Version: v6 - stable

Other Data Types

Apart from the most common data types mentioned in the Model Basics guide, Sequelize provides several other data types.

Ranges (PostgreSQL only)

```
DataTypes.RANGE(DataTypes.INTEGER)    // int4range
DataTypes.RANGE(DataTypes.BIGINT)     // int8range
DataTypes.RANGE(DataTypes.DATE)       // tstzrange
DataTypes.RANGE(DataTypes.DATEONLY)   // daterange
DataTypes.RANGE(DataTypes.DECIMAL)    // numrange
```

Since range types have extra information for their bound inclusion/exclusion it's not very straightforward to just use a tuple to represent them in javascript.

When supplying ranges as values you can choose from the following APIs:

```
// defaults to inclusive lower bound, exclusive upper bound
const range = [
  new Date(Date.UTC(2016, 0, 1)),
  new Date(Date.UTC(2016, 1, 1))
];
// '["2016-01-01 00:00:00+00:00", "2016-02-01 00:00:00+00:00")'

// control inclusion
const range = [
  { value: new Date(Date.UTC(2016, 0, 1)), inclusive: false },
  { value: new Date(Date.UTC(2016, 1, 1)), inclusive: true },
];
// '("2016-01-01 00:00:00+00:00", "2016-02-01 00:00:00+00:00"]'

// composite form
const range = [
  { value: new Date(Date.UTC(2016, 0, 1)), inclusive: false },
  new Date(Date.UTC(2016, 1, 1)),
];
// '("2016-01-01 00:00:00+00:00", "2016-02-01 00:00:00+00:00")'

const Timeline = sequelize.define('Timeline', {
  range: DataTypes.RANGE(DataTypes.DATE)
});

await Timeline.create({ range });
```

However, retrieved range values always come in the form of an array of objects. For example, if the stored value is `("2016-01-01 00:00:00+00:00", "2016-02-01 00:00:00+00:00"]`, after a finder query you will get:

```
[
  { value: Date, inclusive: false },
  { value: Date, inclusive: true }
]
```

You will need to call `reload()` after updating an instance with a range type or use the `returning: true` option.

Special Cases

```
// empty range:
Timeline.create({ range: [] }); // range = 'empty'

// Unbounded range:
Timeline.create({ range: [null, null] }); // range = '[,)'
// range = '[,"2016-01-01 00:00:00+00:00")'
Timeline.create({ range: [null, new Date(Date.UTC(2016, 0, 1))] });

// Infinite range:
// range = '[-infinity,"2016-01-01 00:00:00+00:00")'
Timeline.create({ range: [-Infinity, new Date(Date.UTC(2016, 0, 1))] });
```

Network Addresses

All

Sequelize DataType	PostgreSQL	MariaDB	MySQL	MSSQL	SQLite	Snowflake	db2	ibmi
`CIDR`	`CIDR`	✗	✗	✗	✗	✗	✗	✗
`INET`	`INET`	✗	✗	✗	✗	✗	✗	✗
`MACADDR`	`MACADDR`	✗	✗	✗	✗	✗	✗	✗

Arrays (PostgreSQL only)

```
// Defines an array of DataTypes.SOMETHING.
DataTypes.ARRAY(/* DataTypes.SOMETHING */)

// For example
// VARCHAR(255)[]
DataTypes.ARRAY(DataTypes.STRING)
// VARCHAR(255)[][]
DataTypes.ARRAY(DataTypes.ARRAY(DataTypes.STRING))
```

BLOBs

```
DataTypes.BLOB              // BLOB (bytea for PostgreSQL)
DataTypes.BLOB('tiny')      // TINYBLOB (bytea for PostgreSQL)
DataTypes.BLOB('medium')    // MEDIUMBLOB (bytea for PostgreSQL)
DataTypes.BLOB('long')      // LONGBLOB (bytea for PostgreSQL)
```

The blob datatype allows you to insert data both as strings and as buffers. However, when a blob is retrieved from database with Sequelize, it will always be retrieved as a buffer.

