

**opentext™**

# **COBOL-IT OpenESQL**

**Getting Started**

# Table of contents

---

About COBOL-IT OpenESQL	3
Documentation	3
Overview	4
Pre-requisite knowledge	5
Getting Started Guide	6
Getting Started	6
Working with the Getting Started application – Basic	7
Tutorials and Best Practice Recommendations	11
Runtime Errors and Diagnostics	16
Migration Guide	21
Migrating to CitOESQL	21
User Guide	29
Modes of Operation	30
Command line syntax	32
Source Code Formats	33
Directive Syntax	34
Control statements in source	35
Programming	37
Reference Manual	52
Developing SQL Applications	53
SQL Statements	81
CitOESQL Directives	151
Legal Notice	231
Third-Party Notices	231

# 1. About COBOL-IT OpenESQL

---

Welcome to COBOL-IT OpenESQL (CitOESQL).

## 1.1 Documentation

---

Below are the guides available for CitOESQL:



## Getting Started

Use the Getting Started application to set up an ODBC data source...



## Migration Guide

Learn how to migrate from CitSQL to CitOESQL



## User Guide

Walk through the product features.



## Reference Manual

This guide describes the programming features available for SQL applications.

---

## 1.2 Overview

---

COBOL-IT OpenESQL is an Embedded SQL (ESQL) preprocessor for COBOL-IT. It reads COBOL source code and writes amended source code where EXEC SQL statements are replaced with calls to a runtime library that accesses ODBC data sources.

COBOL-IT OpenESQL can be used as a stand-alone preprocessor that is executed separately and before the COBOL-IT compiler, or in conjunction with COBOL-IT's `-preprocess=` option, in which case it is invoked by the COBOL compiler.

When used stand-alone, the preprocessor provides conditional compilation and copybook expansion. These tasks may be performed by the COBOL compiler when used with the `-preprocess` option.

The debugger will show the code generated by CitOESQL rather than the original EXEC SQL statements. When used with the compiler's `-preprocess` option, debugging of the original source code is available.

## 1.3 Pre-requisite knowledge

---

Some basic knowledge of Embedded SQL is assumed in this document.

It is also assumed that the reader has basic knowledge of the COBOL-IT compiler and debugger and has worked through the getting started process for them. The same applies to COBOL-IT Developer Studio, if this is used.

## 2. Getting Started Guide

---

### 2.1 Getting Started

---

To work with the Getting Started application you must first set up an ODBC data source for the database of your choice. You will need to know the ODBC Data Source Name (DSN) and a valid UserID and Password for the DSN that is able to create and drop tables.

When working with PostgreSQL you must set the following ODBC options in your environment:

UpdatableCursors=0

UseDeclareFetch=1

On Linux you must include the ODBC shared object location in LD\_LIBRARY\_PATH. For example, if the unixODBC driver manager is installed in its default location, this will be:

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

The Getting Started application, sample.cbl, can be found in the following locations:

Windows: %COBOLITDIR%\samples\sql

Linux: \$COBOLITDIR/samples/sql

The application executes a series of SQL statements. Open sample.cbl in a text editor of your choice to view these statements. Comments in the source code provide information about alternative methods for opening the database connection.

To work with the application, open a command line window and execute the appropriate cobol-itsetup script for your environment (see the Installing COBOL-IT section of the COBOL-IT Compiler Suite, [Getting Started With Compiler Suite](#) guide for more details). Then change the directory to the SQL sample directory.

---

## 2.2 Working with the Getting Started application – Basic

First run the precompiler:

```
citoesql sample.cbl
```

The `-g` parameter is required if you want to compile for debugging. You can omit it from release builds.

The `-conf=citoesql.conf` parameter sets compiler options required by CitOESQL including running CitOESQL's preprocessor and linking its runtime library.

You can also run CitOESQL as a standalone preprocessor and then compile the generated output file.

Next compile the file generated by CitOESQL, which will have the same name but a `.cbp` extension.

Windows: `cobc -g -I%COBOLITDIR%\lib\ citoesqlr_dll.lib sample.cbp`

Linux: `cobc -g -L\${COBOLITDIR}/lib -lcitoesqlr sample.cbp`



### Note

The `-l` parameter links CitOESQL's runtime library.

The application can now be executed using `cobcrun`. Enter the DSN, User-ID and Password when prompted. The console output should be as follows:

```
cobcrun sample
Create/insert/update/drop test

Enter data source (Eg odbcdemo) MyDSN
Enter username[( '.'\|'/')password] (Eg admin/) myUserId.myPassword
Drop table
Error(anticipated) : cannot drop table
-00003701
Cannot drop the table 'mfesqltest', because it does not exist
Create table
Insert row
Commit
Update row
Verify data before rollback
Rollback
Verify data after rollback
Drop table
Disconnect
Create table after commit release
Cannot create table as expected
-00019702
Connection name not found.
Test completed without error
```

You can debug the application using `cobcdb`, as detailed in the Using the COBOL-IT Debugger section of the COBOL-IT Compiler Suite, [Getting Started With Compiler Suite](#) guide.

To build an executable program rather than the default dll or shared object add `-x` to the command line as shown in the example below:

```
cobc -x -g -conf=citoesql.conf sample.cbl
```

## 2.2.1 Working with the Getting Started application in Developer Studio

---

### Prerequisites

Set up a new workspace by following the steps documented in the Developer Studio Getting Started manual. In summary:

1. Validate the COBOL-IT license using the menu sequence **Window > Preferences > COBOL** then click the **Browse** button to select the COBOL-IT license file.
  2. Using the menu sequence **Window > Preferences > General > Editors > Text Editor**, ensure **Show line numbers** has been checked.
  3. Using the menu sequence **Window > Preferences > General > Workspace**:
    - De-select **Build automatically**
    - Select **Refresh using native hooks or polling**
    - Select **Refresh on access**
    - Select **Save automatically before build**
  4. Using the menu sequence **Window > Preferences > Run/Debug > Perspectives**, select **Always** for the **Open the associated perspective when launching**.
- 

## 2.2.2 Project creation and import of source code

---

Follow the steps listed to create a new project and import source code.



1. Select **File > New > COBOL Project...** to open the COBOL Project Wizard.
  2. Enter a project name, for example **samples**, and click **Finish**.
  3. Right click on the newly created project and select **New > Folder**, enter the name **object** and click **Finish**.
  4. Right click on the project again and select **Import** followed by **General** and **FileSystem** then click **Next**.
  5. Click **Browse** and navigate to the **COBOL-IT samples > sql directory**, then click **OK**. Select **sample.cbl** and click **Finish**.
  6. Right click on the project again and select **Properties** followed by **COBOL Properties** then check the **Enable source settings** option.
  7. Select **DevOps Tools > Debugging Tools**. Check **Enable source settings** and **Produce debugging metadata in compiled object**.
  8. Select **Dialects > Compiler Configuration Files**. Enter **citoesql.conf** for **Use <file> as configuration file**.
  9. Select **Link > Full Build** and set **Build type** to **Build executable program**.
  10. Select **Standard Options** and check **Save object file in source folder or in <directory>** and enter **object** in the textbox.
  11. Click **Apply** and **Close**.
- 

## Build and Debugging

To build the project use the menu sequence **Project > Clean**, ensure the project is selected for cleaning and building and then click **Clean**.

You now need to set up a debug configuration as follows:

1. Double click on sample.cbl to open the source file.
2. Scroll down to the EXEC SQL CONNECT statement, right click the shaded area to the left of the line number of the EXEC SQL line and choose **Toggle Breakpoint**. A blue circle should appear next to the line number. You can also use double click to set and unset breakpoints as an alternative to right click.
3. Right click on sample.cbl in the project file tree in the left-hand pane, select **Debug As followed by Debug Configurations**.
4. Click on **Cobol Program** followed by the left most icon in the toolbar above (New launch configuration) and enter 'sample' as the configuration name at the top of the dialog box.
5. Check that the **Project Name** is set to the correct project and that **Program** is set to sample.cbl.
6. Click **Debug**.
7. Respond to the prompts from the program with the ODBC Data Source Name, User-ID and Password.
8. The application will stop on the breakpoint. You can now use other debugger features to inspect COBOL variables, manage other breakpoints, use single step code execution, etc. Please consult COBOL-IT Developer Studio, [Getting Started: The Debugger Perspective](#) guide for more information.

## 2.3 Tutorials and Best Practice Recommendations

---

### Note

The tutorial applications are described below for command line use. If you prefer, they can be used in Developer Studio with directives following the pattern described in Working with the Getting Started application in Developer Studio.

You can control the behavior of CitOESQL by setting precompiler directives. This can be done in a variety of ways, on the command line, in source code and in directive files. The order in which sources of directives are used, and hence the reverse order of precedence if a directive is set more than once is:

The command line

Automatic directive files

Source code

Directive files can be specified explicitly with the USE directive, however some directive files, if they exist, are used automatically by CitOESQL. These are recommended as the most convenient way of managing directive use. The automatic directive files, and the order in which they are applied are:

- `%COBOLITDIR%\etc\citoesql.dir` (Windows) or `$COBOLITDIR/etc/citoesql.dir` (Linux and Unix).

This is useful for setting directives that will be applied to all programs.

- `citoesql.dir` in the directory containing the COBOL source code.

This is useful for setting directives that are common to groups of programs.

- A file with the same name and in the same directory as the program being pre-compiled but with a file extension of `'dir'`.

---

### 2.3.1 Performance Tuning

---

#### *Introduction*

Reducing the number of interactions required between components to perform a given task will often improve application performance. In a SQL application, the most significant of these is the number of calls made between the client application and the database management system, often referred to as 'round trips'. In some cases, the database client library or ODBC driver will perform such optimizations transparently on behalf of the client application. In other cases, a client application can use optimization techniques explicitly.

CitOESQL provides several ways to optimize the performance of SQL applications and some of these are demonstrated in the tune.cbl sample application.

#### Note

Benchmark applications such as tune.cbl should be used with great care in predicting the performance of other applications. There are many factors that can affect application performance that benchmarks may fail to consider. Hardware configurations and the relative speeds of CPU, memory, disk and network also play a significant role, as do operating system overheads and other activity on the machine. tune.cbl is intended only to provide an illustrative guide to the relative performance of different coding approaches and the potential usage of the tuning parameters available in CitOESQL.

A prerequisite for running this application is an ODBC data source, see the Getting Started section for more details.

On Windows open a COBOL-IT command line window, on Linux or Unix run the COBOL-IT command line setup script.

- Change directory to the CobolIt (or CobolIT64) samples\sql or samples/sql directory
- Build the application with the following command line:

```
cobc -g -x -conf=citoesql.conf tune.cbl timer.cbl
```

- Run the application with the following command line:

```
tune (Windows) or ./tune (Linux and Unix)
```

- Respond to the prompts for Data Source, User-ID and Password.

You will see elapsed times for;

- Inserting rows via a single row insert statement and an array insert statement.
- Selecting and fetching rows using read only and updatable cursors or SELECT INTO statements using single row or multi-row retrieval.

---

### *Transparent Cursor Prefetch*

When using single row cursor FETCH statements, CitOESQL can prefetch rows. By default, it will use a prefetch of 8 rows for read only cursors and 4 rows for updatable cursors. If you edit tune.dir you will see the prefetch settings at their default values. You can experiment with them as follows:

- Try setting them to 1 to see how much default prefetching improves performance.
- Try commenting them out with a # at the start of the line to confirm their default values.
- Try using larger values. You may find that very large values perform less well than smaller values in some cases and that for a given application there is often a “sweet spot” that performs best.

### *Host Variable Arrays*

CitOESQL supports the use of host variable arrays to insert and fetch multiple rows at a time. When using array host variables you can use a FOR :count clause if you do not want to use the whole array. Edit tune.cbl and familiarize yourself with the syntax. You can change the 78-level constants demoRows, insertArraySize, readOnlyarraySize and forUpdatearraySize to change the number of rows in the table and the size of the host variable arrays. You can experiment with different values to see how different array sizes impact performance and how the BEHAVIOR directive and its subdirectives perform relative to host variable arrays.

It is often best to use a smaller array size for updatable cursors than for read only cursors. Although a larger array size will generally improve performance, it can also increase contention and lock wait delays. When considering array sizes and prefetch sizes for updatable cursors, you should consider if your databases hold FOR UPDATE locks only when a row is available on the client, for example in Microsoft SQL Server, or if FOR UPDATE locks are held until the current transaction terminates, for example in Oracle and PostgreSQL.

### *SQL Statement Cache*

CitOESQL maintains a cache of prepared statements. The default cache size is 20. You can use the STMTCACHE directive to change this. For a large batch, application values up to the low hundreds may be beneficial.

---

## **2.3.2 SQL Syntax Checking Options**

CitOESQL syntax checking is designed to be lightweight and tolerant of server-specific SQL extensions. This means that it may not detect all SQL errors. You can enable more rigorous checking with the CHECK directive. This also requires the DB directive and, in most cases, the PASS directive. You can see how this works with the tune.cbl sample:

- Navigate to %COBOLITDIR%\Samples\sql or \$COBOLITDIR/Sample/sql and open tune.dir in an editor of your choice.
- Remove the # character from the start of the line that starts #db and replace the string <your ODBC Data Source Name> with the name of your ODBC data source.
- Unless you are using operating system authentication, remove the # character from the line that starts #pass and replace the string <UserID> with your database User-ID, and the string <Password> with your database password.
- Remove the # character from the line that starts #check .
- Save the file, and open tune.cbl.
- Search for the first INSERT INTO statement and change the keyword into to intoo .
- Compile tune.cbl and notice the syntax-based error message.
- Change the intoo back to into , the table name oesqldemo to xyz and re-compile. Note the error message is now for a reference to a non-existent table.

Sometimes you cannot avoid errors because a table does not exist in the database, for example if the table is a temporary table that the database automatically drops at disconnect time. CitOESQL can work around this by executing DDL statements at precompile time.

- Navigate to the start of tune.cbl and search for create table .
- Note that the EXEC SQL statement starts with the prefix '[also check]'. This tells the precompiler that this statement should be executed at both precompile and execute time.
- The prefix '[only check]' instructs the precompiler that the statement should only be executed at precompile time.
- Go back to the top of the program and search for 'drop table'. Note the prefix '[also check ignore error]'. This instructs the precompiler to execute the statement at both precompile time and execute time and to ignore any precompile time errors.

There may be other cases where a precompile time server-detected error is unavoidable, in which case the prefix [nocheck] can be used.

- Return to the start of `tune.cbl` and search for 'set :dbms'. Note that immediately after opening a database connection, `tune.cbl` uses a `set :<hostVariable = current database` `CitOESQL` statement to determine the type of database it is connected to.
- Search for `if dbms =` and then scroll down a few lines until you can see two create table statements. `tune.cbl` includes two create table statements to enable it to use the TYPE `VARCHAR` with most databases. `VARCHAR2` is used with Oracle and exploits the create or replace syntax to avoid separate create and drop statements.
- The create table statements have the prefix `[nocheck] [also check ignore error]`. This has the following effect:
  - `[nocheck]` means there will be no syntax check at precompile time.
  - `[also check ignore error]` means the statement will be executed at compile time and potentially also at runtime, however in `tune.cbl` COBOL code ensures only one of the two statements will be executed at runtime. Any precompile time execution error will be ignored.
- Note that this is a somewhat contrived example for demonstration purposes.

When migrating an application to a new database, you can use the `IGNORESCHEMAERRORS` directive in conjunction with the `CHECK` directive. This limits server syntax checking to syntax alone and does not treat missing tables and columns as errors. This may be useful in obtaining a quick appraisal of SQL syntax differences that need to be remediated before the schema has been migrated.

## 2.4 Runtime Errors and Diagnostics

Diagnostic information for SQL errors can be obtained in several ways in CitOESQL.

- The most common method in most embedded SQL applications is via a SQLCA data structure. The SQLCA is supplied by including `exec sql include sqlca endexec`, where SQLCODE contains a numeric error code and SQLSTATE a 5-character string, both of which identify the error condition. SQLERRMC contains a brief error message and SQLERRML contains the length of the message.
- An application may simply declare SQLCODE and/or SQLSTATE without using an SQLCA if it only requires the type of error.
- SQLERRMC is limited to 70 bytes. A longer and more complete error can be obtained by declaring a PIC X(n) field named MFSQLMESSAGETEXT where 'n' can be of any size, but 256 characters is generally sufficient to obtain the complete error message.

### Note

ODBC error messages start with one or more component names in square brackets followed by the message text. These identify where the error message was detected, for example, by the ODBC Driver Manager or the ODBC driver, or the database server. CitOESQL removes any text in square brackets from SQLERRMC, but not from MFSQLMESSAGETEXT.

- Finally, complete diagnostic information, including the possibility of a SQL error returning more than one error, can be obtained via a GET DIAGNOSTICS statement.

All these methods are demonstrated by the `errref.cbl` sample application.

- Build the application in the `sql samples` directory via the following

```
command: cobc -x -g -conf=citoesql.conf errref.cbl
```

- Execute the sample via:

```
errref (Windows) or ./errref (Linux and Unix)
```

### Note

`errref` generates the following common warning and error conditions:

- No data found or returned.
- Too many rows returned for SELECT INTO.
- Character data truncation.
- Attempt to insert a row with a duplicate key.



Open the `erref.cbl` file and inspect the code to see how the output was generated using the methods described above.

You will need the output of the application for the next tutorial

---

## 2.4.1 Diagnostic mapping for database migration

---

When migrating an application from one database to another you will need to address differences in SQL syntax between the two systems.

It may also be required to address cases where application logic or operation procedures have dependencies on the error diagnostics returned by the database. CitOESQL can assist this process by allowing diagnostics returned by the new database for a given error condition to be mapped to the diagnostics returned by the old database system, thus avoiding the need to make changes to the source code. This is achieved by use of an error mapping file. Errors can be mapped for an application via the `ERRORMAP` precompiler directive, or for the current connection by executing a `SET ERRORMAP` statement. These methods are demonstrated by the `errmap.cbl` sample.

- Navigate to the sql samples directory.
- Open login.cpy in a text editor of your choice and edit the svr and usrpas fields to match your ODBC DSN, User ID and Password. If using Operating System authentication usrpas should be SPACES. Save the file.
- Open errmap.dir and note the directive errmap=test. This means that at runtime errmap will load mapping information from a file named test.emap.
- Open test.emap. This is set up to map diagnostics returned by PostgreSQL. You can use the output from errref to update this file for other databases.

#### Note

Error mapping can match diagnostic information to be mapped by any combination of SQLCODE, SQLSTATE and a substring within the error message text. The mapping process will change SQLCODE and SQLSTATE to new values; specify the original values to leave them unchanged. The error message text can be:

Left unchanged by omitting it from the mapping

Replaced by new text

supressed (i.e set to spaces by specifying a single ~ character as the replacement text)

- Replacement error messages start with [test]. This will appear in message text returned in CITSQLEMESSAGETEXT and by GET DIAGNOSTICS, but not in SQLERRMC. It is there as a simple visual check that an error has been mapped successfully.
- Open conn.emap. This is very similar to test.emap, but will be used with a SET ERRORMAP statement rather than the ERRORMAP directive. Replacement error messages start with [conn].
- Set the environment variable CIT\_ERRORMAP\_PATH to the current directory. This can be ''. This will cause CitoESQL to look for the mapping files in the current directory and is useful for testing new mapping files. If not set, the default location of %COBOLITDIR%\etc (Windows) or \%COBOLITDIR/etc (Linux) will be used.
- Compile and run errormap.

```
cobc -g -x -conf=citoesql.conf errmap.cbl
errmap
```

- Inspect the output and verify that errors have been mapped.
- Open errmap.cbl and inspect the code.

#### To deploy error mapping files for general use:

- Edit the files and remove any unwanted leading text in square brackets at the start of replacement error messages.
- Copy the files to the default location: %COBOLITDIR%\etc (Windows) or \$COBOLITDIR/etc (Linux)
- Unset environment variable CIT\_ERRORMAP\_PATH (if set).

## 2.4.2 COBOL-IT CitOESQL files and locations

File name	Windows location relative to %COBOLITDIR%	Linux/Unix location relative to \$COBOLITDIR)	Use
SQLCA.cpy	copy	share/cobol-it/copy	SQLCA definition
SQLDA.cpy	copy	share/cobol-it/copy	SQLDA definition
runcitoesql.bat runcitoesql.sh	bin	bin	Used by cobc - preprocess to execute the citoesql precompiler
citoesql.dir	etc	etc	Global default citoesql directives
citoesql.conf	config	share/cobol-it/config	cobc configuration file for integrated precompilation
citoesqlx.conf	config	share/cobol-it/config	cobc configuration file when citoesql is used without integrated precompilation

File name	Windows location relative to %COBOLITDIR%	Linux/Unix location relative to \$COBOLITDIR)	Use
*.emap	etc	etc	Default location for error mapping files

### 2.4.3 Performance and Diagnostic Aids

CitOESQL provides two execution tracing capabilities that may help you in diagnosing bugs and performance issues. Both are controlled by directives.

Setting ODBCTRACE=ALWAYS will enable ODBC tracing in the ODBC driver manager. This can also be done via the ODBC Administrator (Windows) or by editing the ODBC configuration files odbc.ini and odbcinst.ini (Linux), however when developing or debugging an application you may find the precompiler directive more convenient. Trace information is appended to the trace file if it already exists, so you must remember to delete or clear the trace file between runs. ODBCTRACE traces ODBC entry to and exit from ODBC API calls along with diagnostic error messages. ODBCTRACE writes all trace records to disk immediately and consequently this can have a significant impact on performance.

TRACELEVEL traces the calls an application makes to the CitOESQL's runtime library. The resultant traces can be used to analyse performance issues. TRACELEVEL offers several levels of detail and records directive settings and tuning statistics, such as the number of rows read by a cursor. TRACELEVEL has significantly less performance impact than ODBCTRACE but if an application terminates abnormally not all trace events may be recorded.

# 3. Migration Guide

## 3.1 Migrating to CitOESQL

### 3.1.1 CitSQL and CitOESQL Comparison

Implementation Comparison	CitSQL	CitOESQL
Command Line	<code>&gt;citsql [options]   @optionFile](source file(s)   @sourcesFile)</code>	<code>&gt;citoesql [options]   [@]filesNames(s)</code>  Options may include USE

### 3.1.2 Command line options

CitSQL	CitOESQL
<code>:CloseOnCommit=[True/False]</code>  For Oracle compatibility, cursors after a commit.	<code>-CLOSEONCOMMIT=YES NO</code>  <code>-CLOSEONROLLBACK=YES NO</code>
<code>:CursorSynteticName=[True/False]</code>  Causes the Cursor Name to be generated dynamically at runtime by the RCQ runtime.	Unique cursor names are always generated at runtime.
<code>:DBEncoding=&lt;Codepage or UTF-8]</code>  Specifies what encoding is used by the DB storage.	ODBC drivers provide conversion between the client codepage and the database encoding, it is not necessary to inform the precompiler of the database encoding.

CitSQL	CitOESQL
<p data-bbox="225 237 735 271">:DeallocateCloseCursor=[True/False]</p> <p data-bbox="225 327 804 405">Causes the CLOSE CURSOR statement to also deallocate the DECLARED cursor.</p>	<p data-bbox="879 237 1358 495">No equivalent. CitOESQL maintains a cache of prepared statements to optimize performance, cursors, and other prepared statements. These are deallocated when necessary using a least recently used algorithm.</p>
<p data-bbox="225 533 804 566">:DefaultCCSID=&lt;Valid codepage or UTF-8)</p> <p data-bbox="225 622 767 701">Specifies the default CCSID for string fields that have no explicit CCSID declaration.</p>	<p data-bbox="879 533 1382 656">No equivalent, however, ODBC drivers will automatically convert data between the client and database encodings.</p>
<p data-bbox="225 734 576 768">:DebugMode= [TRUE/FALSE]</p> <p data-bbox="225 824 804 947">Default [FALSE] When set to TRUE, causes log file created when LogMode=TRUE to contain more detail in some situations.</p>	<p data-bbox="879 734 1366 857">The TRACELEVEL option provides varying degrees of logging for diagnostic and optimization purposes.</p> <p data-bbox="879 913 1374 1037">ODBC provides its own API logging capability that can be turned on and off without recompiling an application.</p>
<p data-bbox="225 1077 655 1111">:ForceStringMode=[TRUE/FALSE]</p> <p data-bbox="225 1167 804 1424">When set to True, CitSQL noncompound PIC X data fields are sent to the database as a C-String (where the String is terminated by the character X'00'). When set to FALSE, CitSQL sends PIC X data to the database as a byte-array.</p>	<p data-bbox="879 1077 1150 1111">-[NO]ALLOWNULLCHAR</p>
<p data-bbox="225 1458 663 1491">:FreeFormatOutput=[TRUE/FALSE]</p> <p data-bbox="225 1547 751 1671">Default [FALSE] When set to TRUE, causes output of the precompiler to be created in "free" source format.</p>	<p data-bbox="879 1458 1278 1536">- SOURCEFORMAT=(FIXED FREE VARIABLE)</p>

## CitSQL

`:ImmediateCursor=[True/False]`

(CitSQL for PostgreSQL Only) When set to True, causes a PREPARE EXEC to be executed before the OPEN when a CURSOR is declared with OPEN and FETCH statements

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

(CitSQL for PostgreSQL Only) When set to TRUE, causes a PREPARE EXEC statement to be executed before the OPEN statement when a CURSOR is declared with OPEN and FETCH statements.

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

`:IncludeSearchPath=<Path>`

Default [Current Working Directory] A comma, or semicolon separated list of the directories in which CitSQL will look for Include files.

`:LibName=<libname>`

Defaults are: RCQMYSQL (MySQL) and RCQPGSQL (PostgreSQL).

## CitOESQL

Declare cursor is purely declarative which enables it to be placed in the DATA DIVISION, which some legacy applications depend on.

CitOESQL optimizes cursor behavior for SELECT INTO statements and OPEN CURSOR statements.

COBCPY environment variable plus current directory.

No equivalent. The runtime library `odbcw32.dll` (or its Linux equivalent shared object) may not be renamed.

CitSQL	CitOESQL
<p><code>:LogMode=[ TRUE/FALSE ]</code></p> <p>Default is: [FALSE] When set to TRUE the runtime component creates a log file called RCQDLL.log that traces all SQL operations.</p>	<p><code>-TRACELEVEL=&lt;number&gt;</code></p>
<p><code>:MaxMem=&lt;number megabytes&gt;</code></p> <p>Default is: 100 Allocates memory for the precompilation of very large source files.</p>	<p>No equivalent.</p>
<p><code>:NoRecCode=&lt;numeric&gt;</code></p> <p>Default is 1403. Allows mapping of value returned to indicate the end of a FETCH statement.</p>	<p>Default is 100 as defined by ANSI. - ERRORMAP can be used to override default 100 value.</p>
<p><code>:Prefetch=&lt;numeric&gt;</code></p> <p>Allows for the prefetch of records in a network transaction, where there is a whole number that represents the number of records to read in a networked transaction. The Prefetch option is available only with PGSQL.</p>	<p><code>-PF_RO_CURSOR</code> <code>-PF_UPD_CURSOR</code></p> <p>Prefetch sizes may be set separately for read only and updatable cursors. High values may benefit read only cursors but can cause concurrency conflicts for updatable cursors. Available with all databases.</p>
<p><code>:QuoteTranslation=&lt;pattern&gt;</code></p> <p>Default is: <b>QDB</b></p> <p>Allows mapping of single quotes, double quotes, and back quotes. By default, quotes are unchanged, which corresponds to a default value of QuoteTranslation=QDB.</p>	<p>No equivalent.</p> <p>All major databases allow use of single and double quotes consistently in line with ANSI.</p> <p>Mapping of back quotes could be helpful to applications developed for Microsoft Access being ported to other databases.</p>



CitSQL	CitOESQL
<pre data-bbox="228 237 627 264">:SelectPrepare=[True/False]</pre> <p data-bbox="228 327 802 533">(CitSQL for PostgreSQL Only) Default is TRUE. Now, the preprocessor causes the PREPARE EXEC statement to be executed prior to the OPEN statement and the results stored. When set to False, the former behavior is applied.</p>	No similar requirement.
<pre data-bbox="228 577 580 604">:StandardPrefix=&lt;prefix&gt;</pre> <p data-bbox="228 667 802 741">Default is: [None] Characters prefixed to generated data items.</p>	Currently all generated data items are prefixed by 'PCS-' (for precompiler services).
<pre data-bbox="228 786 512 813">:StepLimit=&lt;numeric&gt;</pre> <p data-bbox="228 875 802 1126">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.</p> <p data-bbox="228 1189 802 1435">The CitSQL parser is based on a Backtracking technology where 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. In general you will not need to use the :StepLimit option.</p>	No similar requirement
<pre data-bbox="228 1480 580 1507">;StrictMode=[TRUE/FALSE]</pre> <p data-bbox="228 1570 802 1731">Default is: [FALSE] When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an SQL syntax in an EXEC statement.</p>	<pre data-bbox="884 1480 1265 1597">-[NO]CHECK (plus -DB=&lt;connectionName&gt; and - PASS=&lt;userid&gt;.&lt;password&gt;)</pre> <p data-bbox="884 1659 1385 1776">When set a database connection will be used at compile time to validate SQL statements.</p>
<pre data-bbox="228 1821 683 1848">:StrictPictureMode=[TRUE/FALSE]</pre> <p data-bbox="228 1910 802 2022">Default is: [FALSE] When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize a PICTURE clause.</p>	<p data-bbox="884 1821 1066 1848">No equivalent.</p> <p data-bbox="884 1910 1385 2067">The preprocessor generates an error if a host variable is used that does not have a data type acceptable for use as a host variable.</p>

CitSQL	CitOESQL
<p data-bbox="225 237 596 271">: TargetPattern=&lt;pattern&gt;</p> <p data-bbox="225 327 807 495">Describes a group of tokens that can be strung together as components to describe the location and naming convention applied to the precompiled target file.</p>	<p data-bbox="879 237 1358 450">No equivalent. Output files are always generated in the same location as source files and with an extension of '.cbp' except when COBOL-IT's preprocess directive is used.</p>
<p data-bbox="225 533 552 566">:TrimMode=[ TRUE/FALSE ]</p> <p data-bbox="225 622 799 835">Default is: [FALSE] When set to TRUE, alphanumeric (PIC X) strings that are passed to the database are first trimmed, (right space removed), so that the data in the database does not have trailing spaces.</p>	<p data-bbox="879 533 1382 611">-PICXBINDING={DEFAULT   PAD   TRIM   TRIMALL   FIXED   VARIABLE}</p>
<p data-bbox="225 869 624 902">:TruncComments=[ TRUE/FALSE ]</p> <p data-bbox="225 958 799 1081">When set to TRUE, Comments are truncated after column 72. When set to FALSE, comments are not truncated after column 72.</p>	<p data-bbox="879 869 1062 902">No equivalent.</p>

**CitSQL**`:UTFInput=[TRUE/FALSE]`

When set to TRUE, specifies that the source code contains literals encoded in UTF-8.

**CitOESQL**

No equivalent.

### 3.1.3 Host Variables

**CitSQL**

CitSQL supports the non-COBOL USAGE Clauses:

USAGE [LONG]VARCHAR  
USAGE [LONG] VARRAW  
USAGE VARYING

**CitOESQL**

CitOESQL supports the non-COBOL USAGE Clauses:

USAGE [LONG]VARCHAR  
USAGE [LONG] VARRAW  
USAGE VARYING

Supported SQL [TYPE] [IS]:

DATE, DATE-RECORD  
TIME, TIME-RECORD  
TIMESTAMP, TIMESTAMP-RECORD  
TIMESTAMP-OFFSET, TIMESTAMPOFFSET-RECORD  
BINARY, VARBINARY, LONG-VARBINARY  
CHAR, LONG-VARCHAR  
CHAR-VARYING

### 3.1.4 ESQL features

CitSQL	CitOESQL
N/A	Passes all the NIST ANSI ESQL compliance tests.
	Includes support for dynamic SQL.
	Multiple documented extensions to ANSI ESQL:
	- Array insert and fetch
	- GET DIAGNOSTICS
	- SAVEPOINT support
	Directives and constants in source code based on MF \$SET syntax.
	Conditional compilation based on MF COBOL conditional compilation built into the precompiler.

### 3.1.5 Dependencies and Limitations

CitSQL	CitOESQL
N/A	Requires ODBC on Windows and Linux.
	Database ODBC drivers may depend on database client libraries.
	Requires 3rd party ODBC Driver Manager for Linux (UnixODBC recommended).
	32 and 64 bit support on Linux and Windows.

### 3.1.6 SQL Statement Differences and Limitations

CitSQL	CitOESQL
N/A	Does not support CitSQL EXEC SQL CONNECT statement syntax directly, however, does provide 6 flexible ODBC alternatives.
	Does not support CitSQL EXEC SQL DECLARE hostvar VARIABLE CCSID xxxxx statement.
	Does not support CitSQL PIC N USAGE VARCHAR USAGE CLAUSE.

## 4. User Guide

---

## 4.1 Modes of Operation

---

CitOESQL can be used as a standalone preprocess that executes before and separately from cobc, or in conjunction with cobc's -preprocess directive. This latter mode is referred to as integrated precompilation.

### 4.1.1 Standalone precompilation

---

When used standalone, you can use \$DISPLAY and \$SET statements in source code to manage CitOESQL directives and set constants. CitOESQL handles copybook expansion, constant setting and conditional compilation.

When debugging, you will see the code generated by CitOESQL.

### 4.1.2 Integrated precompilation

---

With integrated precompilation the following steps take place: - cobc reads the source code in an initial pass that handles constants and conditional compilation and writes the updated source code to a temporary file. This process also adds metadata comment lines to allow the debugger to locate the original, un-preprocessed source code

- cobc executes the script specified by the -preprocess option to execute CitOESQL passing it the name of the temporary file generated in the previous step and another temporary filename to be used for the precompiled output.
- CitOESQL preprocesses EXEC SQL statements and writes the preprocessed output source code to the temporary file requested by cobc.
- cobc executes the remainder of the compilation process.

When using integrated precompilation you will see the original source code in the debugger.

When using integrated precompilation:

- Do not use \$SET statements in source code to set CitOESQL directives, these will be ignored and have no effect.
- Do not use \$DISPLAY statements in the source code, in this case cobc will output a warning.
- Do not use conditional compilation that depends on constants set via CitOESQL directives or on the CitOESQL command line.
- Set all constants required for conditional compilation on the cobc command or in source code using \$SET CONSTANT statements that are compatible with cobc.
- Set CitOESQL directives on the CitOESQL command line or in a directives file for CitOESQL (via USE directive(s) on the CitOESQL command line).

To set CitOESQL directives you must edit the script file used to invoke CitOESQL. A sample script file is provided in %COBOLITDIR%\bin\runcitoesql.bat on Windows and \$COBOLITDIR/bin/runcitoesql.sh on Linux. When opening the script for integrated precompilation, cobc will search the current directory and %COBOLITDIR%\bin\ on Windows and \$COBOLITDIR/bin/ on Linux if no path is specified. Alternatively, you may specify an absolute or relative path for your script file. It will often be convenient to copy and rename the default script file to your source code directory and to make this directory current when compiling.

In the script file you should place CitOESQL directives after the directive and before the final two command line parameters (these are the input and output files specified by cobc).

