Version: v6 - stable

Raw Queries

As there are often use cases in which it is just easier to execute raw / already prepared SQL queries, you can use the `sequelize.query` method.

By default the function will return two arguments - a results array, and an object containing metadata (such as amount of affected rows, etc). Note that since this is a raw query, the metadata are dialect specific. Some dialects return the metadata "within" the results object (as properties on an array). However, two arguments will always be returned, but for MSSQL and MySQL it will be two references to the same object.

```
const [results, metadata] = await sequelize.query("UPDATE users SET y = 42 WHERE x = 12");
// Results will be an empty array and metadata will contain the number of affected rows.
```

In cases where you don't need to access the metadata you can pass in a query type to tell sequelize how to format the results. For example, for a simple select query you could do:

```
const { QueryTypes } = require('sequelize');
const users = await sequelize.query("SELECT * FROM `users`", { type: QueryTypes.SELECT });
// We didn't need to destructure the result here - the results were returned directly
```

Several other query types are available. Peek into the source for details.

A second option is the model. If you pass a model the returned data will be instances of that model.

```
// Callee is the model definition. This allows you to easily map a query to a predefined model
const projects = await sequelize.query('SELECT * FROM projects', {
  model: Projects,
  mapToModel: true // pass true here if you have any mapped fields
});
// Each element of `projects` is now an instance of Project
```

See more options in the query API reference. Some examples:

```
const { QueryTypes } = require('sequelize');
await sequelize.query('SELECT 1', {
  // A function (or false) for logging your queries
  // Will get called for every SQL query that gets sent
  // to the server.
  logging: console.log,

  // If plain is true, then sequelize will only return the first
  // record of the result set. In case of false it will return all records.
  plain: false,

  // Set this to true if you don't have a model definition for your query.
  raw: false,

  // The type of query you are executing. The query type affects how results are formatted before they are passed b
  type: QueryTypes.SELECT
});

// Note the second argument being null!
// Even if we declared a callee here, the raw: true would
// supersede and return a raw object.
console.log(await sequelize.query('SELECT * FROM projects', { raw: true }));
```

"Dotted" attributes and the `nest` option

If an attribute name of the table contains dots, the resulting objects can become nested objects by setting the `nest: true` option. This is achieved with dottie.js under the hood. See below:

- Without `nest: true`:

```
const { QueryTypes } = require('sequelize');
const records = await sequelize.query('select 1 as `foo.bar.baz`', {
  type: QueryTypes.SELECT
});
console.log(JSON.stringify(records[0], null, 2));
```

```
{
  "foo.bar.baz": 1
}
```

- With `nest: true`:

```
const { QueryTypes } = require('sequelize');
const records = await sequelize.query('select 1 as `foo.bar.baz`', {
  nest: true,
  type: QueryTypes.SELECT
});
console.log(JSON.stringify(records[0], null, 2));
```

```
{
  "foo": {
    "bar": {
      "baz": 1
    }
  }
}
```

Replacements

Replacements in a query can be done in two different ways, either using named parameters (starting with `:`), or unnamed, represented by a `?`. Replacements are passed in the options object.

- If an array is passed, `?` will be replaced in the order that they appear in the array
- If an object is passed, `:key` will be replaced with the keys from that object. If the object contains keys not found in the query or vice versa, an exception will be thrown.

```
const { QueryTypes } = require('sequelize');

await sequelize.query(
  'SELECT * FROM projects WHERE status = ?',
  {
    replacements: ['active'],
    type: QueryTypes.SELECT
  }
);

await sequelize.query(
  'SELECT * FROM projects WHERE status = :status',
  {
    replacements: { status: 'active' },
    type: QueryTypes.SELECT
  }
);
```

Array replacements will automatically be handled, the following query searches for projects where the status matches an array of values.

```
const { QueryTypes } = require('sequelize');

await sequelize.query(
  'SELECT * FROM projects WHERE status IN(:status)',
  {
    replacements: { status: ['active', 'inactive'] },
    type: QueryTypes.SELECT
  }
);
```

To use the wildcard operator `%`, append it to your replacement. The following query matches users with names that start with 'ben'.

```
const { QueryTypes } = require('sequelize');

await sequelize.query(
  'SELECT * FROM users WHERE name LIKE :search_name',
  {
    replacements: { search_name: 'ben%' },
    type: QueryTypes.SELECT
  }
);
```

Bind Parameter

Bind parameters are like replacements. Except replacements are escaped and inserted into the query by sequelize before the query is sent to the database, while bind parameters are sent to the database outside the SQL query text. A query can have either bind parameters or replacements. Bind parameters are referred to by either $1, $2, ... (numeric) or $key (alpha-numeric). This is independent of the dialect.

- If an array is passed, `$1` is bound to the 1st element in the array (`bind[0]`)
- If an object is passed, `$key` is bound to `object['key']`. Each key must begin with a non-numeric char. `$1` is not a valid key, even if `object['1']` exists.
- In either case `$$` can be used to escape a literal `$` sign.

The array or object must contain all bound values or Sequelize will throw an exception. This applies even to cases in which the database may ignore the bound parameter.

The database may add further restrictions to this. Bind parameters cannot be SQL keywords, nor table or column names. They are also ignored in quoted text or data. In PostgreSQL it may also be needed to typecast them, if the type cannot be inferred from the context `$1::varchar`.

```
const { QueryTypes } = require('sequelize');

await sequelize.query(
  'SELECT *, "text with literal $$1 and literal $$status" as t FROM projects WHERE status = $1',
  {
    bind: ['active'],
    type: QueryTypes.SELECT
  }
);

await sequelize.query(
  'SELECT *, "text with literal $$1 and literal $$status" as t FROM projects WHERE status = $status',
  {
    bind: { status: 'active' },
    type: QueryTypes.SELECT
  }
);
```