Sequelize

```js
const Sequelize = require('sequelize');

// Create a connection object
const sequelize = new Sequelize(
  // Database name
  'library_db',
  // User
  'root',
  // Password
  'myPassword',
  {
    // Database location
    host: 'localhost',
    dialect: 'mysql',
    port: 3306
  }
);

module.exports = sequelize;
```

```js
const Sequelize = require('sequelize');

// Enable access to .env variables
require('dotenv').config();

// Use environment variables to connect to database
const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    host: 'localhost',
    dialect: 'mysql',
    port: 3306
  }
);

module.exports = sequelize;
```

```json
{
  "name": "library-db",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "mysql2": "^2.2.5",
    "sequelize": "^6.3.5"
  }
}
```

```js
const express = require('express');

// Import the connection object
const sequelize = require('./config/connection');

const app = express();
const PORT = process.env.PORT || 3001;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Connect to the database before starting the Express.js server
sequelize.sync().then(() => {
  app.listen(PORT, () => console.log('Now listening'));
});
```

```sql
DROP DATABASE IF EXISTS library_db;
CREATE DATABASE library_db;
```

```
DB_NAME=library_db
DB_PASSWORD=
DB_USER=
```

```
node_modules
.DS_Store
.env
```

Models

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

// Create a new Sequelize model for books
class Book extends Model {}

Book.init(
  // Define fields/columns on model
  // An `id` can be automatically created by Sequelize,
            though best practice would be to define the primary key ourselves
  {
    // Manually define the primary key
    book_id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    title: {
      type: DataTypes.STRING
    },
    author: {
      type: DataTypes.STRING
    },
    isbn: {
      type: DataTypes.STRING
    },
    pages: {
      type: DataTypes.INTEGER
    },
    edition: {
      type: DataTypes.INTEGER
    },
    // Will become `is_paperback` in table due to `underscored` flag
    isPaperback: {
      type: DataTypes.BOOLEAN
    }
  },
  {
    // Link to database connection
    sequelize,
    // Set to false to remove `created_at` and `updated_at` fields
    timestamps: false,
    // Prevent sequelize from renaming the table
    freezeTableName: true,
    underscored: true,
    modelName: 'book'
  }
);
module.exports = Book;
```

```
const express = require('express');
const sequelize = require('./config/connection');

// Import model to sync table with database
const Book = require('./models/Book');

const app = express();
const PORT = process.env.PORT || 3001;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Force true to drop/recreate table(s) on every sync
sequelize.sync({ force: true }).then(() => {
  app.listen(PORT, () => console.log('Now listening'));
});
```

```
const Sequelize = require('sequelize');

require('dotenv').config();

const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    host: 'localhost',
    dialect: 'mysql',
    port: 3306
  }
);

module.exports = sequelize;
```

```
DROP DATABASE IF EXISTS library_db;
CREATE DATABASE library_db;
```

```
DB_NAME=library_db
DB_PASSWORD=
DB_USER=
```

```
node_modules
.DS_Store
.env
```

Create & Read

```
const router = require('express').Router();
const apiRoutes = require('./api');

// Prefix all routes defined in the api directory with `/api`
router.use('/api', apiRoutes);

module.exports = router;
```

```
const router = require('express').Router();
const bookRoutes = require('./bookRoutes');

// Prefix all routes defined in `bookRoutes.js` with `/books
router.use('/books', bookRoutes);

module.exports = router;
```

```
const router = require('express').Router();

// Import the model
const Book = require('../../models/Book');

// GET all books
router.get('/', (req, res) => {
  // Get all books from the book table
  Book.findAll().then((bookData) => {
    res.json(bookData);
  });
});

// GET all paperback books
router.get('/paperbacks', (req, res) => {
  Book.findAll({
    // Order by title in ascending order
    order: ['title'],
    where: {
      // Only get books that have this boolean set to TRUE
      is_paperback: true
    },
    attributes: {
      // Don't include these fields in the returned data
      exclude: ['is_paperback', 'edition']
    }
  }).then((bookData) => {
    res.json(bookData);
  });
});

// GET a single book
router.get('/:id', (req, res) => {
  // Find a single book by its primary key (book_id)
  Book.findByPk(req.params.id).then((bookData) => {
    res.json(bookData);
  });
});
```

```
// CREATE a book
router.post('/', (req, res) => {
  // Use Sequelize's `create()` method to add a row to the table
  // Similar to `INSERT INTO` in plain SQL
  Book.create({
    title: req.body.title,
    author: req.body.author,
    is_paperback: true
  })
    .then((newBook) => {
      // Send the newly created row as a JSON object
      res.json(newBook);
    })
    .catch((err) => {
      res.json(err);
    });
});