## 4.2 Command line syntax

---

The command line for CitOESQL takes the following form:

```
citoesql [directive1 directive2 ...] [@]filename1 [@] filename2 ...
```

Each directive specified on the command line must be preceded by a hyphen.

filename1 filename2 ... is normally a sequence of filenames. Filenames not preceded by an '@' character are assumed to be COBOL source files. Each COBOL file will be preprocessed and if there are no errors detected an output file with the same name but a .cbp extension will be written.

If a filename starts with an '@' character, the file is assumed to be a text file containing a list of input COBOL filenames, one per line.

If the first precompiler directive on the command line is `P`, then exactly two filenames must be supplied. The first is the input filename and the second is the output filename. This directive should only be used in the script used by the COBOL-IT COBOL compiler in association with the `preprocess=<script> option`.

---

### 4.2.1 Placing Precompiler Directives in Files

---

You can place precompiler directives in a file as well as the command line or in source code via the USE directive.

In a directives file:

- blank lines or lines with a hash symbol (#) or semicolon (;) in column 1 are ignored

- directives optionally be preceded by one or two hyphens(-)

- directives may be grouped together on a line and separated by a space or comma(,)

- directives may be grouped together and wrapped in a SQL directive thus

```
SQL(directive1 [, directive2 ...])
```

---



## 4.3 Source Code Formats

---

### 4.3.1 Source code formats

---

CitOESQL supports the following source code formats:

- Fixed, which corresponds to traditional COBOL with the source code in columns 8-72 and the indicator in column 7.
- Variable, which extends the right margin from column 72 to column 256 and the indicator column remains in column 7.
- Free, or Terminal, format. In this format the indicator in column 1 is optional, source code can start in column 1 or 2 and extend to column 248.

The source code format is specified by the SOURCEFORMAT directive, for example:

```
citoesql -sourceformat=variable gs.cbl
```

on the command line, or

```
$SET SQL(sourceformat=variable)
```

at the start of a source file, or

```
sourceformat=variable
```

or

```
SQL(sourceformat=variable)
```

in directives file.

## 4.4 Directive Syntax

---

### 4.4.1 Directive syntax

---

Directives are case insensitive. The format of a directive is one of:

```
[[NO]directiveName[ =directiveSetting]
```

```
[NO]directiveName[ (directiveSetting)]
```

```
[NO]directiveName[ "directiveSetting"]
```

```
[NO]directiveName[ 'directiveSetting']
```

#### Note

On Windows double quotes on command lines must be escaped by a backslash character.

On a command line a directive must be preceded by one or two hyphens with no whitespace between the hyphen(s) and the directive.

## 4.5 Control statements in source

---

You can control precompilation by placing control statements in source files. The \$SET statement is used to set constants and directives. Other control statements provide support for conditional compilation and message output.

### 4.5.1 Directives

---

You can set directives in source code using a \$SET control statement thus

```
$SET directiveSetting
```

### 4.5.2 Constants

---

You can set a constant in source code using a \$SET CONSTANT control statement.

The following formats are compatible with both standalone and integrated precompilation:

```
$SET CONSTANT constantName [=] constantValue
```

```
$SET CONSTANT constantName [=] "constantValue"
```

CitOESQL also supports the following formats:

```
$SET CONSTANT(constantName [=] constantValue)
```

```
$SET CONSTANT(constantName [=] "constantValue")
```

```
$SET CONSTANT(constantName [=] 'constantValue')
```

To set a constant in a directives file or on the CitOESQL command line use the following formats:

```
[ - ] [ - ] CONSTANT(constantName [=] constantValue)
```

```
[ - ] [ - ] CONSTANT(constantName [=] "constantValue")
```

```
[ - ] [ - ] CONSTANT(constantName [=] 'constantValue')
```

### 4.5.3 Conditional compilation

---

CitOESQL supports conditional compilation using \$IF, \$ELSE and \$END statements in standalone mode. These are compatible with the conditional compilation support offered by cobc.

CitOESQL also support the following variant, which is not supported by cobc:

```
$IF constantName [NOT] DEFINED
```

## 4.5.4 Messages

---

In standalone mode you can output a message at compile time using a \$DISPLAY statement. The message displayed starts with the first non-blank character following \$DISPLAY and ends with the last non-blank character on the same line.

## 4.6 Programming

---

### 4.6.1 Syntax Checking Options

---

CitOESQL is an open implementation for the ANSI Embedded SQL standard, and as such can be used with a wide variety of databases, each of which can accept different and sometimes unique SQL syntax. To accommodate these differences, by default, CitOESQL does minimal SQL syntax checking at compile time. You can increase the level of CitOESQL SQL syntax checking at compile time by using the `SQL(CHECK)` directive with other associated directives.

`SQL(CHECK)` connects to the database during compilation, and asks the database to validate SQL syntax with the existing database SQL objects such as tables, columns, etc.

In addition to `SQL(CHECK)`, you must also specify the `SQL(DB)` directive, and optionally the `SQL(PASS)` directive. This combination ensures a successful connection to your database at compile time and returns applicable SQL syntax errors.

#### Note

When using `SQL(CHECK)`, we suggest that you connect to a local rather than a remote database, as network access could compromise compilation speed.

In addition to the topics below, see the `CHECK` compiler directive topic in the [CitOESQL Reference Manual](#) for more information on the `SQL(CHECK)` compiler directive option.

### 4.6.2 `SQL(CHECK)` and Schema Objects

---

All databases contain Schemas that include SQL objects like tables, columns, views, and temporary tables. If all the SQL objects in a database are available at compile time, using the `SQL(CHECK)` directive ensures that all SQL syntax is fully checked by the database during compilation.

However, some schema objects might not be available in the database at compile time. In this case, CitOESQL offers two solutions:

#### SQL(IGNORESCHEMAERRORS) directive

Use this directive with SQL(CHECK) when tables or other SQL objects do not exist in the database. The addition of SQL(IGNORESCHEMAERRORS) enables CitOESQL to ignore invalid object reference errors returned by the database and continue compilation.

#### [ALSO CHECK] and [ONLY CHECK] statement prefixes

Use these statement prefixes on individual SQL statements in your code if you want CitOESQL to create SQL objects in the database at compile time. With the objects in the database, all invalid object reference errors are returned during compilation, and optionally at run time as well.

For more information on [ALSO CHECK] and [ONLY CHECK], see SQL Statement Prefixes for SQL(CHECK).

### 4.6.3 SQL(CHECK) Command-line Options

---

You can improve SQL syntax checking in some scenarios by combining SQL(CHECK) with additional compiler directive options and/or using a local database.

#### SQL(CHECK) with SQL(DB) and, optionally, SQL(PASS)

When you use the SQL(CHECK) and SQL(DB) directives, together with SQL(PASS) if necessary, CitOESQL opens a connection to a data source at compile time and uses the data source to perform additional checking. This is the recommended way to use CitOESQL and is much more reliable at detecting errors. In addition to detecting syntax errors that are specific to a particular data source, it can also detect misspelled names and invalid use of reserved words.

#### SQL(CHECK), local database, deployment schema

CitOESQL is at its most effective when you use SQL(CHECK) with a local database that uses the same schema that is used when the application is deployed. This combination compiles programs faster than accessing a networked server for compile-time checking.

#### SQL(CHECK), local database, no deployment schema

When you do not have access to a data source with the deployment schema installed you can still use SQL(CHECK) to perform additional syntax checking, but you must also use SQL(IGNORESCHEMAERRORS) to avoid errors for invalid name use.

SQL(IGNORESCHEMAERRORS) is also helpful when your program uses temporary tables that exist only at run time.

### 4.6.4 SQL Statement Prefixes for SQL(CHECK)

---

To enable complete SQL syntax checking at compile time when tables or temporary tables are not in your database, CitOESQL provides SQL statement prefixes that enable you to execute specific SQL statements at compile time and optionally at run time also.

A statement prefix is coded directly into an EXEC SQL statement and executed only when compiled with SQL(CHECK).

**Syntax:**



```
EXEC SQL [statementPrefix] [errorFlag[...]] SQLStatement END-EXEC
```

**Parameters:**

#### statementPrefix

##### [ALSO CHECK]

Statement prefix that instructs SQL(CHECK) to execute the following *SQLStatement* both at compile time and a run time.

##### [ONLY CHECK]

Statement prefix that instructs SQL(CHECK) to execute the following *SQLStatement* at compile time only.

##### [NOCHECK]

Statement prefix that turns off the effects of the SQL(CHECK) directive for the associated *SQLStatement* only.

#### errorFlag

##### [IGNORE ERROR]

Overrides the default behavior of providing a success or error return code upon *SQLStatement* execution, and instead ignores all errors at compile time and proceeds with compilation.

##### [WITH WARNING]

Overrides the default behavior of providing a success or error return code upon *SQLStatement* execution, and instead returns all errors as warnings at compile time, and proceeds with compilation.

#### *SQLStatement*

An SQL statement that conforms to the following:

- A DDL statement such as CREATE TABLE or the specific DML statements INSERT, DELETE or UPDATE
- No host variables used
- Written in a syntax understood by the DBMS vendor

#### Example 1:

Create a SQL Server temporary table at compile time only so that subsequent SQL that references this table is fully syntax checked by the CitoESQL SQL(CHECK) directive during compilation.

```
EXEC SQL [ONLY CHECK]
create table #temp(col1 int,col2 char(32))
END-EXEC
```

#### Example 2:



Create a SQL Server temporary table at both compile time and execution time so that, in addition to full compile-time syntax checking, the table is created at execution time.

```
EXEC SQL [ALSO CHECK]
create table #temp(col1 int,col2 char(32))
END-EXEC
```

**Example 3:**

Create test data at compile time with Oracle whether or not the table exists at compile time.

```
EXEC SQL [ONLY CHECK IGNORE ERROR]
Drop table TT
END-EXEC

EXEC SQL [ONLY CHECK]
create table TT (col1 int)
END-EXEC

EXEC SQL [ONLY CHECK]
Insert into table TT values (1)
END-EXEC
```

## 4.6.5 Tuning Performance

### Cursor Types and Performance

CitOESQL handles ambiguously declared embedded SQL cursors by making them forward and readonly, and incapable of retrieving locks. This change improves performance and efficiency, optimizing CitOESQL as most DBMS SQL cursor access plans do.

You can further optimize CitOESQL by using the BEHAVIOR directive, which enables you to change your embedded SQL cursor characteristics (including prefetch processing) without changing any code in your SQL application sources.

### Statement Cache

Embedded SQL statements are optimized for repeat execution. When first executed a statement is prepared at the data source. This is analogous to program compilation and means that information is retained about how to execute the statement so that it does not have to be recompiled on subsequent executions.

Prepared statements consume memory on both the client and server; CitOESQL maintains a cache to limit how much memory is consumed by prepared statements. The cache is managed on a Least Recently Used (LRU) basis. When the cache reaches its limit the least recently used statement that can safely be removed from the cache is replaced with the statement currently being executed. Any resources used by the replaced prepared statement are freed.

The default cache size is 20 which is quite small, however, this limit has the ability to avoid problems with some database products that have low limits on server resource consumption. The size of the cache can be changed via the STMTCACHE directive.

In practice, a large batch program may benefit from a statement cache size of 300 or possibly more. Many factors affect the optimum cache size, so it is best to experiment. Initial increments will generally improve performance of programs that use connections with long lifespans, but eventually additional increments benefits may reduce performance. Finding the optimum cache size may take a little effort.

---

## Datetime Data Type Handling

By default, CitOESQL supports ODBC/ISO 8601 formats for all input and output character host variables associated with datetime columns in your DBMS.

For example, when using a SQL Server DBMS, the default data type formats for character host variables are:

SQL Server Data Type	ODBC/ISO 8601 Format
date	yyyy-mm-dd
time	hh:mm:ss

SQL Server Data Type	ODBC/ISO 8601 Format
datetime2	yyyy-mm-dd hh:mm:ss.fffffff

In addition to SQL Server, these formats generally apply to other DBMS vendors that accept ISO 8601 formats. CitOESQL also supports alternative formats for both input and output character host variables. We provide several SQL compiler directive options that enable you to specify alternative formats that override the default.

### *Input Host Variables - DETECTDATE*

The DETECTDATE SQL compiler option directive instructs CitOESQL to examine the contents of PIC X character input host variables, looking for data that matches the default ISO 8601 formats. You can override the default formats by specifying one or more additional directives:

#### **Note**

For complete information on each directive, see its corresponding topic under CitOESQL Directives section in the [CitOESQL Reference Manual](#).

#### DATE

Specify an alternative DATE format.

#### DATEDELIM

Specify an alternative delimiter for date columns.

- When used with DATE, the alternative delimiter is applied to the alternative date format specified.
- When used without DATE, the alternative delimiter is applied to the ISO 8601 date format.

#### TIME

Specify an alternative TIME format.

#### TIMDELIM

Specify an alternative delimiter for time columns.

- When used with TIME, the alternative delimiter is applied to the alternative time format specified.
- When used without TIME, the alternative delimiter is applied to the ISO 8601 time format.

#### TSTAMPSEP

Specify a one-character delimiter used between the date and time portions of your input host variables.

The dash character instructs CitOESQL to look for a specific set of delimiters, including a dash, a space, and a T. For example, if you do not specify any alternative date or time formats, and you set TSTAMPSEP to a dash character (-), CitOESQL recognizes the following formats in your input host variables:

- `yyyy-mm-dd-hh.mm.ss.ffffff`
- `yyyy-mm-dd hh.mm.ss.ffffff`
- `yyyy-mm-dd hh:mm:ss.ffffff`
- `yyyy-mm-ddThh.mm.ss.ffffff`
- `yyyy-mm-ddThh:mm:ss.ffffff`

All other characters instruct CitOESQL to search for that specific character between each date and time format, where the date portion is delimited by a dash character (-) and the time portion is delimited by a colon (:).

If you do not specify TSTAMPSEP, CitOESQL defaults to searching for a space character as the delimiter between the date and time formats, where the date portion is delimited by a dash character (-) and the time portion is delimited by a colon (:).

#### Guidelines for using DETECTDATE

Use of the DETECTDATE directive can create significant processing overhead. To minimize this, we recommend that you follow the guidelines presented in the following usage scenarios:

Scenario	DETECTATE option
My application uses date, time and datetimes values in PIC X input host variables, but I am happy with the supported ODBC/ISO 8601 formats and have no use for alternative formats.	Not required. *
My application uses date, time and datetime values in PIC X input host variables, but I only use those values in date, time, or datetime columns in my database.	CLIENT
My application uses ODBC escape sequences for date, time, and datetimes values in PIC X input host variables, but I only use those values in date, time, or datetime columns in my database.	CLIENT

Scenario	DETECTATE option
My application uses date, time, and datetime values in PIC X input host variables, but I only use those values in character columns in my database. Also, my SQL does not use either implicit or explicit characters for date, time, or datetime2 data types.	Not required. Do not use DETECTDATE, DATE, or TIME SQL compiler directive options.
I only use SQLTYPE host variables with date, time and datetime columns, and never use PIC X host variables with date, time or datetime columns.	Not required. **
My application uses date, time and datetime values in PIC X input host variables, and I use those values both in character columns and in date, time and datetime columns in my database, and my character columns might use data in formats that could be confused with the formats for date, time or datetime values.	SERVER

\* We recommend that you use DETECTDATE when also using TIME values with Oracle.

\*\* Optionally, you can use alternative datetime SQL compiler directive options.



### Note

For complete information on all DETECTDATE options, see the DETECTDATE SQL compiler directive option in the [CitOESQL Reference Manual](#).

## Output Host Variables

By default, CitOESQL returns date, time and datetime data types in the ISO 8601 default format. You can override the default format by specifying additional CitOESQL directives as follows:

DBMS Data Type	ODBC/ISO 8601 Format	CitOESQL Directives
date	yyyy-mm-dd	DATE, DATEDELIM

DBMS Data Type	ODBC/ISO 8601 Format	CitOESQL Directives
datetime	yyyy-mm-dd hh:mm:ss.ffffff	TSTAMPSEP

Changing the error code logic in an application containing logic originally designed for a specific database can be cumbersome. Consider these scenarios:

- My code expects Oracle SQLCODE 1403 at end of result set processing and not SQLCODE 100 as produced by other databases.
- My code expects z/OS DB2 SQLCODE-811 when a SELECT INTO statement returns more than one row.
- My code does not expect data truncation warnings after a FETCH statement, but my new database sets SQLCODE 1.
- When inserting a row that results in a duplicate key error, my code expects original database error codes.

These are just a few simple examples, however, error code mapping allows maximum flexibility in preserving the current error handling in your application code. When you provide search criteria based on what the new database returns in error situations, (using value 0 when SQLCODE or SQLSTATE values are returned that do not matter), and specify the error values for the original database, you can ensure that your application receives the error codes it expects.

When error mapping is enabled, it is processed after an embedded SQL statement completes execution. If SQLCODE is non-zero or SQLSTATE is not 00000, the error map is used to determine if SQLCODE, SQLSTATE, and optionally the associated error message, should be replaced with values from the error map. This is done by scanning error map records in order until either of the following conditions are met:

- It reaches the end of the map, in which case SQLCODE, SQLSTATE, and the error message are left unchanged.
- A match is made on some combination of SQLCODE, SQLSTATE, and a substring present in the error message.

### *SQL error mapping files*

You control error mapping using an error mapping file. This is a simple text file that specifies which error conditions to map, the replacement values for SQLCODE and SQLSTATE, and optionally replacement values for the error message, including complete suppression of the error message.

You can specify mappings based on the returned values of SQLCODE, SQLSTATE or a substring within the error message, or any combination of these.

## LOCATION

The default location for mapping files is %COBOLITDIR%\etc (Windows) or \$COBOLITDIR/etc (Linux)

### **Note**

You can override the default location using the CIT\_ERRORMAP\_PATH system environment variable.

## FILENAME

You can name an SQL error mapping file using any prefix you choose; however, all error mapping files must have an `.emap` extension.

## CONTENTS



Each record in a mapping file contains the following values in this order, delimited by commas:

```
{SC-ret-val|0},{SS-ret-val|0},{msg-substr},SC-repl-val,SS-repl-val,[msg-substr-repl-val]
```

**Where:**

*SC-ret-val|0*

The returned database value for SQLCODE, or 0 (zero), to indicate that the returned database SQLCODE value does not matter.

*SS-ret-val|0*

The returned database value for SQLSTATE, or 0 (zero), to indicate that the returned database SQLSTATE value does not matter.

*msg-substr*

The returned database error message substring, if applicable. Specify a string of characters that appear in the message returned by the database. The error is mapped if the substring is present in the error message, and when the SQLCODE and SQLSTATE conditions are also satisfied. The following syntax rules apply when providing a substring:

- If the substring contains a comma, enclose the entire substring in single (') or double (") quotes
- Message substrings are case sensitive.

 **Note**

The full error message is used for the substring search rather than the 70 bytes subset returned in SQLERRMC.

Other than providing a substring, you also have these two options:

- Omit a value by including a space before the next comma delimiter. The original message is returned, effectively switching off error message replacement for substrings.
- Specify a single tilde (~) character. This populates the message-receiving field, consisting of SQLERRMC, MFSQLMESSAGETEXT (or the host variable for

MESSAGE\_TEXT with GET DIAGNOSTICS), with spaces. SQLERRML in the SQLCA is also set to zero rather than the number of characters returned in SQLERRMC.

*SC-repl-val*

Original database replacement value for SQLCODE.

*SS-repl-val*

Original database replacement value for SQLSTATE.

*msg-substr-repl-val*

Replacement value for the error message, if applicable. Syntax rules for msg-substr also apply to msg-substr-repl-val. When omitted, the initial error message is not replaced. Use a single tilde (~) character to completely suppress the message.

### *SQL error mapping record examples*

#### **EXAMPLE 1:**

This example is based on a migration from DB/2 for z/OS to PostgreSQL where a SELECT INTO statement returns more than the expected one row. The following mapping file entry changes the returned SQLCODE from 1 to -811 while leaving the PostgreSQL error intact. It does this when PostgreSQL returns SQLCODE 1 and SQLSTATE 21000 (the matching criteria):

```
1, 21000, , -811, 21000
```

#### **EXAMPLE 2:**

This example is based on a migration from DB/2 for z/OS to PostgreSQL. The record maps errors that contain the string "duplicate" when a primary key or unique constraint error occurs, and changes the error message to "Unique constraint violation". Notice that the return values PostgreSQL provides for both SQLCODE and SQLSTATE do not matter. The search criteria is based solely on the returned PostgreSQL error message alone:

```
0, 00000, "duplicate", -803, 22002, Unique constraint violation
```

**EXAMPLE 3:** ODBC generates a warning when a host variable is smaller than the returned value. If an application tests for SQLCODE being non-zero rather than negative, this can break application logic. To completely suppress the warning condition, including the error message, the following record matches warnings where SQLSTATE has the value 01004:

```
0, 01004, , 0, 00000, ~
```

**EXAMPLE 4:** As an alternative to Example 3, when migrating a legacy application to an environment using UTF-8 and you want to test whether your host variables are large enough, the following record changes this warning to an error with SQLCODE -55 and SQLSTATE "22XYZ":

```
0, 01004, , -55, 22XYZ, Host variable too small
```

### *SQL error mapping enablement*

Use the CitOESQL ERRORMAP compiler directive option to enable error mapping, as documented in the CitOESQL Directive section of the [CitOESQL Reference Manual](#).

If you intend to use multiple error mapping files in one program, use the SQL Statement SET ERRORMAP, as documented in SQL Statements section of the [CitOESQL Reference Manual](#).

# 5. Reference Manual

---

## 5.1 Developing SQL Applications

---

The following topics describe the programming features available for SQL applications in general.

- **Embedded SQL**

Instructions on how to embed SQL statements into your programs.

- **Host Variables**

The purpose of host variables, how to declare them, and how to use them in your SQL applications.

- **Cursors**

The purpose of cursors, how to declare them, and how to use them in SQL applications.

- **Data Structures**

The purpose and use of the SQLCA and SQLDA data structures available for SQL applications.

- **Dynamic SQL**

An explanation of how dynamic SQL works, and a description of its purpose, advantages, and use.

### 5.1.1 Embedded SQL

---

The CItOESQL preprocessor works by taking the SQL statements that you have embedded in your COBOL program and converting them to the appropriate function calls to the database.

**Keywords:**

In your COBOL program, each embedded SQL statement must be preceded by the introductory keywords:

```
EXEC SQL
```

and followed by the keyword:

```
END-EXEC
```

For example:

```
EXEC SQL
  SELECT au_fname INTO :lastname FROM authors
  WHERE au_id = '124-59-3864'
END-EXEC
```

The embedded SQL statement can be broken over as many lines as necessary following the normal COBOL rules for continuation, but between the EXEC SQL and END-EXEC keywords you can only code an embedded SQL statement; you cannot include any ordinary COBOL code.

The case of embedded SQL keywords in your programs is ignored. You can use all upper-case, all lower-case, or a combination of the two. For example, the following are all equivalent:

```
EXEC SQL CONNECT
exec sql connect
Exec Sql Connect
```

### Cursor names, statement names, and connection names:

The case of cursor names, statement names and connection names must match that used when the variable is declared. For example, if you declare a cursor as C1, you must always refer to it as C1 (and not as c1).

The settings for the database determines whether such things as connection names, table and column names, are case-sensitive.

### SQL identifiers:

Hyphens are not permitted in SQL identifiers such as table and column names.

SQL identifiers are typically restricted regarding which characters they support. Typically, unquoted identifiers can only contain A-Z, 0-9 and underscore. Some databases might also allow lower-case characters, and/or @ and # symbols. If your SQL identifiers contain any other characters, such as a grave accent, spaces, or DBCS characters, they must be delimited. Refer to your database vendor documentation for more information, including the character to use as the delimiter.

### SQL statements:

Most vendors provide SQL Reference documentation with their database software that includes full information about embedded SQL statements. Regardless of the database software, you should, for example, be able to perform the following typical operations using the statements shown:

Operation	SQL Statement(s)
Add data to a table	INSERT
Change data in a table	UPDATE
Retrieve a row of data from a table	SELECT
Create a named cursor	DECLARE CURSOR

Operation	SQL Statement(s)
Retrieve multiple rows of data using a cursor	OPEN, FETCH, CLOSE

A full syntax description is given for each of the supported embedded SQL statements, together with an example of its use, in the topics under [Embedded SQL](#).

## 5.1.2 Host Variables

Host variables are data items defined within a COBOL program. They are used to pass values to and receive values from a database. Host variables can be defined in the File Section, Working-Storage Section, Local-Storage Section or Linkage Section of your COBOL program and can be coded using any level number between 1 and 48.

A host variable can be input or output:

Input host variables - to specify data to be transferred from the COBOL program to the database • Output host variables

to hold data to be returned to the COBOL program from the database

To use host variables, you must declare them in your program and then reference them in your SQL statements.

A host variable can be defined as any of the following types:

- [Simple Host Variables](#)

To store and retrieve a single string of data.

- [Host Arrays](#)

To store and retrieve multiple rows of data.

- [Indicator Variables](#)

A companion variable that stores null value and data truncation information.

- [Indicator Arrays](#)

A companion array used to store null value and data truncation information for multiple rows.

### Simple Host Variables

Before you can use a host variable in an embedded SQL statement, you must declare it.

#### DECLARING SIMPLE HOST VARIABLES

Generally, host variable declarations are coded as data items bracketed by the embedded SQL statements BEGIN DECLARE SECTION and END DECLARE SECTION. The following rules also apply:

You can use groups of data items as a single host variable. However, a group item cannot be used in a WHERE clause.

CitOESQL trims trailing spaces from character host variables. If the variable consists entirely of spaces, CitOESQL does not trim the first space character because some servers treat a zero-length string as NULL.

With CitOESQL, you can use COBOL data items as host variables even if they have not been declared using BEGIN DECLARE SECTION and END DECLARE SECTION.

Host variable names must conform to the COBOL rules for data items.

Host variables can be declared anywhere that it is legal to declare COBOL data items.

## REFERENCING SIMPLE HOST VARIABLES

You reference host variables from embedded SQL statements. When you code a host variable name into an embedded SQL statement, it must be preceded by a colon (:) to enable the compiler to distinguish between the host variable and tables or columns with the same name.

### EXAMPLE:

```
EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC
01 id          pic x(4).
01 name        pic x(30).
01 book-title  pic x(40).
01 book-id     pic x(5).
EXEC SQL
  END DECLARE SECTION
END-EXEC
. . .
  display "Type your identification number: "
  accept id.
* The following statement retrieves the name of the
* employee whose ID is the same as the contents of
* the host variable "id". The name is returned in
* the host variable "name".
  EXEC SQL
    SELECT emp_name INTO :name FROM employees
    WHERE emp_id=:id
  END-EXEC
  display "Hello " name.
* In the following statement, :book-id is an input
* host variable that contains the ID of the book to
* search for, while :book-title is an output host
* variable that returns the result of the search.
  EXEC SQL
    SELECT title INTO :book-title FROM titles
    WHERE title_id=:book-id
  END-EXEC
```

## Host Arrays

An array is a collection of data items associated with a single variable name. You can define an array of host variables (called host arrays) and operate on them with a single SQL statement.



You can use host arrays as input variables in INSERT, UPDATE and DELETE statements and as output variables in the INTO clause of SELECT and FETCH statements. This means that you can use arrays with SELECT, FETCH, DELETE, INSERT and UPDATE statements to manipulate large volumes of data.

Some of the benefits to using host arrays include:

You can perform multiple CALL, EXECUTE, INSERT or UPDATE operations by executing only one SQL statement, which can significantly improve performance, especially when the application and the database are on different systems.

You can fetch data in batches, which can be useful when creating a scrolling list of information.

As with simple host variables, you must declare host arrays in your program and then reference them in your SQL statements.

### **DECLARING HOST ARRAYS**

Host arrays are declared in much the same way as simple host variables using BEGIN DECLARE SECTION and END DECLARE SECTION. With host arrays, however, you must use the OCCURS clause to dimension the array.

### **REFERENCING HOST ARRAYS**

The following rules apply to coding host arrays into embedded SQL statements:

Just as with simple host variables, you must precede a host array name with a colon (;).

If the number of rows available is more than the number of rows defined in an array, a SELECT statement returns the number of rows defined in the array, and an SQLCODE message is issued to indicate that the additional rows could not be returned.

Use a SELECT statement only when you know the maximum number of rows to be selected. When the number of rows to be returned is unknown, use the FETCH statement.

If you use multiple host arrays in a single SQL statement, their dimensions must be the same. CitOESQL does not support the mixing of host arrays and simple host variables within a single SQL statement. They must be all simple or all arrays.

For CitOESQL, you must define all host variables within a host array with the same number of occurrences. If one variable has 25 occurrences, all variables in that host array must have 25 occurrences.

- Optionally, use the FOR clause to limit the number of array elements processed to just those that you want. This is especially useful in UPDATE, INSERT and DELETE statements where you may not want to use the entire array. The following rules apply:

If the value of the FOR clause variable is less than or equal to zero, no rows are processed.

The number of array elements processed is determined by comparing the dimension of the host array with the FOR clause variable. The lesser value is used.

## EXAMPLES:

The following example shows typical host array declarations and references.

```
EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC
01 AUTH-REC-TABLES
  05 Auth-id OCCURS 25 TIMES PIC X(12).
  05 Auth-Lname OCCURS 25 TIMES PIC X(40).
EXEC SQL
  END DECLARE SECTION
END-EXEC.
. . .

EXEC SQL
  CONNECT USERID 'user' IDENTIFIED BY 'pwd'
  USING 'db_alias'
END-EXEC
EXEC SQL
  SELECT au-id, au-lname
  INTO :Auth-id, :Auth-Lname FROM authors
END-EXEC
display sqlerrd(3)
```

The following example demonstrates the use of the FOR clause, showing 10 rows (the value of :maxitems) modified by the UPDATE statement:

```

EXEC SQL
  BEGIN DECLARE SECTION
END-EXEC

01 AUTH-REC-TABLES
  05 Auth-id OCCURS 25 TIMES PIC X(12).
  05 Auth-Lname OCCURS 25 TIMES PIC X(40).
01 maxitems PIC S9(4) COMP-5 VALUE 10.
EXEC SQL
  END DECLARE SECTION
END-EXEC.
. . .
EXEC SQL
  CONNECT USERID 'user' IDENTIFIED BY 'pwd'
  USING 'db_alias'
END-EXEC
EXEC SQL
  FOR :maxitems
  UPDATE authors
  SET au_lname = :Auth_Lname
  WHERE au_id = :Auth_id
END-EXEC
display sqlerrd(3)

```

## Indicator Variables

Use indicator variables to:

- Assign null values
- Detect null values
- Detect data truncation

Unlike COBOL, SQL supports variables that can contain null values. A null value means that no entry has been made and usually implies that the value is either unknown or undefined. A null value enables you to distinguish between a deliberate entry of zero (for numerical columns) or a blank (for character columns) and an unknown or inapplicable entry. For example, a null value in a price column does not mean that the item is being given away free, it means that the price is not known or has not been set.

### Important

When a host variable is null, its indicator variable has the value -1; when a host variable is not null, the indicator variable has a value other than -1.

Indicator variables serve an additional purpose if truncation occurs when data is retrieved from a database into a host variable. If the host variable is not large enough to hold the data returned from the database, the warning flag `sqlwarn1` in the `SQLCA` data structure is set and the indicator variable is set to the size of the data contained in the database.

## DECLARING INDICATOR VARIABLES

Indicator variables are always defined as:

```
pic S9(4) comp-5.
```

## REFERENCING INDICATOR VARIABLES

Together, a host variable and its companion indicator variable specify a single SQL value. The following applies to coding a host variable with a companion indicator variable:

Both variables must be preceded by a colon (:).

Place an indicator variable immediately after its corresponding host variable.

- Reference the host variable and indicator variable in a FETCH INTO or SELECT ...INTO statement with or without an INDICATOR clause as follows:

```
:hostvar:indicvar
```

or

```
:hostvar INDICATOR :indicvar
```

You cannot use indicator variables in a search condition. To search for null values, use the is null construct instead.

#### EXAMPLES:

This example demonstrates the declaration of an indicator variable that is used in a FETCH ...INTO statement.

```
EXEC SQL
  BEGIN DECLARE SECTION
  END-EXEC
  01 host-var pic x(4).
  01 indicator-var pic S9(4) comp-5.
EXEC SQL
  END DECLARE SECTION
END-EXEC
. . .

EXEC SQL
  FETCH myCursor INTO :host-var:indicator-var
END-EXEC
```

The following shows an embedded UPDATE statement that uses a saleprice host variable with a companion indicator variable, `saleprice-null`:

```
EXEC SQL
  UPDATE closeoutsale
  SET temp_price = :saleprice:saleprice-null,
  listprice = :oldprice
END-EXEC
```

In this example, if `saleprice-null` has a value of -1, when the UPDATE statement executes, the statement is read as:

```
EXEC SQL
  UPDATE closeoutsale
  SET temp_price = null, listprice = :oldprice
END-EXEC
```

This example demonstrates the use of the `is null` construct to do a search:

```

if saleprice-null equal -1
EXEC SQL
DELETE FROM closeoutsale
WHERE temp_price is null
END-EXEC
else
EXEC SQL
DELETE FROM closeoutsale
WHERE temp_price = :saleprice
END-EXEC
end-if

```

## Indicator Arrays

Just as an indicator variable is used as a companion to a host variable, use an indicator array as a companion to a host array to indicate the null status of each returned row or to store data truncation warning flags.

### EXAMPLES:

In this example, an indicator array is set to -1 so that it can be used to insert null values into a column:

```

01 ix PIC 99 COMP-5.
. . .

EXEC SQL
BEGIN DECLARE SECTION
END-EXEC
01 sales-id OCCURS 25 TIMES PIC X(12).
01 sales-name OCCURS 25 TIMES PIC X(40).
01 sales-comm OCCURS 25 TIMES PIC S9(9) COMP-5.
01 ind-comm OCCURS 25 TIMES PIC S9(4) COMP-5.
EXEC SQL
END DECLARE SECTION
END-EXEC.
. . .
PERFORM VARYING ix FROM 1 BY 1 UNTIL ix > 25
MOVE -1 TO ind-comm (ix)
END-PERFORM.
. . .
EXEC SQL
INSERT INTO SALES (ID, NAME, COMM)
VALUES (:sales_id, :sales_name, :sales_comm:ind-comm)
END-EXEC

```

## COBOL to SQL Data Type Mapping


SQL has a standard set of data types, but the exact implementation of these varies between databases, and many databases do not implement the full set.

Within a program, COBOL host variable declarations can serve both as COBOL host variables and as SQL database variables. To make this possible, the preprocessor converts COBOL data types to their equivalent SQL data types. We sometimes refer to this conversion process as mapping COBOL data types to SQL data types. The preprocessor looks for specific COBOL picture clause formats that identify those that require mapping to SQL data types. For mapping to be successful, you must declare your COBOL host variables using these specific COBOL picture clauses.

We provide SQL data types for the CitOESQL preprocessor. For complete information on each SQL data type and its required COBOL host variable formats, see the SQL Data Types and ODBC SQL/COBOL Data Type Mappings Reference topics.

## SQL TYPES

Manipulating SQL data that involves date, time, or binary data can be complicated using traditional COBOL host variables, and traditional techniques for handling variable-length character data can also be problematic. To simplify working with this data, we provide the SQL TYPE declaration to make it easier to specify host variables that more closely reflect the natural data types of relational data stores. This allows more applications to be built using static rather than dynamic SQL syntax and can also help to optimize code execution.

