PUBLIC
SQL Anywhere Server
Document Version: 17.01.0 – 2021-10-15

SQL Anywhere - Node.js API Reference

THE BEST RUN

Content

1 Node.js Application Programming

The Node.js API can be used to connect to SQL Anywhere databases, issue SQL queries, and obtain result sets.

The Node.js driver allows users to connect and perform queries on the database using JavaScript on Joyent's Node.js software platform. Drivers are available for various versions of Node.js.

The API interface is very similar to the SAP HANA Node.js Client, and allows users to connect, disconnect, execute, and prepare statements.

The driver is available for install through the NPM (Node Packaged Modules) web site: https://npmjs.org/ .

It can also be downloaded from https://github.com/sqlanywhere/node-sqlanywhere .

In this section:

1.1 Connection Class

Represents the connection to the database.

Syntax

```
class Connection
```

Members

All members of Connection, including inherited members.

Methods

Type	Method	Description
	commit(Function) [page 6]	Performs a commit on the connection.
	connect(String, Function) [page 7]	Connect using an existing connection.
	disconnect(Function) [page 8]	Closes the current connection.

Type	Method	Description
Result	exec(String, Array, Function) [page 9]	Executes the specified SQL statement.
Statement	prepare(String, Function) [page 10]	Prepares the specified SQL statement.
	rollback(Function) [page 11]	Performs a rollback on the connection.

Remarks

The following example uses synchronous calls to create a new connection to the database server, issue a SQL query against the server, display the result set, and then disconnect from the server.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( { ServerName: 'demo17', UserID: 'DBA', Password: 'sql' } )
console.log('Connected');
result = client.exec("SELECT * FROM Customers");
console.log( result );
client.disconnect()
console.log('Disconnected');
```

The following example does essentially the same thing using callbacks to perform asynchronous calls. Error checking is included.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql",
   function( err )
   {
       if( err )
       {
           console.error( "Connect error: ", err );
       }
       else
       {
           console.log( "Connected" )
           client.exec( "SELECT * FROM Customers",
               function( err, rows )
               {
                   if( err )
                   {
                       console.error( "Error: ", err );
                   }
                   else
                   {
                       console.log(rows)
                   }
               }
           );
           client.disconnect(
               function( err )
               {
                   if( err )
                   {
                       console.error( "Disconnect error: ", err );
                   }
                   else
                   {
                       console.log( "Disconnected" )
```

```
                        }
                    }
                );
            }
        }
);
```

The following example also uses callbacks but the functions are not inlined and the code is easier to understand.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql", async_connect );
function async_connect( err )
{
   if( err )
   {
       console.error( "Connect error: ", err );
   }
   else
   {
       console.log( "Connected" )
       client.exec( "SELECT * FROM Customers", async_results );
       client.disconnect( async_disco );
   }
}
function async_results( err, rows )
{
   if( err )
   {
       console.error( "Error: ", err );
   }
   else
   {
       console.log(rows)
   }
}
function async_disco( err )
{
   if( err )
   {
       console.error( "Disconnect error: ", err );
   }
   else
   {
       console.log( "Disconnected" )
   }
}
```

You can also pass connection parameters into the createConnection function, and those parameters are combined with those in the connect() function call to get the connection string used for the connection. You can use a hash of connection parameters or a connection string fragment in either call.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection( { uid: 'dba'; pwd: 'sql' } );
client.connect( 'server=MyServer;host=localhost' );
// the connection string that will be used is
// "uid=dba;pwd=sql;server=MyServer;host=localhost"
```

In this section:

connect(String, Function) Method [page 7]
: Connect using an existing connection.

disconnect(Function) Method [page 8]
: Closes the current connection.

exec(String, Array, Function) Method [page 9]
: Executes the specified SQL statement.

prepare(String, Function) Method [page 10]
: Prepares the specified SQL statement.

rollback(Function) Method [page 11]
: Performs a rollback on the connection.

1.1.1 commit(Function) Method

Performs a commit on the connection.

Syntax

```
connection.commit (callback)
```

Parameters

Type	Name	Description
Function	callback	The optional callback function. (type: Function)

Remarks

This method performs a commit on the connection. By default, inserts, updates, and deletes are not committed upon disconnection from the database server.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err ) {
};
```

The following synchronous example shows how to use the commit method.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare(
    "INSERT INTO Departments "
    + "( DepartmentID, DepartmentName, DepartmentHeadID )"
    + "VALUES (?,?,?)" );
result = stmt.exec( [600, 'Eastern Sales', 902] );
result += stmt.exec( [700, 'Western Sales', 902] );
stmt.drop();
console.log( "Number of rows added: " + result );
result = client.exec( "SELECT * FROM Departments" );
console.log( result );
client.commit();
client.disconnect();
```

1.1.2 connect(String, Function) Method

Connect using an existing connection.

Syntax

```
connection.connect (conn_string, callback)
```

Parameters

Type	Name	Description
String	conn_string	A valid connection string (type: String)
Function	callback	The optional callback function. (type: Function)

Remarks

Creates a new connection.

This method creates a new connection using either a connection string or a hash of connection parameters passed in as a parameter. Before the end of the program, the connection should be disconnected using the disconnect method to free up resources.

