MySQL Connector/Node.js Release Notes

Abstract

This document contains release notes for the changes in each release of MySQL Connector/Node.js.

For additional Connector/Node.js documentation, see http://dev.mysql.com/.

Updates to these notes occur as new product features are added, so that everybody can follow the development process. If a recent version is listed here that you cannot find on the download page (https://dev.mysql.com/ downloads/), the version has not yet been released.

The documentation included in source and binary distributions may not be fully up to date with respect to release note entries because integration of the documentation occurs at release build time. For the most up-to-date release notes, please refer to the online documentation instead.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2021-09-01 (revision: 23257)

Table of Contents

Preface and Legal Notices ................................................................. 1
Changes in MySQL Connector/Node.js 8.0 ............................................. 3
  Changes in MySQL Connector/Node.js 8.0.26 (2021-07-20, General Availability) .......... 3
  Changes in MySQL Connector/Node.js 8.0.25 (2021-05-11, General Availability) .......... 4
  Changes in MySQL Connector/Node.js 8.0.24 (2021-04-20, General Availability) .......... 4
  Changes in MySQL Connector/Node.js 8.0.23 (2021-01-18, General Availability) .......... 5
  Changes in MySQL Connector/Node.js 8.0.22 (2020-10-19, General Availability) .......... 5
  Changes in MySQL Connector/Node.js 8.0.21 (2020-07-13, General Availability) .......... 6
  Changes in MySQL Connector/Node.js 8.0.20 (2020-04-27, General Availability) .......... 7
  Changes in MySQL Connector/Node.js 8.0.19 (2020-01-13, General Availability) .......... 7
  Changes in MySQL Connector/Node.js 8.0.18 (2019-10-14, General Availability) .......... 8
  Changes in MySQL Connector/Node.js 8.0.17 (2019-07-22, General Availability) .......... 8
  Changes in MySQL Connector/Node.js 8.0.16 (2019-04-25, General Availability) .......... 9
  Changes in MySQL Connector/Node.js 8.0.15 (2019-02-01, General Availability) .......... 10
  Changes in MySQL Connector/Node.js 8.0.14 (2019-01-21, General Availability) .......... 11
  Changes in MySQL Connector/Node.js 8.0.13 (2018-10-22, General Availability) .......... 11
  Changes in MySQL Connector/Node.js 8.0.12 (2018-07-27, General Availability) .......... 13
  Changes in MySQL Connector/Node.js 8.0.11 (2018-04-19, General Availability) .......... 14
  Changes in MySQL Connector/Node.js 8.0.10 (Skipped version number) .................... 14
  Changes in MySQL Connector/Node.js 8.0.9 (2018-01-30, Release Candidate) ............. 15
  Changes in MySQL Connector/Node.js 8.0.8 (2017-09-28, Development Milestone) ...... 15
  Changes in MySQL Connector/Node.js 8.0.7 (2017-07-10, Development Milestone) ...... 16
Changes in MySQL Connector/Node.js 1.0 ........................................... 17
  Changes in MySQL Connector/Node.js 1.0.6 (2017-03-07, Development Milestone) ........ 17
  Changes in MySQL Connector/Node.js 1.0.5 (2016-11-14, Development Milestone) ........ 18
  Changes in MySQL Connector/Node.js 1.0.4 (2016-10-10, Development Milestone) ........ 18
  Changes in MySQL Connector/Node.js 1.0.3 (2016-06-21, Development Milestone) ........ 18
  Changes in MySQL Connector/Node.js 1.0.2 (2016-04-11, Development Milestone) ........ 18
  Changes in MySQL Connector/Node.js 1.0.1 (Not released, Internal) .................... 19
  Changes in MySQL Connector/Node.js 1.0.0 (Not released, Internal) .................... 19

Preface and Legal Notices

This document contains release notes for the changes in each release of MySQL Connector/Node.js.
Legal Notices

Copyright © 1997, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download
Changes in MySQL Connector/Node.js 8.0

Changes in MySQL Connector/Node.js 8.0.26 (2021-07-20, General Availability)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- The TLSv1.0 and TLSv1.1 connection protocols are now deprecated and support for them is subject to removal in a future Connector/Node.js version. Using them emits warning level error.

- Refactored logic that managed connections to the MySQL server from the high-level API logic to instead implement X DevAPI components like the Session interface.

Bugs Fixed

- Replaced all instances of the deprecated new Buffer() API for the now recommended alternatives (Buffer.alloc(), Buffer.allocUnsafe() and Buffer.from()) to avoid unnecessary warning messages. (Bug #32820267, Bug #103419)

- For addOrReplaceOne() and replaceOne(), an error is now reported if the object provided as the second argument contains an "_id" property that does not match the value provided as the first argument.