 **Note**

For a complete listing of available SQL TYPEs, see the SQL TYPEs reference topic.

**EXAMPLE:**

Defining date, time, and timestamp fields as SQL TYPEs.

This example program shows date, time and timestamp escape sequences being used, and how to redefine them as SQL TYPEs. It applies to CitoESQL:

```

working-storage section.

EXEC SQL INCLUDE SQLCA END-EXEC
01 date-field1      pic x(29).
01 date-field2      pic x(29).
01 date-field3      pic x(29).

procedure division.
EXEC SQL
    CONNECT TO 'Net Express 4.0 Sample 1' USER 'admin'
END-EXEC
* If the Table is there drop it.
EXEC SQL
    DROP TABLE DT
END-EXEC

* Create a table with columns for DATE, TIME, and DATE/TIME
* NOTE: Access uses DATETIME column for all three.
*       Some databases will have dedicated column types.
* If you are creating DATE/TIME columns on another data
* source, refer to your database documentation to see how to * define the columns.

EXEC SQL
    CREATE TABLE DT ( id INT,
                      myDate DATE NULL,
                      myTime TIME NULL,
                      myTimestamp TIMESTAMP NULL)
END-EXEC

* INSERT into the table using the ODBC Escape sequences

EXEC SQL
    INSERT into DT values (1 ,
                          {d '1961-10-08'}, * > Set just the date part
                          {t '12:21:54'}, * > Set just the time part
                          {ts '1966-01-24 08:21:56'} * > Set both parts
                          )
END-EXEC

* Retrieve the values we just inserted

EXEC SQL
    SELECT      myDate
              ,myTime
              ,myTimestamp
    INTO        :date-field1
              ,:date-field2
              ,:date-field3
    FROM DT
    where id = 1
END-EXEC

* Display the results.

display 'where the date part has been set :'
       date-field1
display 'where the time part has been set :'
       date-field2
display 'NOTE, most data sources will set a default '
       'for the date part '
display 'where both parts has been set :'
       date-field3

* Remove the table.

EXEC SQL
    DROP TABLE DT
END-EXEC

* Disconnect from the data source

EXEC SQL
    DISCONNECT CURRENT
END-EXEC

stop run.

```

Alternatively, you can use host variables defined with SQL TYPEs for date/time variables. Define the following host variables:

```

01 my-id          pic s9(08) COMP-5.
01 my-date        sql type is date.
01 my-time        sql type is time.
01 my-timestamp  sql type is timestamp.

```

and replace the INSERT statement with the following code:

```

*> INSERT into the table using SQL TYPE HOST VARS
move 1 to MY-ID
move "1961-10-08" to MY-DATE
move "12:21:54" to MY-TIME
move "1966-01-24 08:21:56" to MY-TIMESTAMP

EXEC SQL
  INSERT into DT value (
    :MY-ID
    ,:MY-DATE
    ,:MY-TIME
    ,:MY-TIMESTAMP )
END-EXEC

```

## 5.1.3 Cursors

When you write code in which the results set returned by a SELECT statement includes more than one row of data, you must declare and use a cursor. A cursor indicates the current position in a results set, in the same way that the cursor on a screen indicates the current position.

A cursor enables you to:

- Fetch rows of data one at a time

- Perform updates and deletions at a specified position within a results set.

The example below demonstrates the following sequence of events:

1. The DECLARE CURSOR statement associates the SELECT statement with the cursor Cursor1.
2. The OPEN statement opens the cursor, thereby executing the SELECT statement.
3. The FETCH statement retrieves the data for the current row from the columns au\_fname and au\_lname and places the data in the host variables first\_name and last\_name.
4. The program loops on the FETCH statement until no more data is available.
5. The CLOSE statement closes the cursor.

```

EXEC SQL DECLARE Cursor1 CURSOR FOR
  SELECT au_fname, au_lname FROM authors
END-EXEC
. . .

EXEC SQL
  OPEN Cursor1
END-EXEC
. . .

perform until sqlcode not = zero
  EXEC SQL
    FETCH Cursor1 INTO :first_name, :last_name
  END-EXEC
  display first_name, last_name
end-perform
. . .

EXEC SQL
  CLOSE Cursor1
END-EXEC

```

### Declaring a Cursor



Before a cursor can be used, it must be declared. This is done using the DECLARE CURSOR statement in which you specify a name for the cursor and either a SELECT statement or the name of a prepared SQL statement.

Cursor names must conform to the rules for identifiers on the database that you are connecting to, for example, some databases do not allow hyphens in cursor names.

```
EXEC SQL
  DECLARE Cur1 CURSOR FOR
  SELECT first_name FROM employee
  WHERE last_name = :last-name
END-EXEC
```

This example specifies a SELECT statement using an input host variable (:last-name). When the cursor OPEN statement is executed, the values of the input host variable are read and the SELECT statement is executed.

```
EXEC SQL
  DECLARE Cur2 CURSOR FOR stmt1
END-EXEC
. . .
move "SELECT first_name FROM emp " &
  "WHERE last_name=?" to prep.
EXEC SQL
  PREPARE stmt1 FROM :prep
END-EXEC
. . .
EXEC SQL
  OPEN Cur2 USING :last-name
END-EXEC
```

In this example, the DECLARE CURSOR statement references a prepared statement (stmt1). A prepared SELECT statement can contain question marks (?) which act as parameter markers to indicate that data is to be supplied when the cursor is opened. The cursor must be declared before the statement is prepared.

## Opening a Cursor

Once a cursor has been declared, it must be opened before it can be used. This is done using the OPEN statement, for example:

```
EXEC SQL
  OPEN Cur1
END-EXEC
```

If the DECLARE CURSOR statement references a prepared statement that contains parameter markers, the corresponding OPEN statement must specify the host variables or the name of an SQLDA structure that will supply the values for the parameter markers, for example:

```
EXEC SQL
  OPEN Cur2 USING :last-name
END-EXEC
```

If an SQLDA data structure is used, the data type, length, and address fields must already contain valid data when the OPEN statement is executed.

## Using a Cursor to Retrieve Data

Once a cursor has been opened, it can be used to retrieve data from the database. This is done using the FETCH statement. The FETCH statement retrieves the next row from the results set produced by the OPEN statement and writes the data returned to the specified host variables (or to addresses specified in an SQLDA structure). For example:

```
perform until sqlcode not = 0
  EXEC SQL
    FETCH Cur1 INTO :first_name
  END-EXEC
  display 'First name: ' fname
  display 'Last name : ' lname
  display spaces
end-perform
```

When the cursor reaches the end of the results set, a value of 100 is returned in SQLCODE in the SQLCA data structure and SQLSTATE is set to "02000".

As data is fetched from a cursor, locks can be placed on the tables from which the data is being selected.

## Closing a Cursor

When your application has finished using the cursor, it should be closed using the CLOSE statement. For example:

```
EXEC SQL
  CLOSE Cur1
END-EXEC
```

Normally, when a cursor is closed, all locks on data and tables are released. If the cursor is closed within a transaction, however, the locks may not be released.

## Positioned UPDATE and DELETE Statements

Positioned UPDATE and DELETE statements are used in conjunction with cursors and include WHERE CURRENT OF clauses instead of search condition clauses. The WHERE CURRENT OF clause specifies the corresponding cursor.

```
EXEC SQL
  UPDATE emp SET last_name = :last-name
  WHERE CURRENT OF Cur1
END-EXEC
```

This will update last\_name in the row that was last fetched from the database using cursor Cur1 .

```
EXEC SQL
  DELETE emp WHERE CURRENT OF Cur1
END-EXEC
```

This example will delete the row that was last fetched from the database using cursor Cur1 .

### CitOESQL:

With some ODBC drivers, cursors that will be used for positioned updates and deletes must include a FOR UPDATE clause. Note that positioned UPDATE and DELETE are part of the Extended ODBC Syntax and are not supported by all drivers.

## Using Cursors

Cursors are very useful for handling large amounts of data; however, there are a number of issues that you should bear in mind when using cursors, namely: data concurrency, integrity, and consistency.

To ensure the integrity of your data, a database server can implement different locking methods. Some types of data access do not acquire any locks, some acquire a shared lock and some an exclusive lock. A shared lock allows other processes to access the data but not update it. An exclusive lock does not allow any other process to access the data.

When using cursors there are three levels of isolation and these control the data that a cursor can read and lock:

- **Level zero**

Level zero can only be used by read-only cursors. At level zero, the cursor will not lock any rows but may be able to read data that has not yet been committed. Reading uncommitted data is dangerous (as a rollback operation will reset the data to its previous state) and is normally called a "dirty read". Not all databases will allow dirty reads.

- **Level one**

Level one can be used by read-only cursors or updateable cursors. With level one, shared locks are placed on the data unless the FOR UPDATE clause is used. If the FOR UPDATE clause is used, exclusive locks are placed on the data. When the cursor is closed, the locks are released. A standard cursor, that is a cursor without the FOR UPDATE clause, will normally be at isolation level one and use shared locks.

- **Level three**

Level three cursors are used with transactions. Instead of the locks being released when the cursor is closed, the locks are released when the transaction ends. With level three it is usual to place exclusive locks on the data.

It is worth pointing out that there can be problems with deadlocks or "deadly embraces" where two processes are competing for the same data. The classic example is where one process locks data A and then requests a lock on data B while a second process locks data B and then requests a lock on data A. Both processes have data that the other process requires. The database server should spot this case and send errors to one or both processes.

### 5.1.4 Data Structures

---

The CitOESQL preprocessor supplied with this system use two data structures:

<b>Data Structure</b>	<b>Description</b>	<b>Function</b>
SQLCA	SQL Communications Area	Returns status and error information.

Data Structure	Description	Function
SQLDA	SQL Descriptor Area	Describes the variables used in dynamic SQL statements.

## SQL Communications Area (SQLCA)

After each embedded SQL statement is executed, error and status information are returned in the SQL Communications Area (SQLCA).

### CitOESQL:

The SQLCA provided with COBOL-IT for use with CitOESQL contains two variables (SQLCODE and SQLSTATE), plus a number of warning flags which are used to indicate whether an error has occurred in the most recently executed SQL statement.

## USING THE SQLCA

The SQLCA structure is supplied in the file `sqlca.cpy`, which by default is located in the default location specified in the **COBOL-IT CitOESQL files and locations** section in the [CitOESQL Getting Started Guide](#). To include it in your program, use the following statement in the data division:

```
EXEC SQL INCLUDE SQLCA END-EXEC
```

If you do not include this statement, the COBOL Compiler automatically allocates an area, but it is not addressable from within your program. However, if you declare either of the data items `SQLCODE` or `SQLSTATE` separately, the COBOL Compiler generates code to copy the corresponding fields in the SQLCA to the user-defined fields after each `EXEC SQL` statement.

If you declare the data item `MFSQLMESSAGETEXT`, it is updated with a description of the exception condition whenever `SQLCODE` is non-zero. `MFSQLMESSAGETEXT` must be declared as a character data item, `PIC X(n)`, where `n` can be any legal value. This is particularly useful as ODBC error messages often exceed the 70-byte SQLCA message field.

### Note

You do not need to declare `SQLCA`, `SQLCODE`, `SQLSTATE` or `MFSQLMESSAGETEXT` as host variables.

## THE SQLCODE VARIABLE

Testing the value of `SQLCODE` is the most common way of determining the success or failure of an embedded SQL statement.

For details of `SQLCODE` values, see the relevant database vendor documentation.

## THE SQLSTATE VARIABLE

The SQLSTATE variable was introduced in the SQL-92 standard and is the recommended mechanism for future applications. It is divided into two components:

The first two characters are called the class code. Any class code that begins with the letters A through H or the digits 0 through 4 indicates a SQLSTATE value that is defined by the SQL standard or another standard.

The last three characters are called the subclass code.

A value of "00000" indicates that the previous embedded SQL statement executed successfully.

For specific details of the values returned in SQLSTATE, see the relevant database vendor documentation.

## SQLWARN FLAGS

Some statements may cause warnings to be generated. To determine the type of warning, your application should examine the contents of the SQLWARN flags.

**W** - The flag has generated a warning.

blank (space) - The flag has not generated a warning.

The value of a flag is set to **W** if that particular warning occurred, otherwise the value is a blank (space).

Each SQLWARN flag has a specific meaning. For more information on the meaning of the SQLWARN flags, see the relevant database vendor documentation.

## THE WHENEVER STATEMENT

Explicitly checking the value of SQLCODE or SQLSTATE after each embedded SQL statement can involve writing a lot of code. As an alternative, check the status of the SQL statement by using a WHENEVER statement in your application.

The WHENEVER statement is not an executable statement. It is a directive to the Compiler to automatically generate code that handles errors after each executable embedded SQL statement.

The WHENEVER statement allows one of three default actions (CONTINUE, GOTO or PERFORM) to be registered for each of the following conditions:

Condition	Value of SQLCODE
NOT FOUND	100
SQLWARNING	+1

Condition	Value of SQLCODE
SQLERROR	\< 0 (negative)

A WHENEVER statement for a particular condition replaces all previous WHENEVER statements for that condition.

The scope of a WHENEVER statement is related to its physical position in the source program, not its logical position in the run sequence. For example, in the following code if the first SELECT statement does not return anything, paragraph A is performed, not paragraph C:

```
EXEC SQL
  WHENEVER NOT FOUND PERFORM A
END-EXEC.
perform B.
EXEC SQL
  SELECT col1 into :host-var1 FROM table1
  WHERE col2 = :host-var2
END-EXEC.
A.
  display "First item not found".
B.
  EXEC SQL
    WHENEVER NOT FOUND PERFORM C.
  END-EXEC.
C.
  display "Second item not found".
```

## SQLERRM

The SQLERRM data area is used to pass error messages to the application from the database server. The SQLERRM data area is split into two parts:

SQLERRML - holds the length of the error message

SQLERRMC - holds the error text.

Within an error routine, the following code can be used to display the SQL error message:

```
if (SQLERRML \> ZERO) and (SQLERRML \< 80)
  display 'Error Message: ', SQLERRMC(1:SQLERRML)
  else
  display 'Error Message: ', SQLERRMC
end-if.
```

## SQLERRD

The SQLERRD data area is an array of six integer status values, set by the database vendor after an SQL error.

```
SQLERRD PIC X9(9) COMP-5 OCCURS 6 VALUE 0.
```

Please consult the relevant database vendor documentation for more detailed information on these values.

## The SQL Descriptor Area (SQLDA)

When either the number of parameters to be passed, or their data types, are unknown at compilation time, you can use an SQL Descriptor Area (SQLDA) instead of host variables.

An SQLDA contains descriptive information about each input parameter or output column. It contains the column name, data type, length, and a pointer to the actual data buffer for each input or output parameter. An SQLDA is ordinarily used with parameter markers to specify input values for prepared SQL statements, but you can also use an SQLDA with the DESCRIBE statement (or the INTO option of a PREPARE statement) to receive data from a prepared SELECT statement.

Although you cannot use an SQLDA with static SQL statements, you can use a SQLDA with a cursor FETCH statement.

## CITOTESQL

The SQLDA structure is supplied in both the `sqllda.cpy` (SQLDA only) and `sqllda78.cpy` (SQLDA plus SQLTYPE definitions) files, which are in the default location specified **COBOL-IT CitOTESQL files and locations** section in the [CitOTESQL Getting Started Guide](#).

You can include the SQLDA in your COBOL program by adding one or both of the following statements to your data division:

```
EXEC SQL
  INCLUDE SQLDA
END-EXEC

EXEC SQL
  INCLUDE SQLDA78
END-EXEC
```

## USING THE SQLDA

Before an SQLDA structure is used, your application must initialise the following fields:

SQLN: This must be set to the maximum number of SQLVAR entries that the structure can hold.

### The PREPARE and DESCRIBE Statements

You can use the DESCRIBE statement (or the PREPARE statement with the INTO option) to enter the column name, data type, and other data into the appropriate fields of the SQLDA structure.

Before the statement is executed, the SQLN and SQLDABC fields should be initialised as described above.

After the statement has been executed, the SQLD field will contain the number of parameters in the prepared statement. A SQLVAR record is set up for each of the parameters with the SQLTYPE and SQLLEN fields completed.

If you do not know how big the value of SQLN should be, you can issue a DESCRIBE statement with SQLN set to 1 and SQLD set to 0. No column detail information is moved into the SQLDA structure, but the number of columns in the results set is inserted into SQLD.

### The FETCH Statement



Before performing a FETCH statement using an SQLDA structure, follow the procedure below:

1. The application must initialize SQLN and SQLDABC as described above.
2. The application must then insert, into the SQLDATA field, the address of each program variable that will receive the data from the corresponding column. (The SQLDATA field is part of SQLVAR).
3. If indicator variables are used, SQLIND must also be set to the corresponding address of the indicator variable.

The data type field (SQLTYPE) and length (SQLLEN) are filled with information from a PREPARE INTO or a DESCRIBE statement. These values can be overwritten by the application prior to a FETCH statement.

### The OPEN or EXECUTE Statements

To use an SQLDA structure to specify input data to an OPEN or EXECUTE statement, your application must supply the data for the fields of the entire SQLDA structure, including the SQLN, SQLD, SQLDABC, and SQLTYPE, SQLLEN, and SQLDATA fields for each variable. The following scenarios require additional attention:

- **SQLTYPE field is an odd number**

If the value of the SQLTYPE field is an odd number, you must also supply the address of the indicator variable using SQLIND.

- **Host variable input is COMP**

When using CITOESQL with a host variable input defined as COMP, add **8192** (x2000) to the SQLTYPE field.

- **SQLTYPE field is an odd number and indicator variable is COMP**

If the SQLTYPE field is an odd number, and the indicator variable is defined as a COMP, add **4096** (x1000) to the SQLTYPE field.

- **Host variable input is COMP-5**

When using CITOESQL with a host variable input defined as COMP-5, no change to the SQLTYPE field is required.

### The DESCRIBE Statement

After a PREPARE statement, you can execute a DESCRIBE statement to retrieve information about the data type, length and column name of each column returned by the specified prepared statement. This information is returned in the SQL Descriptor Area (SQLDA):

```
EXEC SQL
  DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

If you want to execute a DESCRIBE statement immediately after a PREPARE statement, you can use the INTO option on the PREPARE statement to perform both steps at once:

```
EXEC SQL
  PREPARE stmt1 INTO :sqllda FROM :stmtbuf
END-EXEC
```

The following cases could require that you make manual changes to the SQLTYPE or SQLLEN fields in the SQLDA to accommodate differences in host variable types and lengths after executing DESCRIBE:

- **SQLTYPE: Variable-length character types**

For variable-length character types you can choose to define SQLTYPE as a fixed-size COBOL host variable such as PIC X, N, or G, or a variable-length host variable such as a record with level 49 sub-fields for both length and the actual value. The SQLLEN field could be either 16 or 32 bits depending on the SQLTYPE value.

- **SQLTYPE: Numeric types**

For numeric types you can choose to define SQLTYPE as COMP-3, COMP, COMP-5, or to display numeric COBOL host variables with an included or separate, and leading or trailing sign. The value returned by DESCRIBE depends on the data source. Generally, this is COMP-3 for NUMERIC or DECIMAL columns, and COMP-5 for columns of the tinyint, smallint, integer, or bigint integer types.

- **SQLLEN**

DESCRIBE sets SQLLEN to the size of integer columns in COMP and COMP-5 representations, meaning a value of 1, 2, 4, or 18. You might need to adjust this depending on SQLTYPE. For NUMERIC and DECIMAL columns, it encodes the precision and scale of the result.

## 5.1.5 Dynamic SQL

If everything is known about an SQL statement when the application is compiled, the statement is known as a static SQL statement.

In some cases, however, the full text of an SQL statement may not be known when an application is written. For example, you may need to allow the end-user of the application to enter an SQL statement. In this case, the statement needs to be constructed at run-time. This is called a dynamic SQL statement.

### Dynamic SQL Statement Types

There are four types of dynamic SQL statement:

<b>Dynamic SQL Statement Type</b>	<b>Perform Queries?</b>	<b>Return Data?</b>
Execute a statement once	No	No, can only return success or failure
Execute a statement more than once	No	No, can only return success or failure
Select a given list of data with a given set of selection criteria	Yes	Yes

Dynamic SQL Statement Type	Perform Queries?	Return Data?
Select any amount of data with any selection criteria	Yes	Yes

These types of dynamic SQL statement are described more fully in the following sections.

### EXECUTE A STATEMENT ONCE

With this type of dynamic SQL statement, the statement is executed immediately. Each time the statement is executed, it is re-parsed.

### EXECUTE A STATEMENT MORE THAN ONCE

This type of dynamic SQL statement is either a statement that can be executed more than once or a statement that requires host variables. For the second type, the statement must be prepared before it can be executed.

### SELECT A GIVEN LIST OF DATA

This type of dynamic SQL statement is a SELECT statement where the number and type of host variables is known. The normal sequence of SQL statements is:

- Prepare the statement
- Declare a cursor to hold the results
- Open the cursor
- Fetch the variables
- Close the cursor.

### SELECT ANY AMOUNT OF DATA

This type of dynamic SQL statement is the most difficult type to code. The type and/or number of variables is only resolved at run time. The normal sequence of SQL statements is:

- Prepare the statement
- Declare a cursor for the statement
- Describe the variables to be used
- Open the cursor using the variables just described
- Describe the variables to be fetched
- Fetch the variables using their descriptions
- Close the cursor.

If either the input host variables, or the output host variables are known (at compile time), then the OPEN or FETCH can name the host variables and they do not need to be described.

## Preparing Dynamic SQL Statements

The PREPARE statement takes a character string containing a dynamic SQL statement and associates a name with the statement, for example:

```
move "INSERT INTO publishers " &
      "VALUES (?, ?, ?, ?)" to stmtbuf
EXEC SQL
  PREPARE stmt1 FROM :stmtbuf
END-EXEC
```

Dynamic SQL statements can contain parameter markers - question marks (?) that act as a place holder for a value. In the example above, the values to be substituted for the question marks must be supplied when the statement is executed.

Once you have prepared a statement, you can use it in one of two ways:

- You can execute a prepared statement.

- You can open a cursor that references a prepared statement.

## Executing Dynamic SQL Statements

The EXECUTE statement runs a specified prepared SQL statement.

### Note

Only statements that do not return results can be executed in this way.

If the prepared statement contains parameter markers, the EXECUTE statement must include either the "using :hvar" option to supply parameter values using host variables or the "using descriptor :sqlda\_struct" option identifying an SQLDA data structure already populated by the application. The number of parameter markers in the prepared statement must match the number of SQLDATA entries ("using descriptor :sqlda") or host variables ("using :hvar").

```
move "INSERT INTO publishers " &
      "VALUES (?, ?, ?, ?)" to stmtbuf
EXEC SQL
  PREPARE stmt1 FROM :stmtbuf
END-EXEC
...
EXEC SQL
  EXECUTE stmt1 USING :pubid, :pubname, :city, :state
END-EXEC.
```

In this example, the four parameter markers are replaced by the contents of the host variables supplied via the USING clause in the EXECUTE statement.

## EXECUTE IMMEDIATE STATEMENT

If the dynamic SQL statement does not contain any parameter markers, you can use EXECUTE IMMEDIATE instead of PREPARE followed by EXECUTE, for example:

```
move "DELETE FROM emp " &
      "WHERE last_name = 'Smith'" to stmtbuf
EXEC SQL
      EXECUTE IMMEDIATE :stmtbuf
END-EXEC
```

When using EXECUTE IMMEDIATE, the statement is re-parsed each time it is executed. If a statement is likely to be used many times it is better to PREPARE the statement and then EXECUTE it when required.

## Dynamic SQL Statements and Cursors

If a dynamic SQL statement returns a result, you cannot use the EXECUTE statement. Instead, you must declare and use a cursor.

First, declare the cursor using the DECLARE CURSOR statement:

```
EXEC SQL
      DECLARE C1 CURSOR FOR dynamic_sql
END-EXEC
```

In the example above, `dynamic_sql` is the name of a dynamic SQL statement. You must use the PREPARE statement to prepare the dynamic SQL statement before the cursor can be opened, for example:

```
move "SELECT char_col FROM mfesqltest " &
      "WHERE int_col = ?" to sql-text
EXEC SQL
      PREPARE dynamic_sql FROM :sql-text
END-EXEC
```

Now, when the OPEN statement is used to open the cursor, the prepared statement is executed:

```
EXEC SQL
      OPEN C1 USING :int-col
END-EXEC
```

If the prepared statement uses parameter markers, then the OPEN statement must supply values for those parameters by specifying either host variables or an SQLDA structure.

Once the cursor has been opened, the FETCH statement can be used to retrieve data, for example:

```
EXEC SQL
      FETCH C1 INTO :char-col
END-EXEC
```

Finally, the cursor is closed using the CLOSE statement:

```
EXEC SQL
      CLOSE C1
END-EXEC
```

## CALL STATEMENTS

A CALL statement can be prepared and executed as dynamic SQL.

You can use parameter markers (?) in dynamic SQL wherever you use host variables in static SQL

Use of the IN, INPUT, OUT, OUTPUT, INOUT and CURSOR keyword following parameter markers is the same as their use after host variable parameters in static SQL.

- The whole call statement must be enclosed in braces to conform to ODBC canonical stored procedure syntax (the CitOESQL precompiler does this for you in static SQL). For example:

```
move '{call myproc(?, ? out)}' to sql-text
exec sql
    prepare mycall from :sql-text
end-exec
exec sql
    execute mycall using :parm1, :parm2
end-exec
```

- If you use parameter arrays, you can limit the number of elements used with a FOR clause on the EXECUTE, for example:

```
move 5 to param-count
exec sql
    for :param-count
        execute mycall using :parm1, :param2
end-exec
```

#### EXAMPLE:

The following is an example of a program that creates a stored procedure "mfexecsptest" using data source "SQLServer 2000" and then retrieves data from "publishers" table using a cursor "c1" with dynamic SQL.

```

\$$SET SQL
    WORKING-STORAGE SECTION.

EXEC SQL INCLUDE SQLCA END-EXEC

\*\> after an sql error this has the full message text
01 MFSQLEMSGTEXT PIC X(250).
01 IDX          PIC X(04) COMP-5.

EXEC SQL BEGIN DECLARE SECTION END-EXEC
\*\> Put your host variables here if you need to port
| \*\> to other COBOL compilers

01 stateParam    pic xx.
01 pubid         pic x(4).
01 pubname       pic x(40).
01 pubcity       pic x(20).

01 sql-stat      pic x(256).

EXEC SQL END DECLARE SECTION END-EXEC

PROCEDURE DIVISION.

EXEC SQL
    WHENEVER SQLERROR perform OpenESQL-Error
END-EXEC

EXEC SQL
    CONNECT TO 'SQLServer 2000' USER 'SA'
END-EXEC

\*\> Put your program logic/SQL statements here

EXEC SQL
    create procedure mfexecstest
        (@stateParam char(2) = 'NY' ) as

        select pub_id, pub_name, city from publishers
        where state = @stateParam
END-EXEC

exec sql
    declare c1 scroll cursor for dsq12 for read only
end-exec

move "{call mfexecstest(?)}" to sql-stat
exec sql prepare dsq12 from :sql-stat end-exec

move "CA" to stateParam
exec sql
    open c1 using :stateParam
end-exec

display "Testing cursor with stored procedure"
perform until exit
    exec sql
        fetch c1 into :pubid, :pubname, :pubcity
    end-exec

    if sqlcode = 100
        exec sql close c1 end-exec
        exit perform
    else
        display pubid " " pubname " " pubcity
    end-if
end-perform

EXEC SQL close c1 END-EXEC

EXEC SQL DISCONNECT CURRENT END-EXEC
EXIT PROGRAM.
STOP RUN.
*> Default sql error routine / modify to stop program if
*> needed
OpenESQL-Error Section.

    display "SQL Error = " sqlstate " " sqlcode
    display MFSQLEMSGTEXT
    *> stop run
exit.

```



## 5.2 SQL Statements

---

With the exception of INSERT, DELETE(SEARCHED) and UPDATE(SEARCHED), which are included for your convenience, the embedded SQL statements described here work somewhat differently, or are in addition to, standard SQL statements.

SQL Statement	Description
BEGIN DECLARE SECTION	Signals the beginning of the DECLARE section.
BEGIN TRAN	Provides compatibility with Embedded SQL implementations that do not conform to the ANSI SQL standard with respect to transaction management and, in particular, the Micro Focus Embedded SQL Toolkit for Microsoft SQL Server.
CALL	Executes a stored procedure.
CLOSE	Discards unprocessed rows and frees any locks held by the cursor.
COMMIT	Makes any changes made by the current transaction on the current connection permanent in the database.
CONNECT	Attaches to a specific database using the supplied username and password.
DECLARE CURSOR	Associates the cursor name with the specified SELECT statement and enables you to retrieve rows of data using the FETCH statement.
DECLARE DATABASE	Declares the name of a database.
DELETE (Positioned)	Deletes the row most recently fetched by using a cursor.
DELETE (Searched)	Removes table rows that meet the search criteria.
DESCRIBE	Provides information on prepared dynamic SQL statements and describes the result set for an open cursor.
DISCONNECT	Closes the connection(s) to a database. In addition, all cursors opened for that connection are automatically closed.
END DECLARE SECTION	Terminates a host variable declaration section begun by a BEGIN DECLARE SECTION statement.
EXECSP	Executes a stored procedure.
EXECUTE	Processes dynamic SQL statements.

<b>SQL Statement</b>	<b>Description</b>
EXECUTE IMMEDIATE	Immediately executes the SQL statement.
FETCH	Retrieves a row from the cursor's results set and writes the values of the columns in that row to the corresponding host variables (or to addresses specified in the SQLDA data structure).
GET HDBC	Enables you to use ODBC calls that require you to supply the ODBC connection handle.
GET HENV	Enables you to use ODBC calls that require you to supply the ODBC environment handle.
GET NEXT RESULT SET	Makes the next result set available to an open cursor.
INCLUDE	Includes the definition of the specified SQL data structure or source file in the COBOL program.
INSERT	Adds new rows to a table.
INTO	Retrieves one row of results and assigns the values of the items returned by an OUTPUT clause in a SQL Server INSERT, UPDATE, or DELETE statement to the host variables specified in the INTO list.
OPEN	Runs the SELECT statement specified in the corresponding DECLARE CURSOR statement to produce the results set that is accessed one row at a time by the FETCH statement.
PREPARE	Processes dynamic SQL statements.
QUERY ODBC	Delivers a results set in the same way as a SELECT statement, and must therefore be associated with a cursor via DECLARE and OPEN, or DECLARE, PREPARE and OPEN.
RESET CONNECTION	Closes all open cursors, even if the application has not appropriately closed them.
ROLLBACK	Backs out any changes made to the database by the current transaction on the current connection, or partially rolls back changes to a previously set save point.
SAVEPOINT SAVE TRANSACTION RELEASE [TO] SAVEPOINT	Sets a transaction save point to which a current transaction can be rolled back, resulting in a partial roll back.

<b>SQL Statement</b>	<b>Description</b>
SELECT DISTINCT (using DECLARE CURSOR)	Associates the cursor name with the SELECT DISTINCT statement and enables you to retrieve rows of data using the FETCH statement.
SELECT INTO	Retrieves one row of results and assigns the values of the items in a specified SELECT list to the host variables specified in the INTO list.
SET AUTOCOMMIT	Enables you to control ODBC AUTOCOMMIT mode at run time.
SET CONNECTION	Sets the named connection as the current connection.
SET ERRORMAP	Changes the SQL error map file for the current connection.
SET host_variable	Provides information about CitOESQL connections and databases.
SET OPTION	Enables you to set CitOESQL options.
SET TRACELEVEL	Enables you to dynamically set or change the reporting level of CitOESQL traces for native applications.
SET TRANSACTION ISOLATION	Sets the transaction isolation level for the current connection to one of the isolation level modes specified by ODBC.
SYNCPOINT	Closes all open cursors that were not opened using the WITH HOLD clause, even if the application has not appropriately closed them.
UPDATE (Positioned)	Updates the rows most recently fetched by using a cursor.
UPDATE (Searched)	Updates a table or view based on specified search conditions.

SQL Statement	Description
WHENEVER	Specifies the default action after running an Embedded SQL statement when a specific condition is met.

## 5.2.1 BEGIN DECLARE SECTION

Signals the beginning of the DECLARE section.

### Syntax:

```
\>\>---EXEC SQL---BEGIN DECLARE SECTION---END-EXEC---\>\<
```

### Comments:

The BEGIN DECLARE SECTION statement can be included anywhere where COBOL permits variable declaration. Use END DECLARE SECTION to identify the end of a COBOL declaration section.

Declare sections cannot be nested.

Variables must be declared in COBOL, not in SQL.

To avoid conflict, variables inside a declaration section cannot be the same as any outside the declaration section or in any other declaration section, even in other compilation units.

If data structures are defined within a declaration section, only the bottom-level items (with PIC clauses) can be used as host variables. Two exceptions are arrays specified in FETCH statements and record structures specified in SELECT INTO statements.

### Example:

```
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION END-EXEC
01 staff-id pic x(4).
01 last-name pic x(30).
EXEC SQL END DECLARE SECTION END-EXEC
```

## 5.2.2 BEGIN TRAN

Provides compatibility with Embedded SQL implementations that do not conform to the ANSI SQL standard with respect to transaction management and the Micro Focus Embedded SQL Toolkit for Microsoft SQL Server.

### Syntax Format 1:

```
\>\>--EXEC SQL--BEGIN TRAN-.....-END-EXEC---\>\<
\+transaction_name+
```

### Syntax Format 2:

```
\>\>-EXEC SQL-BEGIN TRANSACTION.....-END-EXEC-\>\<
\+transaction_name+
```

Parameters:

## Transaction name

An optional identifier that is ignored.

Comments:

Use the BEGIN TRAN statement in AUTOCOMMIT mode to open a transaction. After you have opened the transaction in AUTOCOMMIT mode, you should execute a COMMIT or ROLLBACK statement to close the transaction and cause a return to AUTOCOMMIT mode.

If you are not opening a transaction in AUTOCOMMIT mode, then this statement has no effect.

Example:

```
EXEC SQL BEGIN TRANSACTION END-EXEC
```

## 5.2.3 CALL

Executes a stored procedure.

Syntax:

```
>>--EXEC SQL--.....->  
+-FOR :host_integer--+ +- :result_hvar -+  
  
>>--CALL stored_procedure_name-.....-END-EXEC-><  
      | +- , -+ |  
      | V | |  
      +(parameter)-+
```

Parameters:

### Host Integer

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

### result\_hvar

A host variable to receive the procedure result.

### stored\_procedure\_name

The name of the stored procedure.

### parameter

A literal, a DECLARE CURSOR statement\*, or a host variable parameter of the form:

```
[keyword=]:param_hvar [IN | INPUT |
```

INOUT | OUT | OUTPUT]

where:

*keyword* The formal parameter name for a keyword parameter. Keyword parameters can be useful as an aid to readability and where the server supports default parameter values and optional parameters.

*param\_hvar* A host variable.

IN An input parameter.

INPUT An input parameter default).

INOUT An input/output parameter.

OUT An output parameter.

OUTPUT An output parameter.

\* Specify DECLARE CURSOR for stored procedures that return a result set. The use of DECLARE CURSOR unbinds the corresponding parameter.

### Comments:

Do not use the FOR clause if the CALL is part of a DECLARE CURSOR statement.

For maximum portability, observe the following as general rules:

Avoid literal parameters

Use host variable parameters

Avoid mixing positional parameters and keyword parameters

If your server supports a mixture of positional and keyword parameters, list keyword parameters after positional parameters

### Examples:

Call a stored procedure using two positional host variables as input parameters:

```
EXEC SQL
  CALL myProc(param1,param2)
END-EXEC
```

Call a stored procedure using a keyword host variable as an input parameter:

```
EXEC SQL
  CALL myProc (namedParam=:paramValue)
END-EXEC
```

Call a stored procedure using a result host variable and a keyword host variable as an input parameter:

```
EXEC SQL
  :myResult = CALL myFunction(namedParam=:paramValue)
END-EXEC
```

Call a stored procedure using two positional host variables, one as an input parameter and one as an output parameter:

```
EXEC SQL
  CALL getDept(:empName IN, :deptName OUT)
END-EXEC
```

Call a stored procedure using a DECLARE CURSOR statement and a positional host variable as an input parameter (Oracle only):

```
EXEC SQL
  DECLARE cities CURSOR FOR CALL locateStores(:userState)
END-EXEC
```

## 5.2.4 CLOSE

Discards unprocessed rows and frees any locks held by the cursor.

## Syntax:

```
\>\>---EXEC SQL---.-----\>  
      \+-AT db_name-+  
  
\>---CLOSE---cursor_name-----END-EXEC---\>\<
```

## Parameters:

### **AT** *db\_name*

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with



the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement. Provided for backward compatibility.

### *cursor\_name*

A previously declared and opened cursor.

#### **Comments:**

The cursor must be declared and opened before it can be closed. All open cursors are closed automatically at the end of the program.

#### **Example:**

```
*Declare the cursor...
EXEC SQL
  DECLARE C1 CURSOR FOR
    SELECT staff_id, last_name
    FROM staff
  END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not declare cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF

EXEC SQL
  OPEN C1
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not open cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT CURRENT END-EXEC
  STOP RUN
END-IF

PERFORM UNTIL sqlcode NOT = ZERO
*SQLCODE will be zero as long as it has successfully fetched data
EXEC SQL
  FETCH C1 INTO :staff-staff-id, :staff-last-name
END-EXEC
IF SQLCODE = ZERO
  DISPLAY "Staff ID: " staff-staff-id
  DISPLAY "Staff member's last name: " staff-last-name
END-IF
END-PERFORM

EXEC SQL
  CLOSE C1
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not close cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
END-IF
```

## 5.2.5 COMMIT

Makes any changes made by the current transaction on the current connection permanent in the database.

#### **Syntax:**

```
>>---EXEC SQL----->
      +-AT db_name-+

>---COMMIT----->
      +-WORK-----+
      +-TRAN-----+
      +-TRANSACTION--+

>---END-EXEC--<
      +-RELEASE--+
```

### Parameters:

#### **AT** *db\_name*

The name of a database that has been declared using DECLARE DATABASE. This clause is optional. If omitted, the current connection is committed. If provided, and the connection specified is

different than the current connection, the commit is performed on the connection associated with the DECLARE CURSOR statement.

## **WORK**

WORK, TRAN, and TRANSACTION are optional and synonymous.

## **RELEASE**

If RELEASE is specified and the transaction was successfully committed, the current connection is closed.

### **Example:**

```

* Ensure that multiple records are not inserted for a
* member of staff whose staff_id is 99
EXEC SQL
DELETE FROM staff WHERE staff_id = 99
END-EXEC

* Insert dummy values into table
EXEC SQL
INSERT INTO staff
(staff_id
,last_name
,first_name
,age
,employment_date)
VALUES
(99
,'Lee'
,'Phil'
,19
,'1992-01-02')
END-EXEC

IF SQLCODE NOT = ZERO
DISPLAY 'Error: Could not insert dummy values.'
DISPLAY SQLERRMC
DISPLAY SQLERRML
EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN
END-IF

EXEC SQL
COMMIT
END-EXEC

* Check it was committed OK
IF SQLCODE = ZERO
DISPLAY 'Error: Could not commit values.'
DISPLAY SQLERRMC
DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN
END-IF

DISPLAY 'Values committed.'

* Delete previously inserted data
EXEC SQL
DELETE FROM staff WHERE staff_id = 99
END-EXEC

IF SQLCODE NOT = ZERO
DISPLAY 'Error: Could not delete dummy values.'
DISPLAY SQLERRMC
DISPLAY SQLERRML
EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN
END-IF

* Check data deleted OK, commit and release the connection
IF SQLCODE NOT = ZERO
DISPLAY 'Error: Could not delete values.'
DISPLAY SQLERRMC
DISPLAY SQLERRML
EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN
END-IF

EXEC SQL
COMMIT WORK RELEASE
END-EXEC

* Check data committed OK and release the connection.
IF SQLCODE NOT = ZERO
DISPLAY 'Error: Could not commit and release.'
DISPLAY SQLERRMC
DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
END-IF

DISPLAY 'Values committed and connection released.'

```

## 5.2.6 CONNECT

Attaches to a specific database using the supplied user name and password.

### Syntax Format 1:

```

>>---EXEC SQL---CONNECT TO----->
                                +-data_source-+

>-----USER----->
+-AS db_name-+      +-user-+-----++
                                +-password-+

>----->
+-WITH-----PROMPT-+ +-RETURNING output_connection-+
+-NO-+

>-----END-EXEC-----<

```

## Syntax Format 2:

```

>>---EXEC SQL---CONNECT user----->
                                +-IDENTIFIED BY password-+
                                +------'/'password-----+

>----->
+-AT db_name--+      +-USING data_source-+

>----->
+-WITH-----PROMPT--+
+-NO-+

>-----END-EXEC-----<
+-RETURNING output_connection-+

```

## Syntax Format 3:

```

>>---EXEC SQL---CONNECT WITH PROMPT----->

>-----END-EXEC-----<
+-RETURNING output_connection -+

```

## Syntax Format 4:

```

>>---EXEC SQL---CONNECT RESET-----END-EXEC-----<
                                +-name--+

```

## Syntax Format 5:

```

>>---EXEC SQL-----CONNECT DSN input_connection----->

>-----END-EXEC-----<
+-RETURNING output_connection -+

```

## Syntax Format 6:

```

>>---EXEC SQL---CONNECT USING input_connection----->

>----->
+-AS db_name-+ +-WITH-----PROMPT-+
                                +-NO-+

>-----END-EXEC-----<
+-RETURNING output_connection-+

```

## Parameters:

### *data\_source*

The name of the ODBC data store. For ODBC data stores, this is the DSN created via the Microsoft ODBC Data Source Administrator. If you omit *data\_source*, the default ODBC data source is assumed. The data source can be specified as a literal or as a host variable.

### *db\_name*

A name for the connection. Connection names can have as many as 30 characters and can include alphanumeric characters and any symbols legal in filenames. The first character must be a letter. Do not use Embedded SQL keywords or CURRENT or DEFAULT or ALL for the connection name; they are

invalid. If *db\_name* is omitted, DEFAULT is assumed. *db\_name* can be specified as a literal or a host variable. When connecting to SQL Server, *db\_name* is the database to which you are connecting.

### *user*

A valid user-id at the specified data source.

### *password*

A valid password for the specified user-id.

### *output\_connection*

A PIC X(n) text string defined by ODBC as the connection string used to connect to a particular data source. Subsequently, you can specify this string as the *input\_connection* in a CONNECT USING statement.

### *input\_connection*

A PIC X(n) text string containing connection information used by ODBC to connect to the data source. The text string can be either a literal or a host variable.

## RESET

Resets (disconnects) the specified connection.

### *name*

You can specify *name* as CURRENT, DEFAULT or ALL.

### OS Authentication:

When using Oracle, DB2 or SQL Server with ODBC, you can achieve OS authentication using either of these two methods:

In the CONNECT statement, specify a user ID consisting of a single forward slash and either omit the password or specify all spaces

Completely omit the user ID and password from the CONNECT statement

For complete information on OS authentication requirements for your DBMS product, consult your DBMS documentation.

### Comments:

If you use only one connection, you do not need to supply a name for the connection. When you use more than one connection, you must specify a name for each connection. Connection names are global within a process. Named connections are shared by separately compiled programs that are linked into a single executable module.

After a successful CONNECT statement, all database transactions other than CONNECT RESET work through this most recently declared current connection. To use a different connection, use the SET CONNECTION statement.

To cause the ODBC run-time module to prompt at run-time for entry or confirmation of connection details, use CONNECT WITH PROMPT.

Use CONNECT DSN and CONNECT USING to simplify administration.

With CONNECT TO, CONNECT, CONNECT WITH PROMPT, CONNECT DSN and CONNECT USING, you can return connection information to the application.

### Note

If the INIT option of the SQL Compiler directive is used, an implicit connection to the database will be made at run time. In this case, it is not necessary to execute an explicit CONNECT statement.

A File DSN cannot contain a password.

#### Example Format 1:

```
MOVE 'servername' TO svr
MOVE 'username.password' TO usr

EXEC SQL
    CONNECT TO :svr USER :usr
END-EXEC
```

#### Example Format 2:

```
EXEC SQL
    CONNECT 'username.password' USING 'servername'
END-EXEC
```

#### Example Format 3:

```
EXEC SQL
    CONNECT WITH PROMPT
END-EXEC
```

#### Example Format 4:

```
EXEC SQL
    CONNECT RESET
END-EXEC
```

#### Example Format 5:

```
EXEC SQL
    CONNECT USING 'FileDSN=Oracle8;PWD=tiger'
END-EXEC
```

The example above uses a File DSN.

#### Example Format 6:



```

01 connectString          PIC X(72) value
                          'DRIVER={Microsoft Excel Driver (*.xls)};'
                          &'DBQ=c:\demo\demo.xls;'
                          &'DRIVERID=22'
                          .
procedure division.

EXEC SQL
CONNECT USING :connectString
END-EXEC

```

The example above connects to an Excel spreadsheet without setting up a data source.

## 5.2.7 DECLARE CURSOR

Associates the cursor name with the specified SELECT statement and enables you to retrieve rows of data using the FETCH statement. **Syntax Format 1:**

```

>>--EXEC SQL-----DECLARE cursor_name----->
      +-AT db_name-+
>----->
      +-SENSITIVE---+   +-FORWARD--+   +-LOCK-----+
      +-INSENSITIVE-+   +-KEYSET---+   +-LOCKCC-----+
      +-DYNAMIC---+   +-DYNAMIC--+   +-OPTIMISTIC---+
      +-STATIC---+   +-STATIC---+   +-OPTCC-----+
      +-SCROLL---+   +-SCROLL---+   +-OPTCCVAL-----+
      +-READ ONLY---+
      +-READONLY----+
      +-FASTFORWARD--+
      +-FAST FORWARD-+
>--CURSOR-----FOR----->
      +----WITH HOLD---+
>-----select_stmt----->
      +---stored_procedure_call_statement---+
      +---prepared_stmt_name-----+
      +---OPTIMIZE FOR n ROWS-----+
>----->
      +-FOR READ ONLY-----+
      +-FOR UPDATE-----+
      +-OF column_list+
>-----END-EXEC-----<

```

### Syntax Format 2:

#### Note

Format 2 is supported for SQL Server only.

```

>>--EXEC SQL-----DECLARE cursor_name----->
      +-AT db_name-+
>--CURSOR FOR---result-set-generating-dml-statement----->
>-----END-EXEC-----<

```

### Parameters:

**AT db\_name**

The name of a database that has been declared using DECLARE DATABASE.

### Note

If you must use AT *db\_name* in a DECLARE CURSOR, the connection for any following statements that reference the cursor automatically switch to the connection associated with the cursor if different than the current connection, but only for the duration of the statement.

### *cursor\_name*

Cursor name used to identify the cursor in subsequent statements. Cursor names can contain any legal filename character and be up to 30 characters in length. The first character must be a letter.

### *select\_stmt*

Any valid SQL SELECT statement, or a QUERY ODBC statement or a CALL statement for a stored procedure that returns a result set.

### *prepared\_stmt\_name*

The name of a prepared SQL SELECT statement or QUERY ODBC statement.

### *stored\_procedure\_call\_stmt*

A valid stored procedure call which returns a result set.

### *n*

The number of rows per block fetched when the cursor is opened. The value of *n* must be less than 1000.

### *column\_list*

A list of column-names, separated by commas.

### *result-set-generating-dmlstatement*

A SQL Server INSERT, non-positioned UPDATE, or DELETE statement with an OUTPUT clause.

### Comments:

Two separately compiled programs cannot share the same cursor. All statements that reference a particular cursor must be compiled together.

The DECLARE CURSOR statement must appear before the first reference to the cursor. The SELECT statement runs when the cursor is opened. The following rules apply to the SELECT statement:

It cannot contain an INTO clause or parameter markers.

It can contain input host variables previously identified in a declaration section.

With some ODBC drivers, the SELECT statement must include a FOR UPDATE clause if positioned updates or deletions are to be performed.

If OPTIMIZE FOR is specified, OpenESQL uses *n* to override the setting of the PREFETCH directive for the cursor. This allows prefetch optimization for individual cursors.

You can specify multiple SELECT statements in a DECLARE CURSOR statement, signifying the return of multiple result sets from either of the following. In either case, the client application must use the GET NEXT RESULT SET statement to retrieve additional result sets.

A COBOL stored procedure for SQL Server

A standard OpenESQL application program

The following applies to the behavior of certain DECLARE CURSOR options:

SCROLL selects a scroll option, other than FORWARD, that is supported by the driver.

LOCKCC and LOCK are equivalent.

READ ONLY and READONLY are equivalent.

OPTIMISTIC selects an optimistic concurrency mode (OPTCC or OPTCCVAL) that is supported by the driver.

If a HOLD cursor is requested and the current connection closes cursors at the end of transactions, the OPEN statement will return an error (SQLCODE = -19520).

If the database is Microsoft SQL Server and the NOANSI92 ENTRY directive setting has been used (this is the default setting), then a Microsoft SQL Server specific ODBC call will be made at connect time to request that cursors are not closed at the end of transactions. This is compatible with the Micro Focus Embedded SQL Toolkit for Microsoft SQL Server. The setting for USECURLIB must not be YES.

FAST FORWARD and FASTFORWARD are equivalent. This is a performance optimization parameter that applies only to FORWARD, READ-ONLY cursors. You can obtain even greater performance gains by also compiling the program with the AUTOFETCH directive; this is the most efficient method of getting a results set into an application. The AUTOFETCH directive enables two optimizations that can significantly reduce network traffic. The most dramatic improvement is seen when processing cursors with relatively small result sets that can be cached in the memory of an application. FASTFORWARD cursors work only with Microsoft SQL Server 2000 or later servers.

**Example:**

```
EXEC SQL DECLARE C1 CURSOR FOR
  ELECT last_name, first_name FROM staff
END-EXEC

EXEC SQL DECLARE C2 CURSOR FOR
  QUERY ODBC COLUMNS TABLENAME 'staff'
END-EXEC
```

## 5.2.8 DECLARE DATABASE

Declares the name of a database.

### Syntax:

```
>>---EXEC SQL---DECLARE db_name---DATABASE---END-EXEC--->>
```

### Parameters:

#### *db\_name*

A name associated with a database. It must be an identifier and not a host variable. It cannot contain quotation marks.

### Comments:

You must DECLARE *db\_name* before using a CONNECT ... AT *db\_name* statement. You cannot use DECLARE DATABASE with EXECUTE IMMEDIATE or with PREPARE and EXECUTE.

## 5.2.9 DELETE (Positioned)

Deletes the row most recently fetched by using a cursor.

### Syntax:

```
>>---EXEC SQL---.----->>
      +-AT db_name-+
>--DELETE---FROM---table_name--->
>--WHERE CURRENT OF--cursor_name---END-EXEC---<
```

### Parameters:

#### AT *db\_name*

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with

the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement.

### *table\_name*

The same table used in the SELECT FROM option (see DECLARE CURSOR).

### *cursor\_name*

A previously declared, opened, and fetched cursor.

#### **Comments:**

ODBC supports positioned delete, which deletes the row most recently fetched by using a cursor in the Extended Syntax (it was in the Core Syntax for ODBC 1.0 but was moved to the Extended Syntax for ODBC 2.0). Not all drivers provide support for positioned delete, although OpenESQL sets ODBC cursor names to be the same as COBOL cursor names to facilitate positioned update and delete.

With some ODBC drivers, the select statement used by the cursor must contain a 'FOR UPDATE' clause to enable positioned delete.

You cannot use host arrays with positioned delete.

The other form of DELETE used in standard SQL statements is known as a searched delete.

Most data sources require specific combinations of SCROLLOPTION and CONCURRENCY to be specified either by SET statements or in the DECLARE CURSOR statement.

The ODBC cursor library provides a restricted implementation of positioned delete which can be enabled by compiling with SQL(USECURLIB=YES) and using SCROLLOPTION STATIC and CONCURRENCY OPTCCVAL (or OPTIMISTIC).

#### **Example:**

```

* Declare a cursor for update
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT staff_id, last_name FROM staff FOR UPDATE
END-EXEC

IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not declare cursor for update.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
END-IF

* Open the cursor
EXEC SQL
    OPEN C1
END-EXEC
IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not open cursor for update.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
END-IF

* Display staff member's details and give user the opportunity
* to delete particular members.
PERFORM UNTIL SQLCODE NOT = ZERO
    EXEC SQL FETCH C1 INTO :staff-id, :last-name END-EXEC
    IF SQLCODE = ZERO
        DISPLAY 'Staff ID: ' staff-id
        DISPLAY 'Last name: ' last-name
        DISPLAY 'Delete <y/n>? ' WITH NO ADVANCING
        ACCEPT usr-input
        IF usr-input = 'y'
            EXEC SQL
                DELETE FROM staff WHERE CURRENT OF C1
            END-EXEC
            IF SQLCODE NOT = ZERO
                DISPLAY 'Error: Could not delete record.'
                DISPLAY SQLERRMC
                DISPLAY SQLERRML
            END-IF
        END-IF
    END-IF
END-UNTIL
END-IF
END-PERFORM

```

## 5.2.10 DELETE (Searched)

Removes table rows that meet the search criteria.

### Syntax:

```

>>--EXEC SQL-- ..----->
    +-FOR :host_integer-+

>-- ..----->
    +-AT db_name--+      +-FROM-+

>-- ..table_name- ..----->
    +-view_name+ +-WHERE search_conditions-+

```

### Parameters:

#### *host\_integer*

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

#### *AT db\_name*

The name of a database that has been declared using DECLARE DATABASE. This clause is optional. If omitted, the current connection is deleted. If provided, and the connection specified is

different than the current connection, the delete is performed on the connection associated with the DECLARE CURSOR statement.

## FROM

An optional keyword. It is required for ANSI SQL 92 conformance.

### *table\_name*

The target table for the delete operation.

### *view\_name*

The target view for the delete operation.

## WHERE

A standard SQL WHERE clause identifying the row to be deleted.

### *search\_conditions*

Any valid expression that can follow the standard SQL WHERE clause.

### Comments:

DELETE is a standard SQL statement. See the documentation supplied with your ODBC driver for the exact syntax.

You cannot mix simple host variables with host arrays in the WHERE clause. If one of the host variables is an array, they must all be arrays.

If you do not specify a WHERE clause, all the rows in the named table are removed.

### Example:

```
EXEC SQL
  DELETE FROM staff WHERE staff_id = 99
END-EXEC
```

## 5.2.11 DESCRIBE

Provides information on prepared dynamic SQL statements and describes the result set for an open cursor.

### Syntax Format 1:

```
>>---EXEC SQL-----DESCRIBE---.----->
      +---SELECT LIST FOR----+
      +---BIND VARIABLES FOR--+
>---prepared_stmt_name---INTO---:sqlda_struct---END-EXEC--
><
```

## Syntax Format 2:

```
>>---EXEC SQL-----DESCRIBE---CURSOR---cursor_name----->  
>---INTO---:sqlda_struct---END-EXEC--><
```

### Parameters:

#### *prepared\_stmt\_name*

The name of a prepared SQL SELECT statement or QUERY ODBC statement. *cursor\_name* The name of an open cursor.

#### *:sqlda\_struct*

A host variable that specifies the output SQLDA data structure to be populated. The colon is optional to provide compatibility with other embedded SQL implementations.

### Comments:

This statement populates the specified SQLDA data structure with the data type, length, and column name of each column returned by the specified prepared statement.

If neither SELECT LIST FOR or BIND VARIABLES FOR is specified, SELECT LIST FOR is used by default. If BIND VARIABLES FOR is specified, information about input parameters is returned in the SQLDA rather than information about results columns.

The DESCRIBE statement inserts the number of columns into the *sqld* field of the SQLDA structure. If a non-select statement was prepared, *sqld* is set to 0. Before DESCRIBE is called, the following fields in the SQLDA data structure must be initialised by the application:

#### *sqln*

The maximum number of *sqlvar* (column descriptor) entries that the structure can accommodate.

#### *sqldabc*

The Maximum size of the SQLDA:

32-bit – Calculated as  $sqln * 44 + 16$

64-bit – Calculated as  $sqln * 56 + 16$

If *sqln* is set to 0, no column descriptor entries are constructed, but *sqld* is set to the number of entries required. The DESCRIBE statement works in a similar way to a PREPARE statement with an INTO clause.



By default, the SQL types for date, time and timestamp are respectively DATE-RECORD, TIMEREORD and TIMESTAMP-RECORD. When you use the BEHAVIOR=OPTIMIZED option for the SQL Compiler directive, CitOESQL mimics the DB2 on the mainframe for these data types, providing character strings (i.e., PIC X(n)) instead of the standard, default record constructs.

## Note

Few drivers fully implement the ODBC calls necessary for DESCRIBE BIND VARIABLES.

### Example:

```
$set sql(behavior=optimized)
working-storage section.
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE SQLDA78 END-EXEC.
EXEC SQL BEGIN DECLARE SECTION END-EXEC
01 statement pic x(80).
01 host-var-block.
   03 host-var-1 pic 99.
   03 host-var-2 pic x(10).
   03 host-var-3 pic x(15).
EXEC SQL END DECLARE SECTION END-EXEC

PROCEDURE DIVISION.

EXEC SQL CONNECT TO ORCL USER scott.tiger END-EXEC

EXEC SQL
  DECLARE C1 CURSOR FOR stmt1
END-EXEC

move "select * from dept" to statement

move 20 to sqln
$IF P64 SET
  compute sqldabc = 16 + 56 * sqln
$ELSE
  compute sqldabc = 16 + 44 * sqln
$END

EXEC SQL
  PREPARE stmt1 FROM :statement
END-EXEC
EXEC SQL
  DESCRIBE stmt1 INTO :sqlda
END-EXEC

* The data structure "sqlda" now contains a description
* of the dynamic SQL statement.
EXEC SQL
  OPEN C1
END-EXEC

* Complete the SQLDA, by adding buffer addresses and lengths
* and changing types, as necessary and appropriate, to
* to match host variables actually used.
*
* The following SQL directives can reduce the amount of effort
* required by specifying how OpenESQL should DESCRIBE varchar
* and date/time SQL data types:
*   DESCRIBEVARCHARPICX
*   DESCRIBEVARCHAR49
*   DESCRIBEDTCHAR
*   DESCRIBEDTREC

move ESQL-UDISP-UNSIGN to sqltype(1)
set sqldata(1) to address of host-var-1
set sqldata(2) to address of host-var-2
set sqldata(3) to address of host-var-3

perform until exit
  EXEC SQL
    FETCH C1 USING DESCRIPTOR :sqlda
  END-EXEC
  if sqlerrd(3) not = 1
    exit perform
  end-if
  display host-var-1 ' ' host-var-2 ' ' host-var-3

end-perform
goback.
```

## 5.2.12 DISCONNECT

Closes the connection(s) to a database. In addition, all cursors opened for that connection are automatically closed.

### Syntax:

```
>>---EXEC SQL---DISCONNECT---.name-----END-EXEC----<
      +-ALL-----+
      +-CURRENT--+
      +-DEFAULT--+
```

### Parameters:

#### *name*

The connection name.

#### ALL

Disconnects all connections (including automatic connections made when the INIT option of the SQL Compiler directive is used).

#### CURRENT

Disconnects the current connection. The current connection is either the most recent connection established by a CONNECT statement or a subsequent connection set by a SET CONNECTION statement.

#### DEFAULT

Disconnects the default connection. This is the connection made by a CONNECT statement which did not specify a connection name.

### Example:

```
EXEC SQL CONNECT TO "srv1" AS server1 USER "sa." END-EXEC EXEC SQL CONNECT TO "srv2" AS server2 USER "sa." END-EXEC
...
EXEC SQL DISCONNECT server1 END-EXEC
EXEC SQL DISCONNECT server2 END-EXEC.
```

## 5.2.13 END DECLARE SECTION

Terminates a host variable declaration section begun by a BEGIN DECLARE SECTION statement.

### Syntax:

```
>>---EXEC SQL---END DECLARE SECTION-----END-EXEC----<
```

### Example:

```

WORKING-STORAGE SECTION.

EXEC SQL BEGIN DECLARE SECTION END-EXEC

01 staff-id      pic x(4).
01 last-name     pic x(30).
EXEC SQL END DECLARE SECTION END-EXEC

```

## 5.2.14 EXECSP

Executes a stored procedure.

### Syntax:

```

>>--EXEC SQL-.....-EXECSP-.....->
      +-FOR :host_integer-+ +:result_hvar -+

>- stored_procedure_name ----->
      | +- , --+ |
      | V | |
      | (parameter)-+
>.....-END-EXEC-----<>
      +-WITH RECOMPILE-+

```

### Parameters:

#### *:HOST\_INTEGER*

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

#### *:RESULT\_HVAR*

A host variable to receive the procedure result.

#### *stored\_procedure\_name*

The name of the stored procedure.

#### *parameter*

A literal or a host variable parameter of the form:

```
[keyword=]:param_hvar [OUT | OUTPUT]
```

where:

*keyword* is the formal parameter name for a keyword parameter.

*:param\_hvar* is a host variable.

OUT specifies an output parameter.

OUTPUT specifies an output parameter.

WITH RECOMPILE Is ignored and has no effect. It is allowed for syntax compatibility only.

### Example:

```
EXEC SQL
  EXECSP myProc param1,param2
END-EXEC

EXEC SQL
  EXECSP :myResult = myFunction namedParam = :paramValue
END-EXEC

EXEC SQL
  EXECSP getDept :empName, :deptName OUT
END-EXEC

EXEC SQL
  DECLARE cities CURSOR FOR EXECSP locateStores :userState
END-EXEC
```

## 5.2.15 EXECUTE

Processes dynamic SQL statements.

### Syntax:

```
>>-EXEC SQL-----EXECUTE----->
      +---FOR :host_integer---+

>-prepared_stmt_name----->
      +-USING DESCRIPTOR :sqlda_struct-+
      | +- , -+ |
      | V | |
      +-USING :hvar-----+

>-----END-EXEC-----<>
```

### Parameters:

#### *:HOST\_INTEGER*

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

#### *prepared\_stmt\_name*

A previously prepared SQL statement.

#### *:SQLDA\_STRUCT*

A previously declared SQLDA data structure containing a description of the input values. The colon is optional to provide compatibility with other embedded SQL implementations.

#### *:HVAR*

One or more input host variables.

### Comments:

Do not use the FOR clause if the EXECUTE is part of a DECLARE CURSOR statement.

The EXECUTE statement runs the specified prepared SQL statement after substituting values for any parameter markers. (Prepared statements are created using the PREPARE statement.) Only statements that do not return results are permitted.

If the prepared statement contains parameter markers, the EXECUTE statement must include either the USING :hvar option with the same number of host variables or the USING DESCRIPTOR :sqlda\_struct option identifying a SQLDA data structure already populated by the application.

The number of parameter markers in the prepared statement must match the number of sqldata entries (USING DESCRIPTOR :sqlda) or host variables (USING :hvar).

### Example:

```
* Store statement to be dynamically executed...
MOVE "INSERT INTO staff VALUES(?,?,?,?,?)" TO stmbuf.

* Ensure attempt is not made to insert an existing record
EXEC SQL
  DELETE FROM staff WHERE staff_id = 99
END-EXEC

* Prepare the statement
EXEC SQL
  PREPARE st FROM :stmbuf
END-EXEC.

MOVE 99 TO staff-id
MOVE 'Lee' TO last-name
MOVE 'Phil' TO first-name
MOVE 19 TO age
MOVE '1997-01-01' TO employment-date

* Execute the statement with current values.
EXEC SQL
  EXECUTE st USING :staff-id, :last-name
                ,:first-name, :age, :employment-date
END-EXEC
IF SQLCODE = ZERO
  DISPLAY 'Statement executed.'
ELSE
  DISPLAY 'Error: Could not execute statement.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF

* Finally, remove the entry
EXEC SQL
  DELETE FROM staff where staff_id = 99
END-EXEC
IF SQLCODE = ZERO
  DISPLAY 'Values deleted.'
ELSE
  DISPLAY 'Error: Could not delete inserted values.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF
```

## 5.2.16 EXECUTE IMMEDIATE

Immediately executes the SQL statement.

### Syntax

```
>>--EXEC SQL--.....--EXECUTE IMMEDIATE-->
      +--FOR :host_integer--+
>----:stmt_hvar----END-EXEC-<>
```

**Parameters:**

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

A character string host variable.

### Comments:

Do not use the FOR clause if the EXECUTE IMMEDIATE is part of a DECLARE CURSOR statement.

The EXECUTE IMMEDIATE statement cannot contain input parameter markers or host variables. It cannot return results; any results returned from this statement are discarded. Additionally, the statement cannot contain SQL keywords that pertain exclusively to Embedded SQL.

If any rows are returned, SQLCODE is set to +1.

EXECUTE IMMEDIATE must be used for SET statements specific to the Microsoft SQL Server, that is, those that are intended to execute at that server.

### Example:

```
EXEC SQL
    DELETE FROM staff WHERE staff_id = 99
END-EXEC

* Put the required SQL statement in prep.
MOVE "insert into staff (staff_id, last_name, first_name ,age,
- "employment_date) VALUES (99, 'Lee', 'Phillip', 19, '1992-
- '01-02')" TO prep
* Note EXECUTE IMMEDIATE does not require the statement to be
* prepared

EXEC SQL
    EXECUTE IMMEDIATE :prep
END-EXEC

* Check it worked...

IF SQLCODE = ZERO
    DISPLAY 'Statement executed OK.'
ELSE
    DISPLAY 'Error: Statement not executed.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
END-IF

* Run through the same procedure again, this time deleting the
* values just inserted
MOVE "delete from staff where staff_id = 99" TO prep
EXEC SQL
    EXECUTE IMMEDIATE :prep
END-EXEC

IF SQLCODE = ZERO
    DISPLAY 'Statement executed OK.'
ELSE
    DISPLAY 'Error: Statement not executed.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    EXEC SQL DISCONNECT ALL END-EXEC
    STOP RUN
END-IF
```

## 5.2.17 FETCH

Retrieves a row from the cursor's results set and writes the values of the columns in that row to the corresponding host variables (or to addresses specified in the SQLDA data structure).

### Syntax:

```

>>--EXEC SQL--.....->
      +-FOR :host_integer-+

>-----FETCH----->

      +-AT db_name--+          +---PREVIOUS---+
                                +---LAST-----+
                                +---PRIOR-----+
                                +---FIRST-----+
                                +---NEXT-----+

>-----cursor_name-----\>
      +-USING DESCRIPTOR :sqlda_struct-----+

      |          +-----+-----+--+|
      |          V          ||
      +-INTO--.:hvar-----+
      +-:hvar:ivar-----+
      +-:hvar--.:ivar--+
              +-INDICATOR-+

>--END EXEC--<<

```

## Parameters:

### [:HOST\\_INTEGER](#)

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

### [AT DB\\_NAME](#)

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement. Provided for backward compatibility.

### [CURSOR\\_NAME](#)

A previously declared and opened cursor.

### [:SQLDA\\_STRUCT](#)

An SQLDA data structure previously populated by the DESCRIBE statement and containing output value addresses. This option is used only with a cursor declared by a prepared SELECT statement.



(SELECT statements are prepared using the PREPARE statement.) The colon is optional to provide compatibility with other embedded SQL implementations.

## **:HVAR**

Specifies either of the following: One or more host variables, each separated by a comma. One or more host variable+indicator variable combinations, each combination separated by a comma.

### **Comments:**

By default, the FETCH statement retrieves the next row, but you can also specify the previous row or last row or prior row or first row. If there are no more rows to fetch SQLCODE is set to 100 and SQLSTATE is set to "02000".

An OPEN *cursor\_name* statement must precede a FETCH statement, and the cursor must be open while FETCH runs. If you use PREVIOUS, LAST, PRIOR, FIRST or NEXT, you must also set the appropriate cursor options via the DECLARE CURSOR statement or the SET SCROLLOPTION and SET CONCURRENCY statements. Also, the data type of the host variable must be compatible with the data type of the corresponding database column.

If the number of columns is less than the number of host variables, the value of SQLWARN3 is set to

W. If an error occurs, no further columns are processed. (Processed columns are not undone.)

Alternatively, the *:hvar* variable can specify a COBOL record that contains several fields, each corresponding to a column in the select list of the cursor declaration statement. To use this form, you must specify the DB2 option of the SQL Compiler directive. (Note that this will cause PREPARE INTO and DESCRIBE statements to be rejected by the COBOL compiler).

If ANSI92ENTRY is set, then attempting to fetch a null value will set SQLCODE to -19425 if there is no null indicator. If ANSI92ENTRY is not set, SQLCODE will be 0. In both cases, SQLSTATE will be 22002 and SQLWARN2 will be W.

If one of the host variables in the INTO clause is an array, they must all be arrays.

After execution, SQLERRD(3) contains the number of elements processed. For FETCH it is the number of rows fetched.

### **Example:**

```

* Declare a cursor for a given SQL statement.

EXEC SQL DECLARE C1 CURSOR FOR
    SELECT last_name, first_name FROM staff
END-EXEC

EXEC SQL OPEN C1 END-EXEC

* Fetch the current values from the cursor into the host variables
* and if everything goes ok, display the values of the host
* variables

PERFORM UNTIL SQLCODE NOT = ZERO
    EXEC SQL
        FETCH C1 INTO :lname, :fname
    END-EXEC
    IF SQLCODE NOT = ZERO AND SQLCODE NOT = 100
        DISPLAY 'Error: Could not perform fetch'
        DISPLAY SQLERRML DISPLAY SQLERRMC
        EXEC SQL DISCONNECT ALL END-EXEC
        STOP RUN
    END-IF
    DISPLAY 'First name: ' fname DISPLAY 'Last name : ' lname
    DISPLAY SPACES
END-PERFORM

```

## 5.2.18 GET HDBC

Enables you to use ODBC calls that require you to supply the ODBC connection handle.

### Syntax:

```
>>---EXEC SQL---GET HDBC---INTO---:hvar---END-EXEC--->
```

### Parameters:

#### HVAR

A host variable to store the ODBC connection handle. Must be declared as PIC X(4) COMP-5.

### Example:

```
EXEC SQL
    GET HDBC INTO :hvar
END-EXEC
```

## 5.2.19 GET HENV

Enables you to use ODBC calls that require you to supply the ODBC environment handle.

### Syntax:

```
>>---EXEC SQL---GET HENV---INTO---:hvar---END-EXEC--->
```

### Parameters:

#### HVAR

A host variable to store the ODBC environment handle. Must be declared as PIC X(4) COMP-5.

### Example:

```
EXEC SQL
  GET HENV INTO :HENV
END-EXEC
```

## 5.2.20 GET NEXT RESULT SET

---

Makes the next result set available to an open cursor.

### Syntax:

```
>>---EXEC SQL---.----->
      +-AT db_name-+
>--GET NEXT RESULT SET FOR---cursor_name---END-EXEC---<
```

### Parameters:

#### **AT DB\_NAME**

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with

the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement.

## CURSOR\_NAME

A previously declared and opened cursor.

### Comments:

GET NEXT RESULT SET makes the next result set available when retrieving multiple result sets from a stored procedure or from a DECLARE CURSOR statement defined with multiple SELECT statements.

If additional result sets are not available, GET NEXT RESULT SET returns an SQLCODE of 100 and sets SQLSTATE to 02000.

### Example:

```
exec sql declare c cursor for
    call TestProc2(:hv-country)
end-exec

exec sql open c end-exec

display " "
display "First result set from proc"
display " "

perform until exit
    exec sql fetch c into
        :CustomerID, :Company, :City
    end-exec
    if sqlcode = 100 or sqlcode < 0
        exit perform
    end-if
    display CustomerID City
end-perform

*> Always get SQLCODE 100 at the end of a result set
*> until you close the cursor or ask for another
result set

exec sql fetch c into
    :CustomerID, :Company, :City
end-exec
if sqlcode not = 100
    display "FAIL: Fetch after SQLCODE 100 OK"
end-if

*> Ask for another result set, SQLCODE 100 if there
isn't one

exec sql get next result set for c end-exec

display " "
display "Second result set from proc"
display " "

perform until exit
    exec sql fetch c into
        :CustomerID, :Company, :City
    end-exec
    if sqlcode = 100 or sqlcode < 0
        exit perform
    end-if display CustomerID " " City
end-perform
```

## 5.2.21 INCLUDE

Includes the definition of the specified SQL data structure or source file in the COBOL program.

### Syntax:

```
>>---EXEC SQL---INCLUDE-----SQLCA-----END-EXEC----<
                                     +-SQLDA----+
                                     +-filename--+
```

## Parameters:

### SQLCA

Indicates that a SQLCA data structure is accessed. SQLDA Indicates that a SQLDA data structure is accessed.

### FILENAME

Indicates that a file should be included in the source at this point (this is equivalent to the COBOL COPY facility).

## Comments:

This statement uses the corresponding .cpy file. Ensure that sqlca.cpy and sqlda.cpy are in the current directory or in the environment variable COBCPY directory.

## Example:

```
EXEC SQL INCLUDE SQLCA END-EXEC
EXEC SQL INCLUDE SQLDA END-EXEC
EXEC SQL INCLUDE MYFILE END-EXEC
```

## 5.2.22 INSERT

Adds new rows to a table.

## Syntax:

```
>>---EXEC SQL--.....->
                                     +-FOR :host_integer-+
>-----INSERT-----table_name----->
+-AT db_name--+      +-INTO--+  +-view_name-->
                                     +------+
                                     v         |
>-----VALUES (constant_expression)----->
+- (column_list)-+
>-----END-EXEC---<
```

## Parameters:

### :HOST\_INTEGER

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

### AT DB\_NAME

The name of a database that has been declared using DECLARE DATABASE. This clause is optional. If omitted, the current connection executes the insert. If provided, and the connection specified is

different than the current connection, the insert is performed on the connection associated with the DECLARE CURSOR statement.

### **TABLE\_NAME**

The table into which rows are to be inserted. *view\_name* The view into which rows are to be inserted.

### **INTO**

An optional keyword. Required for ANSI SQL 92 conformance.

### **COLUMN\_LIST**

A list of one or more columns to which data is to be added. The columns can be in any order, but the incoming data must be in the same order as the columns. The column list is necessary only when some, but not all, columns in the table are to receive data. Enclose the items in the column list parentheses. If no column list is given, all the columns in the receiving table (in CREATE TABLE order) are assumed.

The column list determines the order in which values are entered.

### **VALUES**

Introduces a list of constant expressions.

### **CONSTANT\_EXPRESSION**

Constant or null values for the indicated columns. The values list must be enclosed in parentheses and must match the explicit or implicit columns list.

Enclose non-numeric constants in single or quotation marks.

### Comments:

The INSERT statement is passed directly to the ODBC driver. See the documentation supplied with your ODBC driver for the exact syntax.

If the host variables in the WHERE clause are arrays, the INSERT statement is executed once for each set of array elements.

Use UPDATE to modify column values in an existing row.

You can omit items in the column list and VALUES list providing that the omitted columns are defined to allow null values.

You can select rows from a table and insert them into the same table in a single statement.

After execution, SQLERRD(3) contains the number of elements processed. For INSERT it is the total number of rows inserted.

### Example:

```
DISPLAY "Enter new staff member's id:"
ACCEPT staff-id

DISPLAY "Enter new staff member's last name:"
ACCEPT last-name

DISPLAY "Enter new staff member's first name:"
ACCEPT first-name

DISPLAY "Enter new staff member's age:"
ACCEPT age

DISPLAY "Enter new staff member's employment date(yyyy/mm/dd):"
ACCEPT employment-date

EXEC SQL
  INSERT INTO staff
    (staff_id
    ,last_name
    ,first_name
    ,age
    ,employment_date)
  VALUES
    (:staff-id
    ,:last-name
    ,:first-name
    ,:age
    ,:employment-date)
END-EXEC
```

## 5.2.23 INTO

Retrieves one row of results and assigns the values of the items returned by an OUTPUT clause in a SQL Server INSERT, UPDATE, or DELETE statement to the host variables specified in the INTO list.

### Syntax:

```

>>---EXEC SQL----->
      +-FOR :host_integer-+ +-AT db_name-+
      +- , -+
      V |
>>---INTO--:hvar--result-set-generating-dml-statement---->
>>---END-EXEC---<<

```

### Parameters:

#### *HOST\_INTEGER*

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

#### *DB\_NAME*

The name of a database that has been declared using DECLARE DATABASE. |

#### *HVAR*

A host variable to store the ODBC connection handle. Must be declared as PIC X(4) COMP-5. |

#### *RESULT-SET-GENERATING-DMLSTATEMENT*

A SQL Server INSERT, non-positioned UPDATE, or DELETE statement with an OUTPUT clause. |

### Comments:

INTO is supported for SQL Server only.

## 5.2.24 OPEN

Runs the SELECT statement specified in the corresponding DECLARE CURSOR statement to produce the results set that is accessed one row at a time by the FETCH statement.

### Syntax:

```

>>---EXEC SQL---OPEN---cursor_name----->
>>-----END-EXEC-----<<
      +-USING DESCRIPTOR :sqlda_struct-+
      | +- , -+ |
      | V |
      +-USING :hvar-----+

```

### Parameters:

#### *CURSOR\_NAME*

A previously declared cursor.

#### *:SQLDA\_STRUCT*

An SQLDA data structure previously constructed by the application. The SQLDA data structure contains the address, data type, and length of each input parameter. This option is used only



with a cursor that references a prepared SQL statement in the DECLARE statement. The colon is optional to provide compatibility with other embedded SQL implementations.

### **:HVAR**

One or more input host variables corresponding to parameter markers in the SELECT statement. This option is used only with a cursor that references a prepared SQL statement in the DECLARE statement. |

#### **Comments:**

If the cursor is declared with a static SELECT statement (that is, one that was not prepared), the SELECT statement can contain host variables but not parameter markers. The current values of the host variables are substituted when the OPEN statement runs. For a statically declared cursor, the OPEN statement cannot contain the USING :hvar or USING DESCRIPTOR :sqlda\_struct option.

If the cursor is declared with a dynamic SELECT statement (that is, one that was prepared), the SELECT statement can contain parameter markers but not host variables. Parameter markers can appear wherever column values are allowed in the SELECT statement. If the SELECT statement has parameter markers, the OPEN statement must include either the USING :hvar option with the same number of host variables or the USING DESCRIPTOR :sqlda\_struct option identifying an SQLDA data structure already populated by the application.

With the USING DESCRIPTOR :sqlda\_struct option, values of the program variables are substituted for parameter markers in the SELECT statement. These program variables are addressed by corresponding SQLDATA entries in the SQLDA data structure.

The number of parameter markers in the SELECT statement must match the number of sqldata entries (USING DESCRIPTOR :sqlda\_struct) or host variables (USING :hvar) in the OPEN statement.

#### **Example:**

```

*Declare the cursor...
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT staff_id, last_name
  FROM staff
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not declare cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF

EXEC SQL
OPEN C1
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not open cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT CURRENT END-EXEC
  STOP RUN
END-IF

PERFORM UNTIL sqlcode NOT = ZERO
*SQLCODE will be zero as long as it has successfully fetched data
EXEC SQL
  FETCH C1 INTO :staff-staff-id, :staff-last-name
END-EXEC
IF SQLCODE = ZERO
  DISPLAY "Staff ID: " staff-staff-id
  DISPLAY "Staff member's last name: " staff-last-name
END-IF
END-PERFORM

EXEC SQL
CLOSE C1
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not close cursor.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
END-IF

```

## 5.2.25 PREPARE

Processes dynamic SQL statements.

### Syntax:

```

>>--EXEC SQL--PREPARE--stmt_name--.----->
                                     +-INTO :sqlda-+
>--FROM--:hvar--END-EXEC-----\>\<

```

### Parameters:

#### STMT\_NAME

The prepared statement name. This can be used by a subsequent EXECUTE or OPEN statement, and/or a previous DECLARE CURSOR statement.

#### :SQLDA

The output SQL descriptor area (SQLDA) data structure to be populated. The colon is optional to provide compatibility with other embedded SQL implementations.

#### :HVAR

The host variable that contains the SQL statement.

## Comments:

You can use a prepared statement in one of two ways:

You can open a cursor that references a prepared statement.

You can execute a prepared statement.

If the prepared statement is used by an EXECUTE statement, *:hvar* cannot contain a SQL statement that returns results.

Because singleton SELECT statements (SELECT INTO) are not allowed in dynamic SQL statements, they cannot be prepared.

When using PREPARE, the SQL statement in *:hvar* cannot contain host variables or comments, but it can contain parameter markers. Also, the SQL statement cannot contain SQL keywords that pertain exclusively to Embedded SQL.

The INTO *:sqlda* option merges the functionality of DESCRIBE and PREPARE so that this example code:

```
EXEC SQL
  PREPARE stmt1 INTO :sqlda FROM :stmt-buf
END-EXEC
```

Is identical to:

```
EXEC SQL
  PREPARE stmt1 FROM :stmt-buf
END-EXEC
EXEC SQL
  DESCRIBE stmt1 INTO :sqlda
END-EXEC
```

## Example:

```

PROGRAM-ID. progname.

WORKING-STORAGE SECTION.
EXEC SQL INCLUDE SQLCA END-EXEC
EXEC SQL BEGIN DECLARE SECTION END-EXEC
01 prep          PIC X(80).
01 nme           PIC X(20).
01 car           PIC X(20).
01 n60          PIC x(5).
EXEC SQL END DECLARE SECTION END-EXEC.

PROCEDURE DIVISION.
EXEC SQL CONNECT TO 'srv1' USER 'sa' END-EXEC
IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not connect to database.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
    STOP RUN
END-IF

* Ensure attempt is not made to recreate an existing table...
EXEC SQL DROP TABLE mf_table END-EXEC

* Create a table...
EXEC SQL CREATE TABLE mf_table
        (owner          char(20)
        ,car_col        char(20)
        ,nought_to_60   char(5))
END-EXEC

IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not create table'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN
END-IF

* Insert an SQL statement into host variable prep...
MOVE "insert into mf_table values(?,?,?)" TO prep

* Prepare the statement...
EXEC SQL
    PREPARE prep_stat FROM :prep
END-EXEC

IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not prepare statement'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN
END-IF

MOVE "Owner" TO nme
MOVE "Lamborghini" TO car
MOVE "4.9" TO n60

* Execute the prepared statement using the above host variables...
EXEC SQL
    EXECUTE prep_stat USING :nme, :car, :n60
END-EXEC

IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not execute prepared statement.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN
END-IF

* Finally, drop the now unwanted table...
EXEC SQL
    DROP TABLE mf_table
END-EXEC

IF SQLCODE NOT = ZERO
    DISPLAY 'Error: Could not drop table.'
    DISPLAY SQLERRMC
    DISPLAY SQLERRML
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN
END-IF

DISPLAY 'All statements executed.'
EXEC SQL DISCONNECT CURRENT END-EXEC
STOP RUN.

```

## 5.2.26 QUERY ODBC

Delivers a results set in the same way as a SELECT statement, and must therefore be associated with a cursor via DECLARE and OPEN, or DECLARE, PREPARE and OPEN.

### Syntax Format 1:

```
>>--EXEC SQL--QUERY ODBC--.-COLUMN--.------>
                                     +-COLUMNS-+
>--.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
    +-QUALIFIER qualifier_name-+ +-OWNER owner_name-+
>--.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
    +-TABLENAME table_name-+ +-COLUMNNAME column_name-+
>--END-EXEC--<<>
```

### Syntax Format 2:

```
>>--EXEC SQL--QUERY ODBC--.-DATATYPE--.------>
                                     +-DATATYPES-+
>--.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
    +-TYPE--.-datatype_name--.-.+
    +-BIGINT-----+
    +-BINARY-----+
    +-BIT-----+
    +-CHAR-----+
    +-DATE-----+
    +-DECIMAL-----+
    +-DOUBLE-----+
    +-FLOAT-----+
    +-INTEGER-----+
    +-LONG VARBINARY-+
    +-LONG VARCHAR---+
    +-NUMERIC-----+
    +-REAL-----+
    +-SMALLINT-----+
    +-TIME-----+
    +-TIMESTAMP-----+
    +-TINYINT-----+
    +-VARBINARY-----+
    +-VARCHAR-----+
```

### Syntax Format 3:

```
>>--EXEC SQL--QUERY ODBC--.-TABLE--.------>
                                     +-TABLES-+
>--.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
    +-QUALIFIER qualifier_name-+ +-OWNER owner_name-+
>--.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.->
    +-TABLENAME table_name-+ +-TYPE tabletype_name----+
>--END-EXEC--<<>
```

### Parameters:

#### QUALIFIER\_NAME

A host variable, identifier or literal which specifies a qualifier to be used to select tables. Not all ODBC drivers support qualifiers, and those that do may use them in different ways. For example, if a data source supports multiple databases, a qualifier can be used to specify which

database to use. Alternatively, for drivers providing access to file based data sources, a qualifier can be used to specify a particular directory to be searched. |

### OWNER\_NAME

A host variable, identifier or literal which specifies a table owner to be used to select tables. Not all ODBC drivers support table ownership.

### TABLE\_NAME

A host variable, identifier or literal which specifies tables to be included in the query.

### DATATYPE\_NAME

A host variable, identifier or literal which specifies a data type to be queried.

### TABLETYPE\_NAME

A host variable, identifier or literal which specifies a list of table types to be included in the query.

#### Comments:

Search patterns consist of the legal characters for SQL identifiers plus underscore ( `_` ) which matches any single character, percent ( `%` ) which matches any sequence of zero or more characters, or a driver defined escape character which can be used to allow underscore or percent in a pattern to represent themselves rather than a wildcard.

If a search pattern parameter is not supplied, a pattern of `%` is used, which will match all relevant dictionary entries.

For table queries the following special rules apply:

- If *qualifier-name* is `%` and *owner-name* and *table-name* are empty strings, the results set consists of a list of valid qualifiers at the data source. All columns apart from `TABLE_QUALIFIER` in the results set (see below) will be null.
- If *owner-name* is `%` and *qualifier-name* and *table-name* are empty strings, the results set consists of a list of valid owners at the data source. All columns apart from `TABLE_OWNER` in the results set (see below) will be null.
- If *tabletype-name* is `%` and *qualifier-name*, *owner-name* and *table-name* are empty strings the results set consists of a list of valid table types at the data source. All columns apart from `TABLE_TYPE` in the results set (see below) will be null.
- If *tabletype-name* is not specified, tables of all types will be returned in the results set. If it is specified it must consist of a comma separated list of table types, for example `'TABLE,VIEW'`.

#### Example:

```

EXEC SQL
  DECLARE tcurs CURSOR FOR QUERY ODBC TABLES
END-EXEC

EXEC SQL DECLARE C1 CURSOR FOR
  QUERY ODBC TABLES OWNER :tab-owner TABLETYPE 'TABLE,VIEW'
END-EXEC

MOVE 'staff' to tab-name
EXEC SQL DECLARE C2 CURSOR FOR
  QUERY ODBC COLUMNS TABLENAME :tab-name
END-EXEC

EXEC SQL DECLARE C3 CURSOR FOR
  QUERY ODBC DATATYPES
END-EXEC

```

## QUERY ODBC - Column Query

The results set for a column query is:

<b>TABLE_QUALIFIER</b>	<b>VARCHAR(128)</b>	
TABLE_OWNER	VARCHAR(128)	
TABLE_NAME	VARCHAR(128) NOT NULL	
COLUMN_NAME	VARCHAR(128) NOT NULL	
DATA_TYPE	SMALLINT NOT NULL	See odbccext.cpy and odbc.cpy for constants representing the ODBC data type codes.
TYPE_NAME	VARCHAR(128) NOT NULL	Driver dependent name for the column's data type.
PRECISION	INTEGER	
LENGTH	INTEGER	Amount of memory required for a column value in its native representation.
SCALE	SMALLINT	
RADIX	SMALLINT	For numeric columns either 10 or 2 depending on the data type; otherwise null
NULLABLE	SMALLINT NOT NULL	

<b>TABLE_QUALIFIER</b>	<b>VARCHAR(128)</b>
REMARKS	VARCHAR(254)

### QUERY ODBC - Data Type Query

The results set for a data type query is:

<b>TYPE_NAME</b>	<b>VARCHAR(128) NOT NULL</b>	<b>Driver dependent name for the column's data type.</b>
DATA_TYPE	SMALLINT NOT NULL	See odbcxext.cpy and odbccpy for constants representing the ODBC data type codes.
PRECISION	INTEGER	Maximum precision for columns of this type.
LITERAL_PREFIX	VARCHAR(128)	Character or characters required to prefix literal values for this type.
LITERAL_SUFFIX	VARCHAR(128)	Character or characters required to suffix literal values for this type.
CREATE_PARAMS	VARCHAR(128)	Parameters required when creating a column of this type, for example, 'precision,scale' for decimal types.
NULLABLE	SMALLINT NOT NULL	
CASE_SENSITIVE	SMALLINT NOT NULL	Specifies case sensitivity in comparisons for character data types.
SEARCHABLE	SMALLINT NOT NULL	SQL_UNSEARCHABLE, SQL_LIKE_ONLY or SQL_ALL_EXCEPT_LIKE (these are defined in odbccpy).
UNSIGNED_ATTRIBUTE	SMALLINT	Specifies if a numeric type is signed or unsigned.
MONEY	SMALLINT NOT NULL	Specifies if a numeric type is a money data type.
AUTO_INCREMENT	SMALLINT	Specifies if the data type is auto incrementing.



<b>TYPE_NAME</b>	<b>VARCHAR(128)</b> <b>NOT NULL</b>	<b>Driver dependent name for the column's data type.</b>
LOCAL_TYPE_NAME	VARCHAR(128)	Localized version of the data type name.
MINIMUM_SCALE	SMALLINT	
MAXIMUM_SCALE	SMALLINT	

### QUERY ODBC - Table Query

The results set for a table query is:

<b>TABLE_QUALIFIER</b>	<b>VARCHAR(128)</b>	
TABLE_OWNER	VARCHAR(128)	
TABLE_NAME	VARCHAR(128)	
TABLE_TYPE	VARCHAR(128)	One of TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS, SYNONYM or a data source specific type identifier

<b>TABLE_QUALIFIER</b>	<b>VARCHAR(128)</b>
<b>REMARKS</b>	<b>VARCHAR(254)</b>

## 5.2.27 RESET CONNECTION

Closes all open cursors, even if the application has not appropriately closed them.

### Syntax:

```
>>---EXEC SQL---RESET CONNECTION---END-EXEC---<
```

## 5.2.28 ROLLBACK

Backs out any changes made to the database by the current transaction on the current connection, or partially rolls back changes to a previously set save point.

### Syntax:

```
>>---EXEC SQL----->
      +-AT db_name-+
>--ROLLBACK----->
      +-WORK-----+
      +-TRAN-----+
      +-TRANSACTION--+
>---END-EXEC---<
      +-RELEASE--+
      +-TO-_*name*--+
      +-SAVEPOINT-+
```

### Parameter:

#### AT DB\_NAME

The name of a database that has been declared using DECLARE DATABASE. This clause is optional. If omitted, the current connection is rolled back. If provided, and the connection specified is

different than the current connection, the rollback is performed on the connection associated with the DECLARE CURSOR statement.

### Comments:

When `RELEASE` is specified and the transaction is successfully rolled back, the current connection is closed.

`TO [SAVEPOINT]` rolls the transaction back just to the save point specified by *name*, which must be set by a preceding `SAVEPOINT` statement.

### Example:

```
EXEC SQL
  ROLLBACK
END-EXEC

EXEC SQL
  ROLLBACK WORK RELEASE
END-EXEC

END-EXEC

EXEC SQL
  ROLLBACK TO SP1
END-EXEC
```

## 5.2.29 SAVEPOINT, SAVE TRANSACTION, RELEASE [TO] SAVEPOINT

Sets a transaction save point to which a current transaction can be rolled back, resulting in a partial roll back.

### Syntax:

```
>>---EXEC SQL-----SAVEPOINT-*name*----->
      +-AT db_name-+                +UNIQUE+

>--END-EXEC---<
+-ON ROLLBACK RETAIN CURSORS--.-----+
      +-ON ROLLBACK RETAIN LOCKS--+

>---EXEC SQL--.-----SAVE--TRANSACTION--.name-- ENDEXEC---<
      +-AT db_name-+                +-TRAN-----+

>>---EXEC SQL-----RELEASE--.-----SAVEPOINT-name--- END-EXEC---<
      +-AT db_name-+                +-TO-+
```

### Parameter:

#### AT DB\_NAME

The name of a database that has been declared using `DECLARE DATABASE`. This clause is not required, and if omitted, the connection automatically switches to the connection associated with

the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement.

### Comments:

You can define multiple save points for a single transaction.

When you set a save point using a unique name, and subsequently set another save point using the same unique name, the named save point is reset to the current transaction state.

The behavior of cursors and locks after a rollback to a save point is database-specific. For details, see the documentation provided by your database vendor.

### Example:

```
EXEC SQL
  SAVEPOINT SP1
END-EXEC

EXEC SQL
  SAVEPOINT PHASE2 ON ROLLBACK RETAIN CURSORS
END-EXEC
```

## 5.2.30 SELECT DISTINCT (using DECLARE CURSOR)

Associates the cursor name with the SELECT DISTINCT statement and enables you to retrieve rows of data using the FETCH statement.

### Syntax:

```
>>---EXEC SQL---.-----DEclare cursor_name----->
      +-AT db_name-+

>---CURSOR FOR-----SELECT DISTINCT-----select_list----->

>---FROM-----table_list--.-----2--END-EXEC-----<
      +-select_options--+
```

### Parameters:

#### *DB\_NAME*

The name of a database that has been declared using DECLARE DATABASE.

#### *CURSOR\_NAME*

Cursor name used to identify the cursor in subsequent statements. Cursor names can contain any legal filename character and be up to 30 characters in length. The first character must be a letter.

#### *SELECT\_LIST*

The name of the columns to retrieve.

#### *TABLE\_LIST*

The name of the tables that contain the columns to be retrieved, as specified in *select\_list*.

## SELECT\_OPTIONS

The options specified to limit the number of rows retrieved and/or order the rows retrieved.

### Comments:

Two separately compiled programs cannot share the same cursor. All statements that reference a particular cursor must be compiled together.

The SELECT DISTINCT statement runs when the cursor is opened. The following rules apply to the SELECT DISTINCT statement:

- The statement cannot contain an INTO clause or parameter markers.
- The statement can contain input host variables previously identified in a declaration section.
- With some ODBC drivers, the SELECT DISTINCT statement must include a FOR UPDATE clause if positioned updates or deletions are to be performed.

### Note

Use SELECT DISTINCT instead of SELECT INTO to remove duplicate rows in the row set.

### Example:

```
01 age-array          pic s9(09) comp-5 occurs 10 times.
01 lname-array       pic x(32) occurs 10 times.

MOVE 5 TO staff-id
EXEC SQL
  SELECT DISTINCT last_name
  INTO :lname-array
  FROM staff
  WHERE staff_id > :staff-id
END-EXEC

EXEC SQL
  SELECT DISTINCT age
  INTO :age-array
  FROM staff
  WHERE first_name > 'George'
END-EXEC
```

## 5.2.31 SELECT INTO

Retrieves one row of results and assigns the values of the items in a specified SELECT list to the host variables specified in the INTO list.

### Syntax:

```

>>---EXEC SQL----->
      +-FOR :host_integer-+ +-AT db_name-+
                                     +- , -+
                                     V   |
>>---SELECT----->
      +-select_list-+
                                     INTO--:hvar----->
>>---select_options---END-EXEC---<<

```

## Parameters:

### **:HOST\_INTEGER**

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

### **DB\_NAME**

The name of a database that has been declared using DECLARE DATABASE.

### **SELECT\_LIST**

The portion of the table to retrieve data from.

### **:HVAR**

One or more host variables to receive the *select\_list* items.

### **SELECT\_OPTIONS**

One or more statements or other options that can be used with the SQL SELECT statement (for example, a FROM or WHERE clause).

## Comments:

A singleton SELECT must contain a FROM clause.

If more columns are selected than the number of receiving host variables, the value of *sqlwarn3* is set to 'W'. The data type and length of the host variable must be compatible with the value assigned to it. If data is truncated, the value of *sqlwarn1* is set to 'W'.

If a SELECT INTO statement returns more than one row from the database, all rows except the first one will be discarded and *sqlwarn4* will be set to "W". If you want to return more than the first row, you should use a cursor. Alternatively, you can specify array items in the INTO clause. The array will be populated up to either the maximum size of the array, the value of *host\_integer* or the number of rows returned, whichever is the smallest.

If SELECT INTO returns more rows from the database than the statement in the application is able to accept, CitoESQL returns the following for each of the specified directives:

```

CHECKSINGLETON SQLCODE = -811 SQLSTATE = 21000 SQLWARN4 = W
NOCHECKSINGLETON SQLCODE = 0 SQLSTATE = 00000 SQLWARN4 = space

ANSI92ENTRY SQLCODE = -1 SQLSTATE = 21000 SQLWARN4 = W

```

If SELECT INTO returns more rows from the database than the statement in the application is able to accept, and none of these directives are set, then CitOESQL returns:

```
SQLCODE = +1 SQLSTATE = 21000 SQLWARN4 = W
```

### Note

If any one of the host variables in the INTO clause is an array, then they all must be arrays.

### Example:

```
...
MOVE 99 TO staff-id
EXEC SQL
  SELECT last_name
  INTO :lname
  FROM staff
  WHERE staff_id=:staff-id
END-EXEC
EXEC SQL
  SELECT staff_id
  INTO :staff-id
  FROM staff
  WHERE first_name = 'Phil'
END-EXEC
```

## 5.2.32 SET AUTOCOMMIT

Enables you to control ODBC AUTOCOMMIT mode at run time.

### Syntax:

```
>>--EXEC SQL--SET AUTOCOMMIT---.-ON--.--END-EXEC-->>
+-OFF-+
```

### Parameters:

#### ON

Changes to AUTOCOMMIT mode, whereby each SQL statement is treated as a separate transaction and is committed immediately upon execution.

#### OFF

Switches off AUTOCOMMIT mode. If the ODBC driver you are using supports transactions, statements must be explicitly committed (or rolled back) as part of a transaction.

### Comments:

The SET AUTOCOMMIT statement is useful for data sources which can only execute DDL statements, such as CREATE and DROP, in AUTOCOMMIT mode.

When set to ON, AUTOCOMMIT releases locks when an outermost stored procedure executes a COMMIT or a ROLLBACK statement.

This statement overrides the AUTOCOMMIT SQL compiler directive option.

### Example:

```
EXEC SQL SET AUTOCOMMIT ON END-EXEC
```

## 5.2.33 SET CONNECTION

Sets the named connection as the current connection.

### Syntax:

```
>>--EXEC SQL--SET CONNECTION---.name-----END-EXEC--><  
+-DEFAULT--+
```

### Parameters:

#### NAME

Specifies the name of a database connection. Must match the connection name specified in a previous CONNECT statement. The name can be either the connection's literal name or the name of a host variable containing character values.

#### DEFAULT

If you have established a connection using the CONNECT statement but omitting the connection name, you can refer to the connection established as DEFAULT.

### Comments:

If you are using connections across compilation modules you must use named connections.

### Example:



```

EXEC SQL CONNECT TO "srv1" AS server1 USER "sa." END-EXEC
EXEC SQL CONNECT TO "srv2" AS server2 USER "sa." END-EXEC

* server2 is the current connection
EXEC SQL CREATE TABLE phil1
(charbit CHAR(5))
END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not create table.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF

EXEC SQL INSERT INTO phil1 VALUES('hello') END-EXEC

IF SQLCODE NOT = ZERO
  DISPLAY 'Error: Could not insert data.'
  DISPLAY SQLERRMC
  DISPLAY SQLERRML
  EXEC SQL DISCONNECT ALL END-EXEC
  STOP RUN
END-IF

* set the current connection to server1
EXEC SQL SET CONNECTION server1 END-EXEC
EXEC SQL
  SELECT first_name
  INTO :fname
  FROM staff
  WHERE staff_id = 10
END-EXEC

DISPLAY fname ' says ' WITH NO ADVANCING

* set the current connection back to server2
EXEC SQL SET CONNECTION server2 END-EXEC
EXEC SQL
  SELECT charbit
  INTO :fname
  WHERE charbit = 'hello'
  FROM phil1
END-EXEC

DISPLAY fname
EXEC SQL DISCONNECT server1 END-EXEC
EXEC SQL DISCONNECT server2 END-EXEC
STOP RUN

```

## 5.2.34 SET ERRORMAP

Changes the SQL error map file for the current connection.

### Syntax:

```
>>--EXEC SQL--SET ERRORMAP--map-file-prefix--END-EXEC--<<
```

### Parameters:

#### MAP-FILE-PREFIX

The prefix of the SQL error map file name (no file extension).

### Comments:

SQL Error Mapping must be enabled using the ERRORMAP SQL compiler directive option.

The SET ERRORMAP statement is most useful in applications that use multiple database connections.

## 5.2.35 SET host\_variable

Provides information about CitOESQL connections and databases.

### Syntax:

```
>>--EXEC SQL-- .----->
                +-AT db_name+

>--SET :host_variable = >
>----- .CURRENT CONNECTION----->
+-CURRENT DATABASE-----+
+-OPTION-----+DATE-----+USA-----+
                    +-EUR-----+
                    +-JIS-----+
                    +-ODBC-----+
                    +-EXTERNAL+
                    +-ISO-----+
                    +-DEFAULT--+
+-TIME-----+USA-----+
                    +-EUR-----+
                    +-JIS-----+
                    +-ODBC-----+
                    +-EXTERNAL+
                    +-ISO-----+
                    +-DEFAULT--+
+-DATEDELIM--+char-----+
+-TIMEDELIM--+char-----+
+-TSTAMPSEP--+char-----+
+-DETECTDATE+-CLIENT--+
+-SERVER--+ +-OFF-----+

>---END EXEC---<
```

### Parameters:

#### AT DB\_NAME

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with

the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement.

## HOST\_VARIABLE

A PIC X(*n*) host variable

## CHAR

Any single printable character or space

### Note

For a full description of each option and its corresponding parameters, see the equivalent *SQL Compiler Directive Options* topic as listed in the *Related reference* section below.

### Comments:

This statement returns the name of the current connection or the type of database for the current connection as specified in the EXEC SQL CONNECT statement in the specified host variable.

If there is no current open connection, host\_variable is set to **NONE** for both CURRENT CONNECTION and CURRENT DATABASE.

With CURRENT CONNECTION, you can use host\_variable in a subsequent EXEC SQL SET CONNECTION statement.

With CURRENT DATABASE, if there is a current open connection, host\_variable is set to one of the following values, depending on the connection:

- DB2
- SQLSERVER
- ORACLE
- POSTGRESQL
- OTHER

In addition to the OPTION syntax specified above, you can optionally place an equals (=) sign between the option and its parameter. For example, the following are equivalent:

```
exec sql set :myhostvar option date EUR end-exec
```

and

```
exec sql set :myhostvar option date=EUR end-exec
```

## 5.2.36 SET OPTION

Enables you to set CitOESQL options.

### Syntax:

```
>>---EXEC SQL---SET OPTION-----QUERYTIME----->>
      +-LOGINTIME---+
      +-APPLICATION--+
      +-HOST-----+
      +-DATE-----+USA-----+
                +-EUR-----+
                +-JIS-----+
                +-ODBC-----+
                +-EXTERNAL--+
                +-ISO-----+
                +-DEFAULT--+
      +-TIME-----+USA-----+
                +-EUR-----+
                +-JIS-----+
                +-ODBC-----+
                +-EXTERNAL--+
                +-ISO-----+
                +-DEFAULT--+
      +-DATEDELIM--+char-----+
      +-TIMDELIM--+char-----+
      +-TSTAMPSEP--+char-----+
      +-DETECTDATE--+CLIENT---+
                +-SERVER---+
                +-OFF-----+

>---value---END-EXEC---<<
```

### Parameters:

#### VALUE

A literal or the name of a host variable. The host variable must contain character values for APPLICATION or HOST and numeric values for LOGINTIME or QUERYTIME.

#### QUERYTIME

Sets the number of seconds that the program waits for a response to an CitOESQL statement. The default is 0, meaning forever. This option does not override existing network timeout settings.

#### LOGINTIME

Sets the number of seconds that the program waits for a response to a CONNECT TO statement. The default is 10 seconds. A value of 0 indicates an infinite timeout period.

#### APPLICATION

Sets the application name which is passed by CitOESQL to the data source when a CONNECT TO statement is executed.

#### HOST

Sets the host workstation name which is passed by CitOESQL when a CONNECT TO statement is executed.

### Comments:

The SET OPTION statement is not supported by all ODBC drivers.

As an option, you can place an equals (=) sign between an option and its parameter. For example, the following are equivalent:

```
exec sql set option date EUR end-exec
```

and

```
exec sql set option date=EUR end-exec
```

### Example:

```
EXEC SQL SET OPTION logintime 5 END-EXEC

EXEC SQL CONNECT TO "srv2" USER "sa." END-EXEC

* If the CONNECT statement cannot log in to the server "srv2"
* within five seconds, it will time out and return to the
program.

EXEC SQL SET OPTION querytime 2 END-EXEC

EXEC SQL SELECT name FROM sysobjects INTO :name END-EXEC

* If the SELECT statement does not respond within 2 seconds,
* the query will time out and return to the program.
```

## 5.2.37 SET TRACELEVEL

Enables you to dynamically set or change the reporting level of CitOESQL traces for native applications.

### Syntax:

```
>>--EXEC SQL--SET TRACELEVEL--.-0-----.--END-EXEC--<<
      +-1-----+
      +-2-----+
      +-3-----+
      +-4-----+
      +-5-----+
      +-6-----+
      +-OFF-----+
      +-DEFAULT--+
```

### Parameters:

0

Turns off CitOESQL trace.

1

The following information is written to the trace file:

<b>BEGIN</b>	Traces main SQL directives.
<b>END</b>	Indicates end of run.
<b>DIRECTIVES</b>	Traces per compilation unit directives the first time a compilation unit is encountered at run time
<b>PREPARE</b>	Identifies the original source code when a statement is prepared

<b>DISPOSE</b>	Provides summary information for overall statement usage when a statement is removed from the prepared statement cache at disconnect time
<b>FLUSH</b>	Provides summary information for overall statement usage when a statement is flushed from the cache usually due to a cache overflow

## 2

The following information is written to the trace file in addition to the information written when you set the trace level to 1:

<b>OPEN</b>	
<b>EXECUTE</b>	Provides the number of rows selected, inserted or updated
<b>EXEC_IMMEDIATE EXECUTE</b>	Provides the number of rows selected, inserted or updated
<b>ODBCCLOSE</b>	Provides summary information for the current cursor use
<b>STMT CHANGED</b>	Reports new concurrency and scroll option settings when the ODBC driver uses different settings than those requested by CitOESQL.

## 3

The following information is written to the trace file in addition to the information written when you set the trace level to 2:

<b>ODBCFETCH</b>	Provides the number of rows fetched

**COBOLFETCH**

Provides the number of rows returned to the COBOL application

**4**

The following information is written to the trace file in addition to the information written when you set the trace level to 3:

```
EXEC_SQL_BEGIN
```

```
EXEC_SQL_END
```

**5**

The following information is written to the trace file in addition to the information written when you set the trace level to 4:

```
ODBC_CALL_START
```

```
ODBC_CALL_END
```

**6**

Only the following information is written to the trace file:

```
ODBC_CALL_START
```

```
ODBC_CALL_END
```

**OFF**

Turns off the CitOESQL trace

**DEFAULT**

Resets the trace setting to the value set by the SQL TRACELEVEL directive. If the TRACELEVEL directive was not used to compile a program, this is equivalent to setting this option to OFF. |

**Example:**

```
EXEC SQL SET TRACELEVEL DEFAULT END-EXEC
```

## 5.2.38 SET TRANSACTION ISOLATION

Sets the transaction isolation level for the current connection to one of the isolation level modes specified by ODBC.

**Syntax:**

```
>>--EXEC SQL--SET TRANSACTION ISOLATION----->
>-----, -READ UNCOMMITTED-,-----END-EXEC-----<
+-READ COMMITTED---+
+-REPEATABLE READ---+
+-SERIALIZABLE-----+
```

### Comments:

Transactions can affect each other in the following ways, depending on the setting of the transaction isolation level:

- **Dirty read** - Transaction 1 updates a row. Transaction 2 reads the row before transaction 1 commits. Transaction 1 issues a rollback. Transaction 2's results are based on invalid data.
- **Nonrepeatable read** - Transaction 1 reads a row. Transaction 2 updates or deletes the row and commits the change. If transaction 1 re-reads the row, it will retrieve different values, or may not be able to re-read the row.
- **Phantom** - Transaction 1 reads a set of rows using a select with a where clause. Transaction 2 inserts a row that satisfies the where clause. If transaction 1 repeats the select, it will read a different set of rows.

These situations can be controlled by locking, which means that a transaction might have to wait until another transaction completes, which limits concurrency (sometimes called pessimistic concurrency), or by forcing a transaction to rollback if the situation occurs, which has less of an impact on concurrency but may force work to be repeated (this is sometimes called optimistic concurrency).

In READ UNCOMMITTED mode, dirty reads, nonrepeatable reads and phantoms are all possible.

In READ COMMITTED mode, dirty reads are not possible but nonrepeatable reads and phantoms are.

In REPEATABLE READ mode, dirty reads and nonrepeatable reads are not possible, but phantoms are.

In SERIALIZABLE mode dirty reads, nonrepeatable reads and phantoms are all impossible.

### Note

*A driver might not support all the isolation levels defined by ODBC. If you set a mode that the driver does not support, SQLCODE and SQLSTATE are set accordingly.*

### Example:

```
EXEC SQL SET TRANSACTION ISOLATION READ UNCOMMITTED END-EXEC
```

## 5.2.39 SYNCPOINT



Closes all open cursors that were not opened using the WITH HOLD clause, even if the application has not appropriately closed them.

**Syntax:**

```
>>---EXEC SQL---SYNCPPOINT---END-EXEC---<<
```

## 5.2.40 UPDATE (Positioned)

Updates the row most recently fetched by using a cursor.

**Syntax:**

```
>>---EXEC SQL-----,----->
      +---FOR :host_integer+ +AT db_name+
              +-----+
              V           |
>---UPDATE---table_name---SET--column_expression----->
>--WHERE CURRENT OF--cursor_name---END-EXEC---<<
```

**Parameters:**

### *:HOST\_INTEGER*

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

### *AT DB\_NAME*

The name of a database that has been declared using DECLARE DATABASE. This clause is not required, and if omitted, the connection automatically switches to the connection associated with

the DECLARE CURSOR statement if different than the current connection, but only for the duration of the statement. |

### *TABLE\_NAME*

The table to be updated.

### *COLUMN\_EXPRESSION*

A value for a particular column name. This value can be an expression or a null value.

### *CURSOR\_NAME*

A previously declared, opened, and fetched cursor.

#### **Comments:**

Do not use the FOR clause if the UPDATE is part of a DECLARE CURSOR statement.

After execution, SQLERRD(3) contains the number of elements processed. For UPDATE it is the total number of rows updated.

ODBC supports positioned update, which updates the row most recently fetched by using a cursor, in the Extended Syntax (it was in the core Syntax for ODBC 1.0 but was moved to the Extended Syntax for ODBC 2.0). Not all drivers provide support for positioned update, although CitOESQL sets ODBC cursor names to be the same as COBOL cursor names to facilitate positioned update and delete.

With some ODBC drivers, the select statement used by the cursor must contain a FOR UPDATE clause to enable positioned update.

The other form of UPDATE used in standard SQL statements is known as a searched update.

You cannot use host arrays with positioned update.

#### **Example:**

```

EXEC SQL CONNECT TO 'srv1' USER 'sa' END-EXEC

EXEC SQL DECLARE C1 CURSOR FOR
  SELECT last_name, first_name
  FROM staff
  FOR UPDATE
END-EXEC

EXEC SQL
  OPEN C1
END-EXEC

PERFORM UNTIL SQLCODE NOT = ZERO

  EXEC SQL
    FETCH C1 INTO :fname, :lname
  END-EXEC

  IF SQLCODE = ZERO
    DISPLAY fname " " lname
    DISPLAY "Update?"
    ACCEPT reply
    IF reply = "y"
      DISPLAY "New last name?"
      ACCEPT lname
      EXEC SQL
        UPDATE staff
          SET last_name=:lname WHERE CURRENT OF c1
      END-EXEC
      DISPLAY "update sqlcode=" SQLCODE
    END-IF
  END-IF
END-PERFORM

EXEC SQL DISCONNECT ALL END-EXEC
STOP RUN.

```

## 5.2.41 UPDATE (Searched)

Updates a table or view based on specified search conditions.

### Syntax:

```

>>--EXEC SQL-- .----->
      +-FOR :host_integer+

>----- .table_name-.---->
+-AT db_name--+          +-view_name--+

      +-----+
      V         |
>--SET--column_expression-- .----->
                              +-WHERE search_conditions-+

>----END-EXEC----<

```

### Parameters:

#### **:HOST\_INTEGER**

A host variable that specifies the maximum number of host array elements processed. Must be declared as PIC S9(4) COMP-5 or PIC S9(9) COMP-5.

#### **AT DB\_NAME**

The name of a database that has been declared using DECLARE DATABASE. This clause is optional. If omitted, the current connection executes the update. If provided, and the connection specified is

different than the current connection, the update is performed on the connection associated with the DECLARE CURSOR statement.

### TABLE\_NAME

The table to be updated.

### VIEW\_NAME

The view to be updated.

### COLUMN\_EXPRESSION

A value for a particular column name. This value can be an expression or a null value.

### SEARCH\_CONDITIONS

Any valid expression that can follow the standard SQL WHERE clause.

#### Comments:

UPDATE is a standard SQL statement which is passed directly to the ODBC driver. See the documentation supplied with your ODBC driver for the exact syntax.

If you do not specify a WHERE clause, all the rows in the named table are updated.

If one of the host variables used in the WHERE clause or SET clause is an array, they must all be arrays.

After execution, SQLERRD(3) contains the number of elements processed. For UPDATE it is the total number of rows updated.

#### Example:

```
EXEC SQL
  UPDATE staff
  SET first_name = 'Jonathan'
  WHERE staff_id = 1
END-EXEC

MOVE 'Phil' TO NewName
MOVE 1 TO targetID

EXEC SQL
  UPDATE staff
  SET first_name = :NewName
  WHERE staff_id = :targetID
END-EXEC
```

## 5.2.42 WHENEVER

Specifies the default action after running an Embedded SQL statement when a specific condition is met.

#### Syntax:

```

>>>-EXEC SQL---WHENEVER---.-NOT FOUND--->
      +-SQLERROR---+
      +-SQLWARNING--+

>---.-CONTINUE-----.-END-EXEC---<<
      +-PERFORM label---+
      +-GOTO stmt_label+

```

## Parameters:

### CONTINUE

Causes the next sequential statement in the source program to run.

### PERFORM LABEL

Identifies a paragraph or section of code to be performed when a certain error level is detected.

### GOTO STMT\_LABEL

Identifies the place in the program that takes control when a certain error level is detected. |

## Comments:

The WHENEVER statement specifies the default action after running an Embedded SQL statement on each of the following conditions: NOT FOUND, SQLERROR, SQLWARNING.

Condition	Sqlcode
No error	0
NOT FOUND	100
SQLWARNING	+1

Condition	Sqlcode
SQLERROR	\<0 (negative)

The scope of a WHENEVER statement is related to the position of statements in the source program, not in the run sequence. The default is CONTINUE for all three conditions.

### Example:

The following example shows the WHENEVER statement in use:

```
``` EXEC SQL WHENEVER sqlerror PERFORM errormessage1 END-EXEC
```

```
EXEC SQL
  DELETE FROM staff
  WHERE staff_id = 'hello'
END-EXEC

EXEC SQL
  DELETE FROM students
  WHERE student_id = 'hello'
END-EXEC

EXEC SQL WHENEVER sqlerror CONTINUE END-EXEC

EXEC SQL
  INSERT INTO staff VALUES ('hello')
END-EXEC

DISPLAY 'Sql Code is: ' SQLCODE
EXEC SQL WHENEVER sqlerror PERFORM errormessage2 END-EXEC

EXEC SQL
  INSERT INTO staff VALUES ('hello again')
END-EXEC

STOP RUN.
```

errormessage1 SECTION.

display "SQL DELETE error: " sqlcode  
EXIT.

errormessage2 SECTION.

display "SQL INSERT error: " sqlcode  
EXIT.

## 5.3 CitOESQL Directives

---

Each SQL compiler directive option is used either at compile time, run time, or both. The behavior at run time is described as one of the following:

### Source file

When a source file specifies the directive, the value set in the source file is used. If a source file does not specify the directive, then the process behavior is used.

### Process

These directives affect connections. When the first-encountered EXEC SQL statement is executed, usually an EXEC SQL CONNECT statement, the run-time system uses the directive settings for the source file containing the statement. These settings apply for the remainder of the process lifetime. The run-time behavior varies depending on the type of connection as follows:

- ODBC with THREAD=ISOLATE - Each thread in the process uses its own set of global directive settings
- ODBC with THREAD=SHARE - One set of global directive settings applies for the entire process

For additional information on the scope of each SQL compiler directive option, see its corresponding CitOESQL directive below.

### Process-based CitOESQL Directives:

ALLOWNULLCHAR  
 ANSI92ENTRY  
 AUTOCOMMIT  
 CHECKDUPCURSOR  
 CHECKSINGLETON  
 CLOSE\_ON\_COMMIT  
 CLOSE\_ON\_ROLLBACK  
 CONNECTIONPOOL  
 CURSORCASE  
 DECDEL  
 ISOLATION  
 NIST  
 ODBCTRACE  
 ODBC3  
 PARAMARRAY • PREFETCH  
 RESULTARRAY  
 STMTCACHE  
 TARGETDB  
 THREAD  
 TRACELEVEL  
 USECURLIB

Directive	Description
ALLOWNULLCHAR	Allows programs to use PIC X(n) host variables, and to select/insert/update the null character (x00) into CHAR columns without changing source to use SQL TYPE BINARY host variables
ALLOWSERVERSELECT	Passes unrecognized EXEC SQL SELECT statements through to the server, thus enabling server-specific behavior.
ANSI92ENTRY	If this is set, CitOESQL conforms to the SQL ANSI'92 entry level standard.
AUTOCOMMIT	Regulates an SQL connection's autocommit attribute.



Directive	Description
AUTOFETCH	Sets the AUTOFETCH attribute on SELECT statements that are run on Microsoft SQL Server data sources. Compiling with this directive can help the performance of your application. This directive works only if the program is also compiled with directive SQL(TARGETDB=MSSQLSERVER). It also serves as a primitive directive for the BEHAVIOR directive option.
BEHAVIOR	Instructs CitoESQL to properly match your COBOL cursors with the database you are using, thus maximizing database cursor performance. BEHAVIOR makes use of multiple primitive directives and associates a default value for each primitive directive depending on the target DBMS. You can also override the default setting of any primitive directive.
CHECK	Sends each SQL statement to the database at compilation time.
CHECKDUPCURSOR	Instructs CitoESQL to determine if the cursor has been opened twice, and if so takes action accordingly.
CHECKSINGLETON	Instructs CitoESQL to check if singleton SELECTs return more than one row when executed.
CLOSE_ON_COMMIT	Specifies whether to close cursors not defined WITH HOLD or leave them open for further fetches after a COMMIT.
CLOSE_ON_ROLLBACK	Specifies whether to close cursors or leave them open for further fetches after a ROLLBACK.
CONNECTIONPOOL	Enables use of ODBC 3.0 connection pooling. When a connection is closed, the Driver Manager actually keeps it alive for a timeout period, and saves the overhead of re-establishing a connection from scratch if the application re-opens an identical connection. ODBC allows you to choose between having a pooling for an ODBC environment or for each driver. See your ODBC documentation for details.
CURSORCASE	If ESQVERSION is 2.0, CURSORCASE is implied. NOCURSORCASE means that cursor names are not case sensitive. CURSORCASE means that they are case sensitive.
DATE	Specifies the explicit date format to use when date values are returned from database date columns into character output host variables.

<b>Directive</b>	<b>Description</b>
DATEDELIM	Specifies a single character as the delimiter between the year, month, and day components to override the default delimiter determined by the DATE directive specification, or implicitly based on default ISO 8601 format ( <i>yyyy-mm-dd</i> ).
DB	Identifies a data source that defines database connection information.
DBMAN	Specifies the preprocessor to use. This directive is not required when compiling programs with CitOESQL.
DECDEL	Specifies the decimal delimiter to use for decimal variables.
DESCRIBEDTCHAR	When using dynamic SQL, described or prepared SQL statements with DATE, TIME, and DATETIME columns are suitable for PIC X( <i>n</i> ) character host variables or DATE, TIME, and TIMESTAMP SQL TYPES.
DESCRIBEDTREC	When using dynamic SQL, described or prepared SQL statements with DATE, TIME, and DATETIME, columns are suitable for the DATE, TIME, and TIMESTAMP-RECORD SQL TYPES in ODBC format record structures.
DESCRIBEVARCHAR49	When using dynamic SQL, described or prepared SQL statements with VARCHAR, columns are suitable for VARCHAR host variables with level 49 sub-fields for length and data.
DESCRIBEVARCHARPICX	When using dynamic SQL, described or prepared SQL statements with VARCHAR, columns are suitable for PIC X host variables.
DETECTDATE	Allows datetime values for PIC X character input host variables in an CitOESQL application to be in different formats than the standard ISO 8601 formats.
ERRORMAP	Specifies the name of the error map file to use, and enables SQL error mapping.
IGNORESCHEMAERRORS	Suppresses compile-time errors resulting from missing schema objects.

<b>Directive</b>	<b>Description</b>
INIT	When set without parameters, the preprocessor automatically generates code to make the connection to the database. When set with the PROT parameter, protects the database when an application terminates abnormally.
ISOLATION	This directive specifies the isolation level that CitOESQL uses as a connection attribute. It also serves as a primitive directive for the BEHAVIOR directive option.
NIST	Sets CitOESQL to conform to the NIST interpretation of the SQL ANSI 92 entry level standard.
ODBCTRACE	ODBCTRACE=USER enables you to control ODBC tracing via odbcc.ini from which you can specify the file that the trace goes into.
ODBCV3	This directive causes an application to register itself as an ODBC Version 3 application.
ODBCVER	This directive causes an application to register itself as an ODBC Version 2, 3.x or 3.8 application.
OPTIMIZECURSORS	Optimizes memory consumption when using Oracle, PostgreSQL, DB2, or SQL Server JDBC providers. Also applies the same data integrity rules on all databases for embedded SQL cursors that use WITH HOLD and FOR UPDATE clauses for both DBMAN=ODBC and DBMAN=JDBC.
PARAMARRAY	If PARAMARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all input parameters.
PASS	The login to use to connect to the data source. This option works in conjunction with the INIT and/or CHECK options.
PICXBINARY	Enables programs to use PIC X(n) host variables to receive data from BINARY, VARBINARY, LONGVARBINARY columns in binary format without changing source to use SQL TYPE BINARY host variables.
PICXBINDING	Specifies the handling of fixed-length PIC X(n) host variables.

Directive	Description
PREFETCH	An application can use this directive to request that CitOESQL use block fetches for cursors. This can provide performance benefits, similar to array fetching, without having to change program logic. The performance benefit depends on the value of <i>n</i> and on whether the ODBC driver in use is already configured to use prefetching.
QUALFIX	Causes the CitOESQL preprocessor to append three characters to the name of the host variables when declaring them to SQL. This ensures problems caused by qualification (where two or more host variables have identical names when not qualified) are avoided but has the sideeffect that SQL error messages sometimes display the names of host variables with the three additional characters appended to them.
RESULTARRAY	If RESULTARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all output parameters.
SAVE-RETURN-CODE	Specifies whether or not to save and then restore RETURN-CODE.
STMTCACHE	The number of prepared SQL statements CitOESQL can cache such that the statements never again require preparation during a program run, thus improving performance.
TARGETDB	Set this directive if you want to optimize performance for a specific data source.
THREAD	Specifies the handling of threads with regard to connections.
TIME	Specifies an explicit time format to use when time values are returned from database time columns into character output host variables.
TIMEDELIM	Specifies a single character as the delimiter between the hour, minute, and second components to override the default delimiter determined by the TIME directive specification, or implicitly based on default ISO 8601 format ( <i>hh:mm:ss</i> ).
TRACELEVEL	Produces a statistical analysis of application behavior by tracing certain operations in native applications. The report produced by this directive provides better readability and is inherently more useful than a traditional ODBC trace.

<b>Directive</b>	<b>Description</b>
TRANSACTION	This directive provides CitOESQL with specifications for managing runtime transactions and, in some cases, enabling compile-time checking.
TSTAMPSEP	Specifies a single character to use as the separator between the date and time parts when datetime values are returned from database datetime columns into character output host variables.

Directive	Description
WHERECURRENT	Allows PostgreSQL and MySQL to accept updateable SELECT and CURSOR statements when no positioned UPDATEs or DELETEs are required.

## ALLOWNULLCHAR

Allows programs to use PIC X(*n*) host variables, and to select/insert/update the null character (x00) into CHAR columns without changing source to use SQL TYPE BINARY host variables.

### Syntax:

#### Important

Use ALLOWNULLCHAR with legacy code only, for example, code that uses FOR BIT DATA columns where the cost of converting to BINARY columns in the database and SQL TYPE IS BINARY host variables is prohibitive. Storing binary data in character columns is not a best practice and should be avoided wherever possible.

## [NO]ALLOWNULLCHAR

### Properties:

Default: NOALLOWNULLCHAR

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

### Comments:

- With ALLOWNULLCHAR, CitOESQL does not truncate the contents of the input host variable when the first embedded NULL character is encountered.
- With NOALLOWNULLCHAR, however, the database is presented a truncated value, including all characters up to the first embedded NULL.
- PostgreSQL does not support the embedding of the NULL character into CHAR columns; therefore, ALLOWNULLCHAR is not supported in PostgreSQL CitOESQL applications.

## ALLOWSERVERSELECT

Passes unrecognized EXEC SQL SELECT statements through to the server, thus enabling serverspecific behavior.

**Syntax:**

[NO]ALLOWSERVERSELECT

**Properties:**

Default: NOALLOWSERVERSELECT

**Scope:**

- Used at compile time: No
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

- NOALLOWSERVERSELECT gives an error when CitOESQL does not recognize an EXEC SQL SELECT statement.
- When ALLOWSERVERSELECT is set, CitOESQL simply passes unrecognized EXEC SQL SELECT statements through to the server, thus enabling server-specific behavior.

## ANSI92ENTRY

If this is set, CitOESQL conforms to the SQL ANSI'92 entry level standard.

**Syntax:**

[NO]ANSI92ENTRY

**Properties:**

Default: NOANS192ENTRY

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

- When ANSI92ENTRY is not specified or NOANSI92ENTRY is specified, CitOESQL does not set any error or warning conditions.
- When ANSI92ENTRY is specified and neither CHECKSINGLETON nor NOCHECKSINGLETON is specified (default), CitOESQL returns:
  - SQLCODE = -1 SQLSTATE = 21000 SQLWARN4 = W

In conformance with the ANSI-92 standard, CitOESQL does the following when ANSI92ENTRY is specified:

Sets the isolation level to serializable.

Closes non-HOLD cursors at the end of a transaction.

When an EXEC SQL FETCH statement is executed on an unopened cursor, returns SQLSTATE 24000. If ODBC3 is also set, returns SQLSTATE 07005.

When an EXEC SQL FETCH statement returns a null value but no indicator host variable has been supplied, returns SQLCODE -19425.

When an EXEC SQL OPEN statement is executed on an opened cursor, returns SQLCODE 19516 and SQLSTATE 07005.

## AUTOCOMMIT

Regulates an SQL connection's autocommit attribute.

### Syntax:

[NO]AUTOCOMMIT

### Properties:

Default: NOAUTOCOMMIT

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

### Comments:



- Because NOAUTOCOMMIT is used by default, CitOESQL default behavior explicitly turns off the SQL connection's autocommit attribute regardless of how the connection was created.
- For CitOESQL connections, NOAUTOCOMMIT allows your application to control local transactions with normal COMMIT and ROLLBACK statements.
- When AUTOCOMMIT is specified explicitly, the SQL connection's autocommit attribute is not altered, whether the connection was created by CitOESQL or by some other means. The result for CitOESQL connections is that AUTOCOMMIT commits every SQL statement in your application, operating in "autocommit mode." With external connections, the autocommit attribute is not modified, so your application continues to participate in the transaction managed externally.
- The SET AUTOCOMMIT embedded SQL statement overrides the AUTOCOMMIT compiler directive option.
- An application can programmatically control the autocommit setting for a connection by executing the EXEC SQL SET AUTOCOMMIT statement.
- An application in autocommit mode can start a local database transaction with the EXEC SQL BEGIN TRANSACTION statement. The transaction ends when the next COMMIT or ROLLBACK statement is executed.
- When a transaction ends, if the connection's autocommit attribute is on, the connection reverts to autocommit mode; otherwise a new local database transaction is started automatically.

## AUTOFETCH

Sets the AUTOFETCH attribute on SELECT statements. Compiling with this directive can help the performance of your application. It also serves as a primitive directive for the BEHAVIOR directive option.

### Note

Errors on a cursor OPEN are deferred until the subsequent FETCH when AUTOFETCH is enabled.

BEHAVIOR=OPTIMIZED enables AUTOFETCH

Errors on a cursor OPEN are deferred until the subsequent FETCH when AUTOFETCH is enabled.

### Syntax:

[NO]AUTOFETCH

### Properties:

Default: NOAUTOFETCH

### Scope:

- Used at compile time: No
- Behavior at run time: Source file

See CitOESQL Directives for more information.

## BEHAVIOR

Instructs CitOESQL to properly match your COBOL cursors with the database you are using, thus maximizing database cursor performance. BEHAVIOR makes use of multiple primitive directives and associates a default value for each primitive directive depending on the target DBMS. You can also override the default setting of any primitive directive.

### Note

If you don't specify any BEHAVIOR primitive directives, and you also accept the default primitive directive values listed in the relevant table in the *Default Primitive Directives* section later in this topic, the TARGETDB directive is not required; however, you must specify TARGETDB if you want to use different default values for any primitive directive.

Setting the BEHAVIOR directive can also affect the SQL types returned when using Dynamic SQL. See the DESCRIBE topic for more information.

### Syntax:

```
BEHAVIOR={OPTIMIZED \ UNOPTIMIZED \ ANSI \ MAINFRAME } [*primitivedirective*  
[*value*]]...
```

### Parameters:

OPTIMIZED - Enables CitOESQL to optimize ambiguous cursor declarations; enhances the access speed

UNOPTIMIZED - Provides backward compatibility with earlier Micro Focus products, where ambiguous cursors are both updateable and scrollable

ANSI - Enables CitOESQL to work like the ANSI standard; enhances the access speed

MAINFRAME - Synonym for OPTIMIZED

*primitivedirective* - Optional directives that enable the fine-tuning of BEHAVIOR

### Properties:

Default: BEHAVIOR=OPTIMIZED

### Scope:

- Used at compile time: Yes
- Behavior at run time: N/A

See CitOESQL Directives for more information.

**Default Primitive Directives:**

Table 1. Default Behavior Primitive Table for SQL Server

<b>BEHAVIOR = Primitive Default Values</b>	<b>OPTIMIZED, MAINFRAME</b>	<b>ANSI</b>
AUTOFETCH	ON	OFF
DEF_CURSOR	RO	UPD
RO_CURSOR	FORWARD or IC_FH	FORWARD or IC_FH
PF_RO_CURSOR	1	1
IC_FH_ISOLATION	UR	UR
UPD_CURSOR	DYNAMIC	DYNAMIC
PF_UPD_CURSOR	8	8
UPD_CONCURRENCY	LOCK	LOCK
ISOLATION	CR	CR

Table 2. Default Behavior Primitive Table for SQL Server

<b>BEHAVIOR = Primitive Default Values</b>	<b>OPTIMIZED, MAINFRAME</b>	<b>ANSI</b>
AUTOFETCH	ON	OFF
DEF_CURSOR	RO	UPD
RO_CURSOR	FORWARD	FORWARD
PF_RO_CURSOR	8	8
UPD_CURSOR	KEYSET	KEYSET
PF_UPD_CURSOR	8	8
UPD_CONCURRENCY	LOCK	LOCK
ISOLATION	CR	CR

<b>BEHAVIOR = Primitive Default Values</b>	<b>OPTIMIZED, MAINFRAME</b>	<b>ANSI</b>
AUTOFETCH	ON	OFF

You can fine tune the BEHAVIOR directive and override these default settings by providing alternate values for primitive directives. If you do this, you must set the TARGETDB directive.

**Comments:**

Setting this directive can also affect the SQL types returned when using Dynamic SQL. See the *DESCRIBE Statement* topic for more information.

OpenESQL sets default values for primitive directives depending on the value of BEHAVIOR and your target database. See the topic *Primitive Directives* for more information.

**Primitive Directives**

These directives work only with when you have your target database set and you have specified the BEHAVIOR directive as well. Specify them only when you want to override the default settings for BEHAVIOR. For example:

```
SQL (TARGETDB=INFORMIX BEHAVIOR=OPTIMIZED DEF_CURSOR=UPD)
```

- DEF\_CURSOR

This cursor directive specifies the default cursor type for ambiguous COBOL cursors, either read only or updateable. Use this directive when the DECLARE CURSOR statement has neither the FOR READONLY nor the FOR UPDATE OF clause.

- IC\_FH\_ISOLATION

This directive sets the transaction isolation for independent firehose cursor connections, enabling simultaneous access to firehose cursors.

- PF\_RO\_CURSOR

This is a prefetch directive that enables you to retrieve more than one read only record at a time from the DBMS with one client request.

- PF\_UPD\_CURSOR

This is a prefetch directive that enables you to retrieve more than one updateable record at a time from the DBMS with one client request.

- RO\_CURSOR

This directive determines what type of database cursor your COBOL read-only cursors use.

- UPD\_CONCURRENCY

This directive enables you to choose the level of concurrency for updateable COBOL cursors.

- UPD\_CURSOR

This directive determines what type of database cursor updateable COBOL cursors use.

## DEF\_CURSOR

This cursor directive specifies the default cursor type for ambiguous COBOL cursors, either read only or updateable. Use this directive when the DECLARE CURSOR statement has neither the FOR READONLY nor the FOR UPDATE OF clause.

### Syntax:

```
DEF_CURSOR={RO \ UPD}
```

### Parameters:

- RO Read Only
- UPD Updateable

### Scope:

- Used at compile time: No
- Behavior at run time: Source File

See CitOESQL Directives for more information.

**Comments:**

While the default when BEHAVIOR=ANSI is DEF\_CURSOR=UPD, if the original developer assumed RO, then use of UPD could adversely affect performance.

## IC\_FH\_ISOLATION

This directive sets the transaction isolation for independent firehose cursor connections, enabling simultaneous access to firehose cursors.

**Syntax:**

```
IC_FH_ISOLATION={UR \ CR \ RR \ SZ}
```

**Parameters:**

- UR Uncommitted Read
- CR Committed Read
- RR Repeatable Read
- SZ Serializable isolation in ODBC

**Scope**

- Used at compile time: No
- Behavior at run time: Source File

See CitOESQL Directives for more information.

## PF\_RO\_CURSOR

This is a prefetch directive that enables you to retrieve more than one read only record at a time from the DBMS with one client request.

**Syntax:**

```
PF_RO_CURSOR={*numberofrows* \ *bufferize*K \ *bufferize*B}
```

**Properties:**

Default for SQL Server connections: 8

**Parameters:**

- `numberofrows` - The number of rows to retrieve
- `bufferSizeK` - The size of the buffer you want to fill in kilobytes
- `bufferSizeB` - The size of the buffer you want to fill in bytes

**Scope:**

- Used at compile time: No
- Behavior at run time: Source File

See `CitOESQL Directives` for more information.

**Comments:**

`PF_RO_CURSOR` must be set after `TARGETDB` and `BEHAVIOR`

Does not work with `FORWARD` or `IC_FH` selection in `RO_CURSOR`

SQL Server read-only server cursors now use `PF_RO_CURSOR` consistently across all runtimes

## **PF\_UPD\_CURSOR**

This is a prefetch directive that enables you to retrieve more than one updateable record at a time from the DBMS with one client request.

**Syntax:**

```
PF_UPD_CURSOR={*numberofrows* \ *bufferSizeK \ *bufferSizeB}
```

**Parameters:**

`numberofrows` - The number of rows to retrieve

`bufferSizeK` - The size of the buffer you want to fill in kilobytes *bufferSizeB* The size of the buffer you want to fill in bytes

**Scope:**

- Used at compile time: No
- Behavior at run time: Source File

See `CitOESQL Directives` for more information.

**Comments:**

`PF_UPD_CURSOR` must be set after `TARGETDB` and `BEHAVIOR`

Does not work with `FORWARD` selection in `RO_CURSOR`

## **RO\_CURSOR**

This directive determines what type of database cursor your COBOL read-only cursors use.

**Syntax:**

```
RO_CURSOR={SCROLL \ FORWARD \ DYNAMIC \ KEYSET \ STATIC \ FF \ IC_FH}
```

**Parameters:**

FF - Fast Forward

IC\_FH - Independent Connection Fire Hose

**Scope:**

- Used at compile time: No
- Behavior at run time: Source File

See CitOESQL Directives for more information.

**Comments:**

SCROLL, FORWARD, DYNAMIC, and KEYSET are standard ODBC types.

FF and IC\_FH are cursors specific to MS SQL Server.

Under MS SQL Server, prefetching does not work with FORWARD and IC\_FH.

As its name indicates, an IC\_FH cursor has its own independent connection to MS SQL Server. Because of this, these cursors are sensitive to the isolation level and locking protocol of previous data access in the application program.

When using a SQL Server connection that supports MARS, MARS is automatically enabled and IC\_FH is converted to FORWARD.

## UPD\_CONCURRENCY

This directive enables you to choose the level of concurrency for updateable COBOL cursors.

**Syntax:**

```
UPD_CONCURRENCY={LOCK \ OPTIMISTIC \ OPTCC \ OPTCCVAL}
```

**Scope:**

- Used at compile time: No
- Behavior at run time: Source File

See CitOESQL Directives for more information.

**Comments:**



LOCK, OPTIMISTIC, OPTCC, and OPTCCVAL are standard ODBC types.

## UPD\_CURSOR

This directive determines what type of database cursor updateable COBOL cursors use.

### Syntax:

```
UPD_CURSOR={SCROLL \ FORWARD \ DYNAMIC \ KEYSET \ STATIC}
```

### Scope:

- Used at compile time: No
- Behavior at run time: Source File

See CitoESQL Directives for more information.

### Comments:

SCROLL, FORWARD, DYNAMIC, KEYSET, and STATIC are standard ODBC types.  
Prefetching does not work with FORWARD.

## CHECK

Sends each SQL statement to the database at compilation time.

### Syntax:

```
[NO]CHECK
```

### Properties:

Default NOCHECK

### Dependencies:

CHECK requires that you also:

Set the DB compiler directive option or identify the database using the DB2DBDFT environment variable.

If the connection to the database specified requires authentication, then the PASS compiler directive option is also required.

When using SQL(CHECK) with an ODBC connection to Oracle, you must have at least the same user privileges as are required to execute the SQL statements in the program.

### Scope:

- Used at compile time: Yes
- Behavior at run time: N/A

See CitOESQL Directives for more information.

**Comments:**

- Depending on your driver and/or DBMS and in certain circumstances, CHECK does not flag invalid SQL statements.
- You can specify the [NOCHECK], [ALSO CHECK], or [WITH CHECK] statement prefix to affect specific SQL statements. See *SQL Statement Prefixes for CHECK Directive* for complete information.

## CHECKDUPCURSOR

Instructs CitOESQL to determine if the cursor has been opened twice, and if so takes action accordingly.

**Syntax:**

```
[NO]CHECKDUPCURSOR
```

**Properties:**

Default NOCHECKDUPCURSOR

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

- If the cursor has been opened twice and the NOANSI92ENTRY directive has also been specified, CHECKDUPCURSOR returns SQLCODE -516 and sets SQLSTATE=24000.
- If the cursor has been opened twice and ANSI92ENTRY has not been specified, NOCHECKDUPCURSOR automatically closes the cursor and then re-opens it.

## CHECKSINGLETON

Instructs CitOESQL to check if singleton SELECTs return more than one row when executed.

**Syntax:**

## [NO]CHECKSINGLETON

### Properties:

Default None

### Returns:

DirectiveSQL	CODE returnsSQLSTATE	returns
None (default)*	+1	21000
CHECKSINGLETON	-811	21000
NOCHECKSINGLETON	0	21000

\*When ANSI92ENTRY is specified without CHECKSINGLETON or NOCHECKSINGLETON (default), SQLCODE returns -1 and SQLSTATE returns 21000. See ANSI92ENTRY for additional details.

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

### Comments:

CHECKSINGLETON is provided for DB2 compatibility.

NOCHECKSINGLETON is provided for applications that require SQLCODE 0.

For applications that require SQLCODE -1, specify ANSI92ENTRY without CHECKSINGLETON or NOCHECKSINGLETON.

CitOESQL returns the following SQLDA diagnostics when a singleton SELECT returns more than one row:

SQLCODES	QLSTATESQLWARN4	Flag
-811	21000	W
0	21000	space
+1	21000	W

SQLCODES	QLSTATESQLWARN4	Flag
-1	21000	W

## CLOSE\_ON\_COMMIT

Specifies whether to close cursors not defined WITH HOLD or leave them open for further fetches after a COMMIT.

### Syntax:

```
CLOSE_ON_COMMIT={YES \ NO}
```

### Parameters:

YES - Close cursors on COMMIT

NO - Leave cursors open on COMMIT

### Properties:

Default YES

### Scope:

- Used at compile time: Yes
- Behavior at run time: Process

See CitOESQL Directives for more information.

### Comments:

By default, meaning CLOSE\_ON\_COMMIT is not specified or CLOSE\_ON\_COMMIT=YES is specified, all cursors not declared WITH HOLD are closed after a COMMIT. CLOSE\_ON\_COMMIT works with the BEHAVIOR directive as follows:

- BEHAVIOR=UNOPTIMIZED\*  
CLOSE\_ON\_COMMIT is ignored. Cursor selection is least optimal. Cursor remains open after COMMIT.
- BEHAVIOR=OPTIMIZED or ANSI, CLOSE\_ON\_COMMIT=YES (default) Cursor selection is optimal.  
Cursor is closed after COMMIT.
- BEHAVIOR=OPTIMIZED or ANSI, CLOSE\_ON\_COMMIT=NO  
Cursor selection is optimal. Cursor remains open after COMMIT.

## CLOSE\_ON\_ROLLBACK

Specifies whether to close cursors or leave them open for further fetches after a ROLLBACK.

**Syntax:**

```
CLOSE_ON_ROLLBACK={YES \ NO}
```

**Parameters:**

- YES - Close cursors on ROLLBACK
- NO - Leave cursors open on ROLLBACK

**Properties:**

Default YES

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

By default, meaning CLOSE\_ON\_ROLLBACK is not specified or CLOSE\_ON\_ROLLBACK=YES is specified, all cursors are closed after a ROLLBACK.

CLOSE\_ON\_ROLLBACK works with the BEHAVIOR directive as follows:

- BEHAVIOR=UNOPTIMIZED  
CLOSE\_ON\_ROLLBACK is ignored. Cursor selection is least optimal. Cursor remains open after ROLLBACK.
- BEHAVIOR=OPTIMIZED or ANSI, CLOSE\_ON\_ROLLBACK=YES (default)  
Cursor selection is optimal. Cursor is closed after ROLLBACK.
- BEHAVIOR=OPTIMIZED or ANSI, CLOSE\_ON\_COMMIT=NO  
Cursor selection is optimal. Cursor remains open after ROLLBACK.

## CONNECTIONPOOL

Enables use of ODBC 3.0 connection pooling. When a connection is closed, the Driver Manager keeps it alive for a timeout period and saves the overhead of re-establishing a connection from scratch if the application re-opens an identical connection. ODBC allows you to choose between having a pooling for an ODBC environment or for each driver. See your ODBC documentation for details.

This option is only useful for applications that frequently open and close connections. Note that some environments, such as Microsoft Transaction Server (MTS), control connection pooling themselves. This option will probably improve the performance of ISAPI applications that are not running under MTS.

**Syntax:**

```
CONNECTIONPOOL={DRIVER \ ENVIRONMENT \ NONE}
```

**Properties:**

Default CONNECTIONPOOL=NONE.

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

## CURSORCASE

NOCURSORCASE means that cursor names are not case sensitive. CURSORCASE means that they are case sensitive.

**Syntax:**

```
[NO]CURSORCASE
```

**Properties:**

Default NOCURSORCASE

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

## DATE

Specifies the explicit date format to use when date values are returned from database date columns into character output host variables.

When used in addition to DETECTDATE, also specifies the explicit data format to recognize in character input host variables.

**Syntax:**

```
DATE={ODBC \ ISO \ USA \ EUR \ JIS}
```

### Parameters:

- ODBC - *yyyy-mm-dd* (ISO 8601 default format)
- ISO - *yyyy-mm-dd* (mainframe default format)
- USA - *mm/dd/yyyy*
- EUR - *dd.mm.yyyy*
- JIS - *yyyy-mm-dd*

### Properties:

Default ODBC (ISO 8601 default format)

### Dependencies:

For input host parameters, requires that the DETECTDATE SQL compiler directive option is also set.

### Scope:

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

### Comments:

DATE can be used with the DATEDELIM directive to specify an alternative delimiter that separates day, month, and year components.

DATE, with or without DATEDELIM, changes the display format of output host variables as specified.

When you specify both DATE and DETECTDATE, CitOESQL uses DATE (with or without DATEDELIM) to also recognize date values in your input host variables. See *DETECTDATE* for more information.

#### Important

If you do not specify an alternative format using DATE, CitOESQL returns your date columns using the ISO 8601 default format in your output character host variables, as specified in the *Properties* section of this topic. The same is also true of input character host variables when DETECTDATE is specified in addition to DATE.

## DATEDELIM

Specifies a single character as the delimiter between the year, month, and day components to override the default delimiter determined by the DATE directive specification, or implicitly based on default ISO 8601 format (*yyyy-mm-dd*).

The specified delimiter is used in character output host variables and, if DETECTDATE is also specified, in character input host variables.

**Syntax:**

```
DATEDELIM=*character*
```

**Properties:**

Default None

**Dependencies:**

None; however, DATEDELIM can be used with DATE to specify an alternative delimiter. See *DATE* for details.

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

- DATEDELIM set without DATE overrides the default ISO 8601 delimiter, a dash (-) character, for date values.
- DATEDELIM set with DATE overrides the default delimiter for the specified DATE parameter. For example, the default delimiter for DATE=USA is a forward slash character (/).

See the DATE and DETECTDATE sections along with the **CitOESQL Datetime Data Types Handling** section in the CitOESQL User Guide for more information.

## DB

Identifies a data source that defines database connection information.

**Syntax:**

```
DB=connection
```

NODB

**Parameter:**



- connection

The value for this parameter varies depending on the run time system:

ODBC Run-time - An ODBC DSN

JVM Managed Run-time - The name of a JNDI DataSource object

**Properties:**

Default NODB

**Dependencies:**

At compile time, DB is required when you set SQL(CHECK). At run time, DB is required when you set SQL(INIT).

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file See CitOESQL Directives for more information.

**Comments:**

The .int file generated is the same except for the TARGETDB number embedded in the file.

## DBMAN

Specifies the preprocessor to use. This directive is not required when compiling programs with CitOESQL.

**Syntax:**

```
DBMAN={ODBC}
```

**Parameters:**

ODBC For native code

**Properties:**

Default DBMAN=ODBC for native applications

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Process

## DECDEL

Specifies the decimal delimiter to use for decimal variables.

**Syntax:**

```
DATE={ODBC \ ISO \ USA \ EUR \ JIS}
```

**Parameters:**

PERIOD - Always use a period (.) as a decimal delimiter.

COMMA - Always use a comma (,) as a decimal delimiter.

LOCAL - Call **GetLocaleInfo** one time to get the decimal delimiter.

NODECDEL - Call **GetLocaleInfo** every time a decimal variable is referenced. Use this when your application dynamically changes its effective locale at run time.

**Properties:** Default - LOCAL

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

## DESCRIBEDTCHAR

When using dynamic SQL, described or prepared SQL statements with DATE, TIME, and DATETIME columns are suitable for PIC X(n) character host variables or DATE, TIME, and TIMESTAMP SQL TYPEs.

**Syntax:**

```
DESCRIBEDTCHAR
```

**Properties:**

Default None

**Dependencies:**

BEHAVIOR=OPTIMIZED automatically sets DESCRIBEDTCHAR.

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

For dynamic SQL, by default, CitOESQL expects DATE, TIME and TIMESTAMP columns to be placed into host variables of DATE, TIME, and TIMESTAMP-RECORD SQL TYPEs in ODBC format record structures.

When BEHAVIOR=OPTIMIZED or DESCRIBEDTCHAR is set, CitOESQL expects DATE, TIME, and DATETIME columns to be placed into PIC X(n) character host variables or DATE, TIME, and TIMESTAMP SQL TYPEs.

**DESCRIBEDTREC**

When using dynamic SQL, described or prepared SQL statements with DATE, TIME, and DATETIME, columns are suitable for the DATE, TIME, and TIMESTAMP-RECORD SQL TYPEs in ODBC format record structures.

**Syntax:**

```
DESCRIBEDTREC
```

**Properties:**

Default DESCRIBEDTREC

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

For dynamic SQL, by default, CitOESQL expects DATE, TIME and TIMESTAMP columns to be placed into host variables of DATE, TIME, and TIMESTAMP-RECORD SQL TYPEs in ODBC format record structures.

Use DESCRIBEDTREC to override BEHAVIOR=OPTIMIZED set by DESCRIBEDTCHAR.

**DESCRIBEVARCHAR49**

When using dynamic SQL, described or prepared SQL statements with VARCHAR, columns are suitable for VARCHAR host variables with level 49 sub-fields for length and data.

**Syntax:**

```
DESCRIBEVARCHAR49
```

**Properties:**

Default None

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

For dynamic SQL, by default, CitOESQL expects VARCHAR columns to be placed into PIC X host variables.

Use DESCRIBEVARCHAR49 to enable the use of VARCHAR host variables with level 49 sub-fields for length and data.

### DESCRIBEVARCHARPICX

When using dynamic SQL, described or prepared SQL statements with VARCHAR, columns are suitable for PIC X host variables.

**Syntax:**

```
DESCRIBEVARCHARPICX
```

**Properties:**

Default DESCRIBEVARCHARPICX

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

For dynamic SQL, by default, CitOESQL expects VARCHAR columns to be placed into PIC X host variables.

### DETECTDATE

Allows datetime values for PIC X character input host variables in an CitOESQL application to be in different formats than the standard ISO 8601 formats.

DETECTDATE enables you to specify alternative formats for input host variables. CitOESQL manages the translation between the format specified by DETECTDATE and the format recognized by your DBMS. This is done for each SQL call on both input and output.

For example, instead of the standard ISO formats, you might prefer to use EUR formats for date and time data.

### Important

Use DETECTDATE with extreme caution and only when absolutely necessary. Before using DETECTDATE, carefully review all options specified here, all of the information presented in the **CitOESQL Datetime Data Types Handling** section in the CitOESQL User Guide, and the information in the DATE, DATEDELIM, TIME, and TIMEDELIM topics to help you determine the best fit for your application.

#### Syntax:

```
[NO]DETECTDATE
```

```
DETECTDATE={CLIENT \ SQLTYPE \ SERVER \ PICX}
```

#### Parameters:

- CLIENT

Applies to DBMAN=ODBC only.

- Input Host Variables

For PIC X character input host variables, CitOESQL recognizes specified datetime formats and translates the data into the ISO 8601 format acceptable to your DBMS. Specified datetime formats for input host variables are:

- Date

ISO 8601 default, can be overridden by specifying the DATE directive and/or the DATEDELIM directive.

- Time

ISO 8601 default, can be overridden by specifying the TIME directive and/or the TIMEDELIM directive.

- Datetime

ISO 8601 default, can be overridden by specifying the TSTAMPSEP directive.

The dash character instructs CitOESQL to look for a specific set of delimiters, including a dash, a space, and a T. For example if you do not specify any alternative date or time

formats, and you set TSTAMPSEP to a dash character (-), CitOESQL recognizes the following formats in your input host variables:

```
yyyy-mm-dd-hh.mm.ss.ffffff
```

```
yyyy-mm-dd hh.mm.ss.ffffff
```

```
yyyy-mm-dd hh:mm:ss.ffffff
```

```
yyyy-mm-ddThh.mm.ss.ffffff
```

```
yyyy-mm-ddThh:mm:ss.ffffff
```

All other characters instruct CitOESQL to search for that specific character between each date and time format, where the date portion is delimited by a dash character (-) and the time portion is delimited by a colon (:).

- Output Host Variables\*

For PIC X output character host variables, CitOESQL returns the data for output host variables in the following datetime formats:

- Date

ISO 8601 default, can be overridden by specifying the DATE directive and/or the DATEDELIM directive.

- Time

ISO 8601 default, can be overridden by specifying the TIME directive and/or the TIMEDELIM directive.

- Timestamp

ISO 8601 default, can be overridden by specifying the TSTAMPSEP directive. When TSTAMPSEP is set to a dash (-) character (TSTAMPSEP="-"), CitOESQL returns datetime columns in the following format: *yyyy-mm-dd-hh.mm.ss.ffffff*

- SERVER

Applies to DBMAN=ODBC only.

Issues SQLDescribeParam calls to the DBMS to identify which input and output host variables are associated with specific character or datetime columns in the database. Host variables associated with datetime columns are translated with the datetime formats listed in CLIENT. Character columns are not translated.

- PICX

Identical to the CLIENT option. Provided for backward compatibility.

### Properties:

Defaults: When DETECTDATE is not specified, the default is: NODETECTDATE When DETECTDATE with no argument is specified, the default is: DETECTDATE=CLIENT

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

- Use DETECTDATE=CLIENT only when your CitOESQL application does not use character columns in the database to store data that is a match to any of the alternative formats for the date, time or datetime fields in your DBMS.
- Use DETECTDATE=SERVER when your CitOESQL application uses datetime values in both datetime and character columns of tables in your database.

 **Note**

DETECTDATE=SERVER does cause CitOESQL to perform extra overhead processing on each applicable SQL statement.

- Datetime formats are defined implicitly unless you set them explicitly by specifying the DATE, DATEDELIM, TIME, and/or TIMEDELIM directives. See the DATE, TIME, DATEDELIM and TIMEDELIM sections for more information.

**ERRORMAP**

This directive specifies the name of the error map file to use, and enables SQL error mapping.

**Syntax:**

```
ERRORMAP=*map-name*
```

**Parameter:**

*map-name* - The prefix of the error map filename (no file extension)

**Properties:**

Default None

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.



## IGNORESCHEMAERRORS

Suppresses compile-time errors resulting from missing schema objects.

### Syntax:

```
[NO] IGNORESCHEMAERRORS
```

### Properties:

Default NOIGNORESCHEMAERRORS

### Dependencies:

To use IGNORESCHEMAERRORS, you must also set CHECK.

### Scope:

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

### Comments:

By setting both IGNORESCHEMAERRORS and CHECK, CitOESQL checks for SQL syntax errors without reference to the database schema. This can be helpful when planning an application migration, as you can use the COBOL compiler and CitOESQL to identify the statements that require remediation without having to first migrate the database schema. It also provides enhanced CitOESQL syntax checking during the development phase.

## INIT

When set without parameters, the preprocessor automatically generates code to make the connection to the database. When set with the PROT parameter, protects the database when an application terminates abnormally.

### Syntax:

```
INIT [= { [PROT\]P } ]
```

```
NOINIT
```

### Parameters:

- INIT - Generates a CONNECT statement in shared mode and an exit handling code
- INIT=PROT - Generates exit handling code only.
- INIT=P
- NOINIT - No code generation whatsoever.

**Properties:**

Default NOINIT

**Dependencies:**

If your INIT call generates a connection, you can use it with the DB and PASS SQL compiler directive options.

**Comments:**

- For the following reasons, we strongly recommend that you consider placing an EXEC SQL CONNECT statement into your code instead of using INIT, INIT=S or INIT=X :

INIT stores user credentials in your code, so its use can raise security concerns.

If the DBMS vendor were to change the underlying APIs used to implement a database connection, this could cause compatibility problems when using INIT.

- Set the INIT directive, with or without PROT, only once for each application. Do not set INIT for SQL programs called by other SQL programs. Instead, specify the INIT option for the first SQL program executed in a run unit. Compiling more than one module in an application with the INIT option could cause your program to terminate abnormally.
- INIT is ignored when used within an OO program.

**ISOLATION**

This directive specifies the isolation level that CitOESQL uses as a connection attribute. It also serves as a primitive directive for the BEHAVIOR directive option.

**Syntax:**

```
ISOLATION={UR \ CR \ RR \ SZ}
```

**Parameters:**

UR - Uncommitted Read

CR - Committed Read

RR - Repeatable Read

SZ - Serializable isolation in ODBC

**Properties:**

Default: CR

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

## NIST

Sets CitOESQL to conform to the NIST interpretation of the SQL ANSI 92 entry level standard.

**Syntax:**

```
[NO]NIST
```

**Properties:**

Default: NONIST

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

## ODBCTRACE

ODBCTRACE=USER enables you to control ODBC tracing via `odbc.ini` from which you can specify the file that the trace goes into.

ALWAYS lets you control ODBC tracing via a directive, which is more convenient from within the IDE. ALWAYS generates the trace into `MFSQLTRACE.LOG` in the current directory, regardless of the settings in `odbc.ini`.

NEVER means that the application will never be traced and overrides `odbc.ini`. As ODBC trace files can contain sensitive information, use NEVER in production applications in secure environments. For more information see the database driver documentation.

**Syntax:**

```
ODBCTRACE={ALWAYS \ NEVER \ USER}
```

**Properties:**

Default: ODBCTRACE=USER.

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**ODBCV3**

This directive causes an application to register itself as an ODBC Version 3 application.

**Syntax:**

```
[NO]ODBCV3
```

**Properties:**

Default: NOODBCV3

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

If ODBCV3 is not specified, the application registers itself as an ODBC Version 2 application. There can be a small performance benefit in registering as an ODBC 3 application. However, registering as an ODBC Version 3 application can result in error and warning conditions returning different values for SQLCODE and SQLSTATE.

We recommend that you use this directive with care.

ODBCV3 is an alias for ODBCVER=38.

**ODBCVER**

This directive causes an application to register itself as an ODBC Version 2, 3.x or 3.8 application.

**Syntax:**

```
ODBCVER={20 \ 30 \ 38}
```

**Properties:**

Default: ODBCVER=20

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

If ODBCVER is not specified, the application registers itself as an ODBC version 2 application.

There can be a small performance benefit in registering as an ODBC version 3 or 3.8 application. However, registering as an ODBC version 3 or 3.8 application can result in error and warning conditions returning different values for SQLCODE and SQLSTATE.

We recommend that you use this directive with care.

ODBCV3 is an alias for ODBCVER=38.

**OPTIMIZECURSORS**

Optimizes memory consumption. Also applies the same data integrity rules on all databases for embedded SQL cursors that use WITH HOLD and FOR UPDATE clauses.

**Syntax:**

```
OPTIMIZECURSORS={YES\NO}
```

**Parameters:**

- YES - Optimize cursors. This can result in considerable memory savings and performance gains for large results sets and ensures that SQL cursors that use WITH HOLD and FOR UPDATE clauses have appropriate database locks when positioned updates/deletes occur.
- NO - Provided for backward compatibility.

**Properties:**

Default: YES

**Dependencies:**

To use OPTIMIZECURSORS, you must set DBMAN to ODBC explicitly, or by setting an SQL compiler directive option that sets DBMAN to ODBC implicitly.

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

## PARAMARRAY

If PARAMARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all input parameters.

### Syntax:

[NO]PARAMARRAY

### Properties:

Default: PARAMARRAY.

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

## PASS

The login to use to connect to the data source. This option works in conjunction with the INIT and/or CHECK options.

### Syntax:

```
PASS={*password* \ *userid.password*}
```

```
NOPASS
```

### Properties:

Default: NOPASS


### Scope:

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

## PICXBINARY

Enables programs to use PIC X(*n*) host variables to receive data from BINARY, VARBINARY, LONGVARBINARY columns in binary format without changing source to use SQL TYPE BINARY host variables.

 **Note**

Applications should use SQL TYPE BINARY host variables instead when dealing with BINARY columns data.

**Syntax:**

```
[NO]PICXBINARY
```

**Properties:**

Default: NOPICXBINARY

**Scope:**

- Used at compile time: No
- Behavior at run time: Source file

See CitOESQL Directives for more information.

## PICXBINDING

Specifies the handling of fixed-length PIC X(*n*) host variables.

**Syntax:**

```
PICXBINDING={DEFAULT \ PAD \ TRIM \ TRIMALL \ FIXED \ VARIABLE}
```

**Parameters:**

**DEFAULT**

- SQL Server - SBCS locales

Always passes PIC X(*n*) host variables to SQL Server as fixed-length data, and trims trailing spaces. The ODBC driver pads the data with spaces before presenting it to SQL Server.

- All other DBMSs in all locales

Always passes PIC X(*n*) host variables to the DBMS as fixed-length data, and trims trailing spaces. If the PIC X(*n*) host variable is all spaces, present one space to the DBMS.

**PAD or FIXED**

Always preserves trailing spaces for all DBMSs in all locales and passes the data to the DBMS as fixed-length data.

**TRIM or VARIABLE**

Always trims trailing spaces for all DBMSs in all locales. If the PIC X(*n*) host variable is all spaces, presents one space to the DBMS and passes the data to the DBMS as variable-length data.

**TRIMALL**

Always trims trailing spaces for all DBMSs in all locales. If the PIC X(*n*) host variable is all spaces, presents an empty string to the DBMS and passes the data to the DBMS as variablelength data.

**Properties:**

Default: DEFAULT

**Dependencies:**

ALLOWNULLCHAR is compatible with PICXBINDING only when PICXBINDING is omitted (meaning it is set to DEFAULT by default) or explicitly set to DEFAULT. Because PICXBINDING is designed to be used with PIC X(*n*) host variables that contain character data, not embedded binary data such as the NULL character ('\0'), an attempt to use ALLOWNULLCHAR when PICXBINDING is set to VARIABLE or FIXED could return unpredictable results.

**Scope:**

- Used at compile time: No
- Behavior at run time: Source file

See CitoESQL Directives for more information.

**Comments:**



For PIC X(*n*) host variables that contain all spaces, consider the following effects of PICXBINDING parameters when inserting into VARCHAR columns:

- DEFAULT with SQL Server inserts a space-padded column
- DEFAULT with all non-SQL Server DBMSs inserts one space into the column
- PAD with all DBMSs inserts a space-padded column
- TRIM with all DBMSs inserts one space into the column
- TRIMALL with Oracle inserts a NULL value into the column
- TRIMALL with all non-Oracle DBMSs inserts an empty string into the column

## PREFETCH

An application can use this directive to request that CitoESQL use block fetches for cursors. This can provide performance benefits similar to array fetching, without having to change program logic. The performance benefit depends on the value of *n* and on whether the ODBC driver in use is already configured to use prefetching.

If *n* is less than 1000, it controls the number of rows to be fetched per batch and the same number of rows is fetched for all cursors. If *n* is greater than or equal to 1000, it sets the size of the prefetch buffer for each cursor. All cursors will have the same buffer size but the number of rows prefetched will depend on the overall size of the row returned by the query for each cursor.

When PREFETCH=*n* is used with Microsoft SQL Server, AUTOFETCH is also used for read only cursors. Cursors which are not read only are forced to be keyset cursors and can be used for positioned updates. PREFETCH=*n* is only supported with DB2, Oracle and Microsoft SQL Server.

### Syntax:

```
PREFETCH=*n*
```

### Properties:

Default: PREFETCH=8

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitoESQL Directives for more information.

### Comments:

Much of the functionality provided by PREFETCH is now incorporated into the functionality of the BEHAVIOR SQL compiler directive option. As a result, PREFETCH is likely to be deprecated in a future release.

## QUALFIX

Causes the CitOESQL preprocessor to append three characters to the name of the host variables when declaring them to SQL. This ensures problems caused by qualification (where two or more host variables have identical names when not qualified) are avoided but has the side-effect that SQL error messages sometimes display the names of host variables with the three additional characters appended to them.

### Syntax:

```
[NO]QUALFIX
```

### Properties:

Default: NOQUALFIX

### Scope:

- Used at compile time: Yes
- Behavior at run time: N/A

See CitOESQL Directives for more information.

## RESULTARRAY

If RESULTARRAY is set, ODBC array binding is used, if it is supported by the ODBC driver, for all output parameters.

### Syntax:

```
[NO]RESULTARRAY
```

### Properties:

Default: RESULTARRAY.

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

## SAVE-RETURN-CODE

Specifies whether or not to save and then restore RETURN-CODE.

**Syntax:**

```
[NO] SAVE-RETURN-CODE
```

**Properties:**

Default: SAVE-RETURN-CODE

**Scope:**

- Used at compile time: Yes
- Behavior at run time: NA

See CitOESQL Directives for more information.

**Comments:**

Use NOSAVE-RETURN-CODE if your COBOL dialect does not recognize the COBOL special register RETURN-CODE.

## STMTCACHE

The number of prepared SQL statements CitOESQL can cache such that the statements never again require preparation during a program run, thus improving performance.

**Syntax:**

```
STMTCACHE=*n*
```

**Parameter:**

*n* Any number; however we recommend that you set this to a number between 20 and 300

**Properties:**

Default: STMTCACHE=20

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

Exercise caution with this directive as STMTCACHE is a trade off between performance and memory use.

## TARGETDB

Set this directive if you want to optimize performance for a specific data source.

### Syntax:

```
TARGETDB={MSSQLSERVER \ ORACLE \ INFORMIX \  
          SYBASE \ DB2 \ ORACLE7 \ POSTGRESQL}  
NOTARGETDB
```

### Properties:

Default: NOTARGETDB

### Scope:

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

### Comments:

With the vast majority of databases, the FOR READ ONLY clause has no effect on the access plan that the database server generates for a query. This, in combination with the fact that FOR READ ONLY syntax is not supported by most servers, CitOESQL removes the generated FOR READ ONLY clause at compile time unless TARGETDB is set to DB2. Removing this clause can facilitate code migration between different database servers and can also facilitate the incorporation of code that works with multiple types of database servers.

## THREAD

Specifies the handling of threads with regard to connections.

### Syntax:

```
THREAD={SHARE \ ISOLATE}
```

**Parameters:**

**SHARE**

All SQL connections, cursors, etc. in an application are shared by all threads. For example, if you have a hard-coded CONNECT statement and thread 1 executes it and then thread 2 executes it, thread 2 gets an error because the connection is already open.

**ISOLATE**

ODBC only. All connections, cursors, etc. are local to the thread that creates them. This is required for multi-threaded application server environments

**Properties:**

Default: SHARE

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**TIME**

Specifies an explicit time format to use when time values are returned from database time columns into character output host variables.

When used in addition to DETECTDATE, specifies the explicit time format to recognize in character input host variables.

**Syntax:**

```
TIME={ODBC \ ISO \ USA \ EUR \ JIS}
```

**Parameters:**

- ODBC hh:mm:ss (ISO 8601 default format)
- ISO hh.mm.ss (mainframe default format)
- USA hh:mm (AM \ PM)
- EUR hh.mm.ss
- JIS hh:mm:ss

**Properties:**

Default: ODBC (ISO 8601 default format)

**Dependencies:**

For input host parameters, requires that the DETECTDATE SQL compiler directive is also set.

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

**Comments:**

- TIME can be used with the TIMEDELIM directive to specify an alternative delimiter that separates hour, minute, and second components.
- TIME, with or without TIMEDELIM, changes the display format of output host variables as specified.
- When you specify both TIME and DETECTDATE, CitOESQL uses TIME (with or without TIMEDELIM) to also recognize time values in your input host variables. See *DETECTDATE* for more information. The following apply to USA format:

Minutes can be omitted. For example, 1 PM is equivalent to 1:00 PM.

AM and PM are not case sensitive.

There must be a single blank before AM or PM.

The hour must not be greater than 12 and cannot be 0 except for the special case of 00:00 AM.

**TIMEDELIM**

Specifies a single character as the delimiter between the hour, minute, and second components to override the default delimiter determined by the TIME directive specification, or implicitly based on default ISO 8601 format ( `hh:mm:ss` ).

**Syntax:**

```
TIMEDELIM=*character*
```

**Properties:**

Default: None

**Dependencies:**

For input host parameters, requires that the DETECTDATE SQL compiler directive is also set.

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file See CitOESQL Directives for more information.

**Comments:**

- TIMEDELIM set without TIME overrides the default ISO 8601 delimiter, a colon (:), for time values.
- TIMEDELIM set with TIME overrides the default delimiter for the specified TIME parameter. For example, the default delimiter for TIME=EUR is a dot character (.).
- See the TIME and DETECTDATE sections along with the **CitOESQL Datetime Data Types Handling** section in the CitOESQL User Guide for more information.

## TRACELEVEL

Produces a statistical analysis of application behavior by tracing certain operations in native applications. The report produced by this directive provides better readability and is inherently more useful than a traditional ODBC trace.

The statistical analysis information is written to a logfile named `OpenESQLTrace.processID.log`, which is created the first time you use TRACELEVEL. With each subsequent use of TRACELEVEL, tracing information is appended to the end of the file. A separator record is written at the end of each trace to help identify different traces.

CitOESQL creates the log files under the directory where the application is located. If file/directory permissions prevent file creation in that location, CitOESQL creates the log files under the directory referenced in the %TEMP% environment variable.

All trace records contain the elapsed run time, accurate to one microsecond.

**Syntax:**

```
TRACELEVEL={T \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ D}
```

**Scope:**

- Used at compile time: No
- Behavior at run time: Process

See CitOESQL Directives for more information.

Comments:

### TRACELEVEL=T

When TRACELEVEL=T, the following information is written to the trace file:

BEGIN - traces main SQL directives

END - indicates end of run

DIRECTIVES - traces per compilation unit directives the first time a compilation unit is encountered at run time.

### TRACELEVEL=1

When TRACELEVEL=1, the following information is written to the trace file in addition to the information written when you specify TRACELEVEL=T:

PREPARE - identifies the original source code when a statement is prepared.

DISPOSE - provides summary information for overall statement usage when a statement is removed from the prepared statement cache at disconnect time

FLUSH - provides summary information for overall statement usage when a statement is flushed from the cache usually due to a cache overflow

### TRACELEVEL=2

When TRACELEVEL=2, the following information is written to the trace file in addition to the information written when you specify TRACELEVEL=1:

```
- OPEN
- EXECUTE - provides the number of rows selected, inserted, or updated
- EXEC_IMMED EXECUTE - provides the number of rows selected, inserted, or updated
- ODBC_CLOSE - provides summary information for the current cursor use
- STMT_CHANGED - reports new concurrency and scroll option settings when the ODBC driver uses different settings than those requested by
CitOESQL.
```

### TRACELEVEL=3

When TRACELEVEL=3, the following information is written to the trace file in addition to the information written when you specify TRACELEVEL=2:

```
- ODBC_FETCH - provides the number of rows fetched
- COBOL_FETCH - provides the number of rows returned to the COBOL application
```

### TRACELEVEL=4

When TRACELEVEL=4, the following information is written to the trace file in addition to the information written when you specify TRACELEVEL=3:

```
- EXEC_SQL_BEGIN
- EXEC_SQL_END
```



## TRACELEVEL=5

When TRACELEVEL=5, the following ODBC API call information is written to the trace file in addition to the information written when you specify TRACELEVEL=4:

```
- ODBC_CALL_START  
- ODBC_CALL_END
```

## TRACELEVEL=6

When TRACELEVEL= 6, only the following information is written to the trace file:

```
- ODBC_CALL_START  
- ODBC_CALL_END
```

## TRACELEVEL=D

When TRACELEVEL=D, a reduced version of TRACELEVEL=4 is produced for debugging purposes. It contains only the following information:

```
- BEGIN  
- END  
- EXEC_SQL_BEGIN  
- EXEC_SQL_END  
- DIRECTIVES
```

## TRANSACTION

This directive provides CitOESQL with specifications for managing run-time transactions and, in some cases, enabling compile-time checking.

### Syntax:

```
TRANSACTION={GLOBAL \ LOCAL \ MIXED \ AUTO}
```

### Parameters:

- GLOBAL - Manages transactions via a distributed transaction manager, such as a Java application server, and its external SQL connection. Checks for the following statements that are not compatible with Java application server applications: BEGIN TRANSACTION COMMIT CONNECT DISCONNECT ROLLBACK SET AUTOCOMMIT SET CONNECT SET TRANSACTION [ISOLATION] Reports an error for each incompatible statement found.
- LOCAL - Manages transactions via a data source and its CitOESQL connection.
- MIXED - Manages transactions via a distributed transaction manager (similar to GLOBAL), but does not perform compile-time checking. This allows EXEC SQL CONNECT statements to create local transactions with an CitOESQL connection in addition to the connection provided by the distributed transaction manager.
- AUTO - Use this option when you want to AUTOCOMMIT each statement executed on an CitOESQL connection:

An application can programmatically control the autocommit setting for a connection by executing the EXEC SQL SET AUTOCOMMIT statement

An application in autocommit mode can start a local database transaction with the EXEC SQL BEGIN TRANSACTION statement. The transaction ends when the next COMMIT or ROLLBACK statement is executed

When a transaction ends, if the connection's autocommit attribute is on, the connection reverts to autocommit mode; otherwise a new local database transaction is started automatically.

#### Properties:

Default: TRANSACTION=LOCAL

#### Scope:

- Used at compile time: Yes
- Behavior at run time: N/A

See CitOESQL Directives for more information.

### TSTAMPSEP

Specifies a single character to use as the separator between the date and time parts when datetime values are returned from database datetime columns into character output host variables.

When used in addition to DETECTDATE, specifies the explicit datetime format to recognize in character input host variables.

#### Syntax:

```
TSTAMPSEP= ' *character*
```

## Parameters:

- Character

Any single character, including:

- (dash character)

When TSTAMPSEP='-' and DETECTDATE is not specified

Character output host variables are returned in the following format: `yyyy-mm-dd-hh.mm.ss.ffffff`

When TSTAMPSEP='-' and DETECTDATE is specified\*\*

The dash character instructs CitOESQL to look for a specific set of delimiters, including a dash, a space, and a T. For example, if you do not specify any alternative date or time formats, and you set TSTAMPSEP to a dash character (-), CitOESQL recognizes the following formats in your input host variables:

`yyyy-mm-dd-hh.mm.ss.ffffff`

`yyyy-mm-dd hh.mm.ss.ffffff`

`yyyy-mm-dd hh:mm:ss.ffffff`

`yyyy-mm-ddThh.mm.ss.ffffff`

- `yyyy-mm-ddThh:mm:ss.ffffff`

- Any other character

When DETECTDATE is not also specified:

Character output host variables are returned in the following format: `yyyy-mm-ddChh:mm:ss.ffffff`

Where C is any character except a dash (-) character, and the number of fractional seconds is platform dependent.