ENUMs

The ENUM is a data type that accepts only a few values, specified as a list.

```
DataTypes.ENUM('foo', 'bar') // An ENUM with allowed values 'foo' and 'bar'
```

ENUMs can also be specified with the `values` field of the column definition, as follows:

```
sequelize.define('foo', {
  states: {
    type: DataTypes.ENUM,
    values: ['active', 'pending', 'deleted']
  }
});
```

JSON (SQLite, MySQL, MariaDB, Oracle and PostgreSQL only)

The `DataTypes.JSON` data type is only supported for SQLite, MySQL, MariaDB, Oracle and PostgreSQL. However, there is a minimum support for MSSQL (see below).

Note for PostgreSQL

The JSON data type in PostgreSQL stores the value as plain text, as opposed to binary representation. If you simply want to store and retrieve a JSON representation, using JSON will take less disk space and less time to build from its input representation. However, if you want to do any operations on the JSON value, you should prefer the JSONB data type described below.

JSONB (PostgreSQL only)

PostgreSQL also supports a JSONB data type: `DataTypes.JSONB`. It can be queried in three different ways:

```
// Nested object
await Foo.findOne({
  where: {
    meta: {
      video: {
        url: {
          [Op.ne]: null
        }
      }
    }
  }
});

// Nested key
await Foo.findOne({
  where: {
    "meta.audio.length": {
      [Op.gt]: 20
    }
  }
});

// Containment
await Foo.findOne({
  where: {
    meta: {
      [Op.contains]: {
        site: {
          url: 'https://google.com'
        }
      }
    }
  }
});
```

MSSQL

MSSQL does not have a JSON data type, however it does provide some support for JSON stored as strings through certain functions since SQL Server 2016. Using these functions, you will be able to query the JSON stored in the string, but any returned values will need to be parsed separately.

```
// ISJSON - to test if a string contains valid JSON
await User.findAll({
  where: sequelize.where(sequelize.fn('ISJSON', sequelize.col('userDetails')), 1)
})

// JSON_VALUE - extract a scalar value from a JSON string
await User.findAll({
  attributes: [[ sequelize.fn('JSON_VALUE', sequelize.col('userDetails'), '$.address.Line1'), 'address line 1']]
})

// JSON_VALUE - query a scalar value from a JSON string
await User.findAll({
  where: sequelize.where(sequelize.fn('JSON_VALUE', sequelize.col('userDetails'), '$.address.Line1'), '14, Foo Str
})

// JSON_QUERY - extract an object or array
await User.findAll({
```

```
  attributes: [[ sequelize.fn('JSON_QUERY', sequelize.col('userDetails'), '$.address'), 'full address']]
})
```

Miscellaneous DataTypes

All

Sequelize DataType	PostgreSQL	MariaDB	MySQL	MSSQL	SQLite	Snowflake	db2
`GEOMETRY`	`GEOMETRY`	`GEOMETRY`	`GEOMETRY`	✗	✗	✗	✗
`GEOMETRY('POINT')`	`GEOMETRY(POINT)`	`POINT`	`POINT`	✗	✗	✗	✗
`GEOMETRY('POINT', 4326)`	`GEOMETRY(POINT,4326)`	✗	✗	✗	✗	✗	✗
`GEOMETRY('POLYGON')`	`GEOMETRY(POLYGON)`	`POLYGON`	`POLYGON`	✗	✗	✗	✗
`GEOMETRY('LINESTRING')`	`GEOMETRY(LINESTRING)`	`LINESTRING`	`LINESTRING`	✗	✗	✗	✗
`GEOGRAPHY`	`GEOGRAPHY`	✗	✗	✗	✗	✗	✗
`HSTORE`	`HSTORE`	✗	✗	✗	✗	✗	✗

NOTE

In Postgres, the GEOMETRY and GEOGRAPHY types are implemented by the PostGIS extension.

In Postgres, You must install the pg-hstore package if you use `DataTypes.HSTORE`

Edit this page

*Last updated on **Oct 13, 2022** by **Rik Smale***

Previous
« Other topics

Next
Using sequelize in AWS Lambda »