The CharSet (CS) connection parameter CS=UTF-8 is always appended to the end of the connection string by the driver since it is required that all strings are sent in that encoding.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err )
{
};
```

The following synchronous example shows how to use the connect method. It is not necessary to specify the CHARSET=UTF-8 connection parameter since it is always added automatically.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql;CHARSET=UTF-8" );
```

Related Information

disconnect(Function) Method [page 8]

1.1.3 disconnect(Function) Method

Closes the current connection.

Syntax

```
connection.disconnect (callback)
```

Parameters

Type	Name	Description
Function	callback	The optional callback function. (type: Function)

Remarks

This method closes the current connection and should be called before the program ends to free up resources.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err )
{
};
```

The following synchronous example shows how to use the disconnect method.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
client.disconnect()
```

Related Information

connect(String, Function) Method [page 7]

1.1.4 exec(String, Array, Function) Method

Executes the specified SQL statement.

Syntax

```
connection.exec (sql, params, callback)
```

Parameters

Type	Name	Description
String	sql	The SQL statement to be executed. (type: String)
Array	params	Optional array of bind parameters. (type: Array)
Function	callback	The optional callback function. (type: Function)

Returns

If no callback is specified, the result is returned.

Remarks

This method takes in a SQL statement and an optional array of bind parameters to execute.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err, result )
{
};
```

For queries producing result sets, the result set object is returned as the second parameter of the callback. For insert, update and delete statements, the number of rows affected is returned as the second parameter of the callback. For other statements, result is undefined.

The following synchronous example shows how to use the exec method.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
result = client.exec("SELECT * FROM Customers");
console.log( result );
client.disconnect()
```

The following synchronous example shows how to specify bind parameters.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" );
result = client.exec(
   "SELECT * FROM Customers WHERE ID >=? AND ID <?",
   [300, 400] );
console.log( result );
client.disconnect()
```

1.1.5 prepare(String, Function) Method

Prepares the specified SQL statement.

Syntax

```
connection.prepare (sql, callback)
```

Parameters

Type	Name	Description
String	sql	The SQL statement to be executed. (type: Function)
Function	callback	The optional callback function. (type: Function)

Returns

If no callback is specified, a Statement object is returned.

Remarks

This method prepares a SQL statement and returns a Statement object if successful.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err, Statement )
{
};
```

The following synchronous example shows how to use the prepare method.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
console.log( result );
client.disconnect();
```

1.1.6 rollback(Function) Method

Performs a rollback on the connection.

Syntax

```
connection.rollback (callback)
```

Parameters

Type	Name	Description
Function	callback	The optional callback function. (type: Function)

Remarks

This method performs a rollback on the connection.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err ) {
};
```

The following synchronous example shows how to use the rollback method.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
```

```
stmt = client.prepare(
   "INSERT INTO Departments "
   + "( DepartmentID, DepartmentName, DepartmentHeadID )"
   + "VALUES (?,?,?)" );
result = stmt.exec( [600, 'Eastern Sales', 902] );
result += stmt.exec( [700, 'Western Sales', 902] );
stmt.drop();
console.log( "Number of rows added: " + result );
result = client.exec( "SELECT * FROM Departments" );
console.log( result );
client.rollback();
client.disconnect();
```

1.2 Statement Class

Represents a prepared statement.

Syntax

```
class Statement
```

Members

All members of Statement, including inherited members.

Methods

Type	Method	Description
	drop(Function) [page 13]	Drops the statement.
result	exec(Array, Function) [page 13]	Executes the prepared SQL statement.

In this section:

drop(Function) Method [page 13]
Drops the statement.

exec(Array, Function) Method [page 13]
Executes the prepared SQL statement.

Related Information

prepare(String, Function) Method [page 10]

1.2.1 drop(Function) Method

Drops the statement.

Syntax

```
statement.drop (callback)
```

Parameters

Type	Name	Description
Function	callback	The optional callback function.

Remarks

This method drops the prepared statement and frees up resources.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err )
{
};
```

The following synchronous example shows how to use the drop method on a prepared statement.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
stmt.drop();
console.log( result );
client.disconnect();
```

1.2.2 exec(Array, Function) Method

Executes the prepared SQL statement.

Syntax

```
statement.exec (params, callback)
```

Parameters

Type	Name	Description
Array	params	The optional array of bind parameters.
Function	callback	The optional callback function.

Returns

If no callback is specified, the result is returned.

Remarks

This method optionally takes in an array of bind parameters to execute.

This method can be either synchronous or asynchronous depending on whether or not a callback function is specified. The callback function is of the form:

```
function( err, result )
{
};
```

For queries producing result sets, the result set object is returned as the second parameter of the callback. For insert, update and delete statements, the number of rows affected is returned as the second parameter of the callback. For other statements, result is undefined.

The following synchronous example shows how to use the exec method on a prepared statement.

```
var sqlanywhere = require( 'sqlanywhere' );
var client = sqlanywhere.createConnection();
client.connect( "ServerName=demo17;UID=DBA;PWD=sql" )
stmt = client.prepare( "SELECT * FROM Customers WHERE ID >= ? AND ID < ?" );
result = stmt.exec( [200, 300] );
stmt.drop();
console.log( result );
client.disconnect();
```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP