CONNECT (Type 1)

The CONNECT (Type 1) statement connects an application process to the identified application server according to the rules for remote unit of work.

An application process can only be connected to one application server at a time. This is called the *current server*. A default application server may be established when the application requester is initialized. If implicit connect is available and an application process is started, it is implicitly connected to the default application server. The application process can explicitly connect to a different application server by issuing a CONNECT TO statement. A connection lasts until a CONNECT RESET statement or a DISCONNECT statement is issued or until another CONNECT TO statement changes the application server.

See "Remote Unit of Work Connection Management" on page 31 for concepts and additional details on connection states. See "Options that Govern Distributed Unit of Work Semantics" on page 39 for the precompiler options that determine the framework for CONNECT behavior.

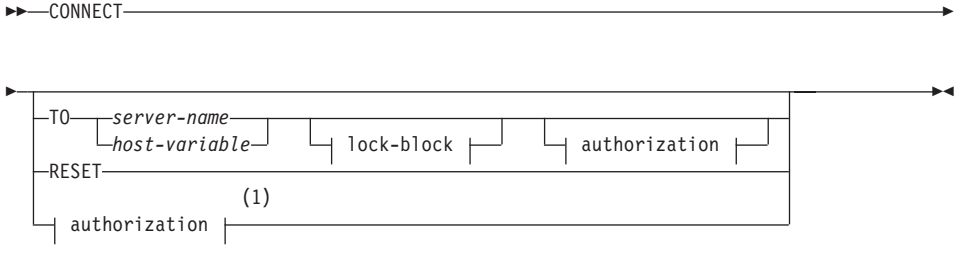
Invocation

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

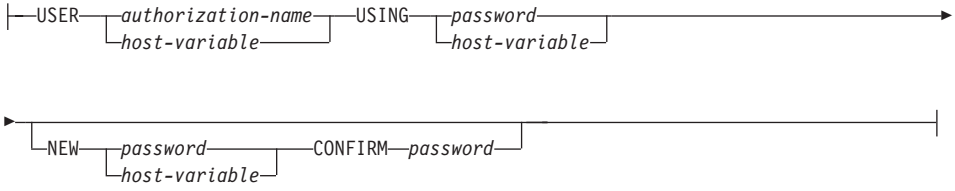
Authorization

The authorization ID of the statement must be authorized to connect to the identified application server. Depending on the authentication setting for the database, the authorization check may be performed by either the client or the server. For a partitioned database, the user and group definitions must be identical across partitions or nodes. Refer to the AUTHENTICATION database manager configuration parameter in the *Administration Guide* for information about the authentication setting.

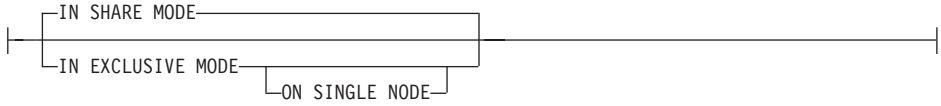
Syntax



authorization:



lock-block:



Notes:

1 This form is only valid if implicit connect is enabled.

Description

CONNECT (with no operand)

Returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described in "Successful Connection".

If a connection state exists, the authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If the authorization ID is longer than 8 bytes, it will be truncated to 8 bytes, and the truncation will be flagged in the SQLWARN0 and SQLWARN1 fields of the SQLCA, with 'W' and 'A', respectively. If the database configuration parameter DYN_QUERY_MGMT is enabled, then the SQLWARN0 and SQLWARN7 fields of the SQLCA will be flagged with 'W' and 'E', respectively.

If no connection exists and implicit connect is possible, then an attempt to make an implicit connection is made. If implicit connect is not available, this attempt results in an error (no existing connection). If no connection, then the SQLERRMC field is blank.

The country code and code page of the application server are placed in the SQLERRMC field (as they are with a successful CONNECT TO statement).

This form of CONNECT:

- Does not require the application process to be in the connectable state.
- If connected, does not change the connection state.

- If unconnected and implicit connect is available, a connection to the default application server is made. In this case, the country code and code page of the application server are placed in the SQLERRMC field, like a successful CONNECT TO statement.
- If unconnected and implicit connect is not available, the application process remains unconnected.
- Does not close cursors.

TO *server-name* or *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the server-name.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

Note: DB2 for MVS supports a 16 byte location-name and both SQL/DS and DB2/400 support a 18 byte target database name. DB2 Version 7 only supports the use of 8 byte database-alias name on the SQL CONNECT statement. However, the database-alias name can be mapped to an 18 byte database name through the Database Connection Service Directory.

When the CONNECT TO statement is executed, the application process must be in the connectable state (see "Remote Unit of Work Connection Management" on page 31 for information about connection states with Type 1 CONNECT).

Successful Connection:

If the CONNECT TO statement is successful:

- All open cursors are closed, all prepared statements are destroyed, and all locks are released from the previous application server.
- The application process is disconnected from its previous application server, if any, and connected to the identified application server.
- The actual name of the application server (not an alias) is placed in the CURRENT SERVER special register.
- Information about the application server is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvvrrm*, where:
  - *ppp* identifies the product as follows:

- DSN for DB2 for MVS
- ARI for SQL/DS
- QSQ for DB2/400
- SQL for DB2 Universal Database

– *vv* is a two-digit version identifier such as '02'

– *rr* is a two-digit release identifier such as '01'

– *m* is a one-digit modification level identifier such as '0'.

For example, if the application server is Version 1 Release 1 of DB2 for OS/2, the value of SQLERRP is 'SQL01010'. [66]

- The SQLERRMC field of the SQLCA is set to contain the following values (separated by X'FF')
  1. the country code of the application server (or blanks if using DDCS),
  2. the code page of the application server (or CCSID if using DDCS),
  3. the authorization ID (up to first 8 bytes only),
  4. the database alias,
  5. the platform type of the application server. Currently identified values are:

Token	Server
QAS	DB2 Universal Database for AS/400
QDB2	DB2 Universal Database for OS/390
QDB2/2	DB2 Universal Database for OS/2
QDB2/6000	DB2 Universal Database for AIX
QDB2/HPUX	DB2 Universal Database for HP-UX
QDB2/LINUX	DB2 Universal Database for Linux
QDB2/NT	DB2 Universal Database for Windows NT
QDB2/PTX	DB2 Universal Database for NUMA-Q
QDB2/SCO	DB2 Universal Database for SCO UnixWare

66. This release of DB2 Universal Database Version 7 is 'SQL07010'.

QDB2/SNI	DB2 Universal Database for Siemens Nixdorf
QDB2/SUN	DB2 Universal Database for Solaris Operating System
QDB2/Windows 95	DB2 Universal Database for Windows 95 or Windows 98
QSQLDS/VM	DB2 Server for VM
QSQLDS/VSE	DB2 Server for VSE

6. The agent ID. It identifies the agent executing within the database manager on behalf of the application. This field is the same as the agent_id element returned by the database monitor.
7. The agent index. It identifies the index of the agent and is used for service.
8. Partition number. For a non-partitioned database, this is always 0, if present.
9. The code page of the application client.
10. Number of partitions in a partitioned database. If the database cannot be partitioned, the value is 0 (zero). Token is present only with Version 5 or later.

- The SQLERRD(1) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. [67]
- The SQLERRD(2) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. [67]
- The SQLERRD(3) field of the SQLCA indicates whether or not the database on the connection is updatable. A database is initially updatable, but is changed to read-only if a unit of work determines the authorization ID cannot perform updates. The value is one of:
  - 1 - updatable
  - 2 - read-only

67. See the "Character Conversion Expansion Factor" section of the "Programming in Complex Environments" chapter in the *Application Development Guide* for details.

- The SQLERRD(4) field of the SQLCA returns certain characteristics of the connection. The value is one of:
  - **0 -** N/A (only possible if running from a down-level client which is one phase commit and is an updater).
  - **1 -** one-phase commit.
  - **2 -** one-phase commit; read-only (only applicable to connections to DRDA1 databases in TP Monitor environment).
  - **3 -** two-phase commit.
- The SQLERRD(5) field of the SQLCA returns the authentication type of the connection. The value is one of:
  - **0 -** Authenticated on the server.
  - **1 -** Authenticated on the client.
  - **2 -** Authenticated using DB2 Connect.
  - **3 -** Authenticated using Distributed Computing Environment security services.
  - **255 -** Authentication not specified.

  See ″Controlling Database Access″ in the *Administration Guide* for details on authentication types.
- The SQLERRD(6) field of the SQLCA returns the partition number of the partition to which the connection was made if the database is partitioned. Otherwise, a value of 0 is returned.
- The SQLWARN1 field in the SQLCA will be set to 'A' if the authorization ID of the successful connection is longer than 8 bytes. This indicates that truncation has occurred. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.
- The SQLWARN7 field in the SQLCA will be set to 'E' if the database configuration parameter DYN_QUERY_MGMT for the database is enabled. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.

Unsuccessful Connection:

If the CONNECT TO statement is unsuccessful:

- The SQLERRP field of the SQLCA is set to the name of the module at the application requester that detected the error. Note that the first three characters of the module name identifies the product. For example, if the application requester is on the OS/2 database manager, the first three characters are 'SQL'.
- If the CONNECT TO statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged.

- If the CONNECT TO statement is unsuccessful because the *server-name* is not listed in the local directory, an error message (SQLSTATE 08001) is issued and the connection state of the application process remains unchanged:
  - If the application requester was not connected to an application server then the application process remains unconnected.
  - If the application requester was already connected to an application server, the application process remains connected to that application server. Any further statements are executed at that application server.
- If the CONNECT TO statement is unsuccessful for any other reason, the application process is placed into the unconnected state.

IN SHARE MODE
Allows other concurrent connections to the database and prevents other users from connecting to the database in exclusive mode.

IN EXCLUSIVE MODE [68]
Prevents concurrent application processes from executing any operations at the application server, unless they have the same authorization ID as the user holding the exclusive lock.

ON SINGLE NODE
Specifies that the coordinator partition is connected in exclusive mode and all other partitions are connected in share mode. This option is only effective in a partitioned database.

RESET
Disconnects the application process from the current server. A commit operation is performed. If implicit connect is available, the application process remains unconnected until an SQL statement is issued.

USER *authorization-name/host-variable*
Identifies the userid trying to connect to the application server. If a host-variable is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The userid that is contained within the *host-variable* must be left justified and must not be delimited by quotation marks.

USING *password/host-variable*
Identifies the password of the userid trying to connect to the application server. *Password* or *host-variable* may be up to 18 characters. If a host variable is specified, it must be a character string variable with a length attribute not greater than 18 and it must not include an indicator variable.

NEW *password/host-variable* **CONFIRM** *password*
Identifies the new password that should be assigned to the userid

68. This option is not supported by DDCS.

identified by the USER option. *Password* or *host-variable* may be up to 18 characters. If a host variable is specified, it must be a character string variable with a length attribute not greater than 18 and it must not include an indicator variable. The system on which the password will be changed depends on how user authentication is set up.

Notes

- It is good practice for the first SQL statement executed by an application process to be the CONNECT TO statement.
- If a CONNECT TO statement is issued to the current application server with a different userid and password then the conversation is deallocated and reallocated. All cursors are closed by the database manager (with the loss of the cursor position if the WITH HOLD option was used).
- If a CONNECT TO statement is issued to the current application server with the same userid and password then the conversation is not deallocated and reallocated. Cursors, in this case, are not closed.
- To use DB2 Universal Database Enterprise - Extended Edition, the user or application must connect to one of the partitions listed in the `db2nodes.cfg` file (see "Data Partitioning Across Multiple Partitions" on page 59 for information about this file). You should try to ensure that not all users use the same partition as the coordinator partition.

Examples

Example 1: In a C program, connect to the application server TOROLAB3, where TOROLAB3 is a database alias of the same name, with the userid FERMAT and the password THEOREM.

```
EXEC SQL  CONNECT TO TOROLAB3 USER FERMAT USING THEOREM;
```

Example 2: In a C program, connect to an application server whose database alias is stored in the host variable APP_SERVER (varchar(8)). Following a successful connection, copy the 3 character product identifier of the application server to the variable PRODUCT (char(3)).

```
EXEC SQL  CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
  strncpy(PRODUCT,sqlca.sqlerrp,3);
```

CONNECT (Type 2)

The CONNECT (Type 2) statement connects an application process to the identified application server and establishes the rules for application-directed distributed unit of work. This server is then the current server for the process.

See "Application-Directed Distributed Unit of Work" on page 35 for concepts and additional details.

Most aspects of a CONNECT (Type 1) statement also apply to a CONNECT (Type 2) statement. Rather than repeating that material here, this section describes only those elements of Type 2 that differ from Type 1.

Invocation

The invocation is the same as "Invocation" on page 550.

Authorization

The authorization is the same as "Authorization" on page 550.

Syntax

The syntax is the same as "Syntax" on page 550. The selection between Type 1 and Type 2 is determined by precompiler options. See "Options that Govern Distributed Unit of Work Semantics" on page 39 for an overview of these options. Further details are provided in the *Command Reference* and *Administrative API Reference* manuals.

Description

TO *server-name/host-variable*
The rules for coding the name of the server are the same as for Type 1.

If the SQLRULES(STD) option is in effect, the *server-name* must not identify an existing connection of the application process, otherwise an error (SQLSTATE 08002) is raised.

If the SQLRULES(DB2) option is in effect and the *server-name* identifies an existing connection of the application process, that connection is made current and the old connection is placed into the dormant state. That is, the effect of the CONNECT statement in this situation is the same as that of a SET CONNECTION statement.

See "Options that Govern Distributed Unit of Work Semantics" on page 39 for information about the specification of SQLRULES.

Successful Connection

If the CONNECT TO statement is successful:

- A connection to the application server is either created (or made non-dormant) and placed into the current and held states.

- If the CONNECT TO is directed to a different server than the current server, then the current connection is placed into the dormant state.
- The CURRENT SERVER special register and the SQLCA are updated in the same way as for Type 1 CONNECT; see page 552.

Unsuccessful Connection

If the CONNECT TO statement is unsuccessful:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.
- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module at the application requester or server that detected the error.

CONNECT (with no operand)**, IN SHARE/EXCLUSIVE MODE, USER,** and **USING**

If a connection exists, Type 2 behaves like a Type 1. The authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If a connection does not exist, no attempt to make an implicit connection is made and the SQLERRP and SQLERRMC fields return a blank. (Applications can check if a current connection exists by checking these fields.)

A CONNECT with no operand that includes USER and USING can still connect an application process to a database using the DB2DBDFT environment variable. This method is equivalent to a Type 2 CONNECT RESET, but permits the use of a userid and password.

RESET

Equivalent to an explicit connect to the default database if it is available. If a default database is not available, the connection state of the application process and the states of its connections are unchanged.

Availability of a default database is determined by installation options, environment variables, and authentication settings. See the *Quick Beginnings* for information on setting implicit connect on installation and environment variables, and the *Administration Guide* for information on authentication settings.

Rules

- As outlined in "Options that Govern Distributed Unit of Work Semantics" on page 39 a set of connection options governs the semantics of connection management. Default values are assigned to every preprocessed source file. An application can consist of multiple source files precompiled with different connection options.

Unless a SET CLIENT command or API has been executed first, the connection options used when preprocessing the source file containing the first SQL statement executed at run-time become the effective connection options.

If a CONNECT statement from a source file preprocessed with different connection options is subsequently executed without the execution of any intervening SET CLIENT command or API, an error (SQLSTATE 08001) is raised. Note that once a SET CLIENT command or API has been executed, the connection options used when preprocessing all source files in the application are ignored.

Example 1 on page 563 illustrates these rules.

- Although the CONNECT TO statement can be used to establish or switch connections, CONNECT TO with the USER/USING clause will only be accepted when there is no current or dormant connection to the named server. The connection must be released before issuing a connection to the same server with the USER/USING clause, otherwise it will be rejected (SQLSTATE 51022). Release the connection by issuing a DISCONNECT statement or a RELEASE statement followed by a COMMIT statement.

Notes

- Implicit connect is supported for the first SQL statement in an application with Type 2 connections. In order to execute SQL statements on the default database, first the CONNECT RESET or the CONNECT USER/USING statement must be used to establish the connection. The CONNECT statement with no operands will display information about the current connection if there is one, but will not connect to the default database if there is no current connection.

Comparing Type 1 and Type 2 CONNECT Statements:

The semantics of the CONNECT statement are determined by the CONNECT precompiler option or the SET CLIENT API (see "Options that Govern Distributed Unit of Work Semantics" on page 39). CONNECT Type 1 or CONNECT Type 2 can be specified and the CONNECT statements in those programs are known as Type 1 and Type 2 CONNECT statements respectively. Their semantics are described below:

Use of **CONNECT TO**:

Type 1	Type 2
Each unit of work can only establish connection to one application server.	Each unit of work can establish connection to multiple application servers.
The current unit of work must be committed or rolled back before allowing a connection to another application server.	The current unit of work need not be committed or rolled back before connecting to another application server.

Type 1	Type 2
The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until changed by another CONNECT.	Same as Type 1 CONNECT if establishing the first connection. If switching to a dormant connection and SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to the current connection is valid and does not change the current connection.	Same as Type 1 CONNECT if the SQLRULES precompiler option is set to DB2. If SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to another application server disconnects the current connection. The new connection becomes the current connection. Only one connection is maintained in a unit of work.	Connecting to another application server puts the current connection into the *dormant state*. The new connection becomes the current connection. Multiple connections can be maintained in a unit of work. If the CONNECT is for an application server on a dormant connection, it becomes the current connection. Connecting to a dormant connection using CONNECT is only allowed if SQLRULES(DB2) was specified. If SQLRULES(STD) was specified, then the SET CONNECTION statement must be used instead.
SET CONNECTION statement is supported for Type 1 connections, but the only valid target is the current connection.	SET CONNECTION statement is supported for Type 2 connections to change the state of a connection from dormant to current.

Use of **CONNECT...USER...USING**:

Type 1	Type 2
Connecting with the USER...USING clauses disconnects the current connection and establishes a new connection with the given authorization name and password.	Connecting with the USER/USING clause will only be accepted when there is no current or dormant connection to the same named server.

Use of **Implicit CONNECT, CONNECT RESET**, and **Disconnecting**:

Type 1	Type 2
CONNECT RESET can be used to disconnect the current connection.	CONNECT RESET is equivalent to connecting to the default application server explicitly if one has been defined in the system. Connections can be disconnected by the application at a successful COMMIT. Prior to the commit, use the RELEASE statement to mark a connection as release-pending. All such connections will be disconnected at the next COMMIT. An alternative is to use the precompiler options DISCONNECT(EXPLICIT), DISCONNECT(CONDITIONAL), DISCONNECT(AUTOMATIC), or the DISCONNECT statement instead of the RELEASE statement.
After using CONNECT RESET to disconnect the current connection, if the next SQL statement is not a CONNECT statement, then it will perform an implicit connect to the default application server if one has been defined in the system.	CONNECT RESET is equivalent to an explicit connect to the default application server if one has been defined in the system.
It is an error to issue consecutive CONNECT RESETs.	It is an error to issue consecutive CONNECT RESETs ONLY if SQLRULES(STD) was specified because this option disallows the use of CONNECT to existing connection.
CONNECT RESET also implicitly commits the current unit of work.	CONNECT RESET does not commit the current unit of work.
If an existing connection is disconnected by the system for whatever reasons, then subsequent non-CONNECT SQL statements to this database will receive an SQLSTATE of 08003.	If an existing connection is disconnected by the system, COMMIT, ROLLBACK, and SET CONNECTION statements are still permitted.
The unit of work will be implicitly committed when the application process terminates successfully.	Same as Type 1.
All connections (only one) are disconnected when the application process terminates.	All connections (current, dormant, and those marked for release pending) are disconnected when the application process terminates.

CONNECT Failures:

Type 1	Type 2
Regardless of whether there is a current connection when a CONNECT fails (with an error other than server-name not defined in the local directory), the application process is placed in the unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.	If there is a current connection when a CONNECT fails, the current connection is unaffected. If there was no current connection when the CONNECT fails, then the program is then in an unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

Examples

Example 1: This example illustrates the use of multiple source programs (shown in the boxes), some preprocessed with different connection options (shown above the code) and one of which contains a SET CLIENT API call.

PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tbl1;
...
```

PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tbl2;
...
```

PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2  SQLRULES DB2  DISCONNECT EXPLICIT [1]
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO
:hv1 FROM tbl3;
...
```

1 Note: not the actual syntax of the SET CLIENT API

PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO
:hv1 FROM tbl4;
...
```

If the application executes PGM1 then PGM2:
- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to QUEBEC fails with SQLSTATE 08001 because both SQLRULES and DISCONNECT are different.

If the application executes PGM1 then PGM3:
- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to LONDON runs: connect=2, sqlrules=DB2, disconnect=EXPLICIT

This is OK because the SET CLIENT API is run before the second CONNECT statement.

If the application executes PGM1 then PGM4:
- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to REGINA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

This is OK because the preprocessor options for PGM1 are the same as those for PGM4.

Example 2:

This example shows the interrelationships of the CONNECT (Type 2), SET CONNECTION, RELEASE, and DISCONNECT statements. S0, S1, S2, and S3 represent four servers.

Sequence	Statement	Current Server	Dormant Connections	Release Pending
0	No statement	None	None	None
1.	SELECT * FROM TBLA	S0 (default)	None	None
2	CONNECT TO S1 SELECT * FROM TBLB	S1 S1	S0 S0	None None
3	CONNECT TO S2 UPDATE TBLC SET ...	S2 S2	S0, S1 S0, S1	None None
4	CONNECT TO S3 SELECT * FROM TBLD	S3 S3	S0, S1, S2 S0, S1, S2	None None
5	SET CONNECTION S2	S2	S0, S1, S3	None
6	RELEASE S3	S2	S0, S1	S3
7	COMMIT	S2	S0, S1	None
8	SELECT * FROM TBLE	S2	S0, S1	None

Sequence	Statement	Current Server	Dormant Connections	Release Pending
9	DISCONNECT S1 SELECT * FROM TBLF	S2 S2	S0 S0	None None

CREATE ALIAS

The CREATE ALIAS statement defines an alias for a table, view, nickname, or another alias.

Invocation

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

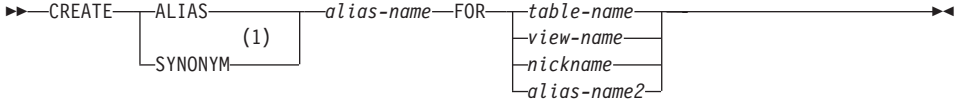
Authorization

The privileges held by the authorization ID of the statement must include as least one of the following:

- SYSADM or DBADM authority
- IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the alias does not exist
- CREATEIN privilege on the schema, if the schema name of the alias refers to an existing schema.

To use the referenced object via the alias, the same privileges are required on that object as would be necessary if the object itself were used.

Syntax



Notes:

1 CREATE SYNONYM is accepted as an alternative for CREATE ALIAS for syntax toleration of existing CREATE SYNONYM statements of other SQL implementations.

Description

alias-name

Names the alias. The name must not identify a table, view, nickname, or alias that exists in the current database.

If a two-part name is specified, the schema name cannot begin with ″SYS″ (SQLSTATE 42939).

The rules for defining an alias name are the same as those used for defining a table name.

FOR *table-name, view-name, nickname,* or *alias-name2*
Identifies the table, view, nickname, or alias for which *alias-name* is defined. If another alias name is supplied (*alias-name2*), then it must not be the same as the new *alias-name* being defined (in its fully-qualified form). The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

Notes

- The definition of the newly created alias is stored in SYSCAT.TABLES.
- An alias can be defined for an object that does not exist at the time of the definition. If it does not exist, a warning is issued (SQLSTATE 01522). However, the referenced object must exist when a SQL statement containing the alias is compiled, otherwise an error is issued (SQLSTATE 52004).
- An alias can be defined to refer to another alias as part of an alias chain but this chain is subject to the same restrictions as a single alias when used in an SQL statement. An alias chain is resolved in the same way as a single alias. If an alias used in a view definition, a statement in a package, or a trigger points to an alias chain, then a dependency is recorded for the view, package, or trigger on each alias in the chain. Repetitive cycles in an alias chain are not allowed and are detected at alias definition time.
- Creating an alias with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

Examples

Example 1: HEDGES attempts to create an alias for a table T1 (both unqualified).

```
CREATE ALIAS A1 FOR T1
```

The alias HEDGES.A1 is created for HEDGES.T1.

Example 2: HEDGES attempts to create an alias for a table (both qualified).

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
```

The alias HEDGES.A1 is created for MCKNIGHT.T1.

Example 3: HEDGES attempts to create an alias for a table (alias in a different schema; HEDGES is not a DBADM; HEDGES does not have CREATEIN on schema MCKNIGHT).

```
CREATE ALIAS MCKNIGHT.A1 FOR MCKNIGHT.T1
```

This example fails (SQLSTATE 42501).

Example 4: HEDGES attempts to create an alias for an undefined table (both qualified; FUZZY.WUZZY does not exist).

```
CREATE ALIAS HEDGES.A1 FOR FUZZY.WUZZY
```

This statement succeeds but with a warning (SQLSTATE 01522).

Example 5: HEDGES attempts to create an alias for an alias (both qualified).

```
CREATE ALIAS HEDGES.A1 FOR MCKNIGHT.T1
CREATE ALIAS HEDGES.A2 FOR HEDGES.A1
```

The first statement succeeds (as per example 2).

The second statement succeeds and an alias chain is created, consisting of HEDGES.A2 which refers to HEDGES.A1 which refers to MCKNIGHT.T1. Note that it does not matter whether or not HEDGES has any privileges on MCKNIGHT.T1. The alias is created regardless of the table privileges.

Example 6: Designate A1 as an alias for the nickname FUZZYBEAR.

```
CREATE ALIAS A1 FOR FUZZYBEAR
```

Example 7: A large organization has a finance department numbered D108 and a personnel department numbered D577. D108 keeps certain information in a table that resides at a DB2 RDBMS. D577 keeps certain records in a table that resides at an Oracle RDBMS. A DBA defines the two RDBMSs as data sources within a federated system, and gives the tables the nicknames of DEPTD108 and DEPTD577, respectively. A federated system user needs to create joins between these tables, but would like to reference them by names that are more meaningful than their alphanumeric nicknames. So the user defines FINANCE as an alias for DEPTD108 and PERSONNEL as an alias for DEPTD577.

