, without having to reopen them each time
- Closqlcsr(*Endjob) lets cursors remain open until IBM i job ends
- Closqlcsr(*Endactgrp) closes cursors when ILE activation group ends

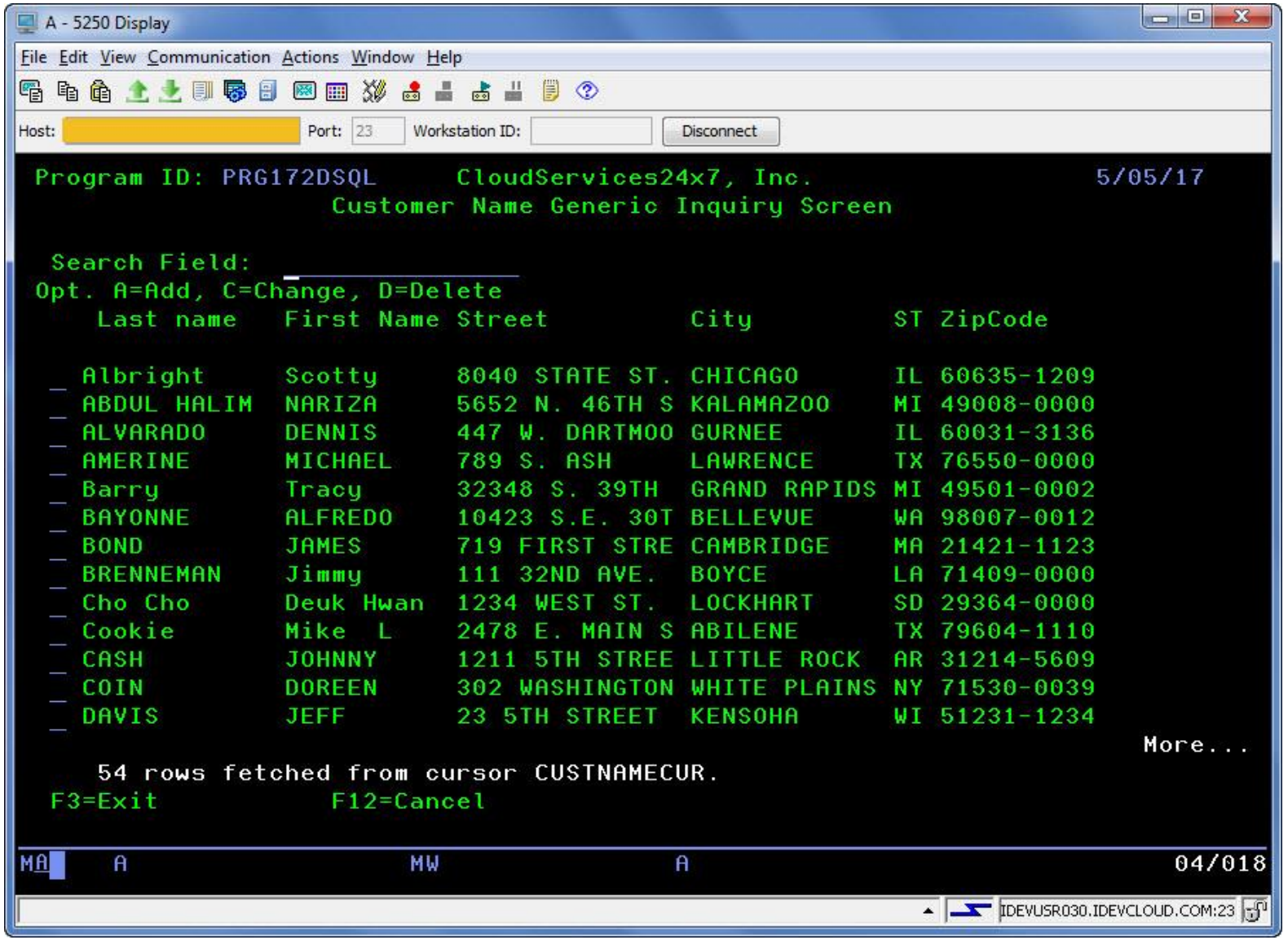
```
Exec SQL Set Option Alwcpydta = *Yes,
                    Closqlcsr = *Endpgm,
                    Commit    = *None;
```