When DETECTDATE is also specified

Character input host variables are returned and scanned using the following format:

`yyyy-mm-ddChh:mm:ss.ffffff`

Where C is any character except a dash (-) character, and the number of fractional seconds is platform dependent.

## Properties:

Default: Space character (ISO 8601 default)

## Scope:

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

TSTAMPSEP changes the display format of output host variables as specified.

You can use TSTAMPSEP directive to override the delimiter used in the output format to separate date and time components.

## WHERECURRENT

Allows PostgreSQL and MySQL to accept updateable SELECT and CURSOR statements when no positioned UPDATES or DELETES are required.

**Syntax:**

```
[NO]WHERECURRENT
```

**Properties:**

Default: WHERECURRENT

**Scope:**

- Used at compile time: Yes
- Behavior at run time: Source file

See CitOESQL Directives for more information.

**Comments:**

- For PostgreSQL, depending on the table definition, the required pseudo column (oid) on a positioned UPDATE or DELETE might be missing.
- For MySQL, depending on the table definition, the required pseudo column (\_rowid) on a positioned UPDATE or DELETE might be missing.
- If positioned UPDATES and DELETES are required, you might need to change your PostgreSQL or MySQL table definition to expose the appropriate pseudo column and make it available.
- When positioned UPDATES and DELETES are not required, use NOWHERECURRENT.

## 5.3.1 SQL Data Types

---

## ODBC SQL/COBOL Data Type Mappings

The following table shows the mappings used by CitOESQL when converting between ODBC SQL and COBOL data types.

ODBC SQL Type	COBOL Picture	Notes
SQL_CHAR( <i>n</i> ) <sup>1</sup>	PIC X( <i>n</i> )	
SQL_NCHAR( <i>n</i> ) <sup>1</sup>	PIC X( <i>n</i> ) or PIC N( <i>n</i> )	
SQL_VARCHAR( <i>n</i> ) <sup>1</sup>	PIC X( <i>n</i> )	
SQL_NVARCHAR( <i>n</i> ) <sup>1</sup>	PIC X( <i>n</i> ) or PIC N( <i>n</i> )	
SQL_LONGVARCHAR <sup>1</sup>	PIC X( <i>max</i> ) or SQL TYPE LONG- VARCHAR( <i>max</i> )	
SQL_NTEXT <sup>1</sup>	PIC X( <i>max</i> ) or PIC N( <i>max</i> )	
SQL_DECIMAL( <i>p</i> , <i>s</i> ) or SQL_NUMERIC( <i>p</i> , <i>s</i> )	PIC S9( <i>p-s</i> )V9( <i>s</i> ) COMP-3	<i>p</i> = precision (total number of digits). <i>s</i> = scale (number of digits after the decimal point). CitOESQL doesn't support using unsigned packed decimal host variables.
SQL_SMALLINT	PIC S9(4) COMP-5	
SQL_INTEGER	PIC S9(9) COMP-5	
SQL_REAL	COMP-1	
SQL_FLOAT	COMP-2	
SQL_DOUBLE	COMP-2	
SQL_BIT	PIC S9(4) COMP-5	
SQL_TINYINT	PIC S9(4) COMP-5	
SQL_BIGINT	PIC S9(18) COMP-3	
SQL_BINARY( <i>n</i> )	PIC X( <i>n</i> ) or SQL TYPE BINARY( <i>n</i> )\	

ODBC SQL Type	COBOL Picture	Notes
SQL_VARBINARY( <i>n</i> )	PIC X( <i>n</i> ) ) or SQL TYPE VARBINARY( <i>n</i> )	
SQL_LONVARBINARY	PIC X( <i>max</i> ) or SQL TYPE LONG- VARBINARY( <i>max</i> )	
SQL_DATE or SQL_TYPE_DATE	PIC X(10) ) or SQL TYPE DATE	yyyy-mm-dd
SQL_TIME or SQL_TYPE_TIME or SQL_SS_TIME2	PIC X(8) or SQL TYPE TIME or SQL TYPE TIME-RECORD	hh:mm:ss
SQL_TIMESTAMP or SQL_TYPE_TIMESTAMP	PIC X(29) or SQL TYPE TIMESTAMP or SQL TYPE TIMESTAMP- RECORD	yyyy-mm- ddhh:mm:ss.ffffff
SQL_SS_TIMESTAMPOFFSET	PIC X(34)	yyyy-mm- ddhh:mm:ss.fffff +/-hh:mm

## SQL Data Types

Integer Data Types

Character Data Types

Numeric Data Types

Binary Data Types

Date and Time Data Types

## Miscellaneous Data Types

### INTEGER DATA TYPES

Small Integer: A small integer (SMALLINT) is a 2-byte integer SQL data type.

#### Host Variable Formats:

##### CITOESQL

```
01 shortint1 PIC S9(4) COMP.  
01 shortint2 PIC S9(4) COMP-4.  
01 shortint3 PIC X(2) COMP-5.  
01 shortint4 PIC S9(4) COMP-5.  
01 shortint5 PIC X(2) COMP-X.  
01 shortint6 PIC 9(4) COMP-X.  
01 shortint7 PIC S9(4) BINARY.
```

### INTEGER

An integer (INT) is a 4-byte integer SQL data type.

#### Host Variable Formats:

##### CITOESQL

```
All of the following definitions are valid for host variables to map directly onto the INT data type.  
01 longint1 PIC S9(9) COMP.  
01 longint2 PIC X(4) COMP-5.  
01 longint3 PIC S9(9) COMP-5.  
01 longint4 PIC X(4) COMP-X.  
01 longint5 PIC 9(9) COMP-X. 01 longint6 PIC S9(9) BINARY.
```

For the most efficient access, we recommend that you declare integers as COMP-5.

### BIG INTEGER

A big integer (BIGINT) is an 8-byte integer SQL data type.

#### Host Variable Formats: CITOESQL

```
01 bigint1 PIC S9(18) COMP-3.  
01 bigint2 PIC S9(18) COMP-5.  
01 bigint3 PIC X(8) COMP-5.  
01 bigint4 PIC X(8) COMP-X.
```

In non-COBOL applications, a BIGINT data type can hold a value larger than PIC S9(18). If you define your host variable for a COBOL data time with a value larger than S9(18), your data might be truncated.

## Character Data Types

### FIXED-LENGTH CHARACTER STRINGS

Fixed-length character strings (CHAR) are SQL data types with a driver-defined maximum length. They are declared in COBOL as:

`PIC X(*n*)` where  $n$  is an integer between 1 and the maximum length.

#### Host Variable Formats:

##### CITUESQL

```
01 char-field1 PIC X(5).  
01 char-field2 PIC X(254).
```

CitUESQL trims trailing spaces from input parameters before sending them to the database server. Trimming the trailing spaces can improve performance when comparing CHAR and VARCHAR values.

#### Important

For CitUESQL, the database server pads the value with spaces as necessary. When space padding is required for a host variable used in an expression, use an explicit SQL CAST function to ensure that the server converts the host variable to the required data type.

```
01 char-field3 SQL TYPE IS CHAR(200).
```

The `char-field3` format uses the CHAR SQL TYPE.

### VARIABLE-LENGTH CHARACTER STRINGS

Variable-length character strings (VARCHAR and LONGVARCHAR) are SQL data types with a variable maximum length.

#### Host Variable Formats:

##### CITUESQL

```
01 varchar1.  
49 varchar1-len PIC 9(4) COMP.  
49 varchar1-data PIC X(200).  
01 longvarchar1.  
49 longvarchar1-len PIC 9(4) COMP.  
49 longvarchar1-data PIC X(30000).  
01 clob1 SQL TYPE IS CLOB(32K).
```



The level number for group items containing only two elementary items must be 49. The first item is a 2-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The length field can be signed or unsigned. The second item is a PIC X(*n*) data type, where *n* is an integer representing the length of the field that holds the data.

SQL statements must reference the group name.

If the data being copied to a SQL CHAR, VARCHAR or LONG VARCHAR data type is longer than the defined length, then the data is truncated and the SQLWARN1 flag in the SQLCA data structure is set. If the data is smaller than the defined length, a receiving CHAR data type might be padded with blanks.

The **glob1** format uses the CLOB SQL TYPE.

In addition to the above definitions, the following definitions are also valid for CITOESQL:

```
01 varchar2 PIC X(20) VARYING.  
01 varchar3 SQL TYPE IS CHAR-VARYING(200).  
01 longvarchar1 SQL TYPE IS LONG-VARCHAR(50000).
```

The **varchar3** format uses the CHAR-VARYING SQL TYPE.

The **longvarchar1** format uses the LONG-VARCHAR SQL TYPE.

## LARGE CHARACTER STRINGS (CLOB)

Large character strings (CLOB) enable you to store large amounts of data in columns.

### Host Variable Formats:

The level number for group items containing only two elementary items must be 49. The first item is a 4-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The length field can be signed or unsigned. The second item is a PIC X(*n*) data type, where *n* is an integer representing the length of the field that holds the data.

SQL statements must reference the group name.

## UNICODE CHARACTER STRINGS

Unicode character strings (UNI) are SQL data types similar to fixed-length character strings, but are encoded using UTF-16 characters instead of single- or mixed-byte characters.

### Host Variable Formats:

CITOESQL

```
03 uni-field1 PIC N(X) USAGE NATIONAL.
```

## UNICODE VARIABLE-LENGTH CHARACTER STRINGS

Unicode variable-length character strings (unichar) are SQL data types similar to variable-length character strings, but are encoded using UTF-16 characters instead of single- or mixed-byte characters.

### Host Variable Formats:

#### CITOESQL

```
01 unichar1.  
49 unichar1-len PIC S9(4) COMP-5.  
49 unichar1-data PIC N(200) USAGE NATIONAL.  
01 unichar2.  
49 unichar1-len PIC S9(4) COMP.  
49 unichar1-data PIC N(200) USAGE NATIONAL.
```

The level number for group items containing only two elementary items must be 49. The first item is a four-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The length field can be signed or unsigned. The second item is a PIC $n$  NATIONAL data type, where  $n$  is an integer representing the length of the field that holds the data.

SQL statements must reference the group name.

## UNICODE LARGE CHARACTER STRINGS (DBCLOB)

Unicode large character strings (DBCLOB) enable you to store large amounts of Unicode data in columns.

### Host Variable Formats:

#### CITOESQL

```
01 dbclob1 SQL TYPE IS DBCLOB(2M).
```

The `dbclob1` format uses the DBCLOB SQL TYPE. The following definition is also valid:

```
01 dbclob2.  
49 dbclob2-len PIC S9(9) COMP-5.  
49 dbclob2-data PIC N(32000) USAGE NATIONAL.
```

The level number for group items containing only two elementary items must be 49. The first item is a four-byte field declared with usage COMP or COMP-5 that represents the effective length of the character string. The length field can be signed or unsigned. The second item is a PIC $n$  NATIONAL data type, where  $n$  is an integer representing the length of the field that holds the data.

SQL statements must reference the group name.

## Numeric Data Types

## APPROXIMATE NUMERIC DATA TYPES

Approximate numeric data types (FLOAT, DOUBLE, and REAL) are SQL data types that enable floating points.

### Host Variable Formats:

#### CITOESQL

```
01 real1 USAGE COMP-1.  
01 float1 USAGE COMP-2.
```

The **real1** format is for the 32-bit (single-precision) SQL floating-point data type, REAL.

The **float1** format is for 64-bit (double-precision) SQL floating-point data types, FLOAT and DOUBLE.

## EXACT NUMERIC DATA TYPES

Exact numeric data types (DECIMAL and NUMERIC) can hold values up to a driver-specified precision and scale.

### Host Variable Formats:

#### CITOESQL

```
01 packed1 PIC S9(8)V9(10) USAGE COMP-3.  
01 packed2 PIC S9(8)V9(10) USAGE PACKED-DECIMAL.  
01 packed3 PIC S9(8)V9(10) USAGE DISPLAY.
```

CitOESQL supports:

- Unsigned and signed DISPLAY numerics

- Leading and trailing signs

## UNICODE NUMERIC DATA TYPES

Unicode numeric data types (signed and unsigned) can hold values up to a driver-specified precision and scale.

### Host Variable Formats:

#### CITOESQL

```
01 uninum-us PIC 9(5)V9(5) USAGE NATIONAL.  
01 uninum-sls PIC S9(5) SIGN LEADING SEPARATE USAGE NATIONAL.  
01 uninum-sts PIC S9(5)V9(5) SIGN TRAILING SEPARATE USAGE NATIONAL.
```

CitOESQL supports:

Unsigned and signed Unicode numerics

Leading and trailing signs

## Binary Data Types

### FIXED-LENGTH BINARY STRINGS

Fixed-length binary data types (RAW, BINARY, and CHAR(x) FOR BIT DATA) are SQL data types with a driver-defined maximum length.

#### Host Variable Formats:

##### CITOESQL

```
03 bin-field1 PIC X(5).  
03 bin-field2 SQL TYPE IS BINARY(200).
```

- SQL BINARY, VARBINARY and IMAGE data are represented in COBOL as PIC X (*n*) fields.
- CITOESQL does not perform data conversion.
- When data is fetched from the database, if the host-variable field is smaller than the amount of data fetched, the data is truncated and the SQLWARN1 field in the SQLCA data structure is set to **W**. If the host-variable field is larger than the amount of data, the field is padded with null (x"00") bytes.
- Any of the following enable you to insert data into BINARY, VARBINARY or LONG-VARBINARY columns:
  - Use dynamic SQL statements
  - Compile your application with the ALLOWNULLCHAR directive
  - Use SQL TYPE host variables
- If you use PIC X host variables, compile your application with the ALLOWNULLCHAR directive to prevent the truncation of transferred data to or from the host variable if a null (x"00") is encountered.
- The **bin-field2** format uses the BINARY SQL TYPE.

### VARIABLE-LENGTH BINARY STRINGS

Variable-length binary data types (VARBINARY, LONG-VARBINARY, and LONG VARCHAR FOR BIT DATA) are SQL data types with a driver-defined maximum variable length.

#### Host Variable Formats:

##### CITOESQL

```

01 varbin-field1 SQL TYPE IS VARBINARY(2000).
01 varbin-field2 SQL TYPE IS LONG-VARBINARY(20000). 01 varbin-field3.
49 varbin-field3-len PIC S9(4) COMP-5.
49 varbin-field3-data PIC X(2000).

```

- SQL BINARY, VARBINARY and IMAGE data are represented in COBOL as PIC X (*n*) fields.
- CITOESQL does not perform data conversion.
- When data is fetched from the database, if the host-variable field is smaller than the amount of data fetched, the data is truncated and the SQLWARN1 field in the SQLCA data structure is set to **W**. If the host-variable field is larger than the amount of data, the field is padded with null (x"00") bytes.
- Any of the following enable you to insert data into BINARY, VARBINARY or LONGVARBINARY columns:
  - Use dynamic SQL statements
  - Compile your application with the ALLOWNULLCHAR directive
  - Use SQL TYPE host variables
- The **varbin-field1** format uses the VARBINARY SQL TYPE.
- The **varbin-field2** format uses the LONG-VARBINARY SQL TYPE.

## LARGE BINARY STRINGS (BLOB)

Large binary string data types (BLOB) are SQL data types that enable you to store large amounts of data, from sources such as JPG files, in binary columns.

### Host Variable Formats:

CITOESQL

```
01 blob1 SQL TYPE IS BLOB(2M).
```

- SQL BINARY, VARBINARY and IMAGE data are represented in COBOL as PIC X (n) fields.
- CITOESQL does not perform data conversion.
- When data is fetched from the database, if the host-variable field is smaller than the amount of data fetched, the data is truncated and the SQLWARN1 field in the SQLCA data structure is set to W. If the host-variable field is larger than the amount of data, the field is padded with null (x"00") bytes.
- Any of the following enable you to insert data into BINARY, VARBINARY or LONGVARBINARY columns:
  - Use dynamic SQL statements
  - Compile your application with the ALLOWNULLCHAR directive
  - Use SQL TYPE host variables

## Date and Time Data Types

COBOL does not support date and time data types directly. Therefore, date and time data columns are converted to COBOL character representations.

### DATE

#### Data Format:

Default and alternative date value formats vary. For CITOESQL, review the *DATE* and *DATEDELIM* SQL compiler directive option topics for more information.

For example, one supported date format is:

```
yyyy-mm-dd
```

An example value for this format is: 1994-05-24

#### Host Variable Formats:

CITOESQL

```
01 date1    PIC X(10) .
01 date2    SQL TYPE IS DATE .
01 date4    PIC X(n) .
```

### DATE1 FORMAT

- Move date data into the host variable using the form:
 

```
MOVE yyyy-mm-dd TO host-var .
```
- Review the *DETECTDATE* SQL compiler directive option topic to determine whether or not it applies to your application.

## DATE2 FORMAT

- Move date data into the host variable using the form:

```
MOVE yyyy-mm-dd TO host-var .
```

- Preferred format; use whenever possible.
- Uses the DATE SQL TYPE.
- Similar to the **date1** format, **date2** never requires DETECTDATE for input host variable processing.

## DATE4 FORMAT

- Move date data into host variables using these forms:

```
MOVE *yyyy* TO *host-var-year*  
MOVE *mm* TO *host-var-month*  
MOVE *dd* to *host-var-day*
```

- Uses the DATE-RECORD SQL TYPE.

## TIME

### Data Format:

Default and alternative time value formats vary. For CitOESQL, review the *TIME* and *TIMEDELIM* SQL compiler directive option topics for more information.

For example, one supported time format is:

```
hh:mm:ss
```

An example value for this format is: 12:34:00

### Host Variable Formats:

- Move time data into a host variable using any of these forms:

```
MOVE hh:mm:ss TO host-var .  
MOVE hh.mm.ss TO host-var .  
MOVE hh:mm PM TO host-var .
```

- Review the *DETECTDATE* SQL compiler directive option topic to determine whether or not it applies to your application.

## TIME2 FORMAT

- Move time data into a host variable using these forms:

```
MOVE `hh:mm:ss` TO `host-var`.
MOVE `hh.mm.ss` TO `host-var`.
MOVE `hh:mm` PM" TO `host-var`.
```

- Preferred format - use whenever possible. - Uses the TIME SQL TYPE. - Similar to the **time1** format, never requires DETECTDATE for input host variable processing.

## TIME4 FORMAT

- Move time data into host variables using this form:

```
MOVE *hh* TO *host-var-hour*
MOVE *mm* TO *host-var-min*
MOVE *ss* TO *host-var-sec*
```

- Uses the TIME-RECORD SQL TYPE.

## TIMESTAMP

Default and alternative timestamp value formats vary. For CITOESQL, review the *TSTAMPSEP* SQL compiler directive option topic for more information. For example, one supported time format is:

`yyyy-mm-dd hh:mm:ss[.f[f[...]]]` where the number of fractional digits is driver-defined. An example value for this format is: 1994-05-24 12:34:00.000

### Host Variable Formats:

#### CITOESQL

```
01 timestamp1 PIC X(29).
01 timestamp2 SQL TYPE IS TIMESTAMP.
```

The timestamp2 format uses the TIMESTAMP SQL TYPE.

```
01 timestamp4 SQL TYPE IS TIMESTAMP-RECORD.
```

## TIMESTAMP1 FORMAT

- Move timestamp data into a host variable using the form:

```
MOVE yyyy-mm-dd hh:mm:ss TO timestamp1
```

- Review the *DETECTDATE* SQL compiler directive option topic to determine whether or not it applies to your application.

## TIMESTAMP2 FORMAT



Move timestamp data into a host variable using the form:

```
MOVE yyyy-mm-dd hh:mm:ss TO timestamp2
```

Preferred format - use whenever possible.

Uses the `TIMESTAMP` SQL TYPE.

Similar to the **timestamp1** format, but when using Visual COBOL version 2.3 or later, never requires `DETECTDATE` for input host variable processing.

## TIMESTAMP4 FORMAT

- Move timestamp data into host variables using these forms:

```
MOVE yyyy TO host-var-year
MOVE mm TO *host-var-month
MOVE dd TO host-var-day
MOVE hh TO host-var-hour
MOVE mm TO host-var-minute
MOVE ss TO host-var-sec
MOVE ff TO host-varc-fra
```

- Uses the `TIMESTAMP-RECORD` SQL TYPE.

## TIMESTAMPOFFSET

If a COBOL output host variable is defined for an SQL timestamp value with an offset, the date and time are specified in the following format:

```
yyyy-mm-dd hh:mm:ss[.f[f[...]]] {+ \ -}hh:mm
```

where the number of fractional digits is driver-defined. For example: 1994-05-24 12:34:00.000  
+02:00

### Host Variable Formats:

#### CITOESQL

```
01 timestampoffset1 PIC X(36).
01 timestampoffset2 SQL TYPE IS TIMESTAMP-OFFSET.
01 timestampoffset3 SQL TYPE IS TIMESTAMP-OFFSET-RECORD.
```

The **timestampoffset2** format uses the `TIMESTAMP-OFFSET` SQL TYPE.

The **timestampoffset3** format uses the `TIMESTAMP-OFFSET-RECORD` SQL TYPE.

## 5.3.2 Miscellaneous Data Types

---

### PIC X VARYING Data Type

#### Syntax:

```
PIC X(*n*) VARYING
```

Where  $n$  is an integer representing the length of the field that holds the data.

#### Example:

```
01  ename PIC X(15) VARYING .
```

generates

```
01  ename .
05  ename-len PIC S9(4) COMP .
05  ename-ARR PIC X(15) .
```

### SQL TYPEs

- BINARY SQL Type
- BLOB SQL Type
- CHAR SQL Type
- CHAR-VARYING SQL Type
- CLOB SQL Type
- DATE SQL Type
- DATE-RECORD SQL Type
- DBCLOB SQL Type
- LONG-VARBINARY SQL Type
- LONG-VARCHAR SQL Type
- TIME SQL Type
- TIME-RECORD SQL Type
- TIMESTAMP SQL Type
- TIMESTAMP-RECORD SQL Type
- TIMESTAMP-OFFSET SQL Type
- TIMESTAMP-OFFSET-RECORD SQL Type

## VARBINARY SQL Type

### BINARY SQL TYPE

#### Syntax:

```
SQL [TYPE] [IS] BINARY(*n*)
```

#### Example:

```
01 hv-name SQL TYPE IS BINARY(200) .
```

generates

```
01 hv-name PIC X(200) .
```

### BLOB SQL TYPE

#### Syntax:

```
SQL [TYPE] [IS] BLOB(*lob-length*)
```

Where `lob-length` is a value between 1 and 1073741823 expressed either as a number or a number followed by K (kilobytes), M (megabytes), or G (gigabytes).

#### Note

Although this SQL TYPE has a theoretical size limitation of 2G, for all practical purposes, the actual limitation is approximately 450M, which is the data size actually allocated to the application program.

#### Example:

```
01 hv-name SQL TYPE IS BLOB(2M) .
```

generates

```
01 hv-name .
03 hv-name-length PIC S9(9) COMP-5 .
03 hv-name-data PIC X(2097152) .
```

### CHAR SQL TYPE

#### Syntax:

```
SQL [TYPE] [IS] CHAR(*n*)
```

#### Example:

```
01 hv-name SQL TYPE IS CHAR(200) .
```

generates

```
01 hv-name PIC X(200) .
```

## CHAR-VARYING SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] CHAR-VARYING(*n*)
```

CHAR-VARYING data is passed to CitOESQL as SQL\_VARCHAR.

Data sent to the data source eliminates trailing spaces except for the first space if the value is all spaces.

Values fetched from the data source are padded with spaces.

### Example:

```
01 hv-name SQL TYPE IS CHAR-VARYING(200) .
```

generates

```
01 hv-name PIC X(200) .
```

## CLOB SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] CLOB(lob-length)
```

Where `lob-length` is a value between 1 and 1073741823 expressed either as a number or a number followed by K (kilobytes), M (megabytes), or G (gigabytes).

### Note

Although this SQL TYPE has a theoretical size limitation of 2G, for all practical purposes, the actual limitation is approximately 450M, which is the data size actually allocated to the application program.

### Example:

```
01 hv-name SQL TYPE IS CLOB(2M) .
```

generates

```
01 hv-name .  
03 hv-name-length PIC S9(9) COMP-5 .  
03 hv-name-data PIC X(2097152) .
```

## DATE SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] DATE
```

Use DATE to generate a single working-storage record to contain all date information.

### Example:

```
01 hv-name SQL TYPE IS DATE .
```

generates

```
01 hv-name PIC X(10) .
```

## DATE-RECORD SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] DATE-RECORD
```

Use DATE-RECORD to generate a group-level record for the date containing individual records for each element of the date as follows:

Year

Month

Day

### Example:

```
01 hv-name SQL TYPE IS DATE .
```

generates

```
01 hv-name PIC X(10) .
```

## DBCLOB SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] DBCLOB(lob-length)
```

Where `lob-length` is a value between 1 and 1073741823 expressed either as a number or a number followed by K (kilobytes), M (megabytes), or G (gigabytes).

## Note

Although this SQL TYPE has a theoretical size limitation of 2G, for all practical purposes, the actual limitation is approximately 450M, which is the data size actually allocated to the application program.

### Example:

```
01 hv-name SQL TYPE IS DBCLOB(2M) .
```

### generates

```
01 hv-name .
03 hv-name-length PIC S9(9) COMP-5 .
03 hv-name-data PIC N(2097152) .
```

## LONG-VARBINARY SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] LONG-VARBINARY(*n*)
```

### Example:

```
01 hv-name SQL TYPE IS LONG-VARBINARY(2000) .
```

### generates

```
01 hv-name .
03 hv-name-len PIC S9(9) COMP-5 .
03 hv-name-val PIC X(2000) .
```

## LONG-VARCHAR SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] LONG-VARCHAR(*n*)
```

### Example:

```
01 hv-name SQL TYPE IS LONG-VARCHAR(65000) .
```

### generates

```
01 hv-name .
03 hv-name-len PIC S9(9) COMP-5 .
03 hv-name-val PIC X(65000) .
```

## TIME SQL TYPE

**Syntax:**

```
SQL [TYPE] [IS] TIME
```

Use TIME to generate a single working-storage record to contain all time information.

**Example:**

```
01 hv-name SQL TYPE IS TIME .
```

generates

```
01 hv-name PIC X(8) .
```

**TIME-RECORD SQL TYPE****Syntax:**

```
SQL [TYPE] [IS] TIME-RECORD
```

Use TIME-RECORD to generate a group-level record for the time containing individual records for each element as follows:

Hour

Minutes -Seconds

To insert data, you must pass valid data in the generated field names.

**Example:**

```
01 hv-name SQL TYPE IS TIME-RECORD .
```

generates

```
01 hv-name .
```

```
03 hv-name-hour PIC 9(4) COMP-5. 03 hv-name-min PIC 9(4) COMP-5. 03 hv-name-sec PIC  
9(4) COMP-5 .
```

**TIMESTAMP SQL TYPE****Syntax:**

```
SQL [TYPE] [IS] TIMESTAMP
```

Use TIMESTAMP to generate a single working-storage record to contain all timestamp information.

Data must be organized in fixed date/time formats.

Fractional seconds are supported up to nine digits. However, this value can vary depending on your target DBMS and your ODBC driver. See your DBMS or ODBC driver documentation for more information.

Fractional data is passed left justified and must contain the number of digits defined in your record. For example, to pass a fractional value of 678 to a record that defines fractional data as nine digits, move the value 678000000.

Fractional data is right justified when returned from a SELECT or FETCH statement.

Because of the way SQL Server stores date/time values, it might round the last fractional digit of a fractional value up or down depending on the digit. For example:

<b>If you pass...</b>	<b>SQL Server returns...</b>
01/01/98 23:59.59.999	1998-01-02 00:00:00.000
01/01/98 23:59.59.995	1998-01-01 23:59:59.997
01/01/98 23:59.59.996	1998-01-01 23:59:59.997
01/01/98 23:59.59.997	1998-01-01 23:59:59.997
01/01/98 23:59.59.998	1998-01-01 23:59:59.997
01/01/98 23:59.59.992	1998-01-01 23:59:59.993
01/01/98 23:59.59.993	1998-01-01 23:59:59.993
01/01/98 23:59.59.994	1998-01-01 23:59:59.993
01/01/98 23:59.59.990	1998-01-01 23:59:59.990



If you pass...	SQL Server returns...
01/01/98 23:59.59.991	1998-01-01 23:59:59.990

**Example:**

```
01 hv-name SQL TYPE IS TIMESTAMP .
```

generates

```
01 hv-name PIC X(29) .
```

## TIMESTAMP-RECORD SQL TYPE

**Syntax:**

```
SQL [TYPE] [IS] TIMESTAMP-RECORD
```

Use TIMESTAMP-RECORD to generate a group-level record for the timestamp containing individual records for each element of the timestamp as follows:

- Year
- Month
- Day
- Hour
- Minute
- Second
- Fractional second

To insert data, you must pass valid data in the generated field names.

Data must be organized in fixed date/time formats.

Fractional seconds are supported up to nine digits. However, this value can vary depending on your target DBMS and your ODBC driver. See your DBMS or ODBC driver documentation for more information.

Fractional data is passed left justified and must contain the number of digits defined in your record. For example, to pass a fractional value of 678 to a record that defines fractional data as nine digits, move the value 678000000.

Fractional data is right justified when returned from a SELECT or FETCH statement.

Because of the way SQL Server stores date/time values, it might round the last fractional digit of a fractional value up or down depending on the digit. For example:

<b>If you pass...</b>	<b>SQL Server returns...</b>
01/01/98 23:59.59.999	1998-01-02 00:00:00.000
01/01/98 23:59.59.995	1998-01-01 23:59:59.997
01/01/98 23:59.59.996	1998-01-01 23:59:59.997
01/01/98 23:59.59.997	1998-01-01 23:59:59.997
01/01/98 23:59.59.998	1998-01-01 23:59:59.997
01/01/98 23:59.59.992	1998-01-01 23:59:59.993
01/01/98 23:59.59.993	1998-01-01 23:59:59.993
01/01/98 23:59.59.994	1998-01-01 23:59:59.993
01/01/98 23:59.59.990	1998-01-01 23:59:59.990

If you pass...	SQL Server returns...
01/01/98 23:59.59.991	1998-01-01 23:59:59.990

### Example:

```
01 hv-name SQL TYPE IS TIMESTAMP-RECORD .
```

### generates

```
01 hv-name.
03 hv-name-year PIC S9(4) COMP-5.
03 hv-name-month PIC 9(4) COMP-5.
03 hv-name-day PIC 9(4) COMP-5. 03 hv-name-hour PIC 9(4) COMP-5. 03 hv-name-min PIC 9(4) COMP-5.
03 hv-name-sec PIC 9(4) COMP-5. 03 hv-name-frac PIC 9(9) COMP-5.
```

## TIMESTAMP-OFFSET SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] TIMESTAMP-OFFSET
```

Use TIMESTAMP-OFFSET to generate a single working-storage record to contain all timestamp information.

### Example:

```
01 hv-name SQL TYPE IS TIMESTAMP-OFFSET .
```

### generates

```
01 hv-name PIC X(36) .
```

## TIMESTAMP-OFFSET-RECORD SQL TYPE

### Syntax:

```
SQL [TYPE] [IS] TIMESTAMP-OFFSET-RECORD
```

Use TIMESTAMP-OFFSET-RECORD to generate a group-level record for the timestamp containing individual records for each element of the timestamp as follows:

Year  
Month  
Day  
Hour  
Minute  
Second  
Fractional second  
Offset hours  
Offset minutes

To insert data, you must pass valid data in the generated field names.

**Example:**

```
01 hv-name SQL TYPE IS TIMESTAMP-OFFSET-RECORD .
```

generates

```
01 hv-name.  
03 hv-name-year PIC S9(4) COMP-5.  
03 hv-name-month PIC 9(4) COMP-5.  
03 hv-name-day PIC 9(4) COMP-5. 03 hv-name-hour PIC 9(4) COMP-5. 03 hv-name-min PIC 9(4) COMP-5.  
03 hv-name-sec PIC 9(4) COMP-5. 03 hv-name-frac PIC 9(9) COMP-5.  
03 hv-name-tz-hour PIC S9(4) COMP-5.  
03 hv-name-tz-min PIC S9(4) COMP-5.
```

## VARBINARY SQL TYPE

**Syntax:**

SQL [TYPE] [IS] VARBINARY(*n*)

**Example:**

```
01 hv-name SQL TYPE IS VARBINARY(2000) .
```

generates

```
01 hv-name.  
49 hv-name-len PIC S9(4) COMP-5.  
49 hv-name-val PIC X(2000).  
49 hv-name-text REDEFINES hv-name-val PIC X(2000).
```

## 5.3.3 USAGE Data Types

CitOESQL supports the following non-COBOL USAGE Clauses:

USAGE VARCHAR

USAGE LONG VARCHAR -USAGE VARRAW

USAGE LONG VARRAW

USAGE VARYING

## USAGE VARCHAR Data Type

### Syntax:

```
USAGE VARCHAR
```

### Example:

```
01 HV-NAME PIC X(30) USAGE VARCHAR .
```

generates

```
01 HV-NAME .  
03 HV-NAME-LEN PIC S9(4) COMP-5 VALUE 0 .  
03 HV-NAME-ARR PIC X(30) .
```

## USAGE LONG VARCHAR DATA TYPE

### Syntax:

```
USAGE VARCHAR
```

### Example:

```
01 HV-NAME PIC X(300) USAGE LONG VARCHAR .
```

generates

```
01 HV-NAME .  
03 HV-NAME-LEN PIC S9(4) COMP-5 VALUE 0 .  
03 HV-NAME-ARR PIC X(300) .
```

## USAGE VARRAW DATA TYPE

### Syntax:

```
USAGE VARRAW
```

### Example:

```
01 HV-NAME PIC X(30) USAGE VARRAW .
```

generates

```
01 HV-NAME.  
03 HV-NAME-LEN PIC S9(9) COMP-5 VALUE 0.  
03 HV-NAME-ARR PIC X(30).
```

## USAGE LONG VARRAW DATA TYPE

### Syntax:

```
USAGE LONG VARRAW
```

### Example:

```
01 HV-NAME PIC X(300) USAGE LONG VARRAW.
```

### generates

```
01 HV-NAME.  
03 HV-NAME-LEN PIC S9(9) COMP-5 VALUE 0.  
03 HV-NAME-ARR PIC X(300).
```

## USAGE VARYING DATA TYPE

### Syntax:

```
USAGE VARYING
```

### Example:

```
01 HV-NAME PIC X(30) USAGE VARYING.
```

### generates

```
01 HV-NAME.  
03 HV-NAME-LEN PIC S9(4) COMP-5 VALUE 0.  
03 HV-NAME-ARR PIC X(30).
```

## 6. Legal Notice

---

For information about legal notices, trademarks, disclaimers, warranties, export and other use restrictions, U.S. Government rights, patent policy, and FIPS compliance, see <https://www.microfocus.com/about/legal/>.

© Copyright 2023 Micro Focus or one of its affiliates.

The only warranties for products and services of Micro Focus and its affiliates and licensors ("Micro Focus") are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

This documentation is derived from COBOL-IT Source code, parts of which are derived from OpenCOBOL.

Copyright (C) 2002-2007 Keisuke Nishida

Copyright (C) 2007 Roger While

### 6.1 Third-Party Notices

---

Additional third-party notices, including copyrights and software license texts, can be found in a 'Third-Party-License-File' file in the root directory of the software.