// CREATE multiple books
router.post('/seed', (req, res) => {
  // Multiple rows can be created with `bulkCreate()` and an array
  // This could also be moved to a separate Node.js script
           to ensure it only happens once
  Book.bulkCreate([    {
      title: 'Make It Stick: The Science of Successful Learning',
      author: 'Peter Brown',
      isbn: '978-0674729018',
      pages: 336,
      edition: 1,
      is_paperback: false
    },
    {
      title: 'Essential Scrum: A Practical Guide to
           the Most Popular Agile Process',
      author: 'Kenneth Rubin',
      isbn: '978-0137043293',
      pages: 500,
      edition: 1,
      is_paperback: true
    },
    {
      title: "White Fragility: Why It's So Hard for
           White People to Talk About Racism",
      author: 'Robin DiAngelo',
      isbn: '978-0807047415',
      pages: 192,
      edition: 2,
      is_paperback: true
    },
    {
      title: 'The Pragmatic Programmer: Your Journey To Mastery',
      author: 'David Thomas',
      isbn: '978-0135957059',
      pages: 352,
      edition: 2,
      is_paperback: false
    },
    {
      title: 'The Art of Computer Programming, Vol. 1:
           Fundamental Algorithms',
      author: 'Donald Knuth',
      isbn: '978-0201896831',
      pages: 672,
      edition: 3,
      is_paperback: false
    },
    {
      title: 'Algorithms of Oppression:
           How Search Engines Reinforce Racism',
      author: 'Safiya Umoja Noble',
      isbn: '978-1479837243',
      pages: 256,
      edition: 1,
      is_paperback: true
    }
  ])
    .then(() => {
      res.send('Database seeded!');
    })
    .catch((err) => {
      res.json(err);
    });
});

module.exports = router;
```

Update & Delete

```
const router = require('express').Router();
const apiRoutes = require('./api');

router.use('/api', apiRoutes);

module.exports = router;
```

```
const router = require('express').Router();
const books = require('./bookRoutes');

router.use('/books', books);

module.exports = router;
```

```
const router = require('express').Router();
const Book = require('../../models/Book');

// GET all books
router.get('/', (req, res) => {
  // Get all books from the book table
  Book.findAll().then((bookData) => {
    res.json(bookData);
  });
});

// GET a book
router.get('/:isbn', (req, res) => {
  // Get one book from the book table
  Book.findOne({ isbn: req.body.isbn }).then((bookData) => {
    res.json(bookData);
  });
});

// Updates book based on its isbn
router.put('/:isbn', (req, res) => {
  // Calls the update method on the Book model
  Book.update(
    {
      // All the fields you can update and the data attached
              to the request body.
      title: req.body.title,
      author: req.body.author,
      isbn: req.body.isbn,
      pages: req.body.pages,
      edition: req.body.edition,
      is_paperback: req.body.is_paperback,
    },
    {
      // Gets the books based on the isbn given
              in the request parameters
      where: {
        isbn: req.params.isbn,
      },
    }
  )
    .then((updatedBook) => {
      // Sends the updated book as a json response
      res.json(updatedBook);
    })
    .catch((err) => res.json(err));
});

// Delete route for a book with a matching isbn
router.delete('/:isbn', (req, res) => {
  // Looks for the books based on isbn given in the request parameters
                and deletes the instance from the database
  Book.destroy({
    where: {
      isbn: req.params.isbn,
    },
  })
    .then((deletedBook) => {
      res.json(deletedBook);
    })
    .catch((err) => res.json(err));
});
router.post('/seed', (req, res) => {
  Book.bulkCreate([
    {
      title: 'Make It Stick: The Science of Successful Learning',
      author: 'Peter Brown',
      isbn: '9780674729018',
      pages: 336,
      edition: 1,
      is_paperback: false,
    },
    {
      title:
        'Essential Scrum: A Practical Guide
              to the Most Popular Agile Process',
      author: 'Kenneth Rubin',
      isbn: '9780137043293',
      pages: 500,
      edition: 1,
      is_paperback: true,
    },
    {
      title:
        "White Fragility: Why It's So Hard for White People
              to Talk About Racism",
      author: 'Robin DiAngelo',
      isbn: '9780807047415',
      pages: 192,
      edition: 2,
      is_paperback: true,
    },
    {
      title: 'The Pragmatic Programmer: Your Journey To Mastery',
      author: 'David Thomas',
      isbn: '9780135957059',
      pages: 352,
      edition: 2,
      is_paperback: false,
    },
    {
      title: 'The Art of Computer Programming, Vol. 1:
              Fundamental Algorithms',
      author: 'Donald Knuth',
      isbn: '9780201896831',
      pages: 672,
      edition: 3,
      is_paperback: false,
    },
    {
      title: 'Algorithms of Oppression:
              How Search Engines Reinforce Racism',
      author: 'Safiya Umoja Noble',
      isbn: '9781479837243',
      pages: 256,
      edition: 1,
      is_paperback: true,
    },
  ]).then(() => {
    res.send('Seeding Success!');
  });
});


module.exports = router;
```

Async—Await

```
const sequelize = require('../config/connection');

const Book = require('../models/Book');
const Library = require('../models/Library');

const bookSeedData = require('./bookSeedData.json');
const librarySeedData = require('./librarySeedData.json');

// Add the `async` keyword to the function `seedDatabase` to make Asynchronous.
const seedDatabase = async () => {
  // Add the `await` keyword infront of the expressions inside the `async` function.
  await sequelize.sync({ force: true });
  // Once JavaScript recognizes the `await` keyword it waits
             for the promise to be fufilled before moving on.
  await Book.bulkCreate(bookSeedData);
  await Library.bulkCreate(librarySeedData);
  process.exit(0);
};

seedDatabase();
```

```
[
  {
   "title": "Make It Stick: The Science of Successful Learning",
   "author": "Peter Brown",
   "isbn": "9780674729018",
   "pages": 336,
   "edition": 1,
   "is_paperback": false
  },
  {
   "title":
    "Essential Scrum: A Practical Guide to the Most Popular Agile Process",
   "author": "Kenneth Rubin",
   "isbn": "9780137043293",
   "pages": 500,
   "edition": 1,
   "is_paperback": true
  },
  {
   "title":
    "White Fragility: Why It's So Hard for White People to Talk About Racism",
   "author": "Robin DiAngelo",
   "isbn": "9780807047415",
   "pages": 192,
   "edition": 2,
   "is_paperback": true
  },
  {
   "title": "The Pragmatic Programmer: Your Journey To Mastery",
   "author": "David Thomas",
   "isbn": "9780135957059",
   "pages": 352,
   "edition": 2,
   "is_paperback": false
  },
  {
   "title": "The Art of Computer Programming, Vol. 1: Fundamental Algorithms",
   "author": "Donald Knuth",
   "isbn": "9780201896831",
   "pages": 672,
   "edition": 3,
   "is_paperback": false
  },
  {
   "title": "Algorithms of Oppression: How Search Engines Reinforce Racism",
   "author": "Safiya Umoja Noble",
   "isbn": "9781479837243",
   "pages": 256,
   "edition": 1,
   "is_paperback": true
  }
 ]
```

```
const router = require('express').Router();
const Book = require('../../models/Book');

// Updates book based on its book_id
router.put('/:book_id', async (req, res) => {
  //Calls the update method on the Book model
  const updatedBook = await Book.update(
    {
      // All the fields you can update and the data attached to the request body.
      title: req.body.title,
      author: req.body.author,
      isbn: req.body.isbn,
      pages: req.body.pages,
      edition: req.body.edition,
      is_paperback: req.body.is_paperback,
    },
    {
      // Gets a book based on the book_id given in the request parameters
      where: {
        book_id: req.params.book_id,
      },
    }
  );

  res.json(updatedBook);
});

// Delete route for a book with a matching book_id
router.delete('/:book_id', async (req, res) => {
  // Looks for the book based on the book_id given in the request parameters
  const deletedBook = await Book.destroy({
    where: {
      book_id: req.params.book_id,
    },
  });
  res.json(deletedBook);
});

module.exports = router;
```

```
const express = require('express');
const routes = require('./routes');
const sequelize = require('./config/connection');

const app = express();
const PORT = process.env.PORT || 3001;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// turn on routes
app.use(routes);

// turn on connection to db and server
sequelize.sync({ force: false }).then(() => {
  app.listen(PORT, () => console.log('Now listening'));
});
```

```
const router = require('express').Router();
const User = require('../../models/User');

// This route uses async/await with '.catch()' for errors
// and no HTTP status codes
router.get('/', async (req, res) => {
  const userData = await User.findAll().catch((err) => {
    res.json(err);
  });
  res.json(userData);
});

