ODBC
Reference Manual
Contents

1 Introduction .............................................................................. 3

1.1 Start and Stop ................................................................. 3
  1.1.1 start_link/[2, 3] ....................................................... 3
  1.1.2 stop/[1, 2] ............................................................... 5

1.2 Utility API ........................................................................ 6
  1.2.1 init_env/[1, 2] ........................................................... 6
  1.2.2 connect/[3, 4, 5, 6] ................................................... 7
  1.2.3 execute_stmt/[3, 4] ................................................. 8
  1.2.4 disconnect/[2, 3] ...................................................... 10
  1.2.5 terminate_env/[2, 3] ............................................... 11

1.3 Basic API .......................................................................... 12
  1.3.1 sql_alloc_handle/[3, 4] ........................................... 12
  1.3.2 sql_bind_col/[4, 5] ................................................ 13
  1.3.3 sql_bind_parameter/[8, 9] ..................................... 14
  1.3.4 sql_close_cursor/[2, 3] .......................................... 16
  1.3.5 sql_connect/[5, 6] .................................................. 16
  1.3.6 sql_describe_col/[4, 5] ......................................... 17
  1.3.7 sql_disconnect/[2, 3] ............................................. 19
  1.3.8 sql_driver_connect/[5, 6] ..................................... 20
  1.3.9 sql_end_tran/[4, 5] ................................................ 21
  1.3.10 sql_exec_direct/[3, 4] ......................................... 22
  1.3.11 sql_fetch/[2, 3] ................................................... 23
  1.3.12 sql_free_handle/[3, 4] ....................................... 24
  1.3.13 sql_get_connect_attr/[4, 5] ................................ 25
  1.3.14 sql_get_diag_rec/[5, 6] ....................................... 27
  1.3.15 sql_num_result_cols/[2, 3] ................................. 28
  1.3.16 sql_row_count/[2, 3] .......................................... 29
  1.3.17 sql_set_connect_attr/[5, 6] ............................... 30
  1.3.18 sql_set_env_attr/[5, 6] ....................................... 31
  1.3.19 alloc_buffer/[3, 4] ............................................. 32
  1.3.20 dealloc_buffer/[2, 3] ......................................... 33
  1.3.21 read_buffer/[2, 3] ............................................. 33
  1.3.22 write_buffer/[3, 4] ............................................. 34

1.4 Error Messages and Exceptions ........................................ 35

1.5 References ..................................................................... 35
1 Introduction

The ODBC API is divided into three parts:

Start and Stop
Starts and stops the server process.

Utility API
Consists of functions that are easier to use than the Basic API. These functions are on a higher level, do more of the job, but allow less control to the application programmer.

Basic API
Gives access to the IDL Interface functions, which are mapped on ODBC functions.

All functions described are synchronous.

The interface supports all ODBC defined SQL data types except binaries. They are all mapped on Erlang strings.

The type string() is a list() of integers representing ASCII codes. The type boolean() is either the macro ?SQL_TRUE or the macro ?SQL_FALSE.

The default Timeout for all functions is 5000 ms, unless otherwise stated.

1.1 Start and Stop

1.1.1 start_link/[2, 3]

Description:
Starts a new ODBC server process, registers it with the supervisor, and links it to the calling process. Opens a unique IDL connection to a new C node on the local host, using the same cookie as is used by the node of the calling process. Links to the process on the C node.

Arguments and Return Value:

start_link(Args, Options) ->
start_link(ServerName, Args, Options) ->
Result

Args -> [Arg]
Arg -> {buffer_size, integer()} | Initialisation arguments.
The initial size of the buffer through which
communication with the C node is done. Doesn’t limit the amount of data that can pass in either direction of a function call, since the buffer will grow dynamically. The default is 32 kb. The minimum is 4 kb.

{max_len_data, integer()} The maximum length of table data, including null-termination, returned from ODBC. This value must be chosen with the buffer size in mind. The default is 8 kb. Used only by the Utility API.

NOTE: The data source or driver may have a lower limit for the max size of returned data. This limit is the value of the optional statement attribute SQL_ATTR_MAX_LENGTH (see [1]).

{max_len_err_msg, integer()} The maximum length of the message part of ODBC error messages, including null-termination. This value must be chosen with the buffer size in mind. The default is 1 kb. Used only by the Utility API.

{max_len_str, integer()} The maximum length of other strings, including null-termination, passed from ODBC to the ODBC server (e.g. column names). Does not limit the size of returned table values. This value must be chosen with the buffer size in mind. The default is 1 kb. Used only by the Utility API.

Options -> [Opt] List of options.
Opt -> {timeout, integer()} | The time in ms allowed for initialisation, see gen_server.

   {debug, [Dbg]}

Dbg -> [trace | Debug options, see gen_server.
   log |
   statistics |
   {log_to_file, FileName} | Debug options, see gen_server.
   {install, {Func, FuncState}}

ServerName -> {local, atom()} | When supplied, causes the server to be registered,
   {global, atom()} locally or globally. If the server is started without a

Result -> {ok, pid()} | The pid of the erl. server.
   {error, Reason} Error tuple.

Reason -> {already_started, pid()} | Server already started.
   timeout | Timeout expired.
   {no_c_node, Info} Can’t start C node. The program may not have

   Info -> string() been found or may not have been executable e.g. More information.

NOTE: There is no default timeout value. Not using the timeout option is equivalent to having an infinite timeout value.

NOTE: Timeout is reported as an error here, not an exception.

NOTE: The debug options are described in the sys module documentation.

1.1.2 stop/[1, 2]

Description:

Stops the ODBC server process as soon as all already submitted requests have been processed. The C node is also stopped.

Arguments and Return Value:

stop(Server) ->
stop(Server, Timeout) ->
   ok
1.2 Utility API

The Utility API uses three maximum string length parameters: the maximum data string length (max_len_data), the maximum error message length (max_len_err_msg), and the maximum length of 'other strings' (e.g. column names) passed from ODBC (max_len_str). These can be set in the call to start[/2, 3] and start_link[/2, 3], but there are default values.

Errors reported by the ODBC API are returned in lists. The relative order of these errors is the same as specified in [1]. Warnings are always ignored and execution proceeds. Should an error occur, execution stops.

1.2.1 init_env/[1, 2]

Description:

Initialises the ODBC environment on the C node. This can also be done through use of functions of the Basic API.

Arguments and Return Value:

init_env(Server) ->
init_env(Server, Timeout) ->
{ok, RefEnvHandle} |
{error, {[Fcn, [Reason]]}}

Server -> pid() |
Name |
{global, Name} |
{Name, Node} | The pid of the server process,
a registered name,
a globally registered name, or
a registered name on a remote node.

Timeout -> integer() |
infinity | Max time (ms) for serving the request.
1.2.2 connect/[3, 4, 5, 6]

Description:

Opens a connection to a data source. There can be only one open data source connection per server. connect/[3, 4] is used when the information that can be supplied through connect/[5, 6] does not suffice.

NOTE: The syntax to be used for ConnectStr is described under SQLDriverConnect in [1]. The ConnectStr must be complete.

Arguments and Return Value:

connect(Server, RefEnvHandle, ConnectStr) ->
connect(Server, RefEnvHandle, ConnectStr, Timeout) ->
connect(Server, RefEnvHandle, DSN, UID, PWD) ->
connect(Server, RefEnvHandle, DSN, UID, PWD, Timeout) ->
{ok, RefConnHandle} |
{error, {Fcn, [Reason]}}

Server -> pid() |
            Name |
            {global, Name} |
            {Name, Node} |
RefEnvHandle -> term()
1.2.3 execute_stmt/[3, 4]

Description:

Executes a single SQL statement. All changes to the data source are automatically committed if successful. Data that is returned for SELECT statements is in string form. Use the Basic API to retrieve binaries and large data (that needs to be divided into smaller chunks).

Arguments and Return Value:

execute_stmt(Server, RefConnHandle, Stmt) ->
execute_stmt(Server, RefConnHandle, Stmt, Timeout) ->
{updated, NRows} |
{selected, [ColName], [Row]} 
{error, {Fcn, [Reason]}}
Server -> pid() | Name | {global, Name} | {Name, Node}

The pid of the server process, a registered name, a globally registered name, or a registered name on a remote node.

RefConnHandle -> term()

Reference to an open connection. Returned by connect/[3, 5].

Stmt -> string()

SQL statement to execute.

Timeout -> integer() | infinity

Max time (ms) for serving the request.

NRows -> integer()

The number of updated rows for update, insert, or delete statements, or -1 if the number is not available. For other statement types the value is driver defined, see [1].

ColName -> string()

The name of a column in the resulting table.

Row -> [Value]

One row of the resulting table.

Value -> string() | null

One value in a row.

Fcn -> atom()

The originating function.

Reason -> {SqlState, MoreInfo}

An ODBC error tuple.

SqlState -> string()

The SQL state, see [1].

MoreInfo -> {NativeCode, Msg, LenMsg}

More error info.

NativeCode -> string()

Data source specific error code.

Msg -> string()

Error message.
LenMsg -> integer() Length of Msg before truncation.

NOTE: {updated, 0} is returned when a statement that does not select or update any rows is successfully executed.

NOTE: The ColNames are ordered the same way as the Values in the Rows (the first ColName is associated with the first Value of each Row etc.). The Rows have no defined order since they represent a set.

NOTE: Column names will be truncated if they are longer than the maximum string length (see option to start_link/[2, 3]). Table values will be truncated if they are longer than the maximum data length, or longer than the value of the statement attribute SQL_ATTR_MAX_LENGTH. If the amount of memory needed to retrieve a table value from a data source can not be determined, the default maximum data length (see start_link/[2, 3]) is used.

1.2.4 disconnect/[2, 3]

Description:
Closes the connection to a data source.

Arguments and Return Value:

disconnect(Server, RefConnHandle) ->
disconnect(Server, RefConnHandle, Timeout) ->
ok |
    {error, [Fcn, [Reason]]}

Server -> pid() |
    Name |
    {global, Name} |
    {Name, Node}

RefConnHandle -> term()

Timeout -> integer() |
    infinity

The pid of the server process, a registered name, a globally registered name, or a registered name on a remote node.
Reference to an open connection. Returned by connect/[3, 5].
Max time (ms) for serving the request.
Fcn -> atom() The originating function.

