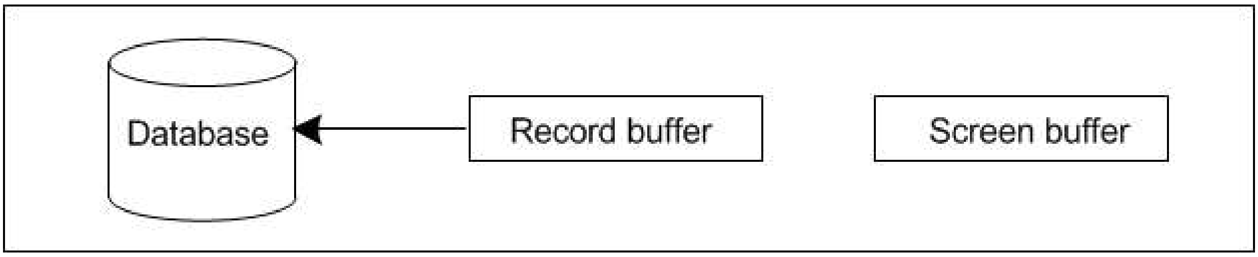
DELETE statement

Removes a record from a record buffer and from the database.

Data movement



Syntax

```
DELETE record
  [ VALIDATE ( condition , msg-expression ) ]
  [ NO-ERROR ]
```

record

The name of a record buffer. You can delete a record only after it has been put into a record buffer by a CREATE, FIND, FOR EACH, or INSERT statement.

If you define an alternate buffer for a table, you can delete a record from that buffer by using the name of the buffer with the DELETE statement.

To delete a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

VALIDATE (*condition*, *msg-expression*)

Use the VALIDATE option to specify a logical value that allows the deletion of a record when TRUE, but does not allow the deletion of a record when FALSE.

The *condition* is a Boolean expression (a constant, field name, variable name, or expression) with a value of TRUE or FALSE.

The *msg-expression* is the message you want to display if the *condition* is FALSE. You must enclose *msg-expression* in quotation marks ("").

You can also describe delete validation criteria for a table in the Data Dictionary. To suppress the Data Dictionary delete validation criteria for a table, use the VALIDATE option as follows:

```
VALIDATE(TRUE,"")
```

If you use the DELETE statement to delete a record in a work table, the AVM disregards any VALIDATE option you use with the DELETE statement.

NO-ERROR

Suppresses ABL errors or error messages that would otherwise occur and diverts them to the ERROR-STATUS system handle. If an error occurs, the action of the statement is not done and execution continues with the next statement. If the statement fails, any persistent side-effects of the statement are backed out. If the statement includes an expression that contains other executable elements, like methods, the work performed by these elements may or may not be done, depending on the order the AVM resolves the expression elements and the occurrence of the error.

To check for errors after a statement that uses the NO-ERROR option:

- Check the ERROR-STATUS:ERROR attribute to see if the AVM raised the ERROR condition.
- Check if the ERROR-STATUS:NUM-MESSAGES attribute is greater than zero to see if the AVM generated error messages. ABL handle methods used in a block **without** a CATCH end block treat errors as warnings and do not raise ERROR, do not set the ERROR-STATUS:ERROR attribute, but do add messages to the ERROR-STATUS system handle. Therefore, this test is the better test for code using handle methods without CATCH end blocks. ABL handle methods used in a block **with** a CATCH end block raise ERROR and add messages to the error object generated by the AVM. In this case, the AVM does not update the ERROR-STATUS system handle.
- Use ERROR-STATUS:GET-MESSAGE(*message-num*) to retrieve a particular message, where *message-num* is 1 for the first message.

If the statement does not include the NO-ERROR option, you can use a CATCH end block to handle errors raised by the statement.

