SQLRPGLE The better file access

SEMiUG Virtual Meeting

Marina Schwenk

Software Developer

About me

- Software Developer at Everbrite LLC, Greenfield WI
- Graduate of Gateway Technical College: Associate Degree IT Software Development
- Graduate of Bellevue University: Bachelors Degree Computer Information Systems - Software Development
- Student programming and writing mentor
- 2019 IBM fresh face
- Member of the WMCPA Women in Information Technology Committee
- Lead Support for the WMCPA Student Organization
- Member of COMMON's Young i Professionals (YiPS) committee

What is embedded SQL?

- SQL statements that are ran inside of your program and utilizes the computing power of the language you are writing in.
- You can embed SQL in multiple languages.
- Gives you easy access to the database from any program.
- Easy to code.

Why is Embedded SQL better?

- SQL is a powerful language that enables you to get at your data easily.
- It helps with onboarding new developers who are unaware of RPGLE but can write SQL statements.
- SQL is flexible.
- Less future enhancements.

SQL Flexibility

- Buildable SQL statements
- Conditioning sql statements
- Upsert statements (will dive in later)

Embedded SQL compared.

- Record level access gives you the entire file, logical file, index or view.
- Record level access depending on how you code you logical files requires you to recompile your program to change the file.
- Record level access is fast. But embedded SQL is faster as it utilizes the SQL engine to assist in the database IO.

How are they similar

- Read Customers is the same as select * from Customers.
- Reade Customers is the same as select * from Customers where custno = customerNumber.

DB access Requirement

SQL Requirements

- Exec Sql

Exec sql

Select Lname from Customer;

Record Level Access

- DCL-F customers
- If not %open Customers;
- Open Customers;
- Chain mykey Customers;
- process data.
- Close Customers;

Select/ Insert Example

Embedded SQL

- Exec SQL insert into Customers2 Lname, Fname, cusno, comno select Lname, Fname, cusno, comno from Customers where Fname = 'Austin';

Record level access

- Dcl-f Customers;
- Dcl-f Customers2;
- If not %open Customers;
- Open Customers;
- If not %open Customers2;
- Open Customers2;
- Customers 1 = Customers2
- Write Customers2.
- Close Customers;
- Close Customers2;

Update Example

Embedded SQL

- Exec SQL update Customers set Lname = ‘Customer ABC’ where fname = ’Customer ab’ and cusno = 123;
- (highlight keywords)

Record level access

- Dcl-f Customers;
- If not %open Customers;
- Open Customers
- Lname = ‘Customer ABC’;
- update Customers.
- Enddo.
- Close Customers;

Delete Example

Embedded SQL

- Exec SQL delete Customers where cusno = :customer;

Record file access.

- Dcl-f Customers;
- If not %open Customers;
- Open Customers
- Chain :cusno Customers
- delete Customers.
- Close Customers;

Upsert Keywords

- Merge into
- Using
- Values
- On
- When not matched then
- When matched then

Upsert - Embedded SQL

- Exec sql Merge into rlsite tgt
  using (
  values(7069435, 10, 267, 'it really works!')
) src (rlsno, rlssq, itmsq, ufcdec) on tgt.rlsno = src.rlsno and tgt.rlssq = src.rlssq and tgt.itmsq = src.itmsq
  when not matched then insert (tgt.rlsno, tgt.rlssq, tgt.itmsq,
  tgt.ufcdec) values(7069435, 10, 267, 'dummy')
  when matched then update set tgt.ufcdec = src.ufcdec ;

Upsert important things

- **TGT** is the table of the table you are inserting into **SRC** is the table you are retrieving from.
- You have to specify **values** after the **using** keyword or otherwise your insert will fail.
- **On** is your join condition
- **When matched then** is where you want your update statement
- **When not matched then** is where you want your insert statement.

Upsert - Record level access

- Dcl-f Customers;
- If not %open Customers;
- Open Customers
- Lname = ‘Customer ABC’;
- update Customers.
- If not found;
- Write customers.
- Endif;
- Enddo.
- Close Customers;

Lets talk joins

- Can you join in record file access?
- Yes but….
- Can you join in SQL?
- Yes, easily.

Joins in SQL

- You join by select customerNumber, CustomerName from Customers c1 join Customers2 c2 on c1.customerNumber = c2.customerNumber
- To embed it in your program you..
- Exec sql select customerNumber, CustomerName from Customers c1 join Customers2 c2 on c1.customerNumber = c2.customerNumber

Joins in record level access

- You either have to use a joined logical or
- Chain multiple files.
- Dcl-f customers
- Dcl-f customers2
- Chain customerNumber customers;
- Chain customerNumber customers2

Joins

- As you can see, joins are much simpler in SQL than record level access.
- You are not limited by the number of files you use.
- You can use sub-selects.
- You can use inner, outer, join, left joins.

Join exceptions in record level access

- No (easy) exception joins.
- No (easy) outter, inner joins.
- No (easy) joins with subselects.

Advanced SQL –Cursors.

- In most cases we use cursors to process through a result set.

- You can do data manipulation and run business logic when you are looping through a cursor.

Declare cursor and open

- exec SQL
  - declare mycursor for

    select Lname, Fname
- from Customers where custno = :customerNumber;

- Open myCursor

Looping through a cursor.

```
dou endofCustomers
        exec SQL
            fetch next from mycursor
            into :Fname;

            if sqstt = 02000;  // row not found.
                leave;
            endIf;

            exec SQL insert into Customers2 ( Fname ) values ( :Fname) ;

        endDo;
Exec SQL close mycursor;
```

Prepared Statements in cursors

- Although not required, you can use prepared statements in your SQL cursors.
- Prepared statements are used when you want to use replaceable parameters.
- To do that, lets look at our cursor again, but this time using a prepared statement.

Declare cursor and open

- Dcl-s sqlStatement varchar(10000);
- Dcl-s firstName varchar(20) inz('Marina');

- Sqlstatement = 'select Lname, Fname
- from Customers where fname = ? ';
- Exec SQL PREPARE myPreparedStatement FROM :sqlStatement using :firstName;

- exec SQL declare mycursor for myPreparedStatement
-
- Open myCursor

Looping through a cursor.

```
dou endofCustomers
        exec SQL
            fetch next from mycursor
            into :Fname;

            if sqstt = 02000;  // row not found.
                leave;
            endIf;

            exec SQL insert into Customers2 ( Fname ) values ( :Fname) ;

        endDo;
Exec SQL close mycursor;
```

Record File Access Equivalent

- To use a cursor equivalent in record file access you must either do a read(E) or set lower limit.
- Dcl-f customers;
- If not %open;
- Open customers;
- Endif;
- Read(e) customers
- Dou %endoffile;
- // business logic.
- Enddo.

Advanced Cursors

- You can go from a simple cursor to a more advanced cursor like joining files together.
- Lets look at another example of the cursors.

Declare cursor and open

- Dcl-s sqlStatement varchar(10000);
- Dcl-s firstName varchar(20) inz('Marina');

- Sqlstatement = 'select Lname, Fname
- from Customers c1 join customers cs on c1.cusno = c2.custno where fname = ? ';
- Exec SQL PREPARE myPreparedStatement FROM :sqlStatement using :firstName;

- exec SQL declare mycursor for myPreparedStatement
-
- Open myCursor

Looping through a cursor.

```
dou endofCustomers
        exec SQL
            fetch next from mycursor
             into :Fname;

           if sqstt = 02000;  // row not found.
               leave;
           endIf;

// business logic

       endDo;
Exec SQL close mycursor;
```

Advanced SQL

- You can go from cursors to procedures in embedded sql.
- Sql procedures are prepared sql statements you can run over and over again.
- Lets look at one simple example.

Procedures

- Exec SQL
- begin
- declare insertName char(150);
- FOR selectCustomers AS customers CURSOR FOR
- SELECT fname from customers where custno = :customerNumber as name
- DO
- SET insertName = name;
- insert into customer2 (fname) values (name) ;
- END FOR;
- end;

Advanced Procedures

- This was just one simple example, lets look at a more advanced example that shows the power of sql.

- This next example does a cleanup of all spool files that we don’t need anymore.

Advanced SQL, Procedures

```
Exec SQL
begin
   declare execCommand char( 150 );
   FOR cleanup AS cleanupspool CURSOR FOR
         SELECT 'DLTSPLF FILE(' concat SPOOLED_FILE_NAME concat ') JOB(' concat JOB_NAME
                  concat
                  ') SPLNBR(' concat FILE_NUMBER concat ')' as command
                   FROM QSYS2.OUTPUT_QUEUE_ENTRIES
                  WHERE DAYS(CURRENT_TIMESTAMP) - DAYS(CREATE_TIMESTAMP)>90
                  and output_queue_name in ( 'QPRINT', 'QPRINTS' )
                  ORDER BY CREATE_TIMESTAMP
   DO
    SET execCommand = command;
       call qcmdexc( command );
   END FOR;
 end;
```

Error Handling

- Condition on SQL code or SQL state
- Its useful to have a utility program to give you a Boolean yes/no when you have encountered an error
- We will look at SQLUTILS our utility program for embedded SQL.

A brief intro to RPG utility programs

- Service program.
- Carrying out a routine function.
- External procedures that can be called from other programs.
- You can add/change procedures as needed.

getSQLStateMessage

```
dcl-proc getSQLStateMessage export;

    dcl-pi getSQLStateMessage varchar( 1024 );
    end-pi;

    dcl-s SQLErrorText varchar( 1024 );

    exec SQL
        get diagnostics condition 1 :SQLerrortext = message_text;

   return SQLErrorText;

end-proc;
```

isSQLError

```
dcl-proc isSQLError export;

    dcl-pi isSQLError ind;
      SQLState varchar( 5 ) value;
    end-pi;

    dcl-s beginningSQLState char( 2 );

    beginningSQLState = %subst( SQLState: 1: 2);

    if ( beginningSQLState <> '' and  beginningSQLState <> '00' and beginningSQLState <> '01' );
      return *on;
    endIf;

    return *off;

end-proc;
```

isSQLWarning

```
dcl-proc isSQLWarning export;

    dcl-pi isSQLWarning ind;
      SQLState varchar( 5 ) value;
    end-pi;

    if ( %subst( SQLState : 1 : 2 )  = '01' );
       return *on;
    endIf;

    return *off;

end-proc;
```

isSQLSucess

```
dcl-proc isSQLSuccess export;

    dcl-pi isSQLSuccess ind;
      SQLState varchar( 5 ) value;
    end-pi;

    if ( %subst( SQLState : 1 : 2 ) = '00' );
      return *on;
    endIf;

    return *off;

end-proc;
```

isRowNotFound

- dcl-proc isSQLRowNotFound export;
- dcl-c SQL_STATE_ROW_NOT_FOUND '02000';
- dcl-pi isSQLRowNotFound ind;
- SQLState varchar(5) value;
- end-pi;

- return (SQLState = SQL_STATE_ROW_NOT_FOUND);

- end-proc;

Declare cursor and open

- exec SQL
  - declare mycursor for
    select Lname, Fname
- from Customers where key = my key;
- If (isSQLError(SQLstt));
  SQLMessage = getSQLErrorMessage();
  log_log(errorId);
- Endif;
- Open myCursor;
- If (isSQLError(SQLstt));
  SQLMessage = getSQLErrorMessage();
  log_log(errorId);
- Endif;

Looping through a cursor.

```
dou isSQLError( SQLstt );
        exec SQL
            fetch next from mycursor
            into :Fname;

            if sqstt = isSQLRowNotFound( SQLstt ) // row not found.
                leave;
            endIf;

            exec SQL insert into Customers2 ( Fname ) values ( :Fname) ;

        endDo;
```

Close cursor

- Exec SQL close mycursor;
- If (isSQLError(SQLstt));

  SQLMessage = getSQLErrorMessage();

  log_log(errorId);
- Endif;

Replace record file access with SQL

- Take the examples that we have gone through and identify where you are using record file access and change those statements to use embedded SQL.
- Remember, your only restriction is what you don’t know about SQL. You can embed SQL anywhere in the program by using exec SQL statements.

Record level access to SQL

- Dcl-f Customers goes away..
- If not %open Customers
- Open Customers;
- Endif;
- This goes away…
- Chain key Customers changes into
- Exec SQL select field from Customers where custno = :customerNumber;
- Close Customers goes away.

Record level access to SQL- continued

- Update mfile goes away.
- Exec SQL update myfield = new value where mykey = key;

Record level access to SQL- continued

- delete Customers goes away.
- Exec SQL delete Customers where mykey = key;

Record level access to SQL

- Any multiple chains to different logical files can be rewritten to use one sql statement.
- Any chains to a joined logical file can be rewritten to use one sql statement.
- Depending on the use you can combine several areas in your program to one area that handles your database access.

Summary

- By comparing record file access to SQL you can see that going to SQL and leveraging the power of SQL will leave you much better off.
- Error handling is easy by using the SQL state codes.
- Its easy to replace your current record file access with embedded SQL.

Any questions?

Thank you!!

My contact Info
Marina Schwenk

- marinaschwenk23@gmail.com
- @marinaschwenk26 on twitter

- Special Thanks:
- Austin Narus, Network Security Administrator, Everbrite
- Bryan Peterson, Application Developer, Everbrite
- Steve Johnson-Evers, Lead Architect, Everbrite
- Rochelle Petty, Application Support, Everbrite
- Debbie Saugen, Owner, Debbie Saugen Consulting, LLC
- Jeff (Red) Koon, RPG Developer