This tutorial explains the key best practices when developing C/C++ applications against the IBM Data Servers; (DB2® for z/OS®; DB2 for i; DB2 for Linux, UNIX, and Windows; and Informix®). It provides details for leveraging several of the features in DB2 Connect™ that pave the way for better performance and align with best-practice recommendations. You can use this information while developing/enhancing existing applications in C/C++ targeting IBM Data Servers.

Best practices for insert operations

The insertion of data rows into data tables is one of the most common operations in data-centric applications targeting IBM Data Servers. Data insert through the application is possible either inserting a row at a time or using multi-row insert. The section below highlights the various scenarios leveraging the multi-row insert capabilities available through the DB2 CLI driver interface.

When the array size is a known value in the application

An application with single-row insert is typically as shown below:

```sql
insert into mytable(id, name) values (1, 'abc')
insert into mytable(id, name) values (2, 'def')
```

When the number of rows is known within the application layer, DB2 CLI provides a mechanism to optimize the insert using array insert as explained below. When the array size is known, `SQL_ATTR_PARAMSET_SIZE` must be set to the array size in `INSERT`, `UPDATE`, `DELETE`, and `MERGE`
statements. We can use this method with all the supported data servers (DB2 for LUW, DB2 for Z/OS, DB2 for i, etc.).

Following are the steps to leverage this capability in an application.

Step 1: Prepare the statement using SQLPrepare() API.

```c
#define NUM_ROWS 3
SQLHANDLE hstmt;
SQLRETURN cliRC;
SQLCHAR *stmt = "insert into mytable(id, name) values(?, ?)";
SQLINTEGER idVal[3] = {1, 2, 3};
SQLINTEGER nameVal[3][4] = {'abc', 'def', 'ghi'};
SQLINTEGER nrows = NUM_ROWS;
SQLINTEGER nRowCount;
SQLLEN piIndforChar[3] = {3, 3, 3};
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);
```

Step 2: Bind the array variables to parameter markers using SQLBindParameter() API.

```c
/* bind parameter 1 to the statement */
cliRC = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &idVal, 0, NULL);
/* bind parameter 2 to the statement */
cliRC = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 4, 0, nameVal, 4, &piIndforChar);
```

Step 3: Specify the number of rows in the array to server.

```c
cliRC = SQLSetStmtAttr (hstmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER) nrows, SQL_IS_INTEGER);
```

Step 4: Send out the value to the server using SQLExecute().

```c
cliRC = SQLExecute(hstmt);
```

Step 5: Call the SQLRowCount() API to find out the number of rows affected/inserted.

```c
cliRC = SQLRowCount(hstmt, nRowCount);
```

With the above steps you will now be able to perform the multi-row insert operation.

Atomicity of array execution

During multi-row insert as elaborated above, DB2 CLI provides two options to control the array execution in case of errors.

ATOMIC: While doing the array execution, if the server encounters an error with any rows, it will roll back all the rows inserted so far on that statement. If no error, it will successfully insert all the rows.

NON_ATOMIC: Regardless of the error, the statement execution for array insert will proceed and will insert only successful rows. It will report back the failed rows to the client.
DB2 CLI gives the option for the application developer to select the atomicity using SQL_ATTR_PARAMOPT_ATOMIC, setting SQL_ATTR_PARAMOPT_ATOMIC to SQL_ATOMIC_YES; this will turn your array execution to atomic. Specifying to SQL_ATOMIC_NO will turn it to non-atomic. By default, DB2 CLI does the atomic array execution. Required syntax to specify the atomicity is as follows:

```c
cliRC = SQLSetStmtAttr(hstmt, (SQLPOINTER)SQL_ATOMIC_NO, SQL_IS_INTEGER);
```

**How to identify the failed rows**

In case of non-atomic array execution, if the server commits only part of the rows, DB2 CLI provides the mechanism to know the row numbers successfully executed and which failed. To know the status of each row, the server that processed application has to call SQLSetStmtAttr with an attribute argument of SQL_ATTR_PARAMS_PROCESSED_PTR to specify the address of a variable in which the driver can return the number of sets of parameters processed, including error sets.

The variable should be an array of short integers whose array size should be the same as the value specified through SQL_ATTR_PARAMSET_SIZE. The parameter status array is allocated by the application and filled in by the driver. Its elements indicate whether the SQL statement was executed successfully for the row of parameters or whether an error occurred while processing the set of parameters. If an error occurred, the driver sets the corresponding value in the parameter status array to SQL_PARAM_ERROR and returns SQL_SUCCESS_WITH_INFO.

The required syntax to specify the param status array for array execution is as follows:

```c
SQLSMALLINT statusPtr[NUM_ROWS]; /* number of parameters processed
cliRC = SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER) statusPtr, SQL_IS_POINTER);
```

If the SQL_ATTR_PARAM_STATUS_PTR statement attribute has been set, SQLExecute of SQLExecDirect fills the parameter status array, which provides the status of each set of parameters.

The application can check the status array to determine which rows were processed. Using the row number, the application can often correct the error and resume processing.

IBM DB2 CLI updates the elements in the parameter status array with the following values:

1. **SQL_PARAM_SUCCESS**— The SQL statement was successfully executed for this set of parameters.
2. **SQL_PARAM_SUCCESS_WITH_INFO**— The SQL statement was successfully executed for this set of parameters, but warning information is available in the diagnostics data structure.
3. **SQL_PARAM_ERROR**— There was an error in processing this set of parameters. Additional error information is available in the diagnostics data structure.
4. **SQL_PARAM_UNUSED**— This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing.
5. **SQL_PARAM_DIAG_UNAVAILABLE**— DB2 CLI treats arrays of parameters as a monolithic unit and so does not generate this level of error information.
When the array size is not known in the application

When the array size is unknown at the instant in the application, or the array size is too large for the application to allocate from its pool of available memory, users can choose to use IBM DB2 CLI chaining and rely on internal optimization to send data in the array form to DRDA. We can use this method with all the supported data servers.

Following are the steps to leverage this capability in an application.

**Step 1**: Prepare the SQL statement for which we need to input the row data.

```c
SQLHANDLE hstmt;
SQLRETURN cliRC;
SQLCHAR *stmt = "insert into mytable(id, name) values(?, ?)";
SQLINTEGER idVal;
SQLINTEGER nameVal[100];
SQLINTEGER nRowCount;
SQLLEN piIndforChar;
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);
```

**Step 2**: Turn on the chaining to send the rows of data.

```c
SQLSetStmtAttr (hstmt, SQL_ATTR_CHAINING_BEGIN, (SQLPOINTER) 1, SQL_IS_INTEGER);
```

**Step 3**: Bind the parameters and issue execute.

```c
nRowCount = 1;
printf("Enter the values for id and Name for %d row\n", nRowCount);
scanf("%d", &idVal);
scanf("%99s", nameVal);
piIndforChar = strlen(nameVal);
/* bind parameter 1 to the statement */
cliRC = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, 0, 0, &idVal, 0, NULL);
/* bind parameter 2 to the statement */
cliRC = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 4, 0, nameVal, 4, piIndforChar);
```

**Step 4**: Repeat Step 3 until you have exhausted all the data in the application.

**Step 5**: End the chaining. This statement will end the SQLExecute() input chain started with SQL_ATTR_CHAINING_BEGIN. It will send all the SQLExecute() requests chained together in single shot. The server will process all the execute requests together.

```c
cliRC = SQLSetStmtAttr (hstmt,SQL_ATTR_CHAINING_END, (SQLPOINTER) 1, SQL_IS_INTEGER);
```

Assume we have sent 3 execute requests. CLI will send all the 3 requests in a single network flow. Return value cliRC represents the success or failure of the chained requests.

If SQL_ERROR or SQL_SUCCESS_WITH_INFO is returned when ending the chain (SQL_ATTR_CHAINING_END), at least one statement in the chain of statements returned SQL_ERROR or SQL_SUCCESS_WITH_INFO when it was executed.
Use the CLI diagnostic functions SQLGetDiagRec() and SQLGetDiagField() to retrieve information about what has caused the error or warning. Application can specify single row or array for execution using SQLExecute() in chaining.

**The ArrayInputChain optimization option with chaining**

The ArrayInputChain capability provides an option for a chain for data rows to be internally optimized to array for increased performance as described below.

**With DB2 for LUW and DB2 for z/OS**

The ArrayInputChain connection parameter control the DB2 CLI driver's ability to convert the individual execute requests in chaining (chaining as explained in the above section) into array execute requests against DB2 for LUW server and DB2 for i.

By default, this variable is set to -1. Setting the ArrayInputChain to -1 or positive value turns on the array input feature. Users can specify up to 32,000 for ArrayInputChain. Users can set the value for this by specifying in the connection string or specific DSN section of db2cli.ini or db2dsdriver.cfg or COMMON section of db2cli.ini and global params section of db2dsdriver.cfg.

If the user has not specified the positive value, CLI will form the array execute requests of array size 1,000. By setting this parameter value to 0, it turns off the array input feature. That means CLI won't convert the execute requests into array, but will send as a separate request. Turning the individual execute requests into array execution will increase the performance of the chain, reducing the times of execution.

**With DB2 for Linux, Unix, Windows and DB2 for i**

ColumnwiseMRI or SQL_ATTR_COLUMNWISE_MRI to SQL_COLUMNWISE_MRI_ON with chaining against DB2 for z/OS server has an ability to convert the individual execute requests in chaining (chaining explained above) into array execute requests. This feature is available from DB2 9.7 Fix Pack 3. An application can set the connection attribute columnWiseMRI to TRUE or SQL_ATTR_COLUMNWISE_MRI on connection level using SQLSetConnectAttr to convert all the chaining requests in the connection or SQL_ATTR_COLUMNWISE_MRI only on particular statements by calling SQLSetstmtAttr against the DB2 for z/OS server group.

Enabling the column-wise MRI using the SQL_ATTR_COLUMNWISE_MRI attribute before turning on the chain can convert the individual execute requests in the chaining into array execute requests. This feature is available from DB2 Connect 9.7 Fix Pack 3 onward only for INSERT SQL statements. Support for MERGE has been added later in DB2 Connect 10.5 Fix Pack 4.

**Limitations**

1. CLI can't enable this feature if the application has any parameters of type LOB or XML. CLI automatically disables the MRI feature in this case and sends all the execute requests chaining together.
2. CLI can't enable this feature if the application is specifying array using single SQLExecute() requests in the chain.
3. Don’t enable this feature if the number of rows to insert are fewer.

**Insert buffering optimization option with chaining with DB2 for LUW server**

Applications can use the insert buffering feature along with chaining to optimize the `INSERT` SQL optimization in a partitioned database environment. Users can enable this feature by setting the `SQL_ATTR_INSERT_BUFFERING` using `SQLSetConnectAttr` to `SQL_ATTR_INSERT_BUFFERING_ON` or `SQL_ATTR_INSERT_BUFFERING_IGD`.

```c
cliRC = SQLSetConnectAttr(hdbc, SQL_ATTR_INSERT_BUFFERING,(SQLPOINTER)SQL_ATTR_INSERT_BUFFERING_IGD,SQL_NTS);
```

By default, this feature is disabled. The user has to explicitly call this to enable this feature. The possible values for `SQL_ATTR_INSERT_BUFFERING` are:

1. `SQL_ATTR_INSERT_BUFFERING_OFF` (default) — Feature is turned off.
2. `SQL_ATTR_INSERT_BUFFERING_ON` — Turns on the insert buffering feature.
3. `SQL_ATTR_INSERT_BUFFERING_IGD` — Enables the insert buffering feature and ignores duplicates while processing the array buffered insert.

**Grouping of SQL statements for performance benefits with DB2 for LUW and DB2 for z/OS servers**

Compound SQL allows multiple SQL statements to be grouped into a single executable block. This block of statements, together with any input parameter values, can then be executed in a single continuous stream, reducing the execution time and network traffic. Users can execute the different SQL statements between `BEGIN COMPOUND` and `END COMPOUND` statement execution in CLI. Those statements must be executed using same statement handle. Each sub-statement executed in the compound SQL must have its own statement handle, belong to the same connection, must have the same isolation level, and must be allocated until `END COMPOUND` statement. `SQLEndTran()` cannot be called for the same connection or any connect requests between `BEGIN COMPOUND` and `END COMPOUND`.

There are two types of compound SQL:

- **Atomic** — The application receives a response from the database manager when all sub-statements have completed successfully or when one ends in an error. If one sub-statement ends in an error, the entire block is considered to have ended in an error. Any changes made to the database within the block are rolled back. Atomic compound SQL is not supported with DB2 Connect.

- **Not-atomic** — The application receives a response from the database manager when all sub-statements have completed. All sub-statements within a block are executed regardless of whether the preceding sub-statement completed successfully. The group of statements can only be rolled back if the unit of work containing the not-atomic compound SQL is rolled back.

For more information about using compound SQL in CLI applications, see Resources.
Utilizing DB2 CLI load or load with async for large bulk insert operations against DB2 for LUW servers

CLI provides the interface for the DB2 LOAD utility using CLI APIs. This LOAD feature is works exclusively against DB2 for LUW servers. This functionality allows you to insert data in CLI using LOAD instead of array insert. LOAD gives significant performance improvement at the server against array insert when inserting the large amount of data. Unlike LOAD utility CLI LOAD won't load data from files; the application has to parse the data from the file and insert it into the appropriate application parameters that correspond to the parameter markers in the prepared statement using SQLBindParameter or SQLExtendedBind API. See Resources for more information.

Starting with DB2 CLI 97 Fix Pack 4, LOAD can be used with the CLI async feature. Due to the asynchronous execution of LOAD in the application, we can get more performance benefits from application. Because with async processing, CLI will execute the LOAD request in a separate thread, which is different from the application thread. The application can use async processing by setting SQL_ATTR_ASYNC_ENABLE to SQL_ASYNC_ENABLE_ON using SQLSetStmtAttr along with the enable of LOAD using SQL_ATTR_USE_LOAD_API.

General recommendations for insert

- If possible, have the client running in the same code page as the database to avoid conversion costs on the server. The database’s code page can be determined by running get db cfg for 'database'.
- CLI will automatically perform data-type conversions in some cases, but this comes with a (small) hidden performance penalty. Try to have the input values already in the most appropriate form for the column being inserted into.

Conclusion

This tutorial has explained key best practices to apply when developing C/C++ applications against the IBM Data Servers. We have provided details for leveraging several features in DB2 Connect that offer improved performance and align with best practices.

Leverage DB2 Connect for insert operations in existing C/C++ IBM Data Server applications
Resources

- Read Executing compound SQL (CLI) statements in CLI applications" to know the steps to leverage compound SQL capability in an application.
- Read Importing data with the CLI LOAD utility in CLI applications" to know detailed steps to execute the LOAD with CLI application.
- The Information Management area on developerWorks provides resources for architects, developers, and engineers.
- Stay current with developer technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Follow developerWorks on Twitter.
- Watch developerWorks demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Get involved in the developerWorks Community. Connect with other developerWorks users while you explore developer-driven blogs, forums, groups, and wikis.
About the authors

Venkatesh Babu KS

Venkatesh Babu KS is a development engineer for the IBM Data Server CLI Driver team with IBM Software Labs, India. He has been working on the DB2 CLI Driver since 2008. He works on new features for the CLI Driver.

Shilu Mathai

Shilu Mathai is a recognized technical expert for CLI and Embedded. He is a lab advocate for DB2 Connect and is leading CLI/embedded driver feature development.

Rajendra Kamath

Rajendra Kamath currently leads the IBM Common Client and Tooling team that includes .NET, CLI/ODBC, and Visual Studio Integration for applications targeting IBM Data Servers. His primary interests is on data driven application and tooling with IBM Data Servers.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)