```
CREATE ALIAS FINANCE FOR DEPTD108
CREATE ALIAS PERSONNEL FOR DEPTD577
```

CREATE BUFFERPOOL

The CREATE BUFFERPOOL statement creates a new buffer pool to be used by the database manager. Although the buffer pool definition is transactional and the entries will be reflected in the catalog tables on commit, the buffer pool will not become active until the next time the database is started.

In a partitioned database, a default buffer pool definition is specified for each partition or node, with the capability to override the size on specific partitions or nodes. Also, in a partitioned database, the buffer pool is defined on all partitions unless nodegroups are specified. If nodegroups are specified, the buffer pool will only be created on partitions that are in those nodegroups.

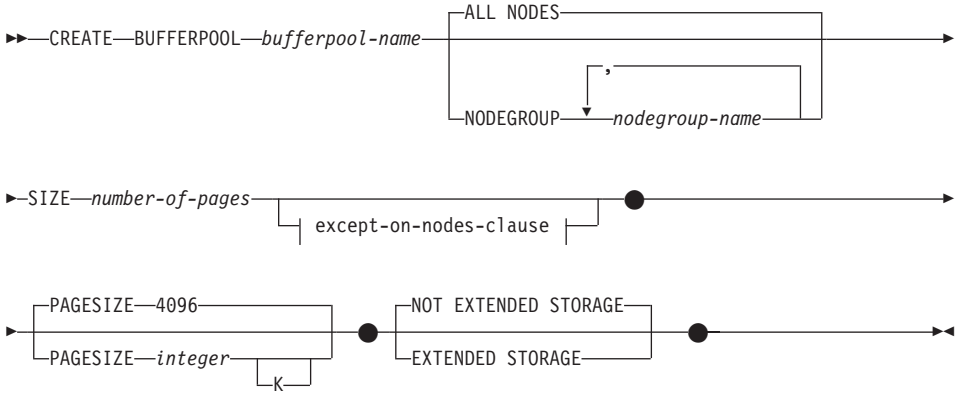
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

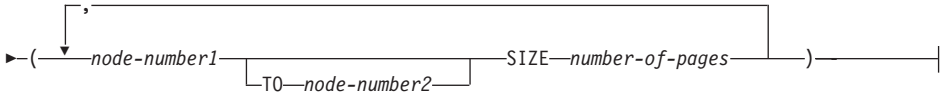
Authorization

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax



except-on-nodes-clause:



Description

bufferpool-name
Names the buffer pool. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *bufferpool-name* must not identify a buffer pool that already exists in a catalog (SQLSTATE 42710). The *bufferpool-name* must not begin with the characters "SYS" or "IBM" (SQLSTATE 42939).

ALL NODES
This buffer pool will be created on all partitions in the database.

NODEGROUP *nodegroup-name, ...*
Identifies the nodegroup or nodegroups to which the buffer pool definition is applicable. If this is specified, this buffer pool will only be created on partitions in these nodegroups. Each nodegroup must currently exist in the database (SQLSTATE 42704). If the NODEGROUP keyword is not specified, then this buffer pool will be created on all partitions (and any partitions subsequently added to the database).

SIZE *number-of-pages*
The size of the buffer pool specified as the number of pages. [69] In a partitioned database, this will be the default size for all partitions where the buffer pool exists.

except-on-nodes-clause
Specifies the partition or partitions for which the size of the buffer pool will be different than the default. If this clause is not specified, then all partitions will have the same size as specified for this buffer pool.

EXCEPT ON NODES
Keywords that indicate that specific partitions are specified. NODE is a synonym for NODES.

node-number1
Specifies a specific partition number that is included in the partitions for which the buffer pool is created.

TO *node-number2*
Specify a range of partition numbers. The value of *node-number2* must be greater than or equal to the value of *node-number1*

69. The size can be specified with a value of (-1) which will indicate that the buffer pool size should be taken from the BUFFPAGE database configuration parameter.

(SQLSTATE 428A9). All partitions between and including the specified partition numbers must be included in the partitions for which the buffer pool is created (SQLSTATE 42729).

SIZE *number-of-pages*
The size of the buffer pool specified as the number of pages.

PAGESIZE *integer* **[K]**
Defines the size of pages used for the bufferpool. The valid values for *integer* without the suffix K are 4 096, 8 192, 16 384 or 32 768. The valid values for *integer* with the suffix K are 4, 8, 16 or 32. An error occurs if the page size is not one of these values (SQLSTATE 428DE). The default is 4 096 byte (4K) pages. Any number of spaces is allowed between *integer* and K, including no space.

EXTENDED STORAGE
If the extended storage configuration is turned on,[70] pages that are being migrated out of this buffer pool will be cached in the extended storage.

NOT EXTENDED STORAGE
Even if the database extended storage configuration is turned on, pages that are being migrated out of this buffer pool, will NOT be cached in the extended storage.

Notes

- Until the next time the database is started, any table space that is created will use an already active buffer pool of the same page size. The database has to be restarted for the table space assignment to the new buffer pool to take effect.
- There should be enough real memory on the machine for the total of all the buffer pools, as well as for the rest of the database manager and application requirements. If DB2 is unable to obtain the total memory for all buffer pools, it will attempt to start up only the default buffer pool. If this is unsuccessful, it will start up a minimal default buffer pool. In either of these cases, a warning will be returned to the user (SQLSTATE 01626) and the pages from all table spaces will use the default buffer pool.

70. Extended storage configuration is turned on by setting the database configuration parameters NUM_ESTORE_SEGS and ESTORE_SEG_SIZE to non-zero values. See *Administration Guide* for details.

CREATE DISTINCT TYPE

The CREATE DISTINCT TYPE statement defines a distinct type. The distinct type is always sourced on one of the built-in data types. Successful execution of the statement also generates functions to cast between the distinct type and its source type and, optionally, generates support for the comparison operators (=, <>, <, <=, >, and >=) for use with the distinct type.

Invocation

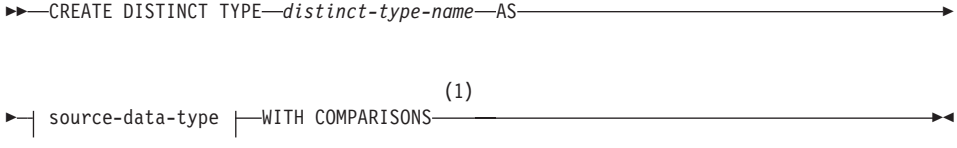
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization

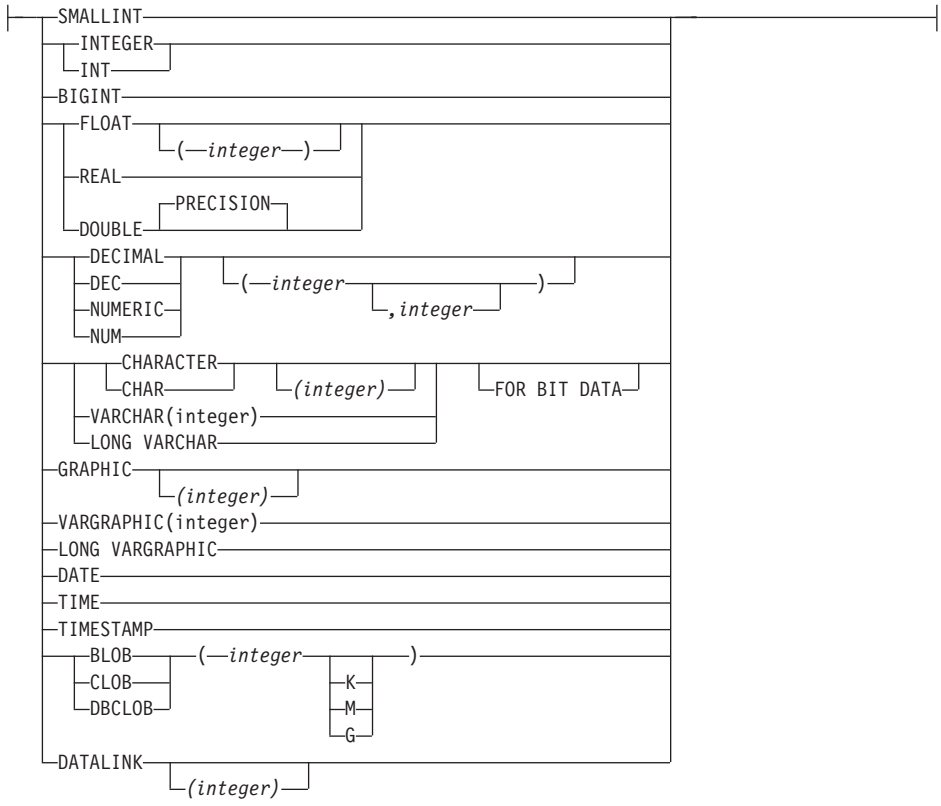
The privileges held by the authorization ID of the statement must include as least one of the following:

- SYSADM or DBADM authority
- IMPLICIT_SCHEMA authority on the database, if the schema name of the distinct type does not refer to an existing schema.
- CREATEIN privilege on the schema, if the schema name of the distinct type refers to an existing schema.

Syntax



source-data-type:



Notes:

1 Required for all source-data-types except LOBs, LONG VARCHAR, LONG VARGRAPHIC and DATALINK which are not supported.

Description

distinct-type-name
Names the distinct type. The name, including the implicit or explicit qualifier must not identify a distinct type described in the catalog. The unqualified name must not be the same as the name of a source-data-type or BOOLEAN (SQLSTATE 42918).

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The schema name (implicit or explicit) must not be greater than 8 bytes (SQLSTATE 42622).

A number of names used as keywords in predicates are reserved for system use, and may not be used as a *distinct-type-name*. The names are SOME, ANY, ALL, NOT, AND, OR, BETWEEN, NULL, LIKE, EXISTS, IN, UNIQUE, OVERLAPS, SIMILAR, MATCH and the comparison operators as described in "Basic Predicate" on page 187. Failure to observe this rule will lead to an error (SQLSTATE 42939).

If a two-part *distinct-type-name* is specified, the schema name cannot begin with "SYS"; otherwise, an error (SQLSTATE 42939) is raised.

source-data-type
Specifies the data type used as the basis for the internal representation of the distinct type. For information about the association of distinct types with other data types, see "Distinct Types" on page 87. For information about data types, see "CREATE TABLE" on page 712.

WITH COMPARISONS
Specifies that system-generated comparison operators are to be created for comparing two instances of a distinct type. These keywords should not be specified if the source-data-type is BLOB, CLOB, DBCLOB, LONG VARCHAR, LONG VARGRAPHIC, or DATALINK, otherwise a warning will be returned (SQLSTATE 01596) and the comparison operators will not be generated. For all other source-data-types, the WITH COMPARISONS keywords are required.

Notes

- Creating a distinct type with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- The following functions are generated to cast to and from the source type:
  - One function to convert from the distinct type to the source type
  - One function to convert from the source type to the distinct type
  - One function to convert from INTEGER to the distinct type if the source type is SMALLINT
  - one function to convert from VARCHAR to the distinct type if the source type is CHAR
  - one function to convert from VARGRAPHIC to the distinct type if the source type is GRAPHIC.

In general these functions will have the following format:

```
CREATE FUNCTION source-type-name (distinct-type-name)
   RETURNS source-type-name ...

CREATE FUNCTION distinct-type-name (source-type-name)
   RETURNS distinct-type-name ...
```

In cases in which the source type is a parameterized type, the function to convert from the distinct type to the source type will have as function name the name of the source type without the parameters (see Table 20 for details). The type of the return value of this function will include the parameters given on the CREATE DISTINCT TYPE statement. The function to convert from the source type to the distinct type will have an input parameter whose type is the source type including its parameters. For example,

```
CREATE DISTINCT TYPE T_SHOESIZE AS CHAR(2)
          WITH COMPARISONS

CREATE DISTINCT TYPE T_MILES AS DOUBLE
          WITH COMPARISONS
```

will generate the following functions:

```
FUNCTION CHAR (T_SHOESIZE) RETURNS CHAR (2)

FUNCTION T_SHOESIZE (CHAR (2))
RETURNS T_SHOESIZE

FUNCTION DOUBLE (T_MILES) RETURNS DOUBLE

FUNCTION T_MILES (DOUBLE) RETURNS T_MILES
```

The schema of the generated cast functions is the same as the schema of the distinct type. No other function with this name and with the same signature may already exist in the database (SQLSTATE 42710).

The following table gives the names of the functions to convert from the distinct type to the source type and from the source type to the distinct type for all predefined data types.

Table 20. CAST functions on distinct types

Source Type Name	Function Name	Parameter	Return-type
CHAR	<distinct>	CHAR (n)	<distinct>
	CHAR	<distinct>	CHAR (n)
	<distinct>	VARCHAR (n)	<distinct>
VARCHAR	<distinct>	VARCHAR (n)	<distinct>
	VARCHAR	<distinct>	VARCHAR (n)
LONG VARCHAR	<distinct>	LONG VARCHAR	<distinct>
	LONG_VARCHAR	<distinct>	LONG VARCHAR
CLOB	<distinct>	CLOB (n)	<distinct>
	CLOB	<distinct>	CLOB (n)
BLOB	<distinct>	BLOB (n)	<distinct>
	BLOB	<distinct>	BLOB (n)

Table 20. CAST functions on distinct types (continued)

Source Type Name	Function Name	Parameter	Return-type
GRAPHIC	<distinct>	GRAPHIC (n)	<distinct>
	GRAPHIC	<distinct>	GRAPHIC (n)
	<distinct>	VARGRAPHIC (n)	<distinct>
VARGRAPHIC	<distinct>	VARGRAPHIC (n)	<distinct>
	VARGRAPHIC	<distinct>	VARGRAPHIC (n)
LONG VARGRAPHIC	<distinct>	LONG VARGRAPHIC	<distinct>
	LONG_VARGRAPHIC	<distinct>	LONG VARGRAPHIC
DBCLOB	<distinct>	DBCLOB (n)	<distinct>
	DBCLOB	<distinct>	DBCLOB (n)
SMALLINT	<distinct>	SMALLINT	<distinct>
	<distinct>	INTEGER	<distinct>
	SMALLINT	<distinct>	SMALLINT
INTEGER	<distinct>	INTEGER	<distinct>
	INTEGER	<distinct>	INTEGER
BIGINT	<distinct>	BIGINT	<distinct>
	BIGINT	<distinct>	BIGINT
DECIMAL	<distinct>	DECIMAL (p,s)	<distinct>
	DECIMAL	<distinct>	DECIMAL (p,s)
NUMERIC	<distinct>	DECIMAL (p,s)	<distinct>
	DECIMAL	<distinct>	DECIMAL (p,s)
REAL	<distinct>	REAL	<distinct>
	<distinct>	DOUBLE	<distinct>
	REAL	<distinct>	REAL
FLOAT(*n*) where *n*<=24	<distinct>	REAL	<distinct>
	<distinct>	DOUBLE	<distinct>
	REAL	<distinct>	REAL
FLOAT(*n*) where *n*>24	<distinct>	DOUBLE	<distinct>
	DOUBLE	<distinct>	DOUBLE
FLOAT	<distinct>	DOUBLE	<distinct>
	DOUBLE	<distinct>	DOUBLE
DOUBLE	<distinct>	DOUBLE	<distinct>
	DOUBLE	<distinct>	DOUBLE

Table 20. CAST functions on distinct types (continued)

Source Type Name	Function Name	Parameter	Return-type
DOUBLE PRECISION	<distinct>	DOUBLE	<distinct>
	DOUBLE	<distinct>	DOUBLE
DATE	<distinct>	DATE	<distinct>
	DATE	<distinct>	DATE
TIME	<distinct>	TIME	<distinct>
	TIME	<distinct>	TIME
TIMESTAMP	<distinct>	TIMESTAMP	<distinct>
	TIMESTAMP	<distinct>	TIMESTAMP
DATALINK	<distinct>	DATALINK	<distinct>
	DATALINK	<distinct>	DATALINK

Note: NUMERIC and FLOAT are not recommended when creating a user-defined type for a portable application. DECIMAL and DOUBLE should be used instead.

The functions described in the above table are the only functions that are generated automatically when distinct types are defined. Consequently, none of the built-in functions (AVG, MAX, LENGTH, etc.) are supported on distinct types until the CREATE FUNCTION statement (see "CREATE FUNCTION" on page 589) is used to register user-defined functions for the distinct type, where those user-defined functions are sourced on the appropriate built-in functions. In particular, note that it is possible to register user-defined functions that are sourced on the built-in column functions.

When a distinct type is created using the WITH COMPARISONS clause, system-generated comparison operators are created. Creation of these comparison operators will generate entries in the SYSCAT.FUNCTIONS catalog view for the new functions.

The schema name of the distinct type must be included in the SQL path (see "SET PATH" on page 1031 or the FUNCPATH BIND option as described in the *Application Development Guide*) for successful use of these operators and cast functions in SQL statements.

Examples

Example 1: Create a distinct type named SHOESIZE that is based on an INTEGER data type.

```
CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS
```

This will also result in the creation of comparison operators (=, <>, <, <=, >, >=) and cast functions INTEGER(SHOESIZE) returning INTEGER and SHOESIZE(INTEGER) returning SHOESIZE.

Example 2: Create a distinct type named MILES that is based on a DOUBLE data type.

```
CREATE DISTINCT TYPE MILES AS DOUBLE WITH COMPARISONS
```

This will also result in the creation of comparison operators (=, <>, <, =, >, >=) and cast functions DOUBLE(MILES) returning DOUBLE and MILES(DOUBLE) returning MILES.

CREATE EVENT MONITOR

The CREATE EVENT MONITOR statement defines a monitor that will record certain events that occur when using the database. The definition of each event monitor also specifies where the database should record the events.

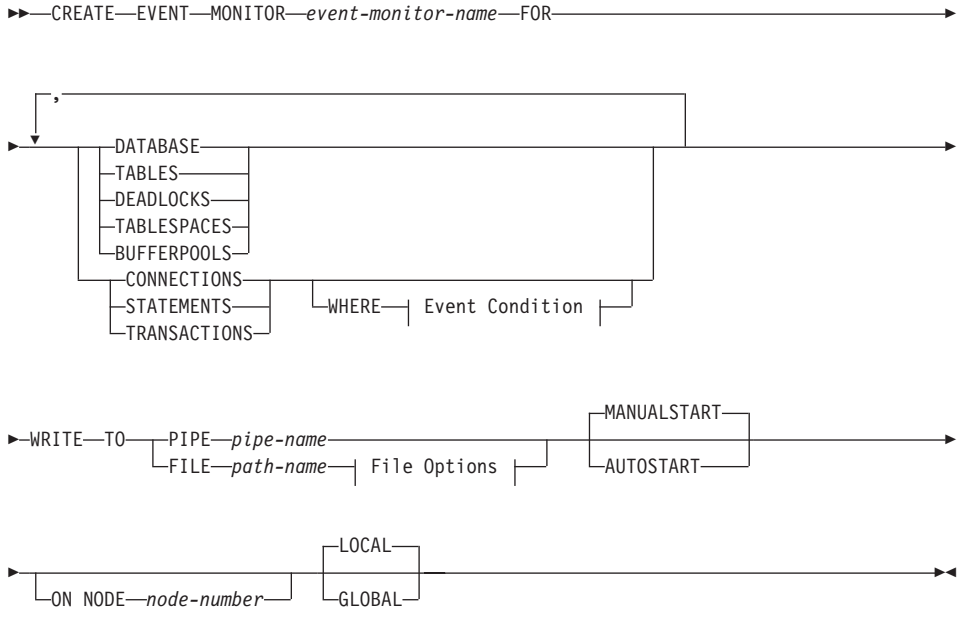
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

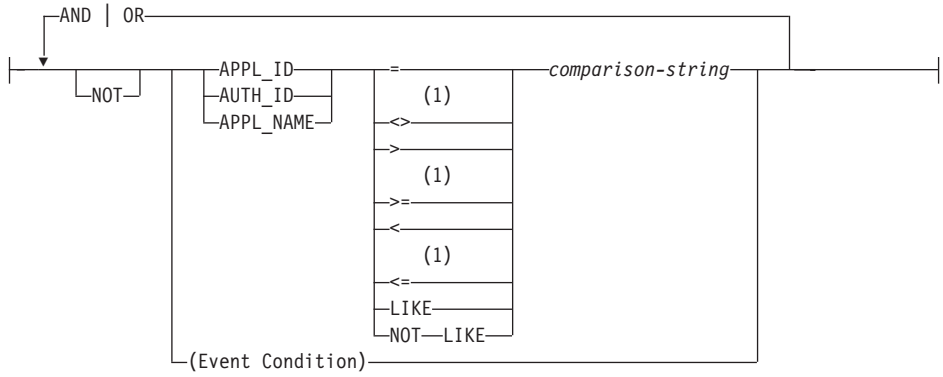
Authorization

The privileges held by the authorization ID must include either SYSADM or DBADM authority (SQLSTATE 42502).

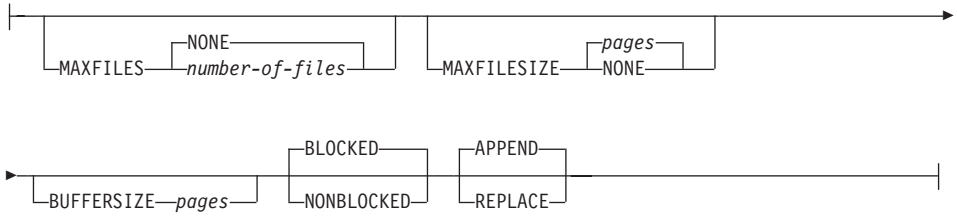
Syntax



Event Condition:



File Options:



Notes:

1 Other forms of these operators are also supported. See "Basic Predicate" on page 187 for more details.

Description

event-monitor-name
Names the event monitor. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *event-monitor-name* must not identify an event monitor that already exists in the catalog (SQLSTATE 42710).

FOR
Introduces the type of event to record.

DATABASE
Specifies that the event monitor records a database event when the last application disconnects from the database.

TABLES
Specifies that the event monitor records a table event for each active table when the last application disconnects from the database. An active table is a table that has changed since the first connection to the database.

DEADLOCKS
Specifies that the event monitor records a deadlock event whenever a deadlock occurs.

TABLESPACES
Specifies that the event monitor records a table space event for each table space when the last application disconnects from the database.

BUFFERPOOLS
Specifies that the event monitor records a buffer pool event when the last application disconnects from the database.

CONNECTIONS
Specifies that the event monitor records a connection event when an application disconnects from the database.

STATEMENTS
Specifies that the event monitor records a statement event whenever a SQL statement finishes executing.

TRANSACTIONS
Specifies that the event monitor records a transaction event whenever a transaction completes (that is, whenever there is a commit or rollback operation).

WHERE *event condition*
Defines a filter that determines which connections cause a CONNECTION, STATEMENT or TRANSACTION event to occur. If the result of the event condition is TRUE for a particular connection, then that connection will generate the requested events.

This clause is a special form of the WHERE clause that should not be confused with a standard search condition.

To determine if an application will generate events for a particular event monitor, the WHERE clause is evaluated:

1. For each active connection when an event monitor is first turned on.
2. Subsequently for each new connection to the database at connect time.

The WHERE clause is not evaluated for each event.

If no WHERE clause is specified then all events of the specified event type will be monitored.

APPL_ID
Specifies that the application ID of each connection should be compared with the *comparison-string* in order to determine if the

connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

AUTH_ID
Specifies that the authorization ID of each connection should be compared with the *comparison-string* in order to determine if the connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

APPL_NAME
Specifies that the application program name of each connection should be compared with the *comparison-string* in order to determine if the connection should generate CONNECTION, STATEMENT or TRANSACTION events (whichever was specified).

The application program name is the first 20 bytes of the application program file name, after the last path separator.

comparison-string
A string to be compared with the APPL_ID, AUTH_ID, or APPL_NAME of each application that connects to the database. *comparison-string* must be a string constant (that is, host variables and other string expressions are not permitted).

WRITE TO
Introduces the target for the data.

PIPE
Specifies that the target for the event monitor data is a named pipe. The event monitor writes the data to the pipe in a single stream (that is, as if it were a single, infinitely long file). When writing the data to a pipe, an event monitor does not perform blocked writes. If there is no room in the pipe buffer, then the event monitor will discard the data. It is the monitoring application's responsibility to read the data promptly if it wishes to ensure no data loss.

pipe-name
The name of the pipe (FIFO on AIX) to which the event monitor will write the data.

The naming rules for pipes are platform specific. On UNIX operating systems pipe names are treated like file names. As a result, relative pipe names are permitted, and are treated like relative path-names (see *path-name* below). However, on OS/2, Windows 95 and Windows NT, there is a special syntax for a pipe name. As a result, on OS/2, Windows 95 and Windows NT absolute pipe names are required.

The existence of the pipe will not be checked at event monitor creation time. It is the responsibility of the monitoring application to have created and opened the pipe for reading at the time that the event monitor is activated. If the pipe is not available at this time, then the event monitor will turn itself off, and will log an error. (That is, if the event monitor was activated at database start time as a result of the AUTOSTART option, then the event monitor will log an error in the system error log.) If the event monitor is activated via the SET EVENT MONITOR STATE SQL statement, then that statement will fail (SQLSTATE 58030).

FILE

Indicates that the target for the event monitor data is a file (or set of files). The event monitor writes out the stream of data as a series of 8 character numbered files, with the extension "evt". (for example, 00000000.evt, 00000001.evt, and 00000002.evt). The data should be considered to be one logical file even though the data is broken up into smaller pieces (that is, the start of the data stream is the first byte in the file 00000000.evt; the end of the data stream is the last byte in the file nnnnnnnn.evt).

The maximum size of each file can be defined as well as the maximum number of files. An event monitor will never split a single event record across two files. However, an event monitor may write related records in two different files. It is the responsibility of the application that uses this data to keep track of such related information when processing the event files.

path-name

The name of the directory in which the event monitor should write the event files data. The path must be known at the server, however, the path itself could reside on another partition or node (for example, in a UNIX-based system, this might be an NFS mounted file). A string constant must be used when specifying the *path-name*.

The directory does not have to exist at CREATE EVENT MONITOR time. However, a check is made for the existence of the target path when the event monitor is activated. At that time, if the target path does not exist, an error (SQLSTATE 428A3) is raised.

If an absolute path (a path that starts with the root directory on AIX, or a disk identifier on OS/2, Windows 95 and Windows NT) is specified, then the specified path will be the one used. If a relative path (a path that does not start with the root) is specified, then the path relative to the DB2EVENT directory in the database directory will be used.

When a relative path is specified, the DB2EVENT directory is used to convert it into an absolute path. Thereafter, no distinction is made between absolute and relative paths. The absolute path is stored in the SYSCAT.EVENTMONITORS catalog view.

It is possible to specify two or more event monitors that have the same target path. However, once one of the event monitors has been activated for the first time, and as long as the target directory is not empty, it will be impossible to activate any of the other event monitors.

File Options
Specifies the options for the file format.

MAXFILES NONE
Specifies that there is no limit to the number of event files that the event monitor will create. This is the default.

MAXFILES *number-of-files*
Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Whenever an event monitor has to create another file, it will check to make sure that the number of .evt files in the directory is less than *number-of-files*. If this limit has already been reached, then the event monitor will turn itself off.

If an application removes the event files from the directory after they have been written, then the total number of files that an event monitor can produce can exceed *number-of-files*. This option has been provided to allow a user to guarantee that the event data will not consume more than a specified amount of disk space.

MAXFILESIZE *pages*
Specifies that there is a limit to the size of each event monitor file. Whenever an event monitor writes a new event record to a file, it checks that the file will not grow to be greater than *pages* (in units of 4K pages). If the resulting file would be too large, then the event monitor switches to the next file. The default for this option is:

- OS/2, Windows 95 and Windows NT - 200 4K pages
- UNIX - 1000 4K pages

The number of pages must be greater than at least the size of the event buffer in pages. If this requirement is not met, then an error (SQLSTATE 428A4) is raised.

MAXFILESIZE NONE
Specifies that there is no set limit on a file's size. If MAXFILESIZE NONE is specified, then MAXFILES 1 must also be specified. This option means that one file will contain all of the event data for a particular event monitor. In this case the only event file will be 00000000.evt.

BUFFERSIZE *pages*
Specifies the size of the event monitor buffers (in units of 4K pages). All event monitor file I/O is buffered to improve the performance of the event monitors. The larger the buffers, the less I/O will be performed by the event monitor. Highly active event monitors should have larger buffers than relatively inactive event monitors. When the monitor is started, two buffers of the specified size are allocated. Event monitors use double buffering to permit asynchronous I/O.

The minimum and default size of each buffer (if this option is not specified) is 4 pages (that is, 2 buffers, each 16 K in size). The maximum size of the buffers is limited by the size of the monitor heap (MON_HEAP) since the buffers are allocated from the heap. If using a lot of event monitors at the same time, increase the size of the MON_HEAP database configuration parameter.

Event monitors that write their data to a pipe also have two internal (non-configurable) buffers that are each 1 page in size. These buffers are also allocated from the monitor heap (MON_HEAP). For each active event monitor that has a pipe target, increase the size of the database heap by 2 pages.

BLOCKED
Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. BLOCKED should be selected to guarantee no event data loss. This is the default option.

NONBLOCKED
Specifies that each agent that generates an event should not wait for the event buffer to be written out to disk if the agent determines that both event buffers are full. NONBLOCKED event monitors do not slow down database operations to the extent of BLOCKED event

monitors. However, NONBLOCKED event monitors are subject to data loss on highly active systems.

APPEND
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. When the event monitor is reactivated, it will resume writing to the event files as if it had never been turned off. APPEND is the default option.

The APPEND option does not apply at CREATE EVENT MONITOR time, if there is existing event data in the directory where the newly created event monitor is to write its event data.

REPLACE
Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will erase all of the event files and start writing data to file 00000000.evt.

MANUALSTART
Specifies that the event monitor not be started automatically each time the database is started. Event monitors with the MANUALSTART option must be activated manually using the SET EVENT MONITOR STATE statement. This is the default option.

AUTOSTART
Specifies that the event monitor be started automatically each time the database is started.

ON NODE
Keyword that indicates that specific partitions are specified.

node-number
Specifies a partition number where the event monitor runs and write the events. With the monitoring scope defined as GLOBAL, all partitions report to the specified partition number. The I/O component will physically run on the specified partition, writing its records to /tmp/dlocks directory on that partition.

GLOBAL
Event monitor reports from all partitions. For a partitioned database in DB2 Universal Database Version 7, only deadlock event monitors can be defined as GLOBAL. The global event monitor will report deadlocks for all nodes in the system.

LOCAL
Event monitor reports only on the partition that is running. It gives a partial trace of the database activity. This is the default.

Rules

- Each of the event types (DATABASE, TABLES, DEADLOCKs,...) can only be specified once in a particular event monitor definition.

Notes

- Event monitor definitions are recorded in the SYSCAT.EVENTMONITORS catalog view. The events themselves are recorded in the SYSCAT.EVENTS catalog view.
- For detailed information on using the database monitor and on interpreting data from pipes and files, see the *System Monitor Guide and Reference*.

Examples

Example 1: The following example creates an event monitor called SMITHPAY. This event monitor, will collect event data for the database as well as for the SQL statements performed by the PAYROLL application owned by the JSMITH authorization ID. The data will be appended to the absolute path /home/jsmith/event/smithpay/. A maximum of 25 files will be created. Each file will be a maximum of 1 024 4K pages long. The file I/O will be non-blocked.

```
CREATE EVENT MONITOR SMITHPAY
   FOR DATABASE, STATEMENTS
   WHERE APPL_NAME = 'PAYROLL' AND AUTH_ID = 'JSMITH'
   WRITE TO FILE '/home/jsmith/event/smithpay'
   MAXFILES 25
   MAXFILESIZE 1024
   NONBLOCKED
   APPEND
```

Example 2: The following example creates an event monitor called DEADLOCKS_EVTS. This event monitor will collect deadlock events and will write them to the relative path DLOCKS. One file will be written, and there is no maximum file size. Each time the event monitor is activated, it will append the event data to the file 00000000.evt if it exists. The event monitor will be started each time the database is started. The I/0 will be blocked by default.

```
CREATE EVENT MONITOR DEADLOCK_EVTS
   FOR DEADLOCKS
   WRITE TO FILE 'DLOCKS'
   MAXFILES 1
   MAXFILESIZE NONE
   AUTOSTART
```

Example 3: This example creates an event monitor called DB_APPLS. This event monitor collects connection events, and writes the data to the named pipe /home/jsmith/applpipe.

```
CREATE EVENT MONITOR DB_APPLS
   FOR CONNECTIONS
   WRITE TO PIPE '/home/jsmith/applpipe'
```