// This route uses async/await with try/catch for errors
// along with HTTP status codes
router.post('/', async (req, res) => {
  try {
    const userData = await User.create(req.body);
    // 200 status code means the request is successful
    res.status(200).json(userData);
  } catch (err) {
    // 400 status code means the server could not understand the request
    res.status(400).json(err);
  }
});

module.exports = router;
```

RESTful Routes

```
const router = require('express').Router();

const apiRoutes = require('./api');

router.use('/api', apiRoutes);

module.exports = router;
```

```
const router = require('express').Router();

const userRoutes = require('./userRoutes');

router.use('/users', userRoutes);

module.exports = router;
```

```
// GET a user
router.get('/:id', async (req, res) => {
  try {
    const userData = await User.findByPk(req.params.id);
    if (!userData) {
      res.status(404).json({ message: 'No user with this id!' });
      return;
    }
    res.status(200).json(userData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// UPDATE a user
router.put('/:id', async (req, res) => {
  try {
    const userData = await User.update(req.body, {
      where: {
        id: req.params.id,
      },
    });
    if (!userData[0]) {
      res.status(404).json({ message: 'No user with this id!' });
      return;
    }
    res.status(200).json(userData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// DELETE a user
router.delete('/:id', async (req, res) => {
  try {
    const userData = await User.destroy({
      where: {
        id: req.params.id,
      },
    });
    if (!userData) {
      res.status(404).json({ message: 'No user with this id!' });
      return;
    }
    res.status(200).json(userData);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

Validation

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

class User extends Model {}

User.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    },
    username: {
      type: DataTypes.STRING,
      // prevents null values
      allowNull: false,
      // will only allow alphanumeric characters
      validate: {
        isAlphanumeric: true,
      },
    },
    email: {
      type: DataTypes.STRING,
      // prevents duplicate email addresses in DB
      unique: true,
      // checks for email format (foo@bar.com)
      validate: {
        isEmail: true,
      },
    },    password: {
      type: DataTypes.STRING,
      allowNull: false,
      // must be longer than 8 characters
      validate: {
        len: [8],
      },
    },
  },
  {
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'user',
  }
);
module.exports = User;
```

```
const sequelize = require('../config/connection');
const User = require('../models/User.js');

test('Checks for null values', async () => {
  const user1 = {};

  const user2 = {
    username: 'test',
    email: 'test@test.com',
    password: '1111111111111',
  };

  const newUser1 = User.build(user1);
  const newUser2 = User.build(user2);

  await expect(newUser1.validate()).rejects.toThrow('notNull');
  await expect(newUser2.validate()).resolves.not.toThrow();
});

test('Checks for short passwords', async () => {
  const user1 = {
    username: 'test',
    email: 'test@test.com',
    password: '123',
  };

  const user2 = {
    username: 'test',
    email: 'test@test.com',
    password: 'password123',
  };

  const newUser1 = User.build(user1);
  const newUser2 = User.build(user2);

  await expect(newUser1.validate()).rejects.toThrow(
    'Validation len on password failed'
  );
  await expect(newUser2.validate()).resolves.not.toThrow();
});

test('Checks for alphanumeric username', async () => {
  const user1 = {
    username: 'test_123',
    email: 'test@test.com',
    password: '123',
  };

  const user2 = {
    username: 'test',
    email: 'test@test.com',
    password: 'password123',
  };

  const newUser1 = User.build(user1);
  const newUser2 = User.build(user2);

  await expect(newUser1.validate()).rejects.toThrow(
    'Validation isAlphanumeric on username failed'
  );
  await expect(newUser2.validate()).resolves.not.toThrow();
});
```

Password Hashing

```json
"dependencies": {
  "bcrypt": "^5.0.0",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mysql2": "^2.2.1",
  "sequelize": "^6.3.5"
}
```

```js
const router = require('express').Router();
const bcrypt = require('bcrypt');
const User = require('../../models/User');

// CREATE a new user
router.post('/', async (req, res) => {
  try {
    const newUser = req.body;
    // hash the password from 'req.body' and save to newUser
    newUser.password = await bcrypt.hash(req.body.password, 10);
    // create the newUser with the hashed password and save to DB
    const userData = await User.create(newUser);
    res.status(200).json(userData);
  } catch (err) {
    res.status(400).json(err);
  }
});

module.exports = router;
```

```js
const router = require('express').Router();
const bcrypt = require('bcrypt');
const User = require('../../models/User');

// Added comments describing the functionality of this `login` route
router.post('/login', async (req, res) => {
  try {
    // we search the DB for a user with the provided email
    const userData = await User.findOne({ where: { email: req.body.email } });
    if (!userData) {
      // the error message shouldn't specify if the login failed
            because of wrong email or password
      res.status(404).json({ message: 'Login failed. Please try again!' });
      return;
    }
    // use `bcrypt.compare()` to compare the provided password and the hashed password
    const validPassword = await bcrypt.compare(
      req.body.password,
      userData.password
    );
    // if they do not match, return error message
    if (!validPassword) {
      res.status(400).json({ message: 'Login failed. Please try again!' });
      return;
    }
    // if they do match, return success message
    res.status(200).json({ message: 'You are now logged in!' });
  } catch (err) {
    res.status(500).json(err);
  }
});
module.exports = router;
```

Hooks