  Previous behavior was unexpected. For example, when calling addOrReplaceOne('1', { _id: '2', name: 'foo' }), a document with "_id" = "1" was created if it did not exist, or if it did exist then any existing property called "name" was updated with the value "foo"; or one was created. (Bug #32763298)

- Now allow SQL statement instances to be executed with different placeholder values. (Bug #32750927)

- Now ensure that all floating point numbers are represented either by JavaScript numbers or by X DevAPI expression literals that are encoded as an X Protocol V_DOUBLE type; this to help avoid precision loss. (Bug #32687374, Bug #103124)

- Prevent column metadata callback functions used as push-based cursors in the execute() method to be unnecessarily called more than once for each result set generated by a statement. (Bug #32631911)
• Upon exceeding `connectTimeout` to a MySQL server, when a connection pool contained an idle connection that previously established a connection to same server and in the meantime the server became unresponsive but reachable; then an application would become unresponsive after trying to acquire a connection from the pool. (Bug #32205365)

• If `connectTimeout` was exceeded as the client tried connecting to a server, the connection socket was not implicitly closed. (Bug #32200234)

Changes in MySQL Connector/Node.js 8.0.25 (2021-05-11, General Availability)

This release contains no functional changes and is published to align the version number with the MySQL Server 8.0.25 release.

Changes in MySQL Connector/Node.js 8.0.24 (2021-04-20, General Availability)

• Functionality Added or Changed
  • Important Change: Starting in this version, Connector/Node.js will ignore compatibility with End-of-life Node.js versions. If you are planning to upgrade, make sure you are using Node.js 12.0.0 or higher.
  • Connector/Node.js no longer guarantees compatibility with End-of-life Node.js versions. Today this means only Node.js 12.0.0 or higher are officially supported.
  • Improved the inline documentation and added functionality for users to generate it locally using JSDoc. See `CONTRIBUTING.md` for instructions on how to generate the HTML documentation.
  • Improved server disconnection handling of X Protocol connections now creates a log entry and returns an error message, as needed, after Connector/Node.js receives a connection-close notice from the server. Connector/Node.js detects three new types of warning notices.
    • `Connection idle notice`: This notice applies to a server connection that remains idle for longer than the relevant timeout setting. Connector/Node.js closes the connection when it receives the notice in an active session or while a new session is being created. An attempt to use the invalid session returns the “Connection closed. Reason: connection idle too long” error message.
    • `Server shutdown notice`: If a connection-close notice is received in a session as a result of a server shutdown, Connector/Node.js terminates the session with the "Connection closed. Reason: server shutdown" error message. All other sessions that are connected to the same endpoint are removed from the pool, if connection pooling is used.
    • `Connection killed notice`: If the connection being killed from another client session, Connector/Node.js closes the connection when it receives the notice in an active session or while a new session is being created. An attempt to use the invalid session returns the “Connection closed. Reason: connection killed by a different session” error message.

• Bugs Fixed
  • The `getSession()` method now releases connections from a pool when an exception is raised while connecting to the server. (Bug #32366743, Bug #101928)
  • The `Session.getDefaultSchema()` method now returned ‘undefined’ instead of a valid Schema instance if a default schema is not set. (Bug #32136490)
Changes in MySQL Connector/Node.js 8.0.23 (2021-01-18, General Availability)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- Added custom debug functionality using the `NODE_DEBUG` environment variable to log and inspect low-level details for the application. Connector/Node.js now supports this feature and uses it, in particular, for logging information about the protocol messages (inbound and outbound) that are exchanged with the MySQL server. Messages sent by the client are available under the protocol:outbound scope, whereas messages sent by the server are available under the protocol:inbound scope.

For example, the following writes a textual protobuf representation of every Mysqlx.Crud.Find and Mysqlx.ResultSet.Row message to stderr:

```
```

Node 10 and later supports wildcard pattern matching, such as `NODE_DEBUG='protocol:inbound:*'` to filter out inbound messages and `NODE_DEBUG='*'` to show all logs. (Bug #31729043)

- Added a deprecation warning to the Connector/Node.js installation process that future Connector/Node.js versions do not guarantee compatibility with End-of-life Node.js versions. If you are planning to upgrade, make sure you are using Node.js 12.0.0 or higher.

Bugs Fixed

- Stored values in a DOUBLE column were truncated when encoded and sent to the MySQL server. For example, a value such as 1.000001 would become 1. They are now encoded properly as double-precision floating numbers to support the full range of 64-bit floating point precision in MySQL columns with the DOUBLE data type (having in mind possible rounding approximations performed by the database engine). (Bug #31734504)

- Improved consistency for method argument error handling when an argument is not set or uses JavaScript's "undefined". (Bug #31709879)

- Refactored code to disable all debugging utilities when debug mode is off. (Bug #31584269)

Changes in MySQL Connector/Node.js 8.0.22 (2020-10-19, General Availability)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- Improved test execution configuration to better align with other connectors. For example, unified environment variable names (such as changing `NODE_TEST_MYSQL_HOST` to `MYSQLX_HOST`). See the Connector/Node.js documentation for usage information.

Bugs Fixed

- Non-BIGINT values stored in BIGINT columns were not decoded properly in result sets. (Bug #31686805, Bug #100324)
• Fetched results from a SET column would only contain one value from the set. (Bug #31654667, Bug #100255)

• Deprecated the dbPassword and dbUser property names; which were aliases to the password and user properties. Their usage now emits deprecation level errors. (Bug #31599660)

• Added a SERVER_GONE error handler to avoid potential circular dependency warnings with Node.js >= 14.0.0. (Bug #31586107, Bug #99869)

• Restricted the offset() method to the CollectionFind and TableSelect APIs, as described in the X DevAPI specification. Using offset() on other APIs yielded this error: "Error: The server has gone away". Instead, this intended behavior is available by using a combination of "sort()" or "orderBy()" and "limit()". (Bug #31418813)

