Externally Described SQL™ -- An SQL iQuery API

Introduced as a beta test API in SQL iQuery v4r7, Externally Described SQL is a simple set of APIs that provide the ability for RPG programmers to leverage the power of SQL iQuery Script in their own RPG applications. *Externally Described SQL* provides the ability to run SQL iQuery Scripts and optionally produce a dynamic SQL statement that is returned to your RPG program. This is accomplished through the use of the SQL iQuery *Externally Described SQL* API or **EDS API** for short.

This Document contains a brief review of the SQL iQuery Scripting language, as well as examples of how to use the Externally Described SQL (EDS) API. The examples presented at the end of this document are included in the SQL iQuery product library IQUERY in the QDEMORPG, QDEMODDS, and QDEMOSQL source files.

Overview of the EDS API

RPG programmers can run SQL iQuery Scripts as well as set and retrieve the value of Session Variables using the EDS APIs. In addition, the final SQL statement within the Script is returned to the caller without being run. This allows the RPG program to run that SQL statement using dynamic embedded SQL or other methods.

The list of available SQL iQuery APIs to support Externally Described SQL is rather short, and includes:

- **iqInit**() -- Optionally initializes the SQL iQuery environment.
- **iqLoadSQL** -- Loads and runs an SQL iQuery Script and returns the resulting SQL statement to the caller without running that final dynamic SQL statement.
- **iqSetVar** - Creates or changes the value assigned to a Session Variable.
    - iqSetVarDec -- Accepts a packed decimal value to assign to a Session Variable.
    - iqSetVarInt -- Accepts an integer value to assign to a Session Variable.
    - iqSetVarDate -- Accepts a date value to assign to a Session Variable.
- **iqAddVar** - Similar to iqSetVar, except if the variable already exists, it automatically transforms the variable into an array and adds the value as the next element.
- **iqDltVar** - Deletes a Session Variable.
- **iqGetVar** - Retrieve the current value of a Session Variable.
- **iqClearVars** -- Clear (delete) all existing Session Variables.
- **iqRUNSQL** -- Dynamically runs the given SQL statement or SQL iQuery Script and routes the output to the designated output setting. (Available Winter 2018).

Session Variables

The key to any good interface is being able to pass data between it and the host language. The externally described SQL API isn't very complex and to keep it simple, only the iqLoadSQL is needed to do anything, but for more flexibility and the ability to pass data between your RPG program and iQuery (also known as parameter passing), an interface to set and retrieve shared data is provided. That interface is called *Session Variables*.

Session Variables are SQL iQuery Script variables and are used to store data. Similar to a regular *field* in RPG, Session Variables may be used to share data between RPG programs and SQL iQuery Scripts. Session Variables are not typed, are automatically created and destroyed, and may contain character, numeric or date information. They have a maximum length of "no practical limit" and auto-grow as needed. SQL iQuery Session Variables or simply "Session Variables" are used as substitution values within the lines of code in any SQL iQuery script.

Session Variables may be assigned any value. For example, a Session Variable named &MAXSALES may contain a value and where ever &MAXSALES is embedded in the SQL iQuery Script, the value stored in that Session Variable is inserted into that line of code in place of &MAXSALES. Let's look at a sample where a value is assigned to &MAXSALES and then it is used in another statement.

```
eval &MAXSALES = 20000;
if (&MAXSALES > 5000);
 // Do something when sales are greater than 5000
endif;
```

The first line assigns the value 20000 to &MAXSALES. Then &MAXSALES is compared against 5000. If it is greater than 5000, we do something, otherwise we don't. Yes, it is that simple to use SQL iQuery Script and Session Variables. Note the EVAL, IF and ENDIF statements are iQuery Script commands and have syntax similar to traditional free-format RPG IV.

Let's use a little more powerful feature, embedding SQL in iQuery scripts. Instead of hard coding 20000 as the &MAXSALES value, let's dynamically retrieve that amount from a database file.

```
select max(sales) INTO &MAXSALES
   FROM datalib.SALESHIST;

if (&SQLState >= '02000');
  // Nothing found!
elseif (&MAXSALES > 5000);
 // Do something when sales are greater than 5000
endif;
```

Now instead of hard coding the maximum sales, we retrieve it from the SALESHIST file and store it in the &MAXSALES Session Variable.

We're all developers, so we might want to log the &MAXSALES value by writing its content to the joblog as follows:

```
select max(sales) INTO &MAXSALES
   FROM datalib.SALESHIST;
if (&maxsales > 5000);
  #joblog &maxSales is higher than expected!
else;
  #joblog &maxSales is lower than expected!
endif;
```

When the above script is run, the user might see something like the following on the Command Entry screen or in the joblog:

```
iQScript: COZTEST/QSQLSRC(EXTSQL)
Ownership of object IQ00000000 in QTEMP type *USRSPC changed.
1566.80 is lower than expected!
```

The first line is what is logged whenever an iQuery Script source is opened for processing. The *USRSPC message is typically only there the first time you run SQL iQuery in your job. The 3rd line is what is written when the #JOBLOG command is run in the SQL iQuery Script.

There are dozens of SQL iQuery "hashtag" commands, and several regular commands (those that don't require a hashtag, such as EVAL), as well as a dozen or so built-in functions. Many of the #commands are available in synonym form, meaning there are two or more #commands that do the same task. For example, #JOBLOG and #MSG are synonyms.

To modify the contents of a Session Variable within an SQL iQuery Script, use any of the following commands:

- EVAL - Similar to RPG's EVAL opcode.
- #define - Set the initial value for a Session Variable.
- #default - Set the value of a Session Variable if the variable doesn't exist.

The EVAL command allows you to assign a value to a Session Variable. The syntax is similar to RPG's EVAL operation code.

The #DEFINE command is a declarative statement. Basically, it is the same as EVAL except gives developers a way to set the value "at the top of the Script". Unlike EVAL, the #DEFINE statement does not need an assignment symbol (equals sign).

The #DEFAULT command is the same as #DEFINE except it checks if the target Session Variable already exists. If it does, it takes no action. If it does not exist, it assigns the value to the Session Variable.

When using the Externally Described SQL (EDS) API, the following APIs are dedicated to creating or deleting Session Variables.

- iqSetVar - Set the value of a Session Variable.
    - iqSetVarDec - Use when the value is a packed or zoned numeric w/decimal positions.
    - iqSetVarInt - Use when the value is a whole number (no decimal positions) or an integer.
    - iqSetVarDate - Use when the value is a Date data type.
- iqDltVar - Deletes a Session Variable.
- iqGetVar - Retrieves the current value of a Session Variable.
- iqClearVars - Deletes all session variables.

Using the iqSETVAR API a character value may be assigned to the variable. In RPG, simply call it with the session variable name (names are case-insensitive) as the first parameter and the value as the 2nd parameter. To assign numeric content, use the RPG built-in functions such as %CHAR to convert it, or use one of the other EDS APIs such as iqSetVarDec or iqSetVarInt.

If you change the content of a Session Variable in your SQL iQuery Script and want to return it to your RPG program, use the iqGetVar or iqGetVarValue APIs.

SQL iQuery Script

As mentioned, the scripting language used by SQL iQuery should be very familiar to RPG developers. It is purposely based on RPG IV syntax, however one notable different is that the comma is used as the parameter separator, which matches the SQL standard.

SQL iQuery Script combines a basic scripting language with SQL and IBM i interfaces. For example, in iQuery Script you can:

1. Run SQL statements.
2. Read and update database tables using SQL.
3. Use conditional logic to control the flow of the script.
4. Read data from *DTAARA or *USRSPC (data areas and user spaces) easily.
5. Retrieve system information, such as System name, Serial Number, IBM i vrm level, etc.
6. Combine SQL with Conditional Logic, e.g., IF (*USRPRF in ('COZZI','QSECOFR','QSYSOPR'));
7. Check if the script is called from a Web app (i.e., is it being run from CGI).
8. Retrieve HTML Web Page Form Field Values as Session Variables.
9. Loop Conditions are supported as well:
    a. FOR (I = 1 to 100);
    b. WHILE (&SQLSTATE <'02000');
    c. FOREACH Select x, y, z INTO &a,&b,&c from qiws.qcustcdt;

For most situations that do not require heavy end-user interaction (such as a Display file, subfile or custom printed forms) you may be able to write the entire application using just SQL iQuery.

SQL iQuery gets you *thinking in SQL* so you can fulfill your business needs much more quickly and make improvements faster and without the need to recompile.

Suppose your application requirements change. For example, in the previous script we compared the maximum sales to 5000, but now the user would like the figure to be 10000. It is easy enough to go into the source member using SEU or RDi and change 5000 to 10000, save it and it's ready. But we could have also setup the script to support values that can be specified at runtime. For example:

```
#default &salesLimit 10000
select max(sales) INTO &MAXSALES
   FROM datalib.SALESHIST;
if (&maxsales > &salesLimit);
  #joblog &maxSales is higher than expected!
else;
  #joblog &maxSales is lower than expected! :(
endif;
```

Now the script uses the &SALESLIMIT Session Variable to hold the sales limit value of 10000. Note the #default command is used here. If the Session Variable &SALESLIMIT does not exist, it will create it and assign the value specified. If it already exists, (because it was set using the iqSetVar API or on the RUNiQRY SETVAR parameter) it does nothing and the variable is unchanged. This is particularly useful when you embedded the RUNiQRY command within a CL program and pass to it user-specified values, such as the SALESLIMIT.

To specify a Session Variable on the RUNiQRY CL command, use the SETVAR parameter, as follows:

```
RUNiQRY SRCMBR(salesdata) SETVAR((saleslimit 10000))
```

Normally SQL iQuery scripts are run using the RUNiQRY CL command. The SRCMBR parameter accepts the name of a source file member that contains the script. Typically, the script is the entire process but with the introduction of the Externally Described SQL API, scripts have an entirely new use.

Let's look at a simple SQL iQuery script named DEMO. This script is included with SQL iQuery and is located in the QIQMACRO source file in the iQuery library.

```
#h2 *MACRO - Macro
#h3 View of QCUSTCDT Demo File
SELECT CUSNUM    as "Customer            Number",
       LSTNAM    as "Last                Name",
       INIT      as "Initials",
       STREET    as "Address",
       CITY      as "City",
       STATE     as "State",
       DIGITS(ZIPCOD) as "Zip Code",
       CDTLMT    as "Credit              Limit",
       BALDUE    as "Balance             Due"
 from qiws.qcustcdt
 order by lstnam;
```

The #H2 and #H3 commands (there are #H1 through #H4) provide a way to specify the Output Headings/Titles. Each #H*n* corresponds to the line number: #H1 is the first line, #H2 the second, etc. #H1 is set to "SQL iQuery for IBM i" if it is not specified. The symbolic value *MACRO is one of several reserved words in SQL iQuery Script. It inserts the name of the source member. Titles are centered above the body of the output; above any Column Headings.

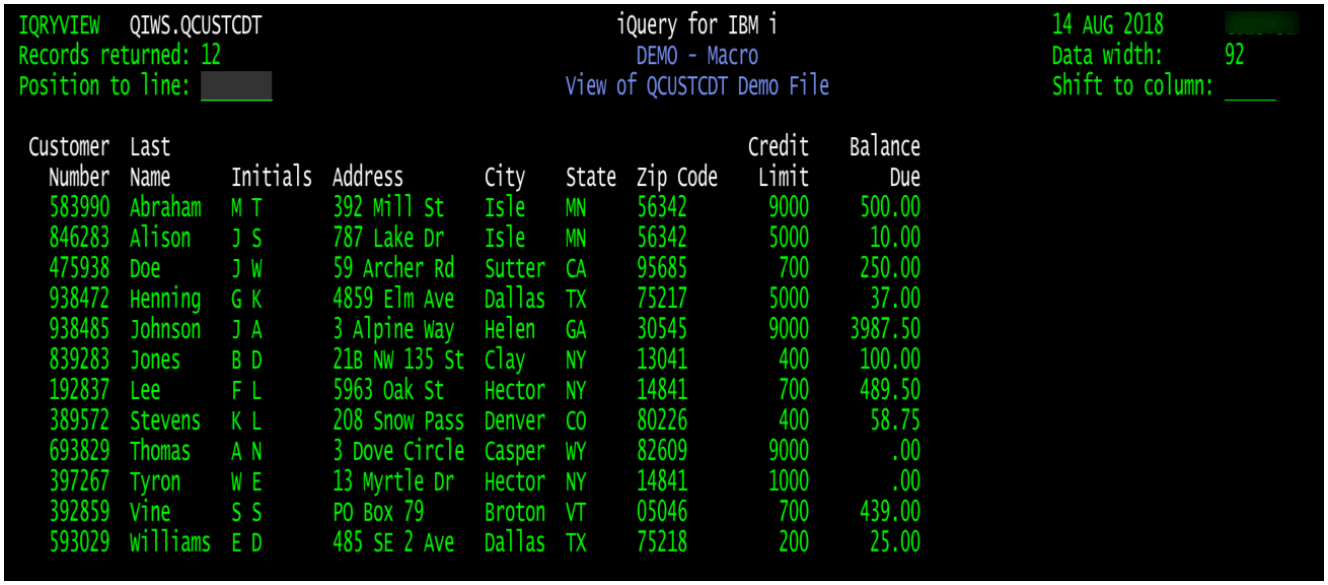
The last statement, which spans many lines is the SQL SELECT statement. It selects the records from the IBM-supplied demo file QCUSTCDT that is shipped in the QIWS library. There is nothing special about this SELECT statement, but we do include custom "column headings" using the "AS" clause on each column. SQL iQuery reads the "AS" clause and builds column headings using it. By the way, this is related to the COLHDG keyword in DDS that supports up to 3 column heading lines. Column headings are broken up into 3, 20-byte segments and shifted downward in the output.

This SELECT statement can be run in any SQL environment, there are no SQL iQuery-unique features in the SQL statements you run with SQL iQuery. Just cut/paste the SQL statement into STRSQL, ACS SQL Script processor or SQL iQuery and the results should be the same. SQL iQuery scripting commands, however are obviously not supported in those environments.

To run this script, we save it into a source file member or an IFS ASCII text file and run it. Assuming it is saved in MYSCRIPT in QSQLSRC in QGPL, the RUNiQRY command to run it would be:

```
runiqry srcmbr(myScript) srcfile(qgpl/qsqlsrc)
```

Since no OUTPUT parameter was specified, it'll default to the display. On the green screen it would look similar to the image below:



If you included column headings when you create your files or don't care about column headings, you can just run the SQL statement right on the RUNiQRY command without a script and achieve similar results.

```
runiqry SQL('select * from qiws.qcustcdt')
```

Using Externally Described SQL in RPG

While using RUNiQRY to run SQL iQuery Scripts is a fantastic tool and provides a way to route the results to the Display, Print (SPOOL), PDF, Excel, CSV, JSON and more, our customers requested a way to pull in a dynamically generated SQL statement from SQL iQuery Script. That way they could pass in substitution values, evoke the Externally Described SQL API, and then process that SQL statement in their own RPG IV program.

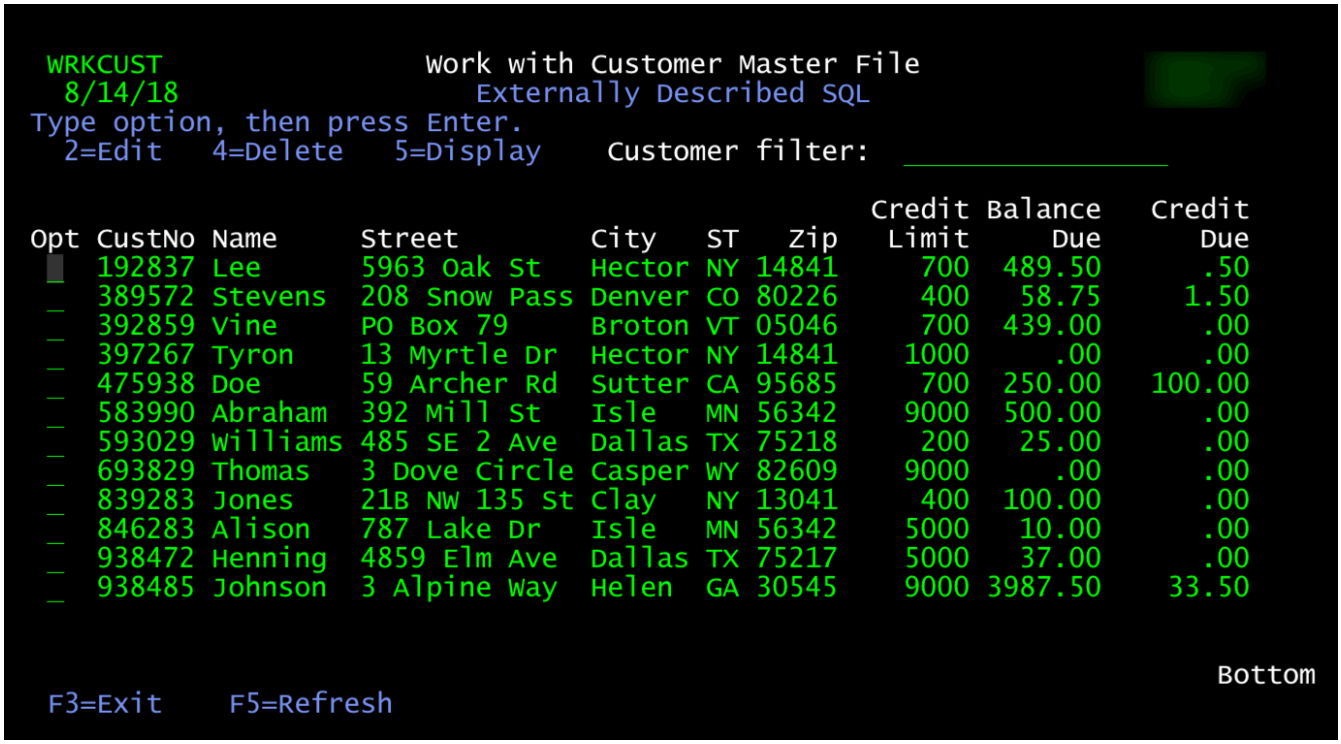
To use the External Described SQL API in RPG, simply **/copy** or **/include** the IQUERYAPI source member into your RPG code.

```
/INCLUDE iQuery/qcpysrc,iQueryAPI
```

Let's looks at an example of using the API in RPG.

Example 1:

A classic *Work With* panel is needed for the QCUSTCDT customer file. We want to start out with a subfile list of the data and provide the user with the ability to subset the results. The display should look something like the following:



The Customer filter field is used to subset the data. Since we want to use it to filter the Name, Street, and City columns, doing so in just regular old RPG IV can be challenging.

In this case, we'll use Externally Described SQL APIs and have SQL iQuery to the difficult part for us. Here's the code for the RPG IV program's LOADDATA subprocedure. It leverages the EDS API to retrieve the SQL statement it needs to extract the correct result set, and then fills up a subfile with the results using a FETCH loop.

Load Subfile using Externally Described SQL API

```
     P loadData        B
     D loadData        PI

      /free
         // A fixed-length return variable receives
         // the Externally Described SQL statement.
         // Typically, 6000 or less is more than enough, but it
         // is up to the programmer to provide the correct size
         // in order to receive the entire SQL statement.
           dcl-s extSQL char(2048);  // The returned SQL variable must be
fixed-length.
           dcl-s dynSQL varchar(2048);


           rrn = 0;
           indds.dspsfl = *off;
           indds.dspsflctl = *off;
           write header;

           if (searchData = '' or searchData = '*ALL');
```

```
          iqDltVar('searchData');
        else;
          iqSetVar('searchData' : %TrimR(searchData));
        endIf;

        // Run the SQL iQuery Script to build the resultSet dynamically
        // Note: The return parm (EXTSQL in this example) must be
        // a fixed length character field.
        iqLoadSQL(extSQL : %size(extSQL) : 'COZTEST/QSQLSRC(WRKCUST)');

        if (extSQL <> '');
          dynSQL = %trimR(extSQL : '; ');  // Trim blanks and semicolon
        else;
           // if no SQL statement was loaded, default to using this stmt.
          dynSQL =  'SELECT CUSNUM, LSTNAM, STREET, +
                       CITY, STATE, ZIPCOD, +
                       CDTLMT, BALDUE, CDTDUE +
                     FROM QIWS.QCUSTCDT +
                     ORDER BY CUSNUM';
        endIf;

        EXEC SQL PREPARE EXTERNSQL FROM :dynSQL;
        EXEC SQL DECLARE CUST CURSOR FOR externSQL;


        EXEC SQL  OPEN   CUST;

        EXEC SQL  FETCH  CUST
                  INTO  :CUSNUM, :LSTNAM,
                        :STREET, :CITY, :STATE, :ZIPCOD,
                        :CDTLMT, :BALDUE, :CDTDUE;


      IF  (SQLSTATE >= '02000'); // Nothing returned?
           errmsg = 'No data returned. Request ignored.';
      else;
        DOW (SQLSTATE < '02000' and rrn < maxRows);
           rrn += 1;
           write(e) detail;
          EXEC SQL  FETCH CUST
                    INTO :CUSNUM, :LSTNAM,
                         :STREET, :CITY, :STATE, :ZIPCOD,
                         :CDTLMT, :BALDUE, :CDTDUE;

        enddo;
      endif;
        EXEC SQL CLOSE CUST;
 /end-free
P loadData        E
```

That's the full routine! Let's review the EDS API section.

```
if (searchData = '' or searchData = '*ALL');
  iqDltVar('searchData');
else;
  iqSetVar('searchData' : %TrimR(searchData));
endIf;

// Run the SQL iQuery Script to build the resultSet dynamically
// Note: the return parm (EXTSQL in this example) must not be VARYING/VARCHAR.
iqLoadSQL(extSQL : %size(extSQL) : 'COZTEST/QSQLSRC(WRKCUST)');

if (extSQL <> '');
  dynSQL = %trimR(extSQL : '; ');  // Trim blanks and semicolo
else;
   // if no SQL statement was loaded, default to using this stmt.
  dynSQL =  'SELECT CUSNUM, LSTNAM, STREET, +
               CITY, STATE, ZIPCOD, +
               CDTLMT, BALDUE, CDTDUE +
             FROM QIWS.QCUSTCDT +
             ORDER BY CUSNUM';
endIf;

EXEC SQL PREPARE EXTSQL FROM :dynSQL;
EXEC SQL DECLARE CUST CURSOR FOR extSQL;
```

A) Delete the previous search criteria, if any, using the iqDLTVAR API.
B) Set the current search criteria by assigning a value to the SEARCHDATA Session Variable, using the iqSETVAR API.
C) Call SQL iQuery Script processor to run the script named WRKCUST.
    a. The first parameter is the receiver variable to receive the resulting SQL statement.
        i. The receiver must be fixed-length character variable with its length passed as the 2nd parameter.
    b. The second parameter is the available length of the first parameter.
    c. The third parameter is the Script source member. A flexible syntax is supported for the member name (see below).
D) If a statement is returned, we trim off trailing blanks and copy it to the DYNSQL variable.
E) If no statement is returned, we default to a safe "select all" statement so it'll continue to work for the end-user.
F) Finally, we pass the dynamic SQL statement to the PREPARE and then DECLARE statements.
    a. The Prepare identifier is named EXTSQL (could be any name) and is using the RPG host variable DYNSQL to access the SQL statement.
    b. After the Prepare completes, the prepared identifier EXTSQL, is passed to the DECLARE statement.

After that the OPEN, FETCH, and CLOSE are processed as usual.

The source code of the full this example program, named WRKCUST is available in the QDEMORPG, QDEMODDS, and QDEMOSQL source members in the iQuery product library. Compile the screen's DDS and then the RPG program, and then just call it: **CALL WRKCUST**

iqLOADSQL - Load and Run an SQL iQuery Script API Syntax

```
1. iqLoadSQL( return-var : length-of-return-var : source-mbr
              [ :  source-file [ : source-library ] } );
             or
              [ :  source-library/sourcefile ] );
2. iqLoadSQL( return-var : length-of-return-var : qualified-source-mbr );
3. iqLoadSQL( return-var : length-of-return-var : ifs-text-file );
```

Example iqLoadSQL Calls

```
1. iqLoadSQL( mySQL : %size(mySQL) : 'SALESRPT' : 'QSQLSRC' : 'OEHIST');
2. iqLoadSQL( mySQL : %size(mySQL) : 'SALESRPT' );
3. iqLoadSQL( mySQL : %size(mySQL) : 'oehist/qsqlsrc(SALESRPT)');
4. iqLoadSQL( mySQL : %size(mySQL) : 'oehist/qsqlsrc,SALESRPT');
5. iqLoadSQL( mySQL : %size(mySQL) : 'myscripts(SALESRPT)');
6. iqLoadSQL( mySQL : %size(mySQL) : 'SALESRPT' : 'oehist/qsqlsrc');
7. iqLoadSQL( mySQL : %size(mySQL) : '/home/reports/salesrpt.sql');
```

The first syntax passes the SQL iQuery script source member name, the source file name and the library.

The second syntax passes on the SQL iQuery script source member SALESRPT. Therefore the API looks for that member in QSQLSRC on the library list.

The third syntax uses the fully qualified name, library/file(member) syntax to access member SALESRPT in file QSQLSRC in library OEHIST.

The fourth syntax is similar to the third, however it uses the RPG /COPY style syntax of **library/file,member** to specify the SQL iQuery script source member name.

The fifth syntax uses the file(member) syntax to access the member SALESRPT in file QSQLSRC on the library list. It tells iqLoadSQL to search the library list for the first file named MYSCRIPTS that contains the SALESRPT iQuery Script source member.

The sixth syntax allows the script member name as parameter 3 followed by a qualified source file name as parameter 4.

The seventh syntax accepts a fully qualified IFS file text name. This file should contain your SQL iQuery script.

Summary

SQL iQuery Script Externally Described SQL (EDS) API is extremely easy to implement.