Commit

- Commitment control is database feature that permits complex transactions to be processed with all-or-nothing architecture
  - If several database updates are required for transaction, then all updates must occur, or none will
- Also allows program to roll back incomplete transactions
  - That have not yet been committed
- SQL typically assumes commitment control is in effect
  - If database does not use commitment control, program should specify Commit(*None)

```
Exec SQL Set Option Alwcpydta = *Yes,
                    Closqlcsr = *Endpgm,
                    Commit    = *None;
```

Subfile Application Example



Subfile Application – Putting it Together

- Comprised of three programs
  - GETSQLDIAG – Service program that:
    - Processes the GET DIAGNOSTICS command
    - Puts the results into a data structure
    - Returns this data structure to the calling program
  - CUSTSRVPGM – Service program that handles SQL I/O
    - SQL INSERT, UPDATE, SELECT and DELETE Code
    - Returns data Structures (Customer and SQL Status)
  - ***PRG175DSQL – Main Driver Program***
    - Runs the 5250 screens
    - Handles the CREATE, READ, UPDATE and DELETE Logic

Dynamic SQL Example

- The driver program uses an example of Dynamic SQL

```
// Purpose: Used for SQL Into Statement
Dcl-ds CustomerDS Ext ExtName('CUSTOMER') Qualified Dim(9999)
End-ds;

// Data Structure for SQL Results
   Dcl-Ds UtilDSSQL inz;
      MessageId Char(10);
      MessageId1 VarChar(7);
      MessageId2 VarChar(7);
      MessageLength int(5);
      MessageText varchar(32740);
      ReturnedSQLCode int(5);
      ReturnedSQLState char(5);
      RowsCount int(10);
   End-Ds;
```

Main Loop

```
// ================================================================
//  main loop
// ================================================================

     ClearFields();          // Clear out the fields
     Open DisplayScreen;
     Dow not exit;
         Clear FooterDs.message;
         Clear CustSflDS.Opt;
         LoadSFL();    // Do SQL processing and load the Subfile
         DisplaySFL(); // Display the subfile and handle any processing
         If CustCtlDS.SearchFld <> *Blanks;
            SearchTerm = '%' + %Trim(CustCtlDS.SearchFld) + '%' ;
         EndIf;
     enddo;
     Close DisplayScreen;
     *inlr = *on;
     return;
```

Building Dynamic SQL Statements – LoadSFL

```
// ============================================================
// load Subfile with Records
// ============================================================
   Dcl-Proc LoadSFL;
      ClearSFL();                  // Clear out the subfile
      BuildSQLStmt();              // Build the SQL Statement
      SelectSuccess =
        GetCUSTOMERRecs_DynSelect(CustomerDS:Statement:SearchTerm:UtilDSSQL);
      If SelectSuccess;
         for rrn = 1 to RowsCount;
            EVAL-CORR CustSflDS = CustomerDS(rrn);
            CustSflDS.nzip = %Dec(CustomerDS(rrn).czip:9:0);
            write DisplayScreen.CustSfl CustSflDS;
            reset CustSflDS;
         Endfor;
      Else;
            HandleSQLMessages();
      Endif;
   End-Proc;
```

Building Dynamic SQL Statement – cont.