• The nextResult() method returned false against an empty result set, and now returns true. Alternatively, use hasData() to check if a result set has data. (Bug #31037211)

• The column.getType() method now returns the stringified type identifier when before it returned its numeric value. For example, DATETIME is now returned instead of 12. (Bug #30922711)

• Improved memory management for work performed by 3rd party APIs. (Bug #30845472)

• Added support for lazy decoding of binary column metadata content. (Bug #30845366)

Changes in MySQL Connector/Node.js 8.0.21 (2020-07-13, General Availability)

• Functionality Added or Changed

• Bugs Fixed

Functionality Added or Changed

• Creating a collection now supports options to enable validation of a JSON schema that documents must adhere to before they are permitted to be inserted or updated. The new ModifyCollection method allows updating the schema of an existing collection. In the createCollection method, the option to re-use an existing collection was renamed from ReuseExistingObject to reuseExisting.

Schema validation is performed by the server, which returns an error message if a document in a collection does not match the schema definition or if the server does not support validation.

If a given collection already exists in the database, the createCollection fails unless the reuseExisting property is enabled in an additional options object such as the following example:

```javascript
const mysqlx = require('@mysql/xdevapi');
mysqlx.getSession('mysqlx://localhost:33060')
    .then(session => {
        return session.getSchema('mySchema').createCollection('myCollection', { reuseExisting: true })
    });
```

You can also use the options object to, for example, create a server-side document validation schema. For that, you can include a schema property matching a valid JSON schema definition within an outer validation object. You should also include the level property where STRICT enables it and OFF disables it. For example:

```javascript
const mysqlx = require('@mysql/xdevapi');
const validation = { schema: { type: 'object', properties: { name: { type: 'string' } } }, level: mysqlx.Schema.ValidationLevel.STRICT }
mysqlx.getSession('mysqlx://localhost:33060')
    .then(session => {
        return session.getSchema('mySchema').createCollection('myCollection', { validation })
    });
```
The same level property logic applies to modifyCollection. Here's an example to enable a JSON schema on an existing collection (or to update it if it already exists) using the STRICT validation level:

```javascript
const mysqlx = require('@mysql/xdevapi');
const validation = { schema: { type: 'object', properties: { name: { type: 'string' } } }, level: mysqlx.Schema.ValidationLevel.STRICT }
mysqlx.getSession('mysqlx://localhost:33060')
  .then(session => {
    return session.getSchema('mySchema').modifyCollection('myCollection', { validation })
  });
```

**Bugs Fixed**

- Row values for columns of type BIGINT were not correctly decoded by Connector/Node.js. Upgrading the google-protobuf library (to 3.11.4) fixed this problem. (Bug #27570685)

**Changes in MySQL Connector/Node.js 8.0.20 (2020-04-27, General Availability)**

**Functionality Added or Changed**

- Added two new connection options that evaluate during the TLS handshake to restrict the negotiated TLS protocols and ciphers; along with those configured on the server that can further restrict the final choices. These new options are `tls-versions` to define the allowed TLS protocol versions, and `tls-ciphersuites` for the allowed cipher suites. These definitions are comma-separated, and accepted by the `getSession()` and `getClient()` methods.

  - **tls-versions**: accepts one or more of the following: TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3. Other values generate an error.
  - **tls-ciphersuites**: accepts IANA cipher suite names, as listed on IANA's [TLS Cipher Suites](https://www.iana.org/assignments/tls-cipher-suites/tls-cipher-suites.xhtml) page. Unsupported or unknown values are ignored.

  Example usage demonstrating both plain JavaScript and JSON configuration object formats:

  ```javascript
  # tls versions:
  mysqlx.getSession('mysqlx://root@localhost?tls-versions=[TLSv1,TLSv1.1,TLSv1.2,TLSv1.3]')
  mysqlx.getSession({ user: 'root', tls: { versions: ['TLSv1', 'TLSv1.1', 'TLSv1.2', 'TLSv1.3'] } })
  
  # tls ciphersuites
  mysqlx.getSession('mysqlx://root@localhost?tls-ciphersuites=[DHE-RSA-AES128-GCM-SHA256,DHE-RSA-AES256-SHA384]')
  mysqlx.getSession({ user: 'root', tls: { ciphersuites: ['DHE-RSA-AES128-GCM-SHA256', 'DHE-RSA-AES256-SHA384'] } })
  ```

  - For X DevAPI applications, when creating a new connection, if the connection data contains several target hosts that have no explicit priority assigned, the behavior of the failover logic now is the same as if all those target hosts have the same priority. That is, the next candidate for making a connection is chosen randomly from the remaining available hosts. If two hosts have the same priority then one is chosen at random.

**Changes in MySQL Connector/Node.js 8.0.19 (2020-01-13, General Availability)**

**Functionality Added or Changed**

- Added DNS SRV support.

**Bugs Fixed**

- For X DevAPI applications, when creating a new connection, if the connection data contains several target hosts that have no explicit priority assigned, the behavior of the failover logic now is the same as if all those target hosts have the same priority. That is, the next candidate for making a connection is chosen randomly from the remaining available hosts. If two hosts have the same priority then one is chosen at random.
Using the `mysqlx+srv` scheme-extension in a connection string (or enabling the `resolveSrv` option in a connection configuration object) allows to automatically resolve any SRV record available in a target DNS server or service discovery endpoint.