Reason -> \{SqlState, MoreInfo\} An ODBC error tuple.

SqlState -> string() The SQL state, see [1].


NativeCode -> string() Data source specific error code.

Msg -> string() Error message.

LenMsg -> integer() Length of Msg before truncation.

1.2.5 terminate_env/[2, 3]

Description:

Cleans up the ODBC environment on the C node. This can also be done through use of functions of the Basic API.

Arguments and Return Value:

\[
\text{terminate_env}(\text{Server, RefEnvHandle}) \rightarrow \text{terminate_env}(\text{Server, RefEnvHandle, Timeout}) \rightarrow \{\text{ok} | \{\text{error, \{Fcn, [Reason]\}}\}
\]

Server -> \text{pid() | Name | \{global, Name\} | \{Name, Node\}}

RefEnvHandle -> \text{term()}

Timeout -> \text{integer() | infinity}

Fcn -> atom()
12(35) Specification

Function.

Reason -> {SqlState, MoreInfo} An ODBC error tuple.

SqlState -> string() The SQL state, see [1].


NativeCode -> string() Data source specific error code.

Msg -> string() Error message.

LenMsg -> integer() Length of Msg before truncation.

1.3 Basic API

To use the Basic API it is necessary to gain a comprehensive understanding of ODBC by studying [1].

ODBC defines the concept of deferred buffers. A deferred buffer is one that exists longer than one function call, so it can be used in several calls. Deferred buffers come in pairs: one data buffer and one length/indicator buffer. The length/indicator buffer is used for communicating the length of data in the data buffer, or to indicate something about the data (e.g. that it is a null-value). The Basic API handles these buffers accordingly: they are allocated, deallocated, read, and written pair-wise.

1.3.1 sql_alloc_handle/[3, 4]

Description:

Allocates memory for an environment, connection, or statement handle. See SQLAllocHandle in [1].

Differences from the ODBC Function:

Allocation of descriptor handles is not supported. The parameters Server and Timeout have been added. The ODBC output parameter OutputHandlePtr has been changed into the returned value RefOutputHandle. Connection pooling is not supported.

Arguments and Return Value:
sql_alloc_handle(Server, HandleType, RefInputHandle) ->

sql_alloc_handle(Server, HandleType, RefInputHandle, Timeout) ->

{Result, RefOutputHandle}

Server -> pid() | The pid of the server,
   Name | a registered name,
   {global, Name} | a globally registered name,
   {Name, Node} | or a registered name on a
   remote node.

HandleType -> ?SQL_HANDLE_ENV | Macros which determine
   ?SQL_HANDLE_DBC | which type of handle to
   ?SQL_HANDLE_STMT allocate.

RefInputHandle -> term() | The context in which the
   ?SQL_NULL_HANDLE new handle is to be
   allocated. When allocating

Timeout -> integer() | Max time (ms) for serving
   infinity the request.

Result -> ?SQL_SUCCESS | Result macro.
   ?SQL_SUCCESS_WITH_INFO |
   ?SQL_INVALID_HANDLE |
   ?SQL_ERROR

RefOutputHandle -> term() | Reference to the allocated
   ?SQL_NULL_HENV | handle, or a value
   ?SQL_NULL_HSTMT

1.3.2 sql_bind_col/[4, 5]

Description:

Assigns storage and data type for a column in a result set (binds a buffer to a column). See SQLBindCol in [1]. Buffers/columns can also be unbound.

NOTE: The memory associated with RefBuf has to be allocated already.

Differences from the ODBC Function:

Neither binding of arrays nor the use of binding offsets is supported. It is not possible to unbind the data buffer without also unbinding the data length buffer. The parameters Server and Timeout have been added. The input parameters TargetType, TargetValuePtr, BufferLength, and StrLen_or_IndPtr of the ODBC function have been replaced with the RefBuf parameter (which represents the same data).
14(35) Specification

Arguments and Return Value:

sql_bind_col(Server, RefStmtHandle, ColNum, RefBuf) ->
sql_bind_col(Server, RefStmtHandle, ColNum, RefBuf, Timeout) ->

Result

Server ->
   pid() | The pid of the server,
   Name | a registered name,
   {global, Name} | a globally registered name,
   {Name, Node} | or a registered name on a remote node.

RefStmtHandle ->
   term() | Reference to the statement handle.

ColNum ->
   integer() | Column number from left to right starting at 1.

RefBuf ->
   integer() |
   NULL_REF | Reference to the buffer where the column data is placed (and to the associated length/indicator buffer).
   NULL_REF removes the binding between a buffer and a column.

Timeout ->
   integer() | Max time (ms) for serving the request.
   infinity |

Result ->
   SQL_SUCCESS |
   SQL_SUCCESS_WITH_INFO |
   SQL_ERROR |
   SQL_INVALID_HANDLE |

1.3.3 sql_bind_parameter/[8, 9]

Description:

Binds a buffer to a parameter marker in an SQL statement. See SQLBindParameter in [1].

NOTE: The memory associated with RefBuf has to be allocated already.

Differences from the ODBC Function:

Only input parameters is supported. Neither binding of arrays nor the use of binding offsets is supported. Multiple values for parameters is not supported. The use of arbitrary 32-bit integers for the ParameterValuePtr argument in the ODBC function (RefBuf here) is not supported. The parameters Server and Timeout have been added. The input parameters ValueType,
ParameterValuePtr, BufferLength, and StrLen_or_IndPtr of the ODBC function have been replaced with the RefBuf parameter (which represents the same data).

Arguments and Return Value:

\[
\text{sql\_bind\_parameter}(Server, \text{RefStmtHandle}, \text{ParNum}, \text{InOutType}, \text{ParSqlType}, \text{ColSize}, \text{DecDigs}, \text{RefBuf}) \rightarrow \\
\text{sql\_bind\_parameter}(Server, \text{RefStmtHandle}, \text{ParNum}, \text{InOutType}, \text{ParSqlType}, \text{ColSize}, \text{DecDigs}, \text{RefBuf}, \text{Timeout}) \rightarrow \\
\]

Result

\[
\begin{align*}
\text{Server} & \rightarrow \text{pid()} \mid \text{Name} \mid \{\text{global, Name}\} \mid \{\text{Name, Node}\} \quad \text{The pid of the server,} \\
& \quad \text{a registered name,} \\
& \quad \text{a globally registered name,} \\
& \quad \text{or a registered name on a remote node.} \\
\text{RefStmtHandle} & \rightarrow \text{term()} \quad \text{Reference to the statement handle.} \\
\text{ParNum} & \rightarrow \text{integer()} \quad \text{Parameter number from left to right starting at 1.} \\
\text{InOutType} & \rightarrow \text{?SQL\_PARAM\_INPUT} \quad \text{Input parameter.} \\
\text{ParSqlType} & \rightarrow \text{integer()} \quad \text{A macro representing a supported SQL data type.} \\
\text{ColSize} & \rightarrow \text{integer()} \quad \text{The maximum size of the parameter. Ignored for certain values of ParSqlType. See also appendix D in [1].} \\
\text{DecDigs} & \rightarrow \text{integer()} \quad \text{The scale of the parameter. Ignored for certain values of ParSqlType. See also appendix D in [1].} \\
\text{RefBuf} & \rightarrow \text{term()} \quad \text{Reference to the buffer where parameters are placed before execution of the statement (and to the associated length/indicator buffer, which must contain one of the following values:} \\
& \quad \text{The length of the data in the data buffer referred to by RefBuf,} \\
& \quad \text{?SQL\_NTS, ?SQL\_NULL\_DATA,} \\
& \quad \text{?SQL\_DATA\_AT\_EXEC, or} \\
& \quad \text{the result of the macro} \\
& \quad \text{?SQL\_LEN\_DATA\_AT\_EXEC(L),} \\
& \quad \text{where L is the length of the data to be sent for the parameter,}
\end{align*}
\]
1.3.4 sql_close_cursor/[2, 3]

Description:

Closes a cursor that has been opened on a statement and discards pending results. See SQLCloseCursor in [1].

Differences from the ODBC Function:

The parameters Server and Timeout have been added.

Arguments and Return Value:

sql_close_cursor(Server, RefStmtHandle) ->
sql_close_cursor(Server, RefStmtHandle, Timeout) ->

Result

Server -> pid() | Name | {global, Name} | {Name, Node} | The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.

RefStmtHandle -> term() | Reference to the statement handle.

Timeout -> integer() | infinity | Max time (ms) for serving the request.


1.3.5 sql_connect/[5, 6]

Description:
Establishes a connection to a driver and a data source. See SQLConnect in [1].

Differences from the ODBC Function:

Connection pooling is not supported.

The parameters Server and Timeout have been added. The input parameters NameLength1, NameLength2, and NameLength3 of the ODBC function have been excluded.

Arguments and Return Value:

```
sql_connect(Server, RefConnHandle, DSN, UID, Auth) ->
sql_connect(Server, RefConnHandle, DSN, UID, Auth, Timeout) ->
Result
```

Server -> pid() | The pid of the server,  
| Name | a registered name,  
| {global, Name} | a globally registered name,  
| {Name, Node} | or a registered name on a remote node.

RefConnHandle -> term() | Reference to the connection handle.

DSN -> string() | The name of the data source.

UID -> string() | The user ID.

Auth -> string() | The user’s password for the data source.

Timeout -> integer() | Max time (ms) for serving the request.

infinity

Result -> ?SQL_SUCCESS | Result macro.

?SQL_SUCCESS_WITH_INFO

?SQL_ERROR

?SQL_INVALID_HANDLE

1.3.6 sql_describe_col/[4, 5]

Description:

Returns the result descriptor – column name, type, column size, decimal digits, and nullability – for one column in the result set. See SQLDescribeCol in [1].
To decide the buffer size needed (how many characters or bytes) to retrieve data for the column it is necessary to calculate the display size (see also appendix D in [1]). The function display_size(SqlType, ColSize) -> integer() does the calculation. The input parameters are returned by sql_describe_col.

Differences from the ODBC Function:

The function does not support retrieval of bookmark column data.

The parameters Server and Timeout have been added. The output parameters ColumnName, NameLengthPtr, DataTypePtr, ColumnSizePtr, DecimalDigitsPtr, and NullablePtr of the ODBC function have been changed into the returned values ColName, LenColName, SqlType, ColSize, DecDigs, and Nullable. BufLenColName must be > 0.

Arguments and Return Value:

sql_describe_col(Server, RefStmtHandle, ColNum, BufLenColName) -> sql_describe_col(Server, RefStmtHandle, ColNum, BufLenColName, Timeout) ->

{Result,
 {ColName, LenColName}, SqlType, ColSize, DecDigs, Nullable}

Server -> pid() | The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.

Name | {global, Name} | {Name, Node}

RefStmtHandle -> term() Reference to the statement handle.

ColNum -> integer() The column number from left to right, starting at 1.

BufLenColName -> integer() Length (>0) of ColName buffer. Allow room for null-termination.

Timeout -> integer() | infinity Max time (ms) for serving the request.

Result -> ?SQL_SUCCESS | ?SQL_SUCCESS_WITH_INFO | Result macro.
1.3.7 sql_disconnect/[2, 3]

Description:

Closes the connection associated with a specific connection handle. See SQLDisconnect in [1].

Differences from the ODBC Function:

ColName -> string()

LenColName -> integer()

SqlType -> integer()

ColSize -> integer()

DecDigs -> integer()

Nullable -> ?SQL_NO_NULLS | ?SQL_NULLABLE | ?SQL_NULLABLE_UNKNOWN

The column name.
Actual length of ColName.
An ODBC SQL data type (ODBC supported data types are supplied through macros, or a driver-specific type (not supplied through macros).
The precision of the column (see appendix D in [1]). If the precision can not be determined, 0 is returned.
The scale of the column (see appendix D in [1]). If the scale can not be determined, or is not applicable, 0 is returned.
Indicates whether the column allows NULL values or not.
Connection pooling is not supported.

The parameters Server and Timeout have been added.

Arguments and Return Value:

sql_disconnect(Server, RefConnHandle) ->
sql_disconnect(Server, RefConnHandle, Timeout) ->

Result

Server -> pid() | The pid of the server,
Name | a registered name,
{global, Name} | a globally registered name,
{Name, Node} | or a registered name on a remote node.

RefConnHandle -> term() Reference to the connection handle.

Timeout -> integer() | Max time (ms) for serving the request.
infinity

Result -> ?SQL_SUCCESS | Result macro.
?SQL_SUCCESS_WITH_INFO |
?SQL_ERROR |
?SQL_INVALID_HANDLE

1.3.8 sql_driver_connect/[5, 6]

Description:

Establishes a connection to a driver and a data source which needs more connection information than SQLConnect offers. See SQLDriverConnect in [1].

Differences from the ODBC Function:

The function does not support prompting with pop-ups, so the connection string supplied must be complete.

The parameters Server and Timeout have been added. The input parameters WindowHandle and StringLength1 of the ODBC function have been excluded. The output parameters OutConnectionString and StringLength2Ptr have been changed into the returned values OutConnStr and LenOutConnStr. BufLenOutConnStr must be > 0.

Arguments and Return Value:
sql_driver_connect(Server, RefConnHandle, InConnStr, BufLenOutConnStr, DrvCompletion) ->

sql_driver_connect(Server, RefConnHandle, InConnStr, BufLenOutConnStr, DrvCompletion, Timeout) ->

\{Result, \{OutConnStr, LenOutConnStr\}\}

Server -> pid() | Name | \{global, Name\} | \{Name, Node\}  
The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.

RefConnHandle -> term()  
Reference to the connection handle.

InConnStr -> string()  
A complete connection string.

BufLenOutConnStr -> integer()  
Length (>0) of OutConnStr buffer. Allow room for null-termination.

DrvCompletion -> \?SQL_DRIVER_NOPROMPT  
No prompting with pop-ups.

Timeout -> integer() | infinity  
Max time (ms) for serving the request.

Result -> \?SQL_SUCCESS | \?SQL_SUCCESS_WITH_INFO | \?SQL_NO_DATA | \?SQL_ERROR | \?SQL_INVALID_HANDLE  
Result macro.

OutConnStr -> string()  
A complete connection string.

LenOutConnStr -> integer()  
The length of OutConnStr before truncation.

1.3.9 sql_end_tran/[4, 5]

Description:

Requests a commit or rollback operation for all active operations on all statement handles associated with a connection. It can also request that a commit or rollback operation be performed for all connections associated with the environment handle. See SQLEndTran in [1].
NOTE: Rollback of transactions may be unsupported by core level drivers.

Differences from the ODBC Function:

The parameters Server and Timeout have been added.

Arguments and Return Value:

sql_end_tran(Server, HandleType, RefHandle, ComplType) ->
sql_end_tran(Server, HandleType, RefHandle, ComplType, Timeout) ->
  Result

  Server -> pid() | The pid of the server,
  Name | a registered name,
  {global, Name} | a globally registered name,
  {Name, Node} | or a registered name on a remote node.
  HandleType -> ?SQL_HANDLE_ENV | The type of handle for which to perform the transaction (all connections associated with an environment or a specific connection).
  ?SQL_HANDLE_DBC
  RefHandle -> term() | Reference to the handle
  ComplType -> ?SQL_COMMIT | Commit operation.
  ?SQL_ROLLBACK | Rollback operation.
  Timeout -> integer() | Max time (ms) for serving the request.
  infinity
  Result -> ?SQL_SUCCESS | Result macro.
  ?SQL_SUCCESS_WITH_INFO |
  ?SQL_ERROR |
  ?SQL_INVALID_HANDLE

1.3.10 sql_exec_direct/[3, 4]

Description:

Executes a preparable statement using the current values of the parameter marker buffers (bound with sql_bind_parameter/[8, 9]) if any parameter markers exist in the statement. See SQLExecDirect in [1].

Differences from the ODBC Function:

?SQL_NO_DATA is returned only in connection with positioned updates, which are not supported.
The parameters Server and Timeout have been added. The input parameter TextLength of the ODBC function has been excluded.

**Arguments and Return Value:**

\[
\text{sql\_exec\_direct}(\text{Server}, \text{RefStmtHandle}, \text{Stmt}) \rightarrow \\
\text{sql\_exec\_direct}(\text{Server}, \text{RefStmtHandle}, \text{Stmt}, \text{Timeout}) \rightarrow
\]

\[
\text{Result}
\]

- **Server** -> pid() | Name | \{global, Name\} | \{Name, Node\}
- **RefStmtHandle** -> term()
- **Stmt** -> string()
- **Timeout** -> integer() | infinity

### 1.3.11 sql\_fetch/[2, 3]

**Description:**

Fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with sql\_bind\_col/[4, 5]. See SQL\_Fetch in [1].

**Differences from the ODBC Function:**

The parameters Server and Timeout have been added.

**Arguments and Return Value:**

\[
\text{sql\_fetch}(\text{Server}, \text{RefStmtHandle}) \rightarrow \\
\text{sql\_fetch}(\text{Server}, \text{RefStmtHandle}, \text{Timeout}) \rightarrow
\]

\[
\text{Result}
\]
Server -> pid() | The pid of the server,  
Name | a registered name,  
{global, Name} | a globally registered name,  
{Name, Node} | or a registered name on a remote node.

RefStmtHandle -> term() Reference to the statement handle.

Timeout -> integer() | Max time (ms) for serving the request.  
infinity

Result -> ?SQL_SUCCESS | Result macro.  
?SQL_SUCCESS_WITH_INFO  
?SQL_NO_DATA  |  
?SQL_ERROR  |  
?SQL_INVALID_HANDLE

1.3.12 sql_free_handle/[3, 4]

Description:

Releases a handle and frees all resources associated with it. See SQLFreeHandle in [1].

Differences from the ODBC Function:

The function does not support deallocation of descriptor handles.

The parameters Server and Timeout have been added.

Arguments and Return Value:

sql_free_handle(Server, HandleType, RefHandle) ->
sql_free_handle(Server, HandleType, RefHandle, Timeout) ->
Result

Server -> pid() | The pid of the server,  
Name | a registered name,  
{global, Name} | a globally registered name,  
{Name, Node} | or a registered name on a remote node.

HandleType -> ?SQL_HANDLE_ENV | Macros which define the type of handle to free.  
?SQL_HANDLE_DBC  
?SQL_HANDLE_STMT

RefHandle -> term() Reference to the handle.

Timeout -> integer() | Max time (ms) for serving the request.  
infinity
Result -> |
| Result macro. |
| ?SQL_SUCCESS |
| ?SQL_ERROR |
| ?SQL_INVALID_HANDLE |

1.3.13 sql_get_connect_attr/[4, 5]

Description:

Returns the current setting of a connection attribute. See SQLGetConnectAttr in [1].

Differences from the ODBC Function:

Only the following attributes, and their possible values, are supported (through macros). More information can be found under SQLSetConnectAttr in [1]. Driver-specific attributes are not supported through macros, but can be retrieved, if they are of character or signed/unsigned long integer types.

- ?SQL_ATTR_ACCESS_MODE
- ?SQL_ATTR_AUTOCOMMIT
- ?SQL_ATTR_ODBC_CURSORS
- ?SQL_ATTR_TRACE
- ?SQL_ATTR_TRACEFILE
- ?SQL_ATTR_TRANSLATE_LIB
- ?SQL_ATTR_TRANSLATE_OPTION

According to [1], BufLen (BufferLength) can be set to SQL_NTS. This is probably not correct, since it would make it impossible for the driver to detect that data needs to be truncated. Hence, the SQL_NTS value has been disallowed.

The function takes a BufType parameter to distinguish between character type attributes and numeric type attributes. For character data the maximum string length must be supplied (allow room for null-termination). For driver-specific numeric type attributes, a subtype must be supplied. The returned value is either a tuple containing the attribute string and its length, or an integer, depending on the specified buffer type.

The parameters Server and Timeout have been added. The output parameters ValuePtr and StringLengthPtr of the ODBC function have been changed into the returned values CharValue and LenCharValue for character type attributes and NumValue for integer types. The input parameter BufferLength has been included in the BufType parameter. BufLen must be > 0.

Arguments and Return Value:
sql_get_connect_attr(Server, RefConnHandle, Attr, BufType) ->
sql_get_connect_attr(Server, RefConnHandle, Attr, BufType, Timeout) ->
{Result, Value}

Server -> pid() | The pid of the server,
Name | a registered name,
{global, Name} | a globally registered
{Name, Node} name, or a registered
name on a remote
node.

RefConnHandle -> term() Reference to the
connection handle.

Attr -> integer() One of the attributes
described above or a
driver-specific
attribute.

BufType -> {?SQL_C_CHAR, BufLen} | The buffer type used
{?SQL_C_ULONG} for retrieving the
data. For character
type data also the
buffer size. For
integer type data that
is driver-specific,
also a subtype.

BufLen -> integer() Buffer size (>0) for
character type data. Allow room for
null-termination

IntType -> {?SQL_IS_UINTEGER | Used only for driver-
?SQL_IS_INTEGER specific attributes. See
SQLGetConnectAttr
in [1].

Timeout -> integer() | Max time (ms) for
infinity serving the request.

Result -> {?SQL_SUCCESS | Result macro.
?SQL_SUCCESS_WITH_INFO |
?SQL_NO_DATA |
?SQL_ERROR |
?SQL_INVALID_HANDLE

Value -> {CharValue, LenCharValue} | Attribute data.
NumValue

CharValue -> string() The value of the
attribute when of
character type.

LenCharValue -> integer() The length of
NumValue -> integer()

CharValue before truncation.
The value of the attribute when of numeric type.

1.3.14 sql_get_diag_rec/[5, 6]

Description:

Retrieves the current values of multiple fields of a diagnostic record that contains error, warning, and status information. See SQLGetDiagRec in [1].

Differences from the ODBC Function:

Retrieving information associated with descriptor handles is not supported.

The parameters Server and Timeout have been added. The output parameters SqlState, NativeErrorPtr, MessageText, and TextLengthPtr of the ODBC function have been changed into the returned values SqlState, NativeErr,ErrMsg, and LenErrMsg. BufLenErrMsg must be > 0.

Arguments and Return Value:

sql_get_diag_rec(Server, HandleType, RefHandle, RecNum, BufLenErrMsg) ->

sql_get_diag_rec(Server, HandleType, RefHandle, RecNum, BufLenErrMsg, Timeout) ->

{Result, SqlState, NativeErr, {ErrMsg, LenErrMsg}}

Server -> pid() | The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.

Name | {global, Name} |

{Name, Node} |

HandleType -> ?SQL_HANDLE_ENV | The type of handle for which to retrieve information.

?SQL_HANDLE_DBC |

?SQL_HANDLE_STMT |

RefHandle -> term() Reference to the handle.

RecNum -> integer() Indicates the status record from which to retrieve information (> 0).

BufLenErrMsg -> integer() Length ofErrMsg buffer (>0). Allow room for null-termination
Specification

1.3.15 sql_num_result_cols/[2, 3]

Description:

Returns the number of columns in a result set. See SQLNumResultCols in [1].

Differences from the ODBC Function:

The parameters Server and Timeout have been added. The output parameter ColumnCountPtr of SQLNumResultCols has been changed into the returned value ColCount.

Arguments and Return Value:

sql_num_result_cols(Server, RefStmtHandle) ->
sql_num_result_cols(Server, RefStmtHandle, Timeout) ->
{Result, ColCount}

Server -> pid() | Name | {global, Name} | {Name, Node} | The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.
RefStmtHandle -> term() | Reference to the statement handle.
Timeout -> integer() | infinity | Max time (ms) for serving the request.
Result -> ?SQL_SUCCESS | ?SQL_SUCCESS_WITH_INFO | Result macro.
1.3.16 sql_row_count/[2, 3]

Description:

Returns the number of rows affected by an UPDATE, INSERT, or DELETE statement. See SQLRowCount in [1].

Differences from the ODBC Function:

The parameters Server and Timeout have been added. The output parameter RowCountPtr of the ODBC function has been changed into the returned value RowCount.

Arguments and Return Value:

sql_row_count(Server, RefStmtHandle) ->
sql_row_count(Server, RefStmtHandle, Timeout) ->
   {Result, RowCount}

Server -> pid() | Name | {global, Name} | {Name, Node}
        The pid of the server, a registered name, a globally registered name, or a registered name on a remote node.

RefStmtHandle -> term()
        Reference to the statement handle.

Timeout -> integer() | infinity
        Max time (ms) for serving the request.

Result -> ?SQL_SUCCESS | ?SQL_SUCCESS_WITH_INFO |
         ?SQL_ERROR | ?SQL_INVALID_HANDLE
        Result macro.

RowCount -> integer()
        The number of affected rows. If the number of affected rows is not available -1 is returned. For exceptions, see SQLRowCount in [1].
1.3.17  sql_set_connect_attr/[5, 6]

Description:

Sets attributes that govern aspects of connections. See SQLSetConnectAttr in [1]. The supported attributes are listed under sql_get_connect_attr/[4, 5].

Driver-specific attributes are not supported through macros, but can be set, if they are strings or signed/unsigned long integers.

Differences from the ODBC Function:

Only character and signed/unsigned long integer attribute types are supported.

The parameters Server and Timeout have been added. The input parameter StringLength of the ODBC function has been replaced with the input parameter BufType.

Arguments and Return Value:

sql_set_connect_attr(Server, RefConnHandle, Attr, Value, BufType) ->
sql_set_connect_attr(Server, RefConnHandle, Attr, Value, BufType, Timeout) ->

Result

Server -> pid() | Name | {global, Name} | {Name, Node}

RefConnHandle -> term()

Attr -> integer()

Value -> string() | integer()

BufType -> ?SQL_C_CHAR |
1.3.18 sql_set_env_attr/[5, 6]

Description:
Sets attributes that govern aspects of environments. See SQLSetEnvAttr in [1].

The following attributes, and their possible values, are supported (through macros). More information can be found under SQLSetEnvAttr in [1]. Other data types than character or unsigned long integer are not supported.

- ?SQL_ATTR_ODBC_VERSION

Differences from the ODBC Function:
Only character and unsigned long integer attribute types are supported.

The parameters Server and Timeout have been added. The input parameter StringLength of the ODBC function has been replaced with the input parameter BufType.

Arguments and Return Value:

sql_set_env_attr(Server, RefEnvHandle, Attr, Value, BufType) ->
sql_set_env_attr(Server, RefEnvHandle, Attr, Value, BufType, Timeout) ->

Result

Server -> pid() | The pid of the server,
Name | a registered name,
{global, Name} | a globally registered
 alloc_buffer/[3, 4]

Description:

Allocates a deferred data buffer and an associated length/indicator buffer.

Arguments and Return Value:

alloc_buffer(Server, BufCType, Size) ->
alloc_buffer(Server, BufCType, Size, Timeout) ->
  {ok, RefBuf}

Server -> pid() | Name | {global, Name} | {Name, Node} | The pid of the server process, a registered name, or a registered name on a remote node.
BufCType -> ?SQL_C_CHAR | ?SQL_C_BINARY | The C data type of the buffer.
Size -> integer() | The buffer size (>0). For character data, allow room for null-termination
Timeout -> integer() | infinity | Max time (ms) for serving the request.
RefBuf -> term() | A handle to the buffer.
1.3.20 dealloc_buffer/[2, 3]

Description:

Deallocates a deferred data buffer and the associated length/indicator buffer.

Arguments and Return Value:

dealloc_buffer(Server, RefBuf) ->
dealloc_buffer(Server, RefBuf, Timeout) ->
    ok

Server ->    pid() | The pid of the server process,  
    Name | a registered name,  
    {global, Name} | a globally registered name, or  
    {Name, Node} | a registered name on a remote node.  
RefBuf -> term() A handle to the buffer.  
Timeout -> integer() | infinity  
            Max time (ms) for serving the request.

1.3.21 read_buffer/[2, 3]

Description:

Returns the contents of a deferred data buffer and its associated length/indicator buffer. Used in connection with sql_fetch/[2, 3].

NOTE: When the returned Value is a binary it is not given that it can be converted into an Erlang term with binary_to_term/1. It can be done if Value was created using term_to_binary/1 and if it was stored in the database through this interface. Value can always be converted into a list with binary_to_list/1.

Arguments and Return Value:

read_buffer(Server, RefBuf) ->
read_buffer(Server, RefBuf, Timeout) ->
    {ok, {Value, LenInd}}

Server ->    pid() | The pid of the server process,  
    Name | a registered name,  
    {global, Name} | a globally registered name, or  
    {Name, Node} | a registered name on a remote node.  
RefBuf -> term() A handle to the buffer.  
Timeout -> integer() | infinity  
            Max time (ms) for serving the request.
Value -> infinity
             string() | Contents of the buffer associated
             binary() with RefBuf. The type is determined
             by the type chosen for RefBuf when it
             was allocated.

LenInd -> integer() | Length/indicator value associated with
            ?SQL_NULL_DATA | RefBuf.
            ?SQL_NO_TOTAL

1.3.22 write_buffer/[3, 4]

Description:

Writes a value to a deferred data buffer.

NOTE: If the function fails, the contents of the deferred data buffer referred
by RefBuf, and the associated length/indicator buffer, is undefined.

Arguments and Return Value:

write_buffer(Server, RefBuf, {Value, LenInd}) ->
write_buffer(Server, RefBuf, {Value, LenInd}, Timeout) ->
   ok

Server -> pid() | The pid of the server,
           Name | a registered name,
           {global, Name} | a globally registered name,
                          {Name, Node} or a registered name on a
                          remote node.

RefBuf -> term() A handle to the buffer.

Value -> string() | Value to write to the buffer
           binary() associated with RefBuf.

LenInd -> integer() | Length/indicator value
                    ?SQL_NULL_DATA | associated with RefBuf.
                    ?SQL_NO_TOTAL
                    ?SQL_DATA_AT_EXEC |
                    no_len_ind

Timeout -> integer() | Max time (ms) for serving
                     infinity the request.
1.4 Error Messages and Exceptions

Errors caused by inability to contact the C node, allocate memory, or otherwise call ODBC functions cause exceptions. Exceptions are common to all functions. Errors caused by ODBC not being able to execute calls are reported through returned errors.

These exceptions terminate the client only.

\{'EXIT', {badarg, M, F, A, ArgNo, Info}\} Argument is of wrong type or out of range.
\{'EXIT', {internal_error, Info}\} Internal error.
\{'EXIT', GenServerSpecificInfo\} Error detected by gen_server.

These cause the ODBC server, and the C node, to terminate as well:

\{'EXIT', {timeout, Info}\} Timeout expired.
\{'EXIT', {stopped, Reason}\} The ODBC server died.

1.5 References