- Program must first prepare SQL statement from program data values
- Can build text string using standard RPG character processing techniques

```
// ============================================================
// Build Select Statement
// ============================================================
 Dcl-Proc BuildSQLStmt;

     statement = 'SELECT * FROM CUSTOMER ';

     If CustCtlDS.SearchFld <> *Blanks; // Search Field has something in it?
        Statement += 'WHERE ' ;
        Statement += 'UPPER(CFNAME) LIKE ? ';
        Statement += 'OR ';
        Statement += 'UPPER(CLNAME) LIKE ? ';
        Statement += 'OR ';
```

Building Dynamic SQL Statements – cont.

```
        Statement += 'UPPER(CSTREET) LIKE ? ';
        Statement += 'OR ';
        Statement += 'UPPER(CCITY) LIKE ? ';
        Statement += 'OR ';
        Statement += 'UPPER(CSTATE) LIKE ? ';
        Statement += 'OR ';
        Statement += 'UPPER(CZIP) LIKE ? ';
        Statement += 'OR ';
        Statement += 'UPPER(CEMAIL) LIKE ? ';
      EndIf;
      Statement += ' ORDER BY CLNAME, CFNAME';

      Statement += ' FOR FETCH ONLY';
  End-Proc;
```

Prepare / Declare / Open / Fetch Statements

- ***Prepare SQL statement*** - validates text string and translates it into executable SQL statement
  - Names prepared SQL statement
- ***Declare Cursor SQL Statement*** – names a cursor and associates it with a SQL statement
- ***Open SQL Statement*** – opens the cursor and process the Select statement embedded in the cursor making the table rows available
- ***Fetch SQL Statement*** – retrieves one of more rows from a result set in to host variables of data structures

CUSTSRVPGM Procedure

Procedure returns a record set to the caller.

```
// Service program's called procedure to return record set

Dcl-Proc GetCUSTOMERRecs_DynSelect Export;

   Dcl-pi GetCUSTOMERRecs_DynSelect Ind;
       CUSTOMERDataDS LIKEDS(CUSTOMER_IODataDS) Dim(9999);
       Statement  varchar(4096);
       SearchTerm VarChar(100);
       WrkUtilDS LikeDS(UtilDSSQL);
   End-pi;
   Dcl-s SelectSuccess ind inz(*off);

   Exec SQL prepare CustSelect from :statement;
   GetDiagnostics(WrkUtilDS);

   Exec SQL declare CustNameCUR Scroll Cursor for CustSelect;
   GetDiagnostics(WrkUtilDS);
....
```

CUSTSRVPGM Procedure

Procedure returns a record set to the caller.

```
....
  If SearchTerm <> *Blanks;
    Exec SQL open CustNameCUR using
         :searchTerm
         , :searchTerm
         , :searchTerm
         , :searchTerm
         , :searchTerm
         , :searchTerm
         , :searchTerm
         ;
  else;
    Exec SQL open CustNameCUR using :statement;
  EndIf;
  GetDiagnostics(WrkUtilDS);
....
```