**Bugs Fixed**

- Improved the CRUD API error messages. (Bug #30423556)
- The `getAffectedItemsCount()` method did not function on result sets in v8.0.18. (Bug #30401962)
- The `Collection.existsInDatabase()` method did not function. (Bug #30401432)

**Changes in MySQL Connector/Node.js 8.0.18 (2019-10-14, General Availability)**

- **Functionality Added or Changed**
- **Bugs Fixed**

**Functionality Added or Changed**

- Implemented the X DevAPI cursor model, which includes adding methods such as `fetchOne()`, `fetchAll()`, `getColumns()`, `hasData()`, and `nextResult()`. For additional details, see the X DevAPI documentation about Working with Result Sets.

  Previously, handling result set data or metadata required specific callback functions when calling `execute()`. With this new interface, the connector automatically switches to this new pull-based cursor model if these callback functions are not provided.

- Improved `Collection.getOne()` performance by making the underlying lookup expression to only parse once, and having subsequent `Collection.getOne()` calls utilize server-side prepared statements.

- Added support to generate test coverage reports by running the likes of `npm run coverage`; see the bundled `CONTRIBUTING.md` for details and requirements. This was added to help users contribute patches.

- Added linter check support to help enforce coding style and convention rules for new contributions by running the likes of `npm run linter`; see the bundled `CONTRIBUTING.md` for details.

**Bugs Fixed**

- Added support for assigning Node.js Buffer values to expression or SQL query placeholders. (Bug #30163003, Bug #96480)

- MySQL column binary values (such as BLOB, BINARY, and VARBINARY) can now convert to proper Node.js Buffer instances. (Bug #30162858, Bug #96478)

- Inserting a raw Node.js Buffer value into MySQL BLOB field resulted in an error as the `content_type` was improperly set; it's now handled as a raw byte string by the X Plugin. (Bug #30158425)

- The padding characters used for fixed-lengthed columns now map to the collation code provided by the column metadata; previously it was based on the JavaScript native type of the values. (Bug #30030159)

**Changes in MySQL Connector/Node.js 8.0.17 (2019-07-22, General Availability)**

- **Functionality Added or Changed**
- **Bugs Fixed**
Functionality Added or Changed

• Document fields containing arrays can now be indexed by setting array to true in an index fields definition.

• Added support for the OVERLAPS and NOT OVERLAPS operators; which is equivalent to the SQL JSON_OVERLAPS() function.

  These binary operators are used with a general "expression operator expression" syntax; and the expressions return a JSON array or object. Example usage: `[1, 2, 3] overlaps $.list`

• Added support for the utf8mb4_0900_bin collation added in MySQL Server 8.0.17.

• The bundled README.md file was split and reformatted with some content moved into the new README.txt and CONTRIBUTING.md files.

Bugs Fixed

• The SQL CAST function did not work as a valid lookup expression. (Bug #29807792)

• Added backtick support for table column identifiers in valid expressions. (Bug #29789818)

• The DIV binary and NOT unary operators are now allowed; and are case-insensitive. (Bug #29771833, Bug #29771027)

• Collection.find() now supports the JavaScript Date type. (Bug #29766014)

• The collection.dropIndex method now silently fails if the index does not exist, as expected, when before it generated a "Can't DROP" error. (Bug #29765589)

• Column.getCollationName() would potentially return the incorrect name. (Bug #29704185)

Changes in MySQL Connector/Node.js 8.0.16 (2019-04-25, General Availability)

• X DevAPI Notes

• Functionality Added or Changed

• Bugs Fixed

X DevAPI Notes

• Connector/Node.js now supports connection attributes as key-value pairs that application programs can pass to the server. Connector/Node.js defines a default set of attributes, which can be disabled or enabled. In addition to these default attributes, applications can also provide their own set of custom attributes.

• Specify connection attributes as a connection-attributes parameter in a connection string, or by using the connectionAttributes property using either a plain JavaScript object or JSON notation to specify the connection configuration options.

  The connection-attributes parameter value must be either empty (the same as specifying true), a Boolean value (true or false to enable or disable the default attribute set), or a list of zero or more key=value pair specifiers separated by commas (to be sent in addition to the default attribute set). Within a list, a missing key value evaluates as NULL.

  The connectionAttributes property allows passing user-defined attributes to the application using either a plain JavaScript object or JSON notation to specify the connection configuration options. Define each attribute in a nested object under connectionAttributes where the property names matches the attribute names, and the property values match the attribute values.
Unlike `connection-attributes`, and while using plain JavaScript objects or JSON notation, if the `connectionAttributes` object contains duplicate keys then no error is thrown and the last value specified for a duplicate object key is chosen as the effective attribute value.

Examples:

Not sending the default client-defined attributes:

```javascript
mysqlx.getSession('{ "user": "root", "connectionAttributes": false }')
```

```javascript
mysqlx.getSession('mysqlx://root@localhost?connection-attributes=false')
```

```javascript
mysqlx.getSession({ user: 'root', connectionAttributes: { foo: 'bar', baz: 'qux', quux: '' } })
```

```javascript
mysqlx.getSession('mysqlx://root@localhost?connection-attributes=[foo=bar,baz=qux,quux]')
```

Application-defined attribute names cannot begin with `_` because such names are reserved for internal attributes.

If connection attributes are not specified in a valid way, an error occurs and the connection attempt fails.

For general information about connection attributes, see [Performance Schema Connection Attribute Tables](#).

**Functionality Added or Changed**

- Optimized the reuse of existing connections through `client.getSession()` by only re-authenticating if required.

- For X DevAPI, performance for statements that are executed repeatedly (two or more times) is improved by using server-side prepared statements for the second and subsequent executions. This happens internally; applications need take no action and API behavior should be the same as previously. For statements that change, repreparation occurs as needed. Providing different data values or different `offset()` or `limit()` values does not count as a change. Instead, the new values are passed to a new invocation of the previously prepared statement.

**Bugs Fixed**

- Idle pooled connections to MySQL Server were not reused, and instead new connections had to be recreated. (Bug #29436892)

- Executing `client.close()` would not close all associated connections in the connection pool. (Bug #29428477)

- `connectTimeout` instead of `maxIdleTime` determined whether idle connections in the connection pool were reused rather than creating new connections. (Bug #29427271)

- Released connections from the connection pool were not being reset and reused; instead new connections were being made. (Bug #29392088)

- Date values in documents were converted to empty objects when inserted into a collection. (Bug #29179767, Bug #93839)

- A `queueTimeout` value other than 0 (infinite) prevented the acquisition of old released connections from the connection pool. (Bug #29179372, Bug #93841)

**Changes in MySQL Connector/Node.js 8.0.15 (2019-02-01, General Availability)**

This release contains no functional changes and is published to align version number with the MySQL Server 8.0.15 release.
Changes in MySQL Connector/Node.js 8.0.14 (2019-01-21, General Availability)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- Removed deprecation notices from the count() methods.
- Setting the default schema via the connection now sets the default schema on the server; meaning, subsequent queries executed using session.sql() do not need to specify the schema.

Bugs Fixed

- Setting the default schema with the connection URI using a schema name that contained special characters (that would need to be percent-encoded) would result in the percent-encoded name being used instead of the original one (e.g. "%25%26%5E%5E_" instead of "%&^*_"). (Bug #28990682)
- An error is once again thrown if sslOption's 'ca' is different than the certificate authority used to sign the server certificate, or if the server certificate has been revoked. (Bug #28977649)
- Attempting to use false-like values such as 0, false, null, and undefined would emit errors when updating or inserting documents in a collection or rows in a table. Additionally, now boolean values become numeric values (true=1, false=0) while null and undefined are converted to MySQL's NULL type. (Bug #28970727, Bug #93315)
- Collection.existsInDatabase() always returned true if any other collection existed in the database. (Bug #28745240)
- Configuring a default schema from the connection string would create the schema if it did not exist. Now, an "Unknown database" error is thrown instead.
- An unexpected notice could result in an unexpected halt of the client.

Changes in MySQL Connector/Node.js 8.0.13 (2018-10-22, General Availability)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- To go with the existing asynchronous mysqlx.getSession(conn_str) method, a new synchronous mysqlx.getClient(conn_str, options) method was added that creates a connection pool handler that provides an asynchronous getSession() method to create and retrieve connections from the pool. The collection pooling options are:
  - enabled: enables or disables connection pooling; boolean and defaults to true.
  - maxSize: maximum number of connections available in the pool; positive integer and defaults to 25.
  - maxIdleTime: maximum number of milliseconds a connection can be idle in the queue before being closed; integer >= 0 and defaults to 0 (infinite).
  - queueTimeout: maximum number of milliseconds a request will wait for a connection to become available; integer >= 0 and defaults to 0 (infinite).
This is different than connectTimeout that’s used for non-pooling. In a pooling scenario, there might already be connections in the pool and queueTimeout controls how long to wait for a connection in the pool.

Example usage:

```javascript
var mysqlx = require('@mysql/xdevapi')
var client = mysqlx.getClient(
  { user: 'root', host: 'localhost', port: 33060 },
  { pooling: { enabled: true, maxIdleTime: 5000, maxSize: 25, queueTimeout: 20000 } }
);

client.getSession()
  .then(session => {
    console.log(session.inspect())
    return session.close() // the connection becomes idle in the client pool
  })
  .then(() => {
    return client.getSession()
  })
  .then(session => {
    console.log(session.inspect())
    return client.close() // closes all connections and destroys the pool
  });
```

Closing a session attached to the pool makes the connection available in the pool for subsequent `getSession()` calls, while closing (destroying) the pool effectively closes all server connections.

- Added a connection timeout query parameter. This defines the length of time (milliseconds) the client waits for a MySQL server to become available in the given network addresses. It was added to both the `mysqlx.getSession()` (non-pooling sessions) and `mysqlx.getClient()` (pooling sessions) interfaces. This option defaults to 10000 (10 seconds). The value 0 disables the timeout so the client will wait until the underlying socket (platform dependent) times out.

Similar to other option formatting rules, this option defined as `connection-timeout` (kabab-style) for URI definitions and `connectionTimeout` (camelCase) for plain JavaScript configuration objects.

Example usage:

```javascript
const mysqlx = require('@mysql/xdevapi');
var client = mysqlx.getClient('root@localhost?connect-timeout=5000')
client.getSession()
  .catch(err => {
    console.log(err.message) // "Connection attempt to the server was aborted. Timeout of 5000 ms was exceeded."
  })

// Or

const mysqlx = require('@mysql/xdevapi');
var client = mysqlx.getClient('mysqlx://root:passwd@[localhost:33060, 127.0.0.1:33060]?connect-timeout=5000')
client.getSession()
  .catch(err => {
    // connection could not be established after 10 seconds (5 seconds for each server)
    console.log(err.message); // All server connection attempts were aborted. Timeout of 5000 ms was exceeded.
  });
```

In a multi-host scenario, the `connect-timeout` value applies to each individual host.

**Bugs Fixed**

- Improved the handling of X Protocol global notices by properly logging and then ignoring non-fatal errors, and making the connection unusable for subsequent operations in the case of a fatal error. (Bug #28653781)
MySQL Connector/Node.js Release Notes

- Calling `getCollationName()` on non-textual fields, such as INT, threw the following error: "TypeError: Cannot read property 'collation' of undefined". (Bug #28608923)

- The `fields()` method did not function with valid expressions generated by the `expr()` method. (Bug #28409639)

- The returned `Session.inspect()` object now includes the 'user' property in addition to the 'dbUser' property but containing the same value. (Bug #28362115)

Changes in MySQL Connector/Node.js 8.0.12 (2018-07-27, General Availability)

- **X DevAPI Notes**

- **Bugs Fixed**

**X DevAPI Notes**

- To increase compliance with the X DevAPI, these Connector/Node.js changes were made:
  - **Collection**: Deprecated: `count()`. Changed: `getSchema()` now returns a Schema instance instead of the schema name.
  - **CollectionModify**: Deprecated: `limit(x, y)`'s second parameter, and `arrayDelete()`.
  - **CollectionFind**: Deprecated: `limit(x, y)`'s second parameter. Added: `limit(x).offset(y)`.
  - **CollectionRemove**: Deprecated: `limit(x, y)`'s second parameter.
  - **Table**: Deprecated: `count()` and `insert(Document)` API. Updated: `getSchema()` now returns a Schema instance instead of the Schema name. Removed: `as()`.
  - **TableSelect**: Deprecated: `limit(x, y)`'s second parameter. Added: `limit(x).offset(y)`.
  - **TableDelete**: Deprecated: `limit(x, y)`'s second parameter, and `delete(x)`'s parameter in favor of using `where(x)` instead.
  - **TableUpdate**: Deprecated: `limit(x, y)`'s second parameter, and `update(x)`'s parameter in favor of using `where(x)` instead.
  - **SqlExecute**: Deprecated: `sqlExecute()` in favor of `sql()`. Added: `bind()`.
  - **Column**: Added `isNumberSigned()`, `getCollationName()`, `getCharacterSetName()`, and `isPadded()`

**Bugs Fixed**

- The Promise returned by the `session.sql().execute()` method resolved to a plain JavaScript object rather than a proper Result instance. This meant it lacked access to the API with methods such as `getAffectedItemsCount()` and `getWarnings()`. (Bug #28146988)

- Retrieving rows with NULL fields would emit an unexpected AssertionError. (Bug #27978594)

- The `session.close()` method is now asynchronous by returning a JavaScript Promise, when before it returned immediately. (Bug #27893001)

- The right-padding mechanism was improved. (Bug #27839295, Bug #28275595, Bug #91503)

- While calling `getSession()` without arguments yields an "Invalid parameter." error, passing in '{}' yielded a "Cannot read property 'length' of undefined." error. Now '{}' is allowed, and `getSession()` defaults to using " as the user name. (Bug #27730748)
- Improved performance for expression parsing and protocol message encoding.

**Changes in MySQL Connector/Node.js 8.0.11 (2018-04-19, General Availability)**

- **Functionality Added or Changed**
- **Bugs Fixed**

**Functionality Added or Changed**

- The protobuf.js library was replaced with the official google-protobuf npm package.
- Added `NOWAIT` and `SKIP_LOCKED` support to the `lockShared()` and `lockExclusive()` methods. Example usage: `lockShared(mysqlx.LockContention.SKIP_LOCKED)`.
- Added the X DevAPI SHA256_MEMORY authentication mechanism.
- Auto-generated document `_id` values generated by the MySQL server, introduced in MySQL Server 8.0.11, are now supported.

**Bugs Fixed**

- Running a select query against a table containing BIGINT values and using those values as filtering criteria could fail to function. This was because those values were converted to JavaScript numbers when encoding the protobuf message, and lost precision since the maximum safe integer in JavaScript is $2^{53} - 1$. (Bug #27570761)
- Row values from columns using the `FLOAT` type were not rounded according to the maximum number of displayable decimal digits defined by the schema. For example, a column with type `FLOAT(3,2)` containing a value of 1.23456789 would display as 1.2300000190734863 instead of the expected 1.23. (Bug #27570541)
- Row values from columns using the `BIT` data type were decoded as their sign integer counterpart instead of unsigned values. For example, `b'111'` was decoded as -4 instead of 7. (Bug #27570462)
- Row values for columns of any type of UNSIGNED integer (TINYINT, SMALLINT, MEDIUMINT, INT or BIGINT) were being interpreted by the connector as their SIGNED integer value counterpart. (Bug #27570342)
- The `sort()` method was added to the following operations: CollectionFind, CollectionRemove, and CollectionModify. (Bug #27429922)
- While adding a document, the expression parser was rejecting valid escaped literally strings that constituted properties of the document, and it threw unexpected errors. (Bug #27429852)
- Messages split into multiple fragments (either because they exceeded the MTU or the maximum size of V8 buffers) were improperly reconstructed and could not be decoded. This behavior would throw an error similar to "Uncaught SyntaxError: Unexpected token". (Bug #27429429)
- Several methods returned plain JavaScript objects that now return iterable arrays. `Schema.getCollections()` now returns an array of Collection instances, `Schema.getTables()` now returns an array of Table instances, and `Session.getSchemas()` now returns an array of Schema instances. (Bug #27294362, Bug #27221114)
- The expression parser was executed every time a document was added but now requires `mysqlx.expr()` to be explicitly called. For example, before `collection.add({ name: 'foo' })` would parse the "name" property. Now, to parse it, use `collection.add({ name: mysqlx.expr("foo") })`. (Bug #27177864)

**Changes in MySQL Connector/Node.js 8.0.10 (Skipped version number)**

There are no release notes for this skipped version number.
Changes in MySQL Connector/Node.js 8.0.9 (2018-01-30, Release Candidate)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- Improved the UUID generation algorithm to implement the design improvements suggested in RFC 4122. Before the chance of duplicated values during a small time frame was too high. (Bug #26120588)

- **X DevAPI:** In the process of refining the definition of the X DevAPI to cover the most relevant usage scenarios, the following API components have been removed from the X DevAPI implementation for Connector/Node.js:
  - API components that support session configurations, such as the `SessionConfig` and `SessionConfigManager` classes.
  - The `mysqlx.config` namespace and all methods of the namespace, `save()`, `get()`, `list()`, `delete()`, and more.
  - The `createTable()`, `foreignKey()`, `dropTable()`, `createView()`, `dropView()`, and `alterView()` methods from the `Schema` class.

- The following methods were added:
  - `Session.setSavePoint`: accepts a name or generates one of the form connector-nodejs-{uuid}, and returns a Promise.
  - `Session.releaseSavePoint`: releases a specific savepoint.
  - `Session.rollbackTo`: rollbacks to a specified savepoint.
  - The `createIndex()` method was added to the Collection API.

Bugs Fixed

- The expression parser used by the CRUD API was replaced with a new implementation written in pure JavaScript. This fixes several grammar related bugs. (Bug #26729768, Bug #26636956, Bug #25036336, Bug #23148246)

- The `CollectionFind.fields()` method was updated to support flexible parameters to follow standard X DevAPI conventions. (Bug #22084545)

Changes in MySQL Connector/Node.js 8.0.8 (2017-09-28, Development Milestone)

- Functionality Added or Changed
- Bugs Fixed

Functionality Added or Changed

- The following Collection methods were added: `replaceOne()`, `addOrReplaceOne()`, `getOne()`, `removeOne()`. For details, see Tutorial: Working with Documents.

- Added row locking support by adding the `lockExclusive()` and `lockShared()` methods to the `CollectionFind` and `TableSelect` classes. For additional information, see Tutorial: Row Locking.
MySQL Connector/Node.js Release Notes

- Extended Authentication support, including SHA-256. For additional information, see Tutorial: Secure Sessions.

- Added "contains" operator support for objects and arrays. This allows additional types of expressions such as `$x IN [x, y, z]` and `$x IN { "x": "foo", "y": "bar" }`, and also referencing field names that map to arrays and objects, such as `$someArray IN $.field` and `$someObject IN $.field`.

**Bugs Fixed**

- Added support for the parentheses-based IN syntax, such as `$IN (x, y, z, ...)`, as defined in the X DevAPI. (Bug #26666817)

**Changes in MySQL Connector/Node.js 8.0.7 (2017-07-10, Development Milestone)**

MySQL Connectors and other MySQL client tools and applications now synchronize the first digit of their version number with the (highest) MySQL server version they support. This change makes it easy and intuitive to decide which client version to use for which server version.

Connector/Node.js 8.0.7 is the first release to use the new numbering. It is the successor to Connector/Node.js 1.0.6.

- **Functionality Added or Changed**
- **Bugs Fixed**

**Functionality Added or Changed**

- It is no longer permitted to pass an empty search condition, such as the NULL value or an empty string, to the `Collection.modify()` and `Collection.remove()` methods.

- A number of changes have been implemented for the "drop" methods:
  
  - The “drop” methods are now made available at the same level as the corresponding “create” methods. For example, the `dropCollection()` and `dropTable()` methods have been removed from the `XSession` class (which has now been consolidated into the `Session` class) and moved under the `Schema` class; under the same principle, the `drop()` method has been removed from the `Collection` and `Table` classes.
  
  - The “drop” methods now succeed even if the objects to be dropped do not exist.

  - `dropView()` is now asynchronous and behaves exactly like `dropTable()` and `dropCollection()` by implicitly executing the operation and returning a promise that will hold the result of the drop operation.

- A configuration handler interface, `mysqlx.config`, has been created for managing persisted session configurations. See MySQL Connector/Node.js with X DevAPI for details.

- There are a few changes with regard to encrypted connections to MySQL servers:
  
  - Connections are now encrypted by default.

  - The connection option `ssl-enable` has been replaced by the `ssl-mode` option, which has `DISABLED`, `REQUIRED` (default), and `VERIFY_CA` as its permitted values.

  - Using the `ssl-crl` option requires the use of the `ssl-ca` option and that `ssl-mode=VERIFY_CA`; this is due to an internal requirement of the Node.js core platform.

  - Consolidated the `BaseSession`, `NodeSession`, and `XSession` into a single `Session` class. The following related changes were also made:
• The mysqlx.getNodeSession() method is renamed to getSession and returns a Session object.

• The DatabaseObject.getSession() now returns a Session object.

• A new client-side failover feature has been implemented: when creating a new connection, multiple hosts now can be specified in the connection string, and Connector/Node.js tries each host until a successful connection is established or until all hosts have been tried. See Tutorial: Getting Started for details

• Connector/Node.js now supports connecting to a local server using Unix sockets. See Tutorial: Getting Started for details.

• The format of the document ID value generated when adding a document to a collection has changed. It is still a string of 32 hexadecimal digits based on a UUID, but the order of digits has been changed to match the requirement of a stable ID prefix.

**Bugs Fixed**

• It was not possible to create a new session for a user with a SHA256 password via the PLAIN authentication mechanism. (Bug #26117627)

• The handling of large JSON arrays was problematic, and would cause an exception to be thrown. (Bug #26084604)

• Attempting to use bind when removing a document from a collection would not succeed, and an exception would be thrown. (Bug #26029551)

• The Table.update() implementation did not require a SearchConditionStr parameter, and not using this parameter could result in updating all the rows of a given table. A client-side exception is now thrown if the SearchConditionStr parameter is undefined or empty. (Bug #25993174)

• The Table.delete() implementation did not require a SearchConditionStr parameter, and not using this parameter could result in deleting all the rows of a given table. A client-side exception is now thrown if the SearchConditionStr parameter is undefined or empty. (Bug #25992969)

**Changes in MySQL Connector/Node.js 1.0**

**Changes in MySQL Connector/Node.js 1.0.6 (2017-03-07, Development Milestone)**

• Functionality Added or Changed

• Bugs Fixed

**Functionality Added or Changed**

• Added support for validating the server certificate with a given CA and/or CRL.

• Expanded support for creating TLS sessions with a URI or connection string.

• Added support for creating IPv6 sessions with a URI or connection string.

• Added support for single array or multiple argument function calls on the public API.

**Bugs Fixed**

• Fixed issues with collection.bind(). (Bug #23236379)

• Expanded support to create sessions based on unified connection strings. This also fixed parsing issues on URI and connection string corner-cases.
• Updated behavior of `collection.add([])` to avoid confusing exceptions.
  Providing an empty array as an argument should always succeed, even without an active connection to the server.

Changes in MySQL Connector/Node.js 1.0.5 (2016-11-14, Development Milestone)

Functionality Added or Changed

• Added APIs for Transaction handling, which includes the `session.startTransaction()`, `session.commit()` and `session.rollback()` functions.

• Added a Table creation API.

Changes in MySQL Connector/Node.js 1.0.4 (2016-10-10, Development Milestone)

• Functionality Added or Changed
• Bugs Fixed

Functionality Added or Changed

• Changed package name from `mysqlx` to `@mysql/xdevapi`.

• The connector can be installed into your project using Node.js's `npm` tool. Install from the download file by issuing:

  ```sh
npm install mysql-connector-nodejs-1.0.4.tar.gz
  ```

  Or install directly from the npm repository by issuing:

  ```sh
  npm install @mysql/xdevapi
  ```

  For more information on `npm` see [http://npmjs.com](http://npmjs.com).

Bugs Fixed

• The Connector/Node.JS version number can now be retrieved from the API. For example, `"ver=require('@mysql/mysqlx/package').version;`. (Bug #24571220)

• Added the `Schema.getCollectionAsTable()` method.

• Added the `Collection.count()` and `Table.count()` methods.

• Added support for the URI type string format for connections.

• Added `View DDL` support.

Changes in MySQL Connector/Node.js 1.0.3 (2016-06-21, Development Milestone)

Bugs Fixed

• Connector/Node.JS was unable to create a session when SSL was enabled. (Bug #23118665)

Changes in MySQL Connector/Node.js 1.0.2 (2016-04-11, Development Milestone)

MySQL Connector/Node.js is a new Node.js driver for use with the X DevAPI. This release, v1.0.2 M1, is the first development release of the MySQL Connector/Node.js 1.0 series.
The X DevAPI enables application developers to write code that combines the strengths of the relational and document models using a modern, NoSQL-like syntax that does not assume previous experience writing traditional SQL.

To learn more about how to write applications using the X DevAPI, see X DevAPI User Guide. For more information about how X DevAPI is implemented in MySQL Connector/Node.js, and its usage, see https://dev.mysql.com/doc/dev/connector-nodejs/.

Please note that the X DevAPI requires at least MySQL Server version 5.7.12 or higher with the X Plugin enabled. For general documentation about how to get started using MySQL as a document database, see Using MySQL as a Document Store.

Functionality Added or Changed

- Internal bug fixes, and released as the first developmental milestone preview release.

Changes in MySQL Connector/Node.js 1.0.1 (Not released, Internal)

Version 1.0.1 has no release notes, or they have not been published because the product version has not been released.

Changes in MySQL Connector/Node.js 1.0.0 (Not released, Internal)

Version 1.0.0 has no release notes, or they have not been published because the product version has not been released.