Some other important usage notes on the NO-ERROR option:

- NO-ERROR does not suppress errors that raise the STOP or QUIT condition.
- A CATCH statement, which introduces a CATCH end block, is analogous to a NO-ERROR option in that it also suppresses errors, but it does so for an entire block of code. It is different in that the error messages are contained in a class-based error object (generated by the AVM or explicitly thrown), as opposed to the ERROR-STATUS system handle. Also, if errors raised in the block are not handled by a compatible CATCH block, ON ERROR phrase, or UNDO statement, then the error is not suppressed, but handled with the default error processing for that block type.
- When a statement contains the NO-ERROR option and resides in a block with a CATCH end block, the NO-ERROR option takes precedence over the CATCH block. That is, an error raised on the statement with the NO-ERROR option will not be handled by a compatible CATCH end block. The error is redirected to the ERROR-STATUS system handle as normal.
- If an error object is thrown to a statement that includes the NO-ERROR option, then the information and messages in the error object will be used to set the ERROR-STATUS system handle. This interoperability feature is important for those integrating code that uses the traditional NO-ERROR technique with the newer, structured error handling that features error objects and CATCH end blocks.

Examples

The `r-delet.p` procedure deletes all the records in the Customer table.

r-delet.p

```
FOR EACH Customer:
  DELETE Customer.
END.
```

The `r-delet2.p` procedure prompts the user for a Customer number and then displays the name of that Customer. It then prompts the user to press y to confirm the deletion of the Customer record. The user's response is stored in the del variable. If the value of the del variable is y, the procedure deletes the Customer record.

r-delet2.p

```
DEFINE VARIABLE del AS LOGICAL NO-UNDO FORMAT "y/n".

REPEAT:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.Name.
  del = NO.
  UPDATE del LABEL "Enter ""y"" to confirm delete".
  IF del THEN DELETE Customer.
END.
```

The `r-delval.p` procedure prompts the user for a Customer number. The procedure displays the name of the Customer and prompts the user: Do you want to delete this Customer? If the user answers no, the procedure prompts the user for another Customer number. If the user answers yes, the procedure checks whether the Customer has orders, using the VALIDATE option. If they do have orders, the procedure displays this message: This Customer has outstanding orders and cannot be deleted. If the Customer has no orders, the procedure deletes the Customer.

r-delval.p

```
DEFINE VARIABLE ans AS LOGICAL NO-UNDO.

REPEAT WITH 1 DOWN:
  PROMPT-FOR Customer.CustNum.
  FIND Customer USING Customer.CustNum.
  DISPLAY Customer.Name.
  ans = NO.
  DISPLAY "Do you want to delete this Customer ?"
    WITH FRAME f-query.
  UPDATE ans WITH FRAME f-query NO-LABELS.
  IF ans THEN
    DELETE Customer VALIDATE(NOT(CAN-FIND(Order OF Customer)),
    "This Customer has outstanding orders and cannot be deleted.").
END.
```

Notes

- When you run procedures that delete large numbers of records (for example, a month-end table purge), the process runs much faster if you use the No Crash Protection (-i) parameter in single-user mode. (You must back up your database before using this option.) See *OpenEdge Deployment: Startup Command and Parameter Reference* for more information on startup parameters.

- Deleting records does not change the amount of space the database takes up on the disk. The AVM re-uses ROWIDs. It does not delete the ROWID when a record is deleted. To recover disk space, you must dump and reload your database.

- The DELETE statement causes any related database DELETE triggers to execute. All DELETE triggers execute before the AVM actually deletes the record. While a DELETE trigger is executing, all FIND requests for the record (even within the trigger) fail, as if the record were already deleted. If a DELETE trigger fails (or executes a RETURN ERROR statement), the corresponding record is not deleted.

- If a table has both a DELETE trigger and delete VALIDATION, the DELETE trigger executes before the validation is performed.

- If you have previously retrieved *`record`* with a field list, the DELETE statement rereads the complete record before deleting it.

See also

CREATE statement, FIND statement, FOR statement, INSERT statement

< >