CUSTSRVPGM Procedure

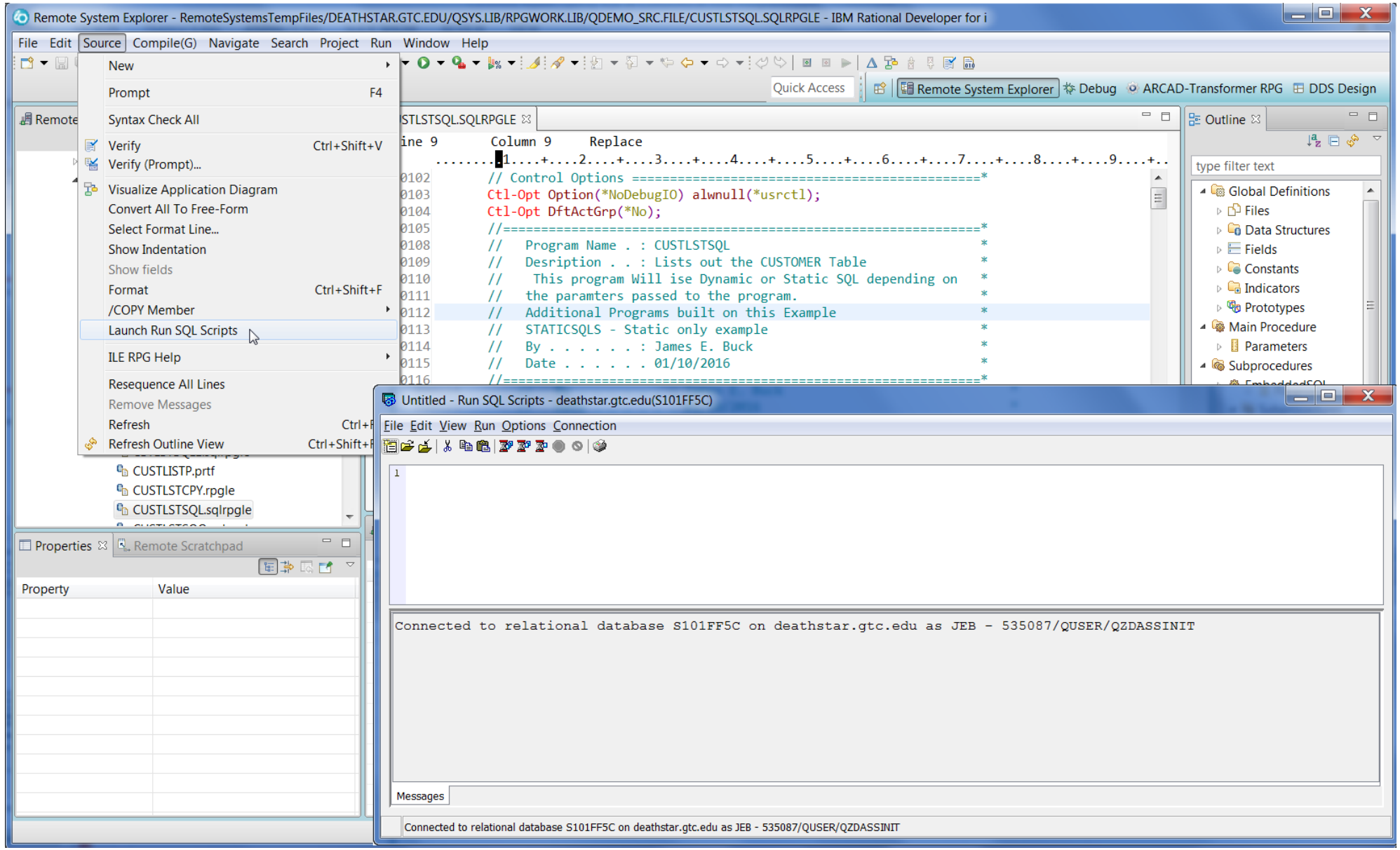
Procedure returns a record set to the caller.

```
...
 Exec SQL fetch CustNameCUR for 9999 rows into :CUSTOMERDataDS;
  GetDiagnostics(WrkUtilDS);

  If ReturnedSQLCode = 000;
         SelectSuccess = *on;
  Else;
         SelectSuccess = *off;
  EndIf;
  exec sql close CustNameCUR;

  return SelectSuccess;
End-Proc;
```

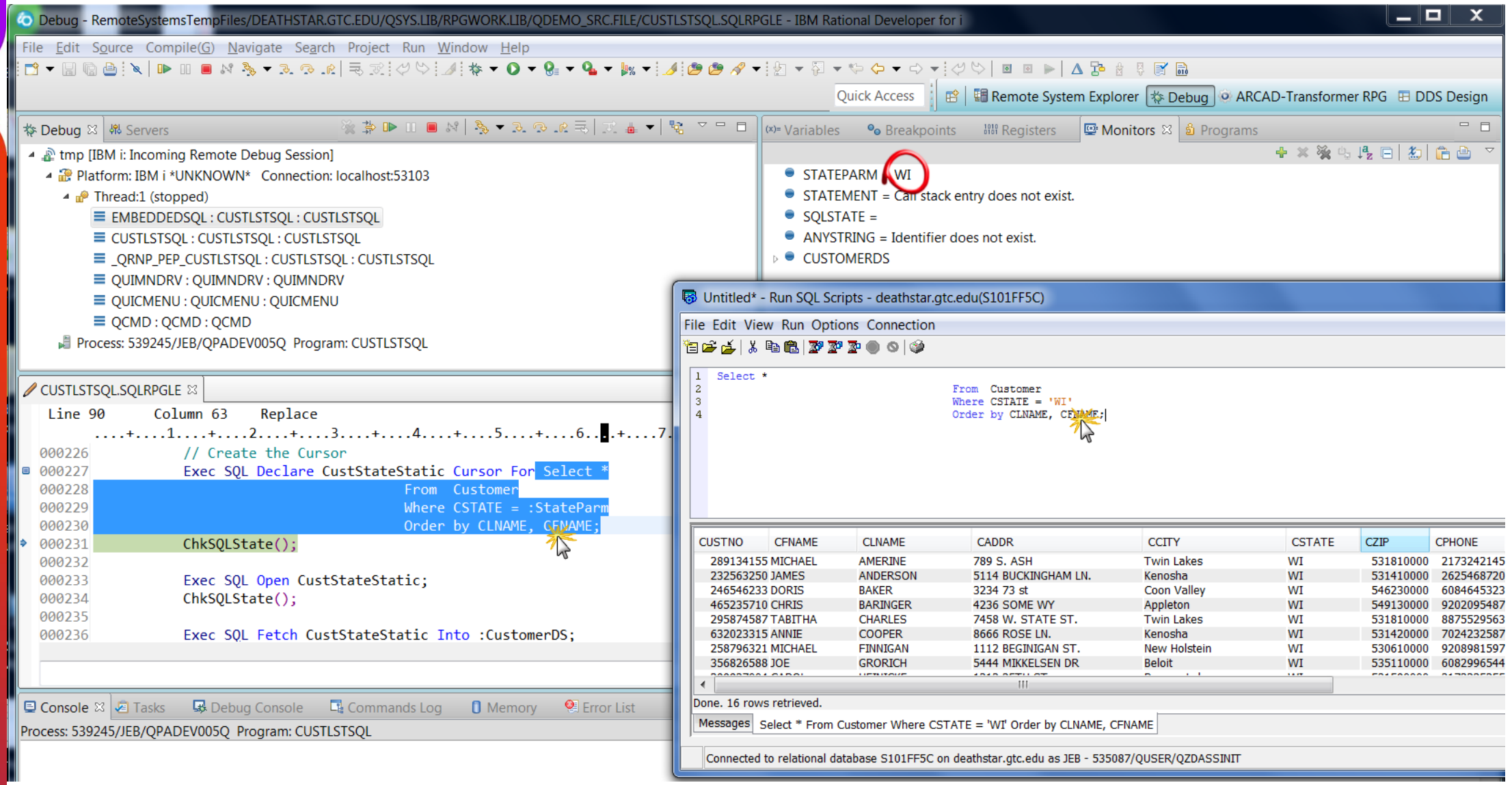
Using RDi to Debug SQL Statements

- ***Most problems when using SQL in RPG are caused by:***
- ***Statement being incorrectly built:***
  - *Especially with Dynamic SQL*
- ***Run SQL Scripts***– No longer just in iNav for Windows
  - Included in Access Client Solutions – Cool!
  - Added to Rational Developer for I – Very Cool!

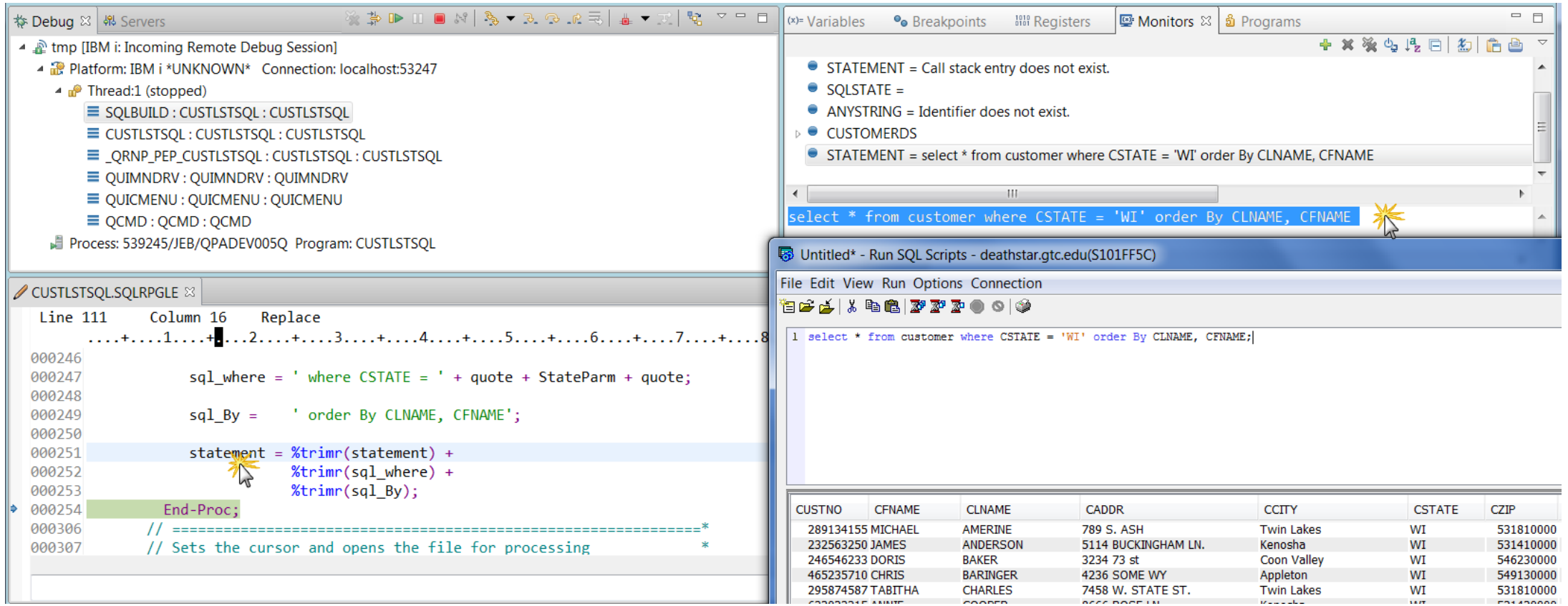
Run SQL Scripts in RDi 9.5.1.2



Debugging a Static SQL statement

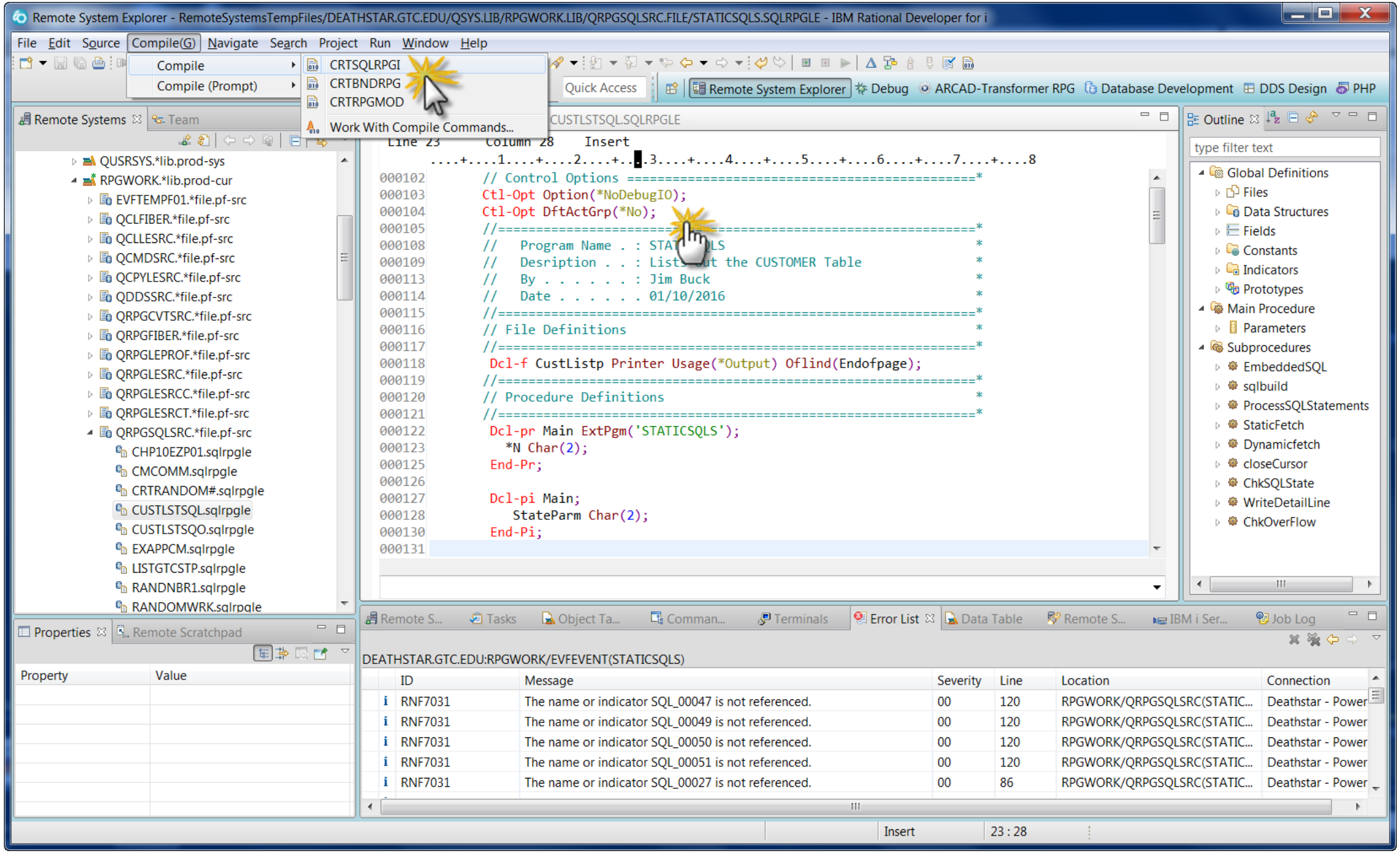


Debugging a Dynamic SQL statement



Creating SQLRPGLE Programs

- Enter source code into source file member
  - Member type SQLRPGLE
- CRTSQLRPGI (Create SQL ILE RPG Object) CL command compiles source and creates program
  - Or module or service program
- CRTSQLRPGI uses precompiler to translate SQL portions of source code before actually compiling it
  - Converts SQL statements to equivalent RPG code to calls database functions
  - Creates temporary source member
  - Compiles (and binds) temporary source member, creating resulting object