```
hooks: {
  beforeCreate: async (newUserData) => {
    newUserData.password = await bcrypt.hash(newUserData.password, 10);
    return newUserData;
  },
  beforeUpdate: async (updatedUserData) => {
    updatedUserData.password = await bcrypt.hash(updatedUserData.password, 10);
    return updatedUserData;
  },
},
```

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

class User extends Model {}

User.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    },
    username: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true,
      },
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        len: [8],
      },
    },
  },
  {
    // When adding hooks via the init() method, they go below
    hooks: {
      // Use the beforeCreate hook to work with data before a new instance is created
      beforeCreate: async (newUserData) => {
        // In this case, we are taking the user's email address, and making all letters lower case before adding it to the database.
        newUserData.email = await newUserData.email.toLowerCase();
        return newUserData;
      },
      // Here, we use the beforeUpdate hook to make all of the characters lower case in an updated email address, before updating the database.
      beforeUpdate: async (updatedUserData) => {
        updatedUserData.email = await updatedUserData.email.toLowerCase();
        return updatedUserData;
      },
    },
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'user',
  }
);

module.exports = User;
```

Instance Method

```
const { Model, DataTypes } = require('sequelize');
const bcrypt = require('bcrypt');
const sequelize = require('../config/connection');

class User extends Model {
  // This instance method uses a conditional statement to check if a user has pets
  hasPets() {
    if (this.numberOfPets > 0) {
      return true;
    } else {
      return false;
    }
  }
}

User.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    },
    username: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true,
      },
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        len: [8],
      },
    },
    numberOfPets: {
      type: DataTypes.INTEGER,
    },
  },
  {
    hooks: {
      beforeCreate: async (newUserData) => {
        newUserData.password = await bcrypt.hash(newUserData.password, 10);
        return newUserData;
      },
      beforeUpdate: async (updatedUserData) => {
        updatedUserData.password = await bcrypt.hash(updatedUserData.password, 10);
        return updatedUserData;
      },
    },
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'user',
  }
);

module.exports = User;
```

```
class User extends Model {
  checkPassword(loginPw) {
    return bcrypt.compareSync(loginPw, this.password);
  }
}
```

One to One

```
"dependencies": {
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mysql2": "^2.2.1",
  "sequelize": "^6.3.5"
},
"devDependencies": {
  "nodemon": "^2.0.4"
}
```

```
const { UUIDV4, Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

class License extends Model {}

License.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    },
    // Use Sequelize's UUID functionality to generate a unique number
          for the license instead of making us do it ourselves
    license_number: {
      type: DataTypes.UUID,
      defaultValue: UUIDV4,
    },
    is_donor: {
      type: DataTypes.BOOLEAN,
      defaultValue: true,
    },
    // This column will store a reference of the `id` of the `Driver`
          that owns this License
    driver_id: {
      type: DataTypes.INTEGER,
      references: {
        // This references the `driver` model, which we set in `Driver.js`
          as its `modelName` property
        model: 'driver',
        key: 'id',
      },
    },
  },
  {
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'license',
  }
);

module.exports = License;
```

```
const sequelize = require('../config/connection');
const { Driver, License } = require('../models');
const driverSeedData = require('./driverSeedData.json');

const seedDatabase = async () => {
  await sequelize.sync({ force: true });
  const drivers = await Driver.bulkCreate(driverSeedData);
  for (const { id } of drivers) {
    // Need to include a valid driver_id when creating a license
    const newLicense = await License.create({
      driver_id: id,
    });
  }
  process.exit(0);
};

seedDatabase();
```

```
const Driver = require('./Driver');
const License = require('./License');

// Define a Driver as having one License to create a foreign key in the `license` table
Driver.hasOne(License, {
  foreignKey: 'driver_id',
  // When we delete a Driver, make sure to also delete the associated License.
  onDelete: 'CASCADE',
});

// We can also define the association starting with License
License.belongsTo(Driver, {
  foreignKey: 'driver_id',
});

// We package our two models and export them as an object
          so we can import them together and use their proper names
module.exports = { Driver, License };
```

One to Many

```
const Driver = require('./Driver');
const License = require('./License');
const Car = require('./Car');

Driver.hasOne(License, {
  foreignKey: 'driver_id',
  onDelete: 'CASCADE',
});

License.belongsTo(Driver, {
  foreignKey: 'driver_id',
});

// