CITSQL
Release Notes

Please note: Upgrading to a Major Release (2.0, 3.0, 4.0, etc…) requires recompilation of all programs. Minor Releases (1.x, 2.x, etc…) document recently added features in maintenance releases (1.x.x, 2.x.x, etc…). Note that in exceptional cases, a minor release may contain updates that may require recompilation of all programs. Where this is the case, an *Important note* will be highlighted in the Release Notes.

Release

Contents

1.3

1.3.48

New

1.3.37 ImmediateCursor=[True/False] (CitSQL for PostgreSQL Only)

```
Default is :ImmediateCursor=False
```

Causes as PREPARE EXEC to be executed before the OPEN when where a CURSOR is declared with OPEN and FETCH statements:

SQL EXEC DECLARE xxx CURSOR …
OPEN xxx
FETCH .. INTO

The results of the PREPARE EXEC are stored, and returned by the FETCH.

Attention should be taken when applying the ImmediateCursor preprocessor parameter. Since the full results of the cursor are returned before the OPEN, this parameter should only be applied for cursors returning small numbers of lines.

When not using the ImmediateCursor precompiler option, you can cause the PREPARE EXEC to be executed prior to the OPEN for cursors using the statement :

SQL EXEC DECLARE xxx STATIC CURSOR …

1.3.37 SelectPrepare=[True/False] (CitSQL for PostgreSQL Only)

```
Default is :SelectPrepare=True
```

Affects the code generated by the

EXEC SQL SELECT …. INTO …

Previously, the preprocessor generated a CURSOR > OPEN > 1 FETCH > CLOSE sequence. Now, the preprocessor causes the PREPARE EXEC to be executed prior to the OPEN, and the rsults are stored. The result is an significant improvement in performance.

Setting :SelectPrepare=False causes the preprocessor to apply the former behavior.

1.3.29 Changes to licensing

CitSQL 1.3.29 (and later) require separate license files, with separate serial numbers for each product/platform pairing. As a consequence, users can no longer use the same license for multiple products, on multiple platforms. As a consequence, users deploying multiple products require multiple license files.

As with previous deployments, the default location of the license file(s) is the directory indicated by the COBOLITDIR environment variable, set in the setenv_cobolit.bat (Windows) or cobol-it-

setup.sh (Linux) script files. Alternatively, the license file(s) may be described using the COBOLIT_LICENSE environment variable.

When using the COBOL_LICENSE environment variable, and describing the multiple licenses for multiple products, the license file names should be separated by a semi-colon. As an example:

>set COBOLIT_LICENSE=C:\COBOL-IT\LICDIR\MyLicCompiler.xml;C:\COBOL-IT\LICDIR\MyLicIDE.xml;C:\COBOL-IT\LICDIR\MyLicSQL.xml)

1.3.29 :DeallocateCloseCursor=[True/False] (CitSQL for PostgreSQL Only)

```
Default is :DeallocateCloseCursor=False
```

Causes the CLOSE cursor to also deallocate the DECLARED cursor.

Normally when a program executes an OPEN Cursor, the RCQ runtime makes a DECLARE Cursor the first time the OPEN is executed. Then, all subsequent executions of the same OPEN will reuse the previously declared cursor.

In situations where several COBOL programs use the same CURSOR name or when using CANCEL, this can result in the return of the POSTGres error: “Prepared name xxx already exists” even if the cursor xxx is closed. To fix this, use this flag so that the DECLAREd cursor is deallocated when the CLOSE cursor statement is executed.

1.3.29 :CursorSynteticName=[True/False] (CitSQL for PostgreSQL Only)

```
Default is :CursorSynteticName=False
```

Causes cursor names to be generated at runtime by the RCQ runtime.

This can be required if you use the same cursor name in several modules.

1.3.20 :Prefetch=<numeric>

Allows for the prefetch of <numeric> records in a network transaction, where <numeric> is a whole number that represents the number of records to read in a networked transaction..

Prefetch can improve performance where networked transactions are concerned.
The Prefetch=<numeric> option can either be implemented on the PGSQL command line, or dynamically by populating the RCQ-CFG-PREFETCH variable before performing an EXEC-SQL FETCH statement.

As an example of a dynamic implementation of the prefetch functionality:
MOVE <numeric> TO RCQ-CFG-PREFETCH
…
EXEC-SQL FETCH …

Where <numeric> is a whole number that represents the number of records to read in a networked transaction. The Prefetch option is available only with PGSQL.

1.3.6 :CloseOnCommit=[True/False]

Default is :CloseOnCommit=False

The :CloseOnCommit option aligns CitSQL behavior with Oracle behavior in cases when a BEGIN

…
END
Construct contains an EXEC SQL SELECT statement followed by a COMMIT statement.

When :CloseOnCommit=True, the preprocessor does not close the SQL cursor when the SELECT statement has executed. Instead, the cursor is closed after the COMMIT statement is executed.

```
BEGIN
EXEC SQL
SELECT aaa FROM X INTO :HostVar
END-EXEC
…
COMMIT
END
```

The default behavior is to open the SQL cursor before the execution of the SELECT statement, and to close the SQL cursor after the execution of the SQL statement.

1.3.4 :ForceStringMode=[True/False]

Default is :ForceStringMode=False unless :DBEncoding is set to UTF-8 or utf8. Then, the default is True.

The :ForceStringMode option affects whether a PIC X data field is sent to the database as a byte-array, or as a C-String.

When set to False, CitSQL sends PIC X data to the database as a byte-array. This is done to respect the COBOL semantic that does not consider the character X'00' as the end of a string. This can be a problem if the user passes a host variable describe with PIC X to a field declared as a DATE, for example. DATEs are stored as 4-byte INTs by Postgres, so Postgres expects the DATE to have a length of 4-bytes.

When set to True, CitSQL sends PIC X data to the database as a C-String (where the String is terminated by the character X'00'). In this case, Postgres can convert the C-String into a DATE. This is not activated by default, as it is counter to the pure COBOL semantic.

Rules:

- When :DBEncoding=UTF-8 or utf8 :ForceStringMode is set to True as UTF-8 does not allow the X'00' character.
- :ForceStringMode=True only affect the behavior of non-compound fields. That is,
  In this case:
  01 COL1 PIC X(10).

  COL1 will be send a C-String

  However, in this case:
  01 GRP1.
  02 COL11 PIC X(10)
  …
  GRP1 will always be send as an array of bytes whatever the :ForceStringMode parameter is.

1.3.3 Valid CCSID values for PIC N fields

For fields described with the PIC N clause, CCSID values 1200, 1201 and UTF-16BE are the only

accepted values. Other values provoke a warning and are ignored.

EXEC SQL DECLARE :HostVar1, :HostVAR2 VARIABLE CCSID xxxx END-EXEC

1.3.2 :DefaultCCSID=<Valid codepage or UTF-8> processes VARCHAR fields

If a CCSID is defined (by default or explicitly for the field)

- STRING fields (PIC X, non-numeric USAGE DISPLAY fields and PIC N fields) are converted from “CCSID” encoding To “DBEncoding” encoding when writing to the database and from “DBEncoding” encoding to “CCSID” encoding when reading from the database.

If a field is described with the PIC N clause, it is considered to be declared as USAGE NATIONAL.

Note- For fields described as PIC N VARCHAR, the length of the -LEN fields is given in number of characters (not in number of bytes).

For clarification, in this example, the length of « 12345 » in FIELDA-ARR is 5, while the number of bytes in storage is 10:

01 FIELDA PIC N(100) VARCHAR.

MOVE « 12345 » TO FIELDA-ARR
MOVE 5 TO FIELDA-LEN

1.3.1 :NationalCodePage is replaced by :DBEncoding

The functionality provided by the :NationalCodePage option has been enhanced and replaced by the :DBEncoding option.

1.3.1 :DBEncoding=<Valid codepage or UTF-8>

The :DBEncoding option specifies what encoding is used by the database storage.

IF a “DBEncoding” is defined :

- NATIONAL PIC N fields are converted from NATIONAL to “DBEncoding” encoding when writing to the database and from “DBEncoding” encoding to NATIONAL when reading from the database.

Valid codepages are those accepted by the COBOL intrinsic functions NATIONAL-OF and DISPLAY-OF. For a complete list of the valid codepages recognized by the COBOL-IT compiler, run the command:

> cobc –list-codepage

1.3.1 :DefaultCCSID=<Valid codepage or UTF-8>

The :DefaultCCSID option specifies the default CCSID for string fields that have no explicit CCSID declaration.

Valid codepages are those accepted by the COBOL intrinsic functions NATIONAL-OF and DISPLAY-OF. For a complete list of the valid codepages recognized by the COBOL-IT compiler, run the command:

> cobc –list-codepage

Different CCSIDs can be declared for different fields in the source code. The CCSID of individual fields may be explicitly declared using the IBM syntax:

EXEC SQL DECLARE :<HOSTVARIABLE> VARIABLE CCSID <Valid codepage or UTF-8> END-EXEC.

If a CCSID is defined (by default or explicitly for the field)

- STRING fields (PIC X and non-numeric USAGE DISPLAY fields) are converted from "CCSID" encoding To "DBEncoding" encoding when writing to the database and from "DBEncoding" encoding to "CCSID" encoding when reading from the database.
- VARCHAR fields are not processed.

1.3.0 :NoRecCode=<numeric>

The :NoRecCode option sets the value returned to indicate the end of a FETCH statement, when no more records are found. The default value is 1403, which corresponds to the Oracle SQLCODE for a NOT FOUND condition.

1.3.0 :NationalCodePage=xxxx

Note- The :NationalCodePage option is replaced by the :DBEncoding option in version 1.3.1.

The :NationalCodePage option takes as a setting one of the codepages recognized by the COBOLIT compiler. For a full list of the codepages recognized by the COBOL-IT compiler, use the command:

>cobc –list-codepage

When the :NationalCodePage=xxxx option is set, input fields declared as PIC N USAGE NATIONAL will be converted in a buffer (using the DISPLAY-OF function) before being sent to the database, and output fields will be returned to a buffer and converted (using the NATIONAL-OF function) before being returned to the output field.

A typical usage would be :NationalCodePage=UTF-8

Limitation: PIC N fields described as USAGE VARCHAR are not converted.

Fixes

1.3.44 SQLERRD(3) not registering after an update

ID : 1150212241

It was possible in versions 1.3.39 through 1.3.42 of CitSQL for PostgreSQL, for the « min » symbol to be reported as undefined, causing programs to abort when checking SQLERRD(3).

This has been corrected.

1.3.43 Memory corruption when a long error message is returned by PostgreSQL

ID : 1150212119

When an SQL error was returned by PostgreSQL to SQLERRMC, which was more than 70 characters, CITSQL was not truncating the message, and as a consequence memory adjacent to SQLCA could be corrupted.

This has been corrected.

1.3.41 Regression errors fixed in Windows

ID : 1150207405

Under some conditions, some corrections made in prior versions were seen not operating as intended in PostgreSQL Windows environments.

This has been corrected.

1.3.28 Problems with NULL indicators using POSTGRESQL

ID: 1150041018

Under certain conditions, it was possible to insert a NULL into a column, then when retrieving the value, to have the operation return a zero.

In some cases, this is due to user error, but has been corrected.

1.3.28 Problem recognizing HOST indicated by a variable

ID: 150042120

In some situations, the insertion of a HOST associated with a variable would produce unexpected results.

In some cases, this is due to user error, but has been corrected.

1.3.27 Function DISPLAY-OF / NATIONAL-OF causes data conversions unnecessarily

ID: 1150031860

In an example, in which the DBEncoding and DefaultCCSID were both set to LATIN9, conversions of DISPLAY-OF and NATIONAL-OF data elements were done unnecessarily.

This has been corrected.

1.3.26 Warning during recompile

ID: 1149984738

During a pre-compilation some warning messages could be difficult to interpret.

This has been corrected.

1.3.26 Precompilation errors when using StrictMode=TRUE option

ID: 1149988064

When precompiling with the option :StrictMode=True, some SQL errors could be returned incorrectly.

This has been corrected.

1.3.25 Insert doesn't work with host variable

ID: 1149953544

Using PostgreSQL, performing an INSERT using a host variable could return a syntax error (SQLSTATE 42601).

As an example :

```
MOVE '03' TO WS-ELEMENT.
EXEC SQL
    INSERT INTO tab1
        (col_1)
    SELECT :WS-ELEMENT
END-EXEC.
```

This has been corrected.

1.3.24 Problem compiling TIEMSTAMP WITHOUT TIMEZONE

ID: 1149936034

In some situations, the TIMESTAMP WITHOUT TIME ZONE clause can cause the CitSQL pre-compiler to return a non-recoverable parse error.

This has been corrected.

1.3.21 DBENcoding option causes problem using LIKE clause with Host variables

ID# 1149807630

The LIKE clause could produce incorrect results when used with Host variables and when using the CITSQL pre-compiler options :DBEncoding=UTF-8.

This has been corrected.

1.3.21 DBEncoding option causes problem with SUBSTRING operation

ID# 1149839016

The SUBSTRING operation could produce incorrect results when using the CITSQL precompiler option :DEncoding=UTF-8.

This has been corrected.

1.3.18 SQLERRD(3) not returned

ID# 1149625672
ID# 1149676236

In application using CITSQL for PostgreSQL, the SQLERRD(3) was not updated with number of rows that qualified to be DELETEd/UPDATEd after DELETE/UPDATE statements.

This has been corrected.

1.3.16 Pre-compiler incorrectly generates extra space

ID: 1149611752

It was possible for the pre-compiler to generate an extra space in some situations, which would be detected when compiling the pre-compiled source code.

This has been corrected.

1.3.16 Precompiler parsing error

ID: 1149616568

A case was detected where a comment line inserted into an SQL statement caused the pre-compiler

to incorrectly interpret the SQL statement and as a result to incorrectly translate the SQL statement.

This has been corrected.

1.3.15 Stmt structure is null error when more than 100 cursors open

ID: 1149404106

Using PostgreSQL, when more than 100 cursors were open, a running program could abort with the error: SELECT 1 |SQLCODE=-000000001:SQLSTATE: ->the stmt structure is null

This has been corrected.

1.3.14 Non-recoverable parse error on position function

ID: 1149354666

ID: 1149496728

The position function, (as an example, …`and position(a in b.c)>0` could cause a non-recoverable parse error.

This has been corrected.

1.3.13 Select replace used with E'\\0x00' notation incorrectly translated

ID: 1149259154

The select replace function used with the E'\\0x00' notation was incorrectly translated by the pre-compiler, creating RCQ query text which included an extra space.

This has been corrected.

1.3.12 Typedef causes parsing error

ID: 1149040784

A Typedef caused the pre-compiler to generate a parsing error.

This has been corrected.

1.3.11 Error code 1, No SQLSTATE returned

ID: 1149130694

In some situations a SELECT could return an Error code 1, NO SQLSTATE returned message.

This has been corrected

1.3.11 SQLSTATE 34000 after second START of a transaction

ID: 1149215240

In version 1.3.9 of PostgreSQL, it is possible to return an SQLSTATE: 34000 after the second START of a transaction.
Following the trace file, it appears that a CLOSE statement is executed, which was not executed in version 1.3.8.

This has been corrected.

1.3.6 Improved management for SELECT FOR UPDATE in CitSQL for PostgreSQL

Locking behavior when there was concurrent execution of the same "FOR UPDATE" cursor was

not depending on the NOWAIT Statement.

This has been corrected and the cursor declared FOR UPDATE now works correctly.

The scope of this fix is currently limited to syntax located inside a transaction (BEGIN/COMMIT/ROLLBACK) block.

Syntax must conform to the PostgreSQL documentation (https://www.postgresql.org/docs/9.5/static/sql-select.html).

The FOR UPDATE statement must be the last statement of the SELECT. Note that in previous versions, this restriction did not exist.

1.3.5 Error parsing PIC N field with hex value

ID: 1149095526

A PIC N field declared with a hex value could cause CitSQL to generate a parsing error.

This has been corrected.

1.3.5 Error handling return of SQLCODE 100 in PostgreSQL

ID: 1149024522

In some cases, where an SQL statement returned with SQLSTATE = 00000, the SQLCODE returned was set to 0 instead of 100.

This has been corrected.

1.2

1.2.22

New

1.2.22 RCQLOG=<logfile>

ID# 1148813398

The RCQLOG runtime environment variable allows a filename to be associated with the logfile produced by CitSQL when precompilig with :LogMode=True :DebugMode=True.

<logfile> can include a complete path name followed by a file name. In the absence of a complete path name, the file name will be applied, and the file will be generated in the current working directory.

When the RCQLOG runtime environment variable is not set, the logfile is named RCQDDL.LOG, and created in the current working directory. The RCQLOG runtime environment variable must be set prior to the use of CitSQL.

1.2.21 :StepLimit=<Nr> option

Usage: :StepLimit=<Nr>

The CitSQL parser is based on a Backtracking technology. In order to do this, it must set a limit on the number of cases it must be able to consider. You can control this limit with the :StepLimit option. Normally you will not need to use the :StepLimit option. Howver if, when pre-compiling your source code, you receive a Step limit parse error, as in the example below, then you should include the :StepLimit option in your pre-compilation command, and increase the value above the default.

As an example:
citpgsql :IncludeSearchPath=/opt/distribPGSQL-64/include :StepLimit=2000000 ./src /testit.cbl

Example of Step limit parse error:
1:./src /testit.cbl: Step limit parse error: ./src/testit.cbl

Fixes

1.2.22 Error Fetch Cursor

ID# 1148812850

SQL statements that required more than 4K would return a “QUERY TOO LONG” error.

This has been corrected. The buffer for dynamically allocated memory is no longer restricted.

1.2.21 Syntax error returned on date command using interval clause

ID# 1148798390

It was possible for a SELECT statement that contained an interval clause to cause the pre-compiler to return a syntax error.

Example:
SELECT PAR_FMOD, date(PAR_FMOD + 1 * interval '1 day')
….

Could return a syntax error as follows:

SQLERRMC [0x7fe62bc9dc5a] = ERROR: syntax error at or near "$1"
LINE 1: SELECT PAR_FMOD,PAR_FMOD+
SQLERRP [0x7fe62bc9dca0] = 1*interv

This has been corrected.

1.2.16 Literal encoded in UTF-8 in copyfile incorrectly translated

ID# 1148712024
ID# 1148733322

A literal in a VALUE clause contained in a copyfile encoded in UTF-8 was not correctly rendered in the generated .cbp file.

As an example:
01 sample-text PIC X(46) VALUE
'âäàáãåéêëèíîïìôöòóõûüùúÂÄÀÁÃÅÉÊÈÍÎÏÌÔÖÒÓÕÛÜÙÚ'.

Was rendered in the generated .cbp file as:
01 sample-text PIC X(46) VALUE
'Ã¢Ã¤Ã Ã¡Ã£Ã¥Ã©ÃªÃ«Ã¨Ã-Ã®Ã¯Ã¬Ã´Ã¶Ã²Ã³Ãµ»Ã¼Ã¹ÃºÃ<82>Ã<84>Ã<80>Ã<81>Ã<83>Ã<85>Ã<89>

This has been corrected.

1.2.15 Pre-compile file, but returns Return code = 0

ID# 1148676582

It was possible for a pre-compile to fail, and therefore produce no output file, and terminate with a return code of 0.

This has been corrected.

1.2.15 Non-recoverable parse error processing casting command

ID# 1148695028
It was possible for a casting, such as SELECT (123.25) ::int; to return an error:

Example:
src/testit.cbl.cbl:876:67:Non-recoverable parse error
msol_to_char((MOD(BCD_NUMBCDLIVIA,4))::int)

This has been corrected.

1.2.15 Non-recoverable parse error processing position command

ID# 1148704042
It was possible for a possibition command, such as position(CAR_CODCRE in PAR_VALOR) = 0 to return an error;

Example:
src/testit.cbl:1304:48:Non-recoverable parse error
(position(CAR_CODCRE in PAR_VALOR) = 0 OR

This has been corrected.

1.2.14 Line number not provided with unsupported statement error

ID# 1148507576
It was possible, when returning an “Unsupported” error statement, that the line number would not be provided.

Example :
Unsupported: SqlSelectModifyStatement

1.2.13 Expected offset 2 actual offset 0

ID# 1148563168

In some cases, the precompiler could erroneously return the error “Expected offset 2 actual offset 0” when encountering an unsupported statement.

This has been corrected.

1.2.11 After restoring connection, CONNECT fails

ID# 1146080526
MySQL After losing and subsequently restoring a connection to the database, it was possible that the execution of a CONNECT statement would not work. CitSQL optimizations could keep statements in memory which were not being refreshed after the disconnect/restore.

This has been corrected.

1.2.11 CLOSE CURSOR aborts program

ID# 1146630198
MySQL If you perform a CLOSE CURSOR without first having OPENed the CURSOR, the program would abort.

This has been corrected. This condition will now return an SQL error.

1.2.11 CitSQL generates code with literal extending past column 72

ID: 1146058932
The PostgreSQL precompiler converted a variable declaration such that the representation of the VALUE clause extended past column 72, creating a compiler error when compiled by the COBOL-IT compiler.

This has been corrected.

1.2.11 EXEC SQL DECLARE CURSOR statement not translated properly

ID: 1148483512
The EXEC SQL DECLARE CURSOR statement could be flagged by CitSQL pre-compiler as a warning in some situations, and not be translated appropriately.

This has been corrected.

1.2.11 SQL DISCONNECT ALL statement not translated correctly

ID: 1146180792

The SQL 'DISCONNECT ALL' to do a 'DEALLOCATE ALL" which is the command PostgreSQL uses to purge prepared transactions, was not recognized.

This has been corrected.

1.2.11 Declairing a READ ONLY CURSOR not supported

ID: 1147073726

When defining a cursor without keys "read only", and operating an update statement in a loop that used the cursor, an error message would be returned that "the resource is locked from another process." As result, the update would fail.

This has been corrected.

1.1

1.1.3

New

citpgsql –V, citmysql –V, citodbc –V

The distributions of CitSQL report the license information from your product license.

Changes to the CitSQL Distribution

The distribution of CitSQL now includes binary executables for the precompiler and runtime component, as well as database client libraries, with the MySQL and PostgreSQL distributions. The distribution of CitSQL no longer includes source code, and no longer includes the samples/c folder.

Updates to the Getting Started Manual

The Getting Started with CitSQL manual has been expanded to include more examples, and to include Port Lists for all of the products.

:FreeFormatOutput=True

The FreeFormatOutput option, when set to TRUE, causes the ouput of the precompiler to be created in "free" source format. Free source format, or "terminal format" applies different rules for establishing the location of the Area A, the Indicator Area, Area B, and the Identification Area. COBOL-IT source programs that are written in Terminal Format must be compiled with the –free compiler flag. For more details on the free source format, see Source Code Management Topics in the Reference Manual.

1.1.0

New

Support of VARCHAR, LONG VARCHAR, VARRAW

CITSQL (MySql, Odbc and Postgre) provides support for variables described with USAGE VARCHAR, USAGE LONG VARCHAR, and USAGE VARRAW.
As an example:

The cobol declaration :

```
01 CUST-RECORD.
  05 COL-4 PIC X(5000) USAGE VARRAW.
  05 COL-5 PIC X(50) USAGE VARCHAR.
  05 COL-6 PIC X(5000) USAGE LONG VARCHAR.
```

Is converted by the CITSQL Preprocessor as follows: :

```
*****RC*SQL**************
*       05 COL-4  PIC X(5000)  USAGE VARRAW.
*****RC*SQL**************
```

```
        05 COL-4.
          06 COL-4-LEN PIC S9(8) COMP-5 VALUE 0.
          06 COL-4-ARR PIC X(5000).
*****RC*SQL**************
*****RC*SQL**************
*        05 COL-5  PIC X(50)    USAGE VARCHAR.
*****RC*SQL**************
        05 COL-5.
          06 COL-5-LEN PIC S9(4) COMP-5 VALUE 0.
          06 COL-5-ARR PIC X(50).
*****RC*SQL**************
*****RC*SQL**************
*        05 COL-6  PIC X(5000)  USAGE LONG VARCHAR.
*****RC*SQL**************
        05 COL-6.
          06 COL-6-LEN PIC S9(8) COMP-5 VALUE 0.
          06 COL-6-ARR PIC X(5000).
*****RC*SQL**************
```

Furthering the example :

Consider the case where your program executes a SELECT into COL-4 (described above) :

```
SELECT COL4 FROM TAB1 INTO :COL-4;
```

In this case, the actual data length read from the database is stored in COL4-LEN and the data is stored in COL-4-ARR.

Consider the case where your program executes an INSERT of COL-4 (described above):

```
INSERT into TAB1
            (
              COL_4
            )
           VALUES
            (
              :COL-4
            )
```

In this case, the COL4-LEN field is updated before the execution of the SQL Statement, so that it contains the actual length of data (in bytes) in COL4-ARR that must be written to the database .

Note that for CITSQL for MYSQL, and CITSQL for ODBC, the implementation of varaiables described with USAGE VARRAW differs from the implementation of variables described with USAGE LONG VARCHAR, in that no conversion is performed on USAGE VARRAW data either while storing or while reading the data.

CITSQL (MySql, Odbc and PostgreSQL support for variables described with
USAGE VARCHAR, USAGE LONG VARCHAR, and USAGE VARRAW can be useful in the following cases:

- In MySQL, support for data described as BLOB or VARCHAR.
- In SQL Server, support for data described as VARBINARY and VarChar.

- In PostgreSQL, support for data described as ByteA and VarChar..

Fixes

SQL error on statement in copybook did not reference copybook name in error message
When a error in SQL statement was detected, and the SQL statement was in a copybook, the copybook filename was not displayed in the error message, so the line number displayed, which was derived from the copybook, was not a useful reference.

This has been corrected. When an SQL error is detected, and the SQL statement is in a copybook, the name of the copybook and the line number in the copybook are now displayed in the error message.