Remote System Explorer - RemoteSystemsTempFiles/DEATHSTAR.GTC.EDU/QSYS.LIB/RPGWORK.LIB/QRPGSQLSRC.FILE/STATICSQLS.SQLRPGLE - IBM Rational Developer for i
File Edit Source Compile(G) Navigate Search Project Run Window Help
Compile
Compile (Prompt)
CRTSQLRPGI
CRTBNDRPG
CRTRPGMOD
Work With Compile Commands...
Quick Access
Remote System Explorer
Debug
ARCAD-Transformer RPG
Database Development
DDS Design
PHP
Remote Systems
Team
CUSTLSTSQL.SQLRPGLE
Line 23 Column 28 Insert
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
000102 // Control Options =====*
000103 Ctl-Opt Option(*NoDebugIO);
000104 Ctl-Opt DftActGrp(*No);
000105 //=====*
000108 // Program Name . : STATICSQLS *
000109 // Desription . . : Lists out the CUSTOMER Table *
000113 // By . . . . . : Jim Buck *
000114 // Date . . . . . 01/10/2016 *
000115 //=====*
000116 // File Definitions *
000117 //=====*
000118 Dcl-f CustListp Printer Usage(*Output) Oflind(Endofpage);
000119 //=====*
000120 // Procedure Definitions *
000121 //=====*
000122 Dcl-pr Main ExtPgm('STATICSQLS');
000123 *N Char(2);
000125 End-Pr;
000126
000127 Dcl-pi Main;
000128 StateParm Char(2);
000130 End-Pi;
000131
QUSRSYS.*lib.prod-sys
RPGWORK.*lib.prod-cur
EVFTEMPF01.*file.pf-src
QCLFIBER.*file.pf-src
QCLLESRC.*file.pf-src
QCMDSRC.*file.pf-src
QCPYLESRC.*file.pf-src
QDDSSRC.*file.pf-src
QRPGCVTSRC.*file.pf-src
QRPGFIBER.*file.pf-src
QRPGLEPROF.*file.pf-src
QRPGLESRC.*file.pf-src
QRPGLESRCC.*file.pf-src
QRPGLESRCT.*file.pf-src
QRPGSQLSRC.*file.pf-src
CHP10EZP01.sqlrpgle
CMCOMM.sqlrpgle
CRTRANDOM#.sqlrpgle
CUSTLSTSQL.sqlrpgle
CUSTLSTSQO.sqlrpgle
EXAPPCM.sqlrpgle
LISTGTCSTP.sqlrpgle
RANDNBR1.sqlrpgle
RANDOMWRK.sqlrpgle
Outline
type filter text
Global Definitions
Files
Data Structures
Fields
Constants
Indicators
Prototypes
Main Procedure
Parameters
Subprocedures
EmbeddedSQL
sqlbuild
ProcessSQLStatements
StaticFetch
Dynamicfetch
closeCursor
ChkSQLState
WriteDetailLine
ChkOverFlow
Properties
Remote Scratchpad
Property
Value
Remote S...
Tasks
Object Ta...
Comman...
Terminals
Error List
Data Table
Remote S...
IBM i Ser...
Job Log
DEATHSTAR.GTC.EDU:RPGWORK/EVFEVENT(STATICSQLS)
ID
Message
Severity
Line
Location
Connection
RNF7031 The name or indicator SQL_00047 is not referenced. 00 120 RPGWORK/QRPGSQLSRC(STATIC... Deathstar - Power
RNF7031 The name or indicator SQL_00049 is not referenced. 00 120 RPGWORK/QRPGSQLSRC(STATIC... Deathstar - Power
RNF7031 The name or indicator SQL_00050 is not referenced. 00 120 RPGWORK/QRPGSQLSRC(STATIC... Deathstar - Power
RNF7031 The name or indicator SQL_00051 is not referenced. 00 120 RPGWORK/QRPGSQLSRC(STATIC... Deathstar - Power
RNF7031 The name or indicator SQL_00027 is not referenced. 00 86 RPGWORK/QRPGSQLSRC(STATIC... Deathstar - Power
Insert
23 : 28

Questions or Comments?

imPower Technologies

IBM i Services

Jim Buck
Phone 262-705-2832
Email – jbuck@impowertechnologies.com
Twitter - @j_buck51