

SQL Anywhere サーバ
文書バージョン: 17 - 2016-05-11

SQL Anywhere - Node.js API リファレンス

目次

1	Node.js アプリケーションプログラミング	3
1.1	Connection クラス.....	3
	commit(Function) メソッド.....	6
	connect(String, Function) メソッド.....	7
	disconnect(Function) メソッド.....	8
	exec(String, Array, Function) メソッド.....	9
	prepare(String, Function) メソッド.....	10
	rollback(Function) メソッド.....	11
1.2	Statement クラス.....	12
	drop(Function) メソッド.....	13
	exec(Array, Function) メソッド.....	14
2	このマニュアルの印刷、再生、および再配布	15

1 Node.js アプリケーションプログラミング

Node.js API は、SQL Anywhere データベースへの接続、SQL クエリの発行、結果セットの取得に使用できます。

Node.js ドライバを使用すると、Joyent の Node.js ソフトウェアプラットフォームで JavaScript を使用してデータベースに接続し、クエリを実行できます。様々なバージョンの Node.js のドライバが使用可能です。

API インタフェースは SAP HANA Node.js Client にきわめて似ており、文の接続、接続解除、実行、準備が可能です。

ドライバは、NPM (Node Packaged Modules) Web サイト (<https://npmjs.org/>) で入手できます。

また、<https://github.com/sqlanywhere/node-sqlanywhere> からダウンロードすることもできます。

i 注記

主な **SQL Anywhere** マニュアルをお探しですか。マニュアルをローカルにインストールした場合は、Windows のスタートメニューを使用してアクセスするか (Microsoft Windows)、C:\Program Files\SQL Anywhere 17\Documentation にナビゲートします。

また、DocCommentXchange の Web で、主な SQL Anywhere API リファレンスマニュアルにアクセスすることもできます。<http://dcx.sap.com>

このセクションの内容:

[Connection クラス \[3 ページ\]](#)

データベースへの接続を表します。

[Statement クラス \[12 ページ\]](#)

準備された文を表します。

1.1 Connection クラス

データベースへの接続を表します。

構文

```
class Connection
```

メンバー

Connection のすべてのメンバー (継承されたメンバーも含みます) を次に示します。

メソッド

タイプ	メソッド	説明
	commit(Function) [6 ページ]	接続に対してコミットを実行します。
	connect(String, Function) [7 ページ]	既存の接続を使用して接続します。
	disconnect(Function) [8 ページ]	現在の接続を閉じます。
結果	exec(String, Array, Function) [9 ページ]	指定された SQL 文を実行します。
文	prepare(String, Function) [10 ページ]	指定された SQL 文を準備します。
	rollback(Function) [11 ページ]	接続に対してロールバックを実行します。

備考

次の例では、同期呼び出しを使用してデータベースサーバへの新規接続を作成し、サーバに対して SQL クエリを発行し、結果セットを表示し、サーバとの接続を切断します。

```
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');
```

次の例では、コールバックを使用したほぼ同じ非同期呼び出しを実行します。エラーチェックが含まれています。

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

```

次の例では、コールバックを使用しますが、関数はインライン化されておらず、分かりやすいコードになっています。

```

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" )
    }
}
}

```

接続パラメータを createConnection 関数に渡すことができます。また、それらのパラメータを connect() 関数呼び出しと組み合わせると、接続で使用する接続文字列を取得できます。接続パラメータのハッシュまたは接続文字列フラグメントをどの呼び出しでも使用できます。

```

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"

```

このセクションの内容:

[commit\(Function\) メソッド \[6 ページ\]](#)

接続に対してコミットを実行します。

[connect\(String, Function\) メソッド \[7 ページ\]](#)

既存の接続を使用して接続します。

[disconnect\(Function\) メソッド \[8 ページ\]](#)

現在の接続を閉じます。

[exec\(String, Array, Function\) メソッド \[9 ページ\]](#)

指定された SQL 文を実行します。

[prepare\(String, Function\) メソッド \[10 ページ\]](#)

指定された SQL 文を準備します。

[rollback\(Function\) メソッド \[11 ページ\]](#)

接続に対してロールバックを実行します。

1.1.1 commit(Function) メソッド

接続に対してコミットを実行します。

構文

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

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは、接続に対してコミットを実行します。デフォルトでは、データベースサーバからの切断時に挿入、更新、削除はコミットされません。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

次の同期の例は、commit メソッドの使用方法を示します。

```
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) メソッド

既存の接続を使用して接続します。

構文

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

パラメータ

タイプ	名前	説明
String	conn_string	有効な接続文字列 (型: 文字列)。
Function	callback	オプションコールバック関数 (型: 関数)。

備考

新しい接続を作成します。

このメソッドは、パラメータとして渡される接続文字列または接続パラメータのハッシュを使用して新しい接続を作成します。プログラムの終了前に、disconnect メソッドを使用してリソースを解放し、接続を切断する必要があります。

CharSet (CS) 接続パラメータ CS=UTF-8 は、すべての文字列がそのエンコーディングで送信する必要があるため、ドライバによって接続文字列の最後に常に付加されます。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err )
```

```
{
};
```

次の同期の例は、connect メソッドの使用方を示します。CHARSET=UTF-8 接続パラメータは自動的に追加されるため、指定する必要はありません。

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

関連情報

[disconnect\(Function\) メソッド \[8 ページ\]](#)

1.1.3 disconnect(Function) メソッド

現在の接続を閉じます。

構文

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

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは現在の接続を閉じます。リソースを解放するためにプログラムを終了する前に呼び出す必要があります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

次の同期の例は、disconnect メソッドの使用方を示します。

```
var sqlanywhere = require( 'sqlanywhere' );
```



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

関連情報

[connect\(String, Function\) メソッド \[7 ページ\]](#)

1.1.4 exec(String, Array, Function) メソッド

指定された SQL 文を実行します。

構文

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

パラメータ

タイプ	名前	説明
String	sql	実行される SQL 文 (型: 文字列)。
Array	params	バインドパラメータのオプション配列 (型: 配列)。
Function	callback	オプションコールバック関数 (型: 関数)。

戻り値

コールバックが指定されていない場合、結果が返されます。

備考

このメソッドは、SQL 文および実行するバインドパラメータのオプション配列を取ります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

```
function( err, result )
```

```
{
};
```

結果セットを生成するクエリの場合、結果セットオブジェクトがコールバックの 2 番目のパラメータとして返されます。insert、update、delete 文の場合、影響を受けるローの数がコールバックの 2 番目のパラメータとして返されます。他の文の場合、結果は未定義です。

次の同期の例は、exec メソッドの使用法を示します。

```
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();
```

次の同期の例は、バインドパラメータを指定する方法を示します。

```
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) メソッド

指定された SQL 文を準備します。

構文

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

パラメータ

タイプ	名前	説明
String	sql	実行される SQL 文 (型: 関数)。
Function	callback	オプションコールバック関数 (型: 関数)。

戻り値

コールバックが指定されていない場合、Statement オブジェクトが返されます。

備考

このメソッドは、SQL 文を準備し、成功した場合に Statement オブジェクトを返します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

次の同期の例は、prepare メソッドの使用方法を示します。

```
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) メソッド

接続に対してロールバックを実行します。

構文

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

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数 (型: 関数)。

備考

このメソッドは、接続に対してロールバックを実行します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

次の同期の例は、rollback メソッドの使用方法を示します。

```
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 クラス

準備された文を表します。

構文

```
class Statement
```

メンバー

Statement のすべてのメンバー (継承されたメンバーも含まれます) を次に示します。

メソッド

タイプ	メソッド	説明
	drop(Function) [13 ページ]	文を削除します。
result	exec(Array, Function) [14 ページ]	準備された SQL 文を実行します。

このセクションの内容:

[drop\(Function\) メソッド \[13 ページ\]](#)

文を削除します。

[exec\(Array, Function\) メソッド \[14 ページ\]](#)

準備された SQL 文を実行します。

関連情報

[prepare\(String, Function\) メソッド \[10 ページ\]](#)

1.2.1 drop(Function) メソッド

文を削除します。

構文

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

パラメータ

タイプ	名前	説明
Function	callback	オプションコールバック関数。

備考

このメソッドは準備文を削除し、リソースを解放します。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

次の同期の例は、準備された文で drop メソッドを使用する方法を示します。

```
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) メソッド

準備された SQL 文を実行します。

構文

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

パラメータ

タイプ	名前	説明
Array	params	バインドパラメータのオプション配列。
Function	callback	オプションコールバック関数。

戻り値

コールバックが指定されていない場合、結果が返されます。

備考

このメソッドは、実行するバインドパラメータの配列を取ります。

このメソッドは、コールバック関数を指定するかどうかによって、同期の場合も非同期の場合もあります。コールバック関数は次の形式を取ります。

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

結果セットを生成するクエリの場合、結果セットオブジェクトがコールバックの 2 番目のパラメータとして返されます。insert、update、delete 文の場合、影響を受けるローの数がコールバックの 2 番目のパラメータとして返されます。他の文の場合、結果は未定義です。

次の同期の例は、準備された文で exec メソッドを使用する方法を示します。

```
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();
```

2 このマニュアルの印刷、再生、および再配布

次の条件に従うかぎり、このマニュアルの全部または一部を使用、印刷、再生、配布することができます。

1. ここに示したものとそれ以外のすべての著作権と商標の表示をすべてのコピーに含めること。
2. マニュアルに変更を加えないこと。
3. SAP 以外の人間がマニュアルの著者または情報源であるかのように示す一切の行為をしないこと。

ここに記載された情報は事前の通知なしに変更されることがあります。

重要免責事項および法的情報

コードサンプル

この文書に含まれるソフトウェアコード及び / 又はコードライン / 文字列 (「コード」) はすべてサンプルとしてのみ提供されるものであり、本稼働システム環境で使用することが目的ではありません。「コード」は、特定のコードの構文及び表現規則を分かりやすく説明及び視覚化することのみを目的としています。SAP は、この文書に記載される「コード」の正確性及び完全性の保証を行いません。更に、SAP は、「コード」の使用により発生したエラー又は損害が SAP の故意又は重大な過失が原因で発生させたものでない限り、そのエラー又は損害に対して一切責任を負いません。

アクセシビリティ

この SAP 文書に含まれる情報は、公開日現在のアクセシビリティ基準に関する SAP の最新の見解を表明するものであり、ソフトウェア製品のアクセシビリティ機能の確実な提供方法に関する拘束力のあるガイドラインとして意図されるものではありません。SAP は、この文書に関する一切の責任を明確に放棄するものです。ただし、この免責事項は、SAP の意図的な違法行為または重大な過失による場合は、適用されません。さらに、この文書により SAP の直接的または間接的な契約上の義務が発生することは一切ありません。

ジェンダーニュートラルな表現

SAP 文書では、可能な限りジェンダーニュートラルな表現を使用しています。文脈により、文書の読者は「あなた」と直接的な呼ばれ方をされたり、ジェンダーニュートラルな名詞 (例:「販売員」又は「勤務日数」) で表現されます。ただし、男女両方を指すとき、三人称単数形の使用が避けられない又はジェンダーニュートラルな名詞が存在しない場合、SAP はその名詞又は代名詞の男性形を使用する権利を有します。これは、文書を分かりやすくするためです。

インターネットハイパーリンク

SAP 文書にはインターネットへのハイパーリンクが含まれる場合があります。これらのハイパーリンクは、関連情報を見いだすヒントを提供することが目的です。SAP は、この関連情報の可用性や正確性又はこの情報が特定の目的に役立つことの保証は行いません。SAP は、関連情報の使用により発生した損害が、SAP の重大な過失又は意図的な違法行為が原因で発生したものでない限り、その損害に対して一切責任を負いません。すべてのリンクは、透明性を目的に分類されています (<http://help.sap.com/disclaimer> を参照)。

[go.sap.com/registration/
contact.html](http://go.sap.com/registration/contact.html)

© 2016 SAP SE or an SAP affiliate company. All rights reserved.

本書のいかなる部分も、SAP SE 又は SAP の関連会社の明示的な許可なくして、いかなる形式でも、いかなる目的にも複製又は伝送することはできません。本書に記載された情報は、予告なしに変更されることがあります。SAP SE 及びその頒布業者によって販売される一部のソフトウェア製品には、他のソフトウェアベンダーの専有ソフトウェアコンポーネントが含まれています。製品仕様は、国ごとに変わる場合があります。

これらの文書は、いかなる種類の表明又は保証もなしで、情報提供のみを目的として、SAP SE 又はその関連会社によって提供され、SAP 又はその関連会社は、これら文書に関する誤記脱落等の過失に対する責任を負うものではありません。SAP 又はその関連会社の製品及びサービスに対する唯一の保証は、当該製品及びサービスに伴う明示的な保証がある場合に、これに規定されたものに限られます。本書のいかなる記述も、追加の保証となるものではありません。

本書に記載される SAP 及びその他の SAP の製品やサービス、並びにそれらの個々のロゴは、ドイツ及びその他の国における SAP SE (又は SAP の関連会社) の商標若しくは登録商標です。本書に記載されたその他のすべての製品およびサービス名は、それぞれの企業の商標です。

商標に関する詳細の情報や通知については、<http://www.sap.com/corporate-en/legal/copyright/index.epx> をご覧ください。