SET PASSTHRU

The SET PASSTHRU statement opens and closes a session for submitting a data source's native SQL directly to that data source. The statement is not under transaction control.

Invocation

This statement can be issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privileges held by the authorization ID of the statement must provide authorization to:

- Pass through to the data source.
- Satisfy security measures at the data source.

Syntax

Description

server-name
Names the data source for which a pass-through session is to be opened. *server-name* must identify a data source that is described in the catalog.

RESET
Closes a pass-through session.

Notes

Refer to "Pass-Through Facility Processing" on page 1256 for guidelines and restrictions on using pass-through.

Examples

Example 1: Start a pass-through session to data source BACKEND.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
```

Example 2: Start a pass-through session with a PREPARE statement.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL PREPARE STMT FROM :PASS_THRU;
EXEC SQL EXECUTE STMT;
```

Example 3: End a pass-through session.

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

Example 4: Use the PREPARE and EXECUTE statements to end a pass-through session.

```
strcpy (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL PREPARE STMT FROM :PASS_THRU_RESET;
EXEC SQL EXECUTE STMT;
```

Example 5: Open a session to pass through to a data source, create a clustered index for a table at this data source, and close the pass-through session.

```
strcpy (PASS_THRU,"SET PASSTHRU BACKEND");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU;
EXEC SQL PREPARE STMT                pass-through mode
   FROM "CREATE UNIQUE
         CLUSTERED INDEX TABLE_INDEX
         ON USER2.TABLE              table is not an
         WITH IGNORE DUP KEY";        alias
EXEC SQL EXECUTE STMT;
STRCPY (PASS_THRU_RESET,"SET PASSTHRU RESET");
EXEC SQL EXECUTE IMMEDIATE :PASS_THRU_RESET;
```

SET PATH

The SET PATH statement changes the value of the CURRENT PATH special register. It is not under transaction control.

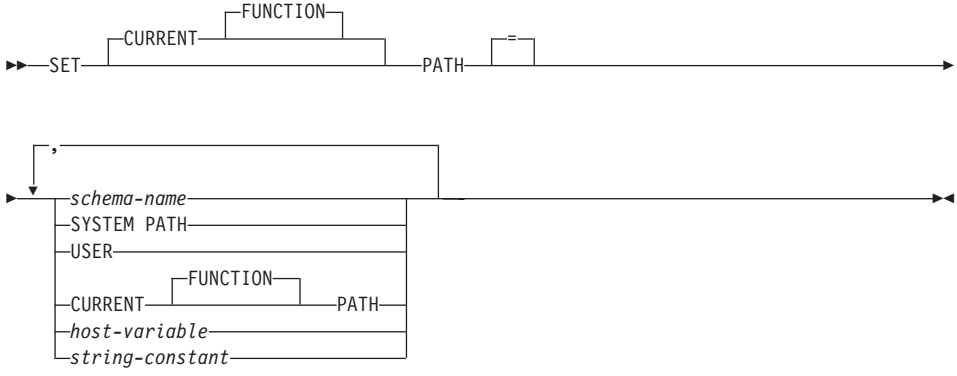
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

No authorization is required to execute this statement.

Syntax



Description

schema-name
This one-part name identifies a schema that exists at the application server. No validation that the schema exists is made at the time that the path is set. If a *schema-name* is, for example, misspelled, it will not be caught, and it could affect the way subsequent SQL operates.

SYSTEM PATH
This value is the same as specifying the schema names *"SYSIBM","SYSFUN"*.

USER
The value in the USER special register.

CURRENT PATH
The value of the CURRENT PATH before the execution of this statement. CURRENT FUNCTION PATH may also be specified.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 30 bytes (SQLSTATE 42815). It cannot be set

to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left justified. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 8.

Rules

- A schema name cannot appear more than once in the function path (SQLSTATE 42732).
- The number of schemas that can be specified is limited by the total length of the CURRENT PATH special register. The special register string is built by taking each schema name specified and removing trailing blanks, delimiting with double quotes, doubling quotes within the schema name as necessary, and then separating each schema name by a comma. The length of the resulting string cannot exceed 254 bytes (SQLSTATE 42907).

Notes

- The initial value of the CURRENT PATH special register is "SYSIBM","SYSFUN","X" where X is the value of the USER special register.
- The schema SYSIBM does not need to be specified. If it is not included in the SQL path, it is implicitly assumed as the first schema (in this case, it is not included in the CURRENT PATH special register).
- The CURRENT PATH special register specifies the SQL path used to resolve user-defined data types, procedures and functions in dynamic SQL statements. The FUNCPATH bind option specifies the SQL path to be used for resolving user-defined data types and functions in static SQL statements. See the *Command Reference* for further information on the use of FUNCPATH option in BIND command.

Example

Example 1: The following statement sets the CURRENT FUNCTION PATH special register.

```
SET PATH = FERMAT, "McDrw #8", SYSIBM
```

Example 2: The following example retrieves the current value of the CURRENT PATH special register into the host variable called CURPATH.

```
EXEC SQL VALUES (CURRENT PATH) INTO :CURPATH;
```

The value would be "FERMAT","McDrw #8","SYSIBM" if set by the previous example.

SET SCHEMA

The SET SCHEMA statement changes the value of the CURRENT SCHEMA special register. It is not under transaction control. If the package is bound with DYNAMICRULES BIND option, this statement has no effect.

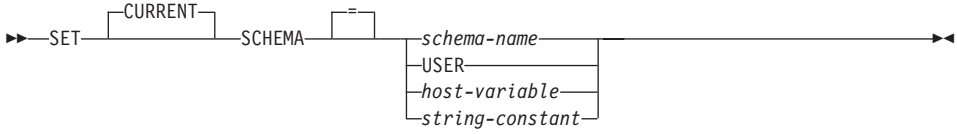
Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

No authorization is required to execute this statement.

Syntax



Description

schema-name
This one-part name identifies a schema that exists at the application server. The length must not exceed 30 bytes (SQLSTATE 42815). No validation that the schema exists is made at the time that the schema is set. If a *schema-name* is misspelled, it will not be caught, and it could affect the way subsequent SQL operates.

USER
The value in the USER special register.

host-variable
A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 30 (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left justified. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 30.

Rules

- If the value specified does not conform to the rules for a *schema-name*, an error (SQLSTATE 3F000) is raised.

- The value of the CURRENT SCHEMA special register is used as the schema name in all dynamic SQL statements, with the exception of the CREATE SCHEMA statement, where an unqualified reference to a database object exists.
- The QUALIFIER bind option specifies the schema name for use as the qualifier for unqualified database object names in static SQL statements (see the *Command Reference* for further information on use of the QUALIFIER option).

Notes

- The initial value of the CURRENT SCHEMA special register is equivalent to USER.
- Setting the CURRENT SCHEMA special register does not effect the CURRENT PATH special register. Hence, the CURRENT SCHEMA will not be included in the SQL path and functions, procedures and user-defined type resolution may not find these objects. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.
- CURRENT SQLID is accepted as a synonym for CURRENT SCHEMA and the effect of a SET CURRENT SQLID statement will be identical to that of a SET CURRENT SCHEMA statement. No other effects, such as statement authorization changes, will occur.

Examples

Example 1: The following statement sets the CURRENT SCHEMA special register.

```
SET SCHEMA RICK
```

Example 2: The following example retrieves the current value of the CURRENT SCHEMA special register into the host variable called CURSCHEMA.

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

The value would be RICK, set by the previous example.

SET SERVER OPTION

The SET SERVER OPTION statement specifies a server option setting that is to remain in effect while a user or application is connected to the federated database. When the connection ends, this server option's previous setting is reinstated. This statement is not under transaction control.

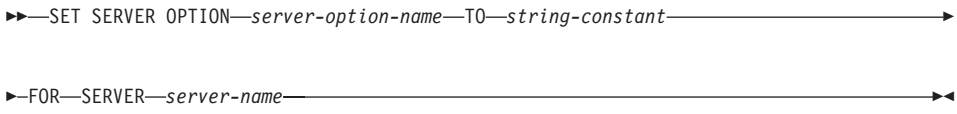
Invocation

This statement can be issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The authorization ID of the statement must have either SYSADM or DBADM authority on the federated database.

Syntax



Description

server-option-name
Names the server option that is to be set. Refer to"Server Options" on page 1249 for descriptions of the server options.

TO *string-constant*
Specifies the setting for *server-option-name* as a character string constant. Refer to "Server Options" on page 1249 for descriptions of possible settings.

SERVER *server-name*
Names the data source to which *server-option-name* applies. It must be a server described in the catalog.

Notes

- Server option names can be entered in uppercase or lowercase.
- SET SERVER OPTION currently only supports the password, fold_id, and fold_pw server options.
- One or more SET SERVER OPTION statements can be submitted when a user or application connects to the federated database. The statement (or statements) must be specified at the start of the first unit of work that is processed after the connection is established.

Examples

Example 1: An Oracle data source called RATCHIT is defined to a federated database called DJDB. RATCHIT is configured to disallow plan hints. However, the DBA would like plan hints to be enabled for a test run of a new application. When the run is over, plan hints will be disallowed again.

```
CONNECT TO DJDB;
strcpy(stmt,"set server option plan_hints to 'Y' for server ratchit");
EXEC SQL EXECUTE IMMEDIATE :stmt;
strcpy(stmt,"select c1 from ora_t1 where c1 > 100"); /*Generate plan hints*/
EXEC SQL PREPARE s1 FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO :hv;
```

Example 2: You have set the server option PASSWORD to 'Y' (yes, validate passwords at the data source) for all Oracle 8 data sources. However, for a particular session in which an application is connected to the federated database in order to access a specific Oracle 8 data source—one defined to the federated database DJDB as ORA8A—passwords will not need to be validated.

```
CONNECT TO DJDB;
strcpy(stmt,"set server option password to 'N' for server ora8a");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL EXECUTE STMT_NAME FROM :stmt;
strcpy(stmt,"select max(c1) from ora8a_t1");
EXEC SQL PREPARE STMT_NAME FROM :stmt;
EXEC SQL DECLARE c1 CURSOR FOR STMT_NAME;
EXEC SQL OPEN c1; /*Does not validate password at ora8a*/
EXEC SQL FETCH c1 INTO :hv;
```

SET transition-variable

The SET transition-variable statement assigns values to new transition variables. It is under transaction control.

Invocation

This statement can only be used as a triggered SQL statement in the triggered action of a BEFORE trigger whose granularity is FOR EACH ROW (see "CREATE TRIGGER" on page 780).

Authorization

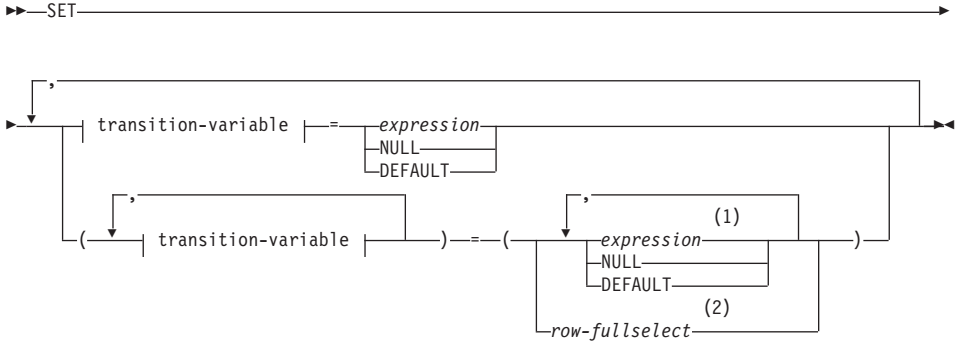
The privileges held by the authorization ID of the creator of the trigger must include at least one of the following:

- UPDATE of the columns referenced on the left hand side of the assignment and SELECT for any columns referenced on the right hand side.
- CONTROL privilege on the table (subject table of the trigger)
- SYSADM or DBADM authority.

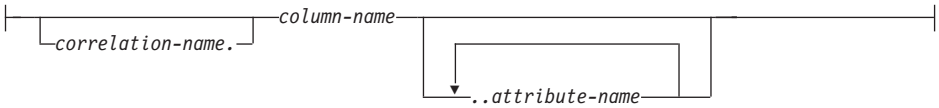
To execute this statement with a *row-fullselect* as the right hand side of the assignment, the privileges held by the authorization ID of the creator of the trigger must also include at least one of the following for each table or view referenced:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM.

Syntax



transition-variable:



Notes:

1 The number of expressions, NULLs and DEFAULTs must match the number of *transition-variable*s.

2 The number of columns in the select list must match the number of *transition-variable*s.

Description

transition-variable
Identifies a column in the set of affected rows for the trigger.

correlation-name
The *correlation-name* given for referencing the NEW transition variables. This *correlation-name* must match the correlation name specified following NEW in the REFERENCING clause of the CREATE TRIGGER.

If OLD is not specified in the REFERENCING clause, the *correlation-name* will default to the *correlation-name* specified following NEW. If both NEW and OLD are specified in the REFERENCING clause, then a *correlation-name* is required with each *column-name* (SQLSTATE 42702).

column-name
Identifies the column to be updated. The *column-name* must identify a column of the subject table of the trigger (SQLSTATE 42703). A column must not be specified more than once (SQLSTATE 42701).

..attribute name
Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*). The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The *attribute-name* must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute name* clause is referred to as a conventional assignment.

expression
Indicates the new value of the column. The expression is any expression of the type described in "Expressions" on page 157. The expression can not include a column function except when it occurs within a scalar fullselect (SQLSTATE 42903). An *expression* may contain references to OLD and NEW transition variables and must be qualified by the *correlation-name* to specify which transition variable (SQLSTATE 42702).

NULL
Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9), unless it was specifically cast to the data type of the attribute.

DEFAULT
Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined using the WITH DEFAULT clause, then the value is set to the default defined for the column (see default-clause in "ALTER TABLE" on page 477).
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined without specifying the WITH DEFAULT clause, the IDENTITY clause, or the NOT NULL clause, then the value is NULL.
- If the column was defined using the NOT NULL clause and the IDENTITY clause is not used, or the WITH DEFAULT clause was not used or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of column-names specified for assignment. The values are assigned to each corresponding column-name. If the result of the row-fullselect is no rows, then null values are assigned. A *row-fullselect* may contain references to OLD and NEW transition variables which must be qualified by the *correlation-name* to specify which transition variable to use. (SQLSTATE 42702). An error is returned if there is more than one row in the result (SQLSTATE 21000).

Rules

- The number of values to be assigned from expressions, NULLs and DEFAULTs or the *row-fullselect* must match the number of columns specified for assignment (SQLSTATE 42802).
- If the statement is used in a BEFORE UPDATE trigger, the *column-name* specified as a *transition-variable* cannot be a partitioning key column (SQLSTATE 42997).

Notes

- If more than one assignment is included, all the *expression*s and *row-fullselect*s are evaluated before the assignments are performed. Thus

references to columns in an expression or row fullselect are always the value of the transition variable prior to any assignment in the single SET transition-variable statement.

- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column.[106]
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

```
SET NEW.EMPNO = DEFAULT
```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.
- See "INSERT" on page 938 for more information on consuming values of a generated sequence for an identity column.
- See "INSERT" on page 938 for more information on exceeding the maximum value for an identity column.

Examples

Example 1: Set the salary column of the row for which the trigger action is currently executing to 50000.

```
SET NEW_VAR.SALARY = 50000;
or
SET (NEW_VAR.SALARY) = (50000);
```

Example 2: Set the salary and the commission column of the row for which the trigger action is currently executing to 50000 and 8000 respectively.

```
SET NEW_VAR.SALARY = 50000, NEW_VAR.COMM = 8000;
or
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (50000, 8000);
```

Example 3: Set the salary and the commission column of the row for which the trigger action is currently executing to the average of the salary and of the commission of the employees of the updated row's department respectively.

```
SET (NEW_VAR.SALARY, NEW_VAR.COMM)
   = (SELECT AVG(SALARY), AVG(COMM)
        FROM EMPLOYEE E
        WHERE E.WORKDEPT = NEW_VAR.WORKDEPT);
```

Example 4: Set the salary and the commission column of the row for which the trigger action is currently executing to 10000 and the original value of salary respectively (i.e., before the SET statement was executed).

```
SET NEW_VAR.SALARY = 10000, NEW_VAR.COMM = NEW_VAR.SALARY;
or
SET (NEW_VAR.SALARY, NEW_VAR.COMM) = (10000, NEW_VAR.SALARY);
```

106. There is no casting of the previous value to the source type prior to the computation.

SIGNAL SQLSTATE

The SIGNAL SQLSTATE statement is used to signal an error. It causes an error to be returned with the specified SQLSTATE and the specified *diagnostic-string*.

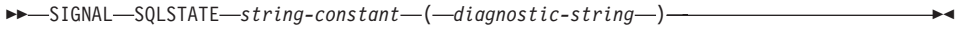
Invocation

The SIGNAL SQLSTATE statement can only be used as a triggered SQL statement within a trigger.

Authorization

No authorization is required to execute this statement.

Syntax



Description

string-constant

The specified *string-constant* represents an SQLSTATE. It must be a character string constant with exactly 5 characters that follow the rules for application-defined SQLSTATEs as follows:

- Each character must be from the set of digits ('0' through '9') or non-accented upper case letters ('A' through 'Z')
- The SQLSTATE class (first two characters) cannot be '00', '01' or '02' since these are not error classes.
- If the SQLSTATE class (first two characters) starts with the character '0' through '6' or 'A' through 'H', then the subclass (last three characters) must start with a letter in the range 'I' through 'Z'
- If the SQLSTATE class (first two characters) starts with the character '7', '8', '9' or 'I' though 'Z', then the subclass (last three characters) can be any of '0' through '9' or 'A' through 'Z'.

If the SQLSTATE does not conform to these rules an error occurs (SQLSTATE 428B3).

diagnostic-string

An expression with a type of CHAR or VARCHAR that returns a character string of up to 70 bytes that describes the error condition. If the string is longer than 70 bytes, it will be truncated.

Example

Consider an order system that records orders in an ORDERS table (ORDERNO, CUSTNO, PARTNO, QUANTITY) only if there is sufficient stock in the PARTS tables.

```
CREATE TRIGGER check_avail
       NO CASCADE BEFORE INSERT ON orders
       REFERENCING NEW AS new_order
       FOR EACH ROW MODE DB2SQL
       WHEN (new_order.quantity > (SELECT on_hand FROM parts
                                   WHERE new_order.partno=parts.partno))
         BEGIN ATOMIC
           SIGNAL SQLSTATE '75001' ('Insufficient stock for order');
         END
```

UPDATE

The UPDATE statement updates the values of specified columns in rows of a table or view. Updating a row of a view updates a row of its base table.

The forms of this statement are:

- The *Searched* UPDATE form is used to update one or more rows (optionally determined by a search condition).
- The *Positioned* UPDATE form is used to update exactly one row (as determined by the current position of a cursor).

Invocation

An UPDATE 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.

Authorization

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

- UPDATE privilege on the table or view where rows are to be updated
- UPDATE privilege on each of the columns to be updated.
- CONTROL privilege on the table or view where rows are to be updated
- SYSADM or DBADM authority.
- If a *row-fullselect* is included in the assignment, at least one of the following for each referenced table or view:
  - SELECT privilege
  - CONTROL privilege
  - SYSADM or DBADM authority.

For each table or view referenced by a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

When the package is precompiled with SQL92 rules [107] and the searched form of an UPDATE includes a reference to a column of the table or view in the

107. The package used to process the statement is precompiled using option LANGLEVEL with value SQL92E or MIA.

right side of the *assignment-clause* or anywhere in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

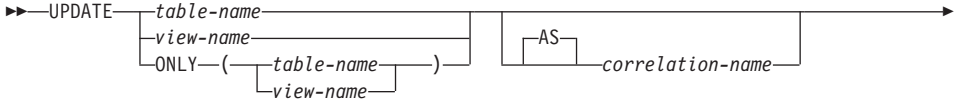
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority.

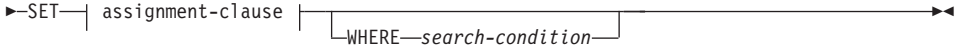
When the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

GROUP privileges are not checked for static UPDATE statements.

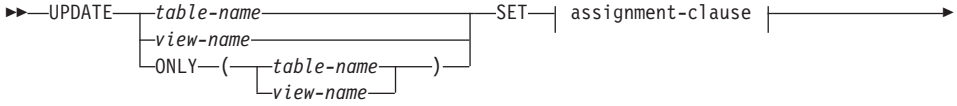
Syntax

Searched UPDATE:

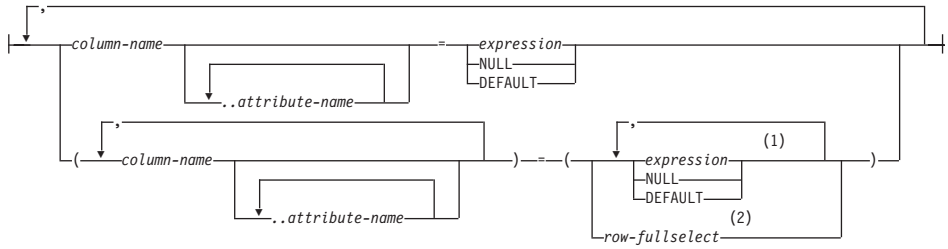




Positioned UPDATE:



assignment-clause:



Notes:

1 The number of expressions, NULLs and DEFAULTs must match the number of *column-name*s.

2 The number of columns in the select list must match the number of *column-name*s.

Description

table-name or *view-name*
Is the name of the table or view to be updated. The name must identify a table or view described in the catalog, but not a catalog table, a view of a catalog table (unless it is one of the updatable SYSSTAT views), a summary table, read-only view, or a nickname. (For an explanation of read-only views, see "CREATE VIEW" on page 823. For an explanation of updatable catalog views, see "Appendix D. Catalog Views" on page 1127.)

If *table-name* is a typed table, rows of the table or any of its proper subtables may get updated by the statement. Only the columns of the specified table may be set or referenced in the WHERE clause. For a positioned UPDATE, the associated cursor must also have specified the same table or view in the FROM clause without using ONLY.

ONLY (*table-name***)**
Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

ONLY (*view-name***)**
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be updated by the statement. For a positioned UPDATE,

the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

AS
Optional keyword to introduce the *correlation-name*.

correlation-name
May be used within search-condition to designate the table or view. (For an explanation of correlation-name, see "Correlation Names" on page 127.)

SET
Introduces the assignment of values to column names.

assignment-clause

column-name
Identifies a column to be updated. The *column-name* must identify an updatable column of the specified table or view.[108] The object ID column of a typed table is not updatable (SQLSTATE 428DZ). A column must not be specified more than once, unless it is followed by an attribute-name (SQLSTATE 42701).

For a Positioned UPDATE:

- If the UPDATE clause was specified in the select-statement of the cursor, each column name in the assignment-clause must also appear in the UPDATE clause.
- If the UPDATE clause was not specified in the select-statement of the cursor and LANGLEVEL MIA or SQL92E was specified when the application was precompiled, the name of any updatable column may be specified.
- If the UPDATE clause was not specified in the select-statement of the cursor and LANGLEVEL SAA1 was specified either explicitly or by default when the application was precompiled, no columns may be updated.

..attribute-name
Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*. The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The attribute-name must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

expression
Indicates the new value of the column. The expression is any

108. A column of a partitioning key is not updatable (SQLSTATE 42997). The row of data must be deleted and inserted to change columns in a partitioning key.

expression of the type described in "Expressions" on page 157. The expression can not include a column function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

NULL
Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9) unless it is specifically cast to the data type of the attribute.

DEFAULT
Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined as a generated column based on an expression, the column value will be generated by the system, based on the expression.
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined using the WITH DEFAULT clause, then the value is set to the default defined for the column (see default-clause in "ALTER TABLE" on page 477).
- If the column was defined without specifying the WITH DEFAULT clause, the GENERATED clause, or the NOT NULL clause, then the value used is NULL.
- If the column was defined using the NOT NULL clause and the GENERATED clause was not used, or the WITH DEFAULT clause was not used, or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

The only value that a generated column defined with the GENERATED ALWAYS clause can be set to is DEFAULT (SQLSTATE 428C9).

The DEFAULT keyword cannot be used as the value in an attribute assignment (SQLSTATE 429B9).

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of *column-name*s specified for

assignment. The values are assigned to each corresponding *column-name*. If the result of the *row-fullselect* is no rows, then null values are assigned.

A *row-fullselect* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated. An error is returned if there is more than one row in the result (SQLSTATE 21000).

WHERE
Introduces a condition that indicates what rows are updated. You can omit the clause, give a search condition, or name a cursor. If the clause is omitted, all rows of the table or view are updated.

search-condition
Is any search condition as described in "Chapter 3. Language Elements" on page 63. Each *column-name* in the search condition, other than in a subquery, must name a column of the table or view. When the search condition includes a subquery in which the same table is the base object of both the UPDATE and the subquery, the subquery is completely evaluated before any rows are updated.

The search-condition is applied to each row of the table or view and the updated rows are those for which the result of the search-condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed only once, whereas a subquery with a correlated reference may have to be executed once for each row.

CURRENT OF *cursor-name*
Identifies the cursor to be used in the update operation. The *cursor-name* must identify a declared cursor as explained in "DECLARE CURSOR" on page 841. The DECLARE CURSOR statement must precede the UPDATE statement in the program.

The table or view named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR" on page 841.)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

This form of UPDATE cannot be used if the target of the update is a view that includes an OLAP function in the select list of the fullselect that defines the view (SQLSTATE 42828).

Rules

- ***Assignment***: Update values are assigned to columns under the assignment rules described in Chapter 3.
- ***Validity***: The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column.

  If a view is used that is not defined using WITH CHECK OPTION, rows can be changed so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

  If a view is used that is defined using WITH CHECK OPTION, an updated row must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW" on page 823.
- ***Check Constraint***: Update value must satisfy the check-conditions of the check constraints defined on the table.

  An UPDATE to a table with check constraints defined has the constraint conditions for each column updated evaluated once for each row that is updated. When processing an UPDATE statement, only the check constraints referring to the updated columns are checked.
- ***Referential Integrity***: The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.

Notes

- If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order in which multiple rows are updated is undefined.
- When an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows updated. The SQLERRD(5) field contains the number of rows inserted, deleted, or updated by all activated triggers. For a description of the SQLCA, see "Appendix B. SQL Communications (SQLCA)" on page 1107.
- Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released, the updated row can only be accessed by the application process that performed the update (except for applications using the

Uncommitted Read isolation level). For further information on locking, see the descriptions of the COMMIT, ROLLBACK, and LOCK TABLE statements.

- If the URL value of a DATALINK column is updated, this is the same as deleting the old DATALINK value then inserting the new one. First, if the old value was linked to a file, that file is unlinked. Then, unless the linkage attributes of the DATALINK value are empty, the specified file is linked to that column.

  The comment value of a DATALINK column can be updated without relinking the file by specifying an empty string as the URL path (for example, as the *data-location* argument of the DLVALUE scalar function or by specifying the new value to be the same as the old value).

  If a DATALINK column is updated with a null, it is the same as deleting the existing DATALINK value.

  An error may occur when attempting to update a DATALINK value if the file server of either the existing value or the new value is no longer registered with the database server (SQLSTATE 55022).
- When updating the column distribution statistics for a typed table, the subtable that first introduced the column must be specified.
- Multiple attribute assignments on the same structured type column occur in the order specified in the SET clause and, within a parenthesized set clause, in left-to-right order.
- An attribute assignment invokes the mutator method for the attribute of the user-defined structured type. For example, the assignment `st..a1=x` has the same effect as using the mutator method in the assignment `st = st..a1(x)`.
- While a given column may be a target column in only one conventional assignment, a column may be a target column in multiple attribute assignments (but only if it is not also a target column in a conventional assignment).
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column.[109]
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.
- See "INSERT" on page 938 for more information on consuming values of a generated sequence for an identity column.

109. There is no casting of the previous value to the source type prior to the computation.

- See "INSERT" on page 938 for more information on exceeding the maximum value for an identity column.

Examples

- *Example 1:* Change the job (JOB) of employee number (EMPNO) '000290' in the EMPLOYEE table to 'LABORER'.

```
UPDATE EMPLOYEE
 SET JOB = 'LABORER'
 WHERE EMPNO = '000290'
```

- *Example 2:* Increase the project staffing (PRSTAFF) by 1.5 for all projects that department (DEPTNO) 'D21' is responsible for in the PROJECT table.

```
UPDATE PROJECT
 SET PRSTAFF = PRSTAFF + 1.5
 WHERE DEPTNO = 'D21'
```

- *Example 3:* All the employees except the manager of department (WORKDEPT) 'E21' have been temporarily reassigned. Indicate this by changing their job (JOB) to NULL and their pay (SALARY, BONUS, COMM) values to zero in the EMPLOYEE table.

```
UPDATE EMPLOYEE
 SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
 WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
```

  This statement could also be written as follows.

```
UPDATE EMPLOYEE
 SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
 WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
```

- *Example 4:* Update the salary and the commission column of the employee with employee number 000120 to the average of the salary and of the commission of the employees of the updated row's department respectively.

```
UPDATE EMPLOYEE EU
 SET (EU.SALARY, EU.COMM)
 =
(SELECT AVG(ES.SALARY), AVG(ES.COMM)
   FROM EMPLOYEE ES
   WHERE ES.WORKDEPT = EU.WORKDEPT)
   WHERE EU.EMPNO = '000120'
```

- *Example 5:* In a C program display the rows from the EMPLOYEE table and then, if requested to do so, change the job (JOB) of certain employees to the new job keyed in.

```
EXEC SQL  DECLARE C1 CURSOR FOR
               SELECT *
                 FROM EMPLOYEE
                 FOR UPDATE OF JOB;

EXEC SQL  OPEN C1;

EXEC SQL  FETCH C1 INTO ...     ;
```

```
if ( strcmp (change, "YES") == 0 )
  EXEC SQL  UPDATE EMPLOYEE
              SET JOB = :newjob
              WHERE CURRENT OF C1;

EXEC SQL  CLOSE C1;
```

- *Example 6:* These examples mutate attributes of column objects.

  Assume that the following types and tables exist:

```
CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
  NOT FINAL WITHOUT COMPARISONS
  MODE DB2SQL

CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE
```

  The following example updates the CIRCLES table by changing the OWNER column and the RADIUS attribute of the CIRCLE column where the ID is 999:

```
UPDATE CIRCLES
  SET OWNER = 'Bruce'
    C..RADIUS = 5
  WHERE ID = 999
```

  The following example transposes the X and Y coordinates of the center of the circle identified by 999:

```
UPDATE CIRCLES
  SET C..CENTER..X = C..CENTER..Y,
    C..CENTER..Y = C..CENTER..X
  WHERE ID = 999
```

  The following example is another way of writing both of the above statements. This example combines the effects of both of the above examples:

```
UPDATE CIRCLES
  SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
    ('Bruce',5,C..CENTER..Y,C..CENTER..X)
  WHERE ID = 999
```

VALUES

The VALUES statement is a form of query. It can be embedded in an application program or issued interactively. For detailed information, see "fullselect" on page 434.

VALUES INTO

The VALUES INTO statement produces a result table consisting of at most one row and assigns the values in that row to host variables.

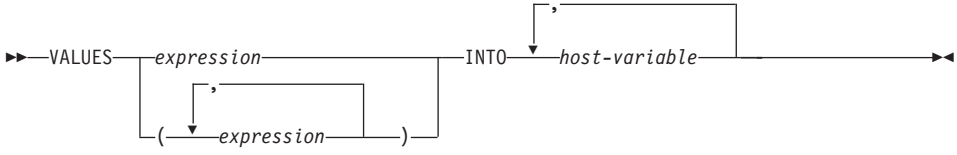
Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



Description

VALUES
Introduces a single row consisting of one of more columns.

expression
An expression that defines a single value of a one column result table.

(expression,...)
One or more expressions that define the values for one or more columns of the result table.

INTO
Introduces a list of host variables.

host-variable
Identifies a variable that is described in the program under the rules for declaring host variables.

The first value in the result row is assigned to the first variable in the list, the second value to the second variable, and so on. If the number of host variables is less than the number of column values, the value 'W' is assigned to the SQLWARN3 field of the SQLCA. (See “Appendix B. SQL Communications (SQLCA)” on page 1107.)

Each assignment to a variable is made according to the rules described in “Assignments and Comparisons” on page 94. Assignments are made in sequence through the list.

If an error occurs, no value is assigned to any host variable.

Examples

Example 1: This C example retrieves the value of the CURRENT PATH special register into a host variable.

```
EXEC SQL VALUES(CURRENT PATH)
         INTO :hvl;
```

Example 2: This C example retrieves a portion of a LOB field into a host variable, exploiting the LOB locator for deferred retrieval.

```
EXEC SQL VALUES (substr(:locator1,35))
         INTO :details;
```

WHENEVER

The WHENEVER statement specifies the action to be taken when a specified exception condition occurs.

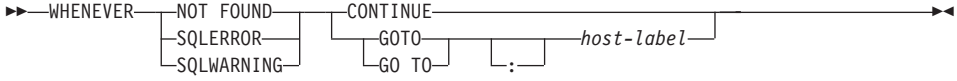
Invocation

This statement can only be embedded in an application program. It is not an executable statement. The statement is not supported in REXX.

Authorization

None required.

Syntax



Description

The NOT FOUND, SQLERROR, or SQLWARNING clause is used to identify the type of exception condition.

NOT FOUND
Identifies any condition that results in an SQLCODE of +100 or an SQLSTATE of '02000'.

SQLERROR
Identifies any condition that results in a negative SQLCODE.

SQLWARNING
Identifies any condition that results in a warning condition (SQLWARN0 is 'W'), or that results in a positive SQL return code other than +100.

The CONTINUE or GO TO clause is used to specify what is to happen when the identified type of exception condition exists.

CONTINUE
Causes the next sequential instruction of the source program to be executed.

GOTO or **GO TO** *host-label*
Causes control to pass to the statement identified by host-label. For host-label, substitute a single token, optionally preceded by a colon. The form of the token depends on the host language.

Notes

There are three types of WHENEVER statements:

- WHENEVER NOT FOUND
- WHENEVER SQLERROR

- WHENEVER SQLWARNING

Every executable SQL statement in a program is within the scope of one implicit or explicit WHENEVER statement of each type. The scope of a WHENEVER statement is related to the listing sequence of the statements in the program, not their execution sequence.

An SQL statement is within the scope of the last WHENEVER statement of each type that is specified before that SQL statement in the source program. If a WHENEVER statement of some type is not specified before an SQL statement, that SQL statement is within the scope of an implicit WHENEVER statement of that type in which CONTINUE is specified.

Example

In the following C example, if an error is produced, go to HANDLERR. If a warning code is produced, continue with the normal flow of the program. If no data is returned, go to ENDDATA.

```
EXEC SQL WHENEVER SQLERROR GOTO HANDLERR;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND GO TO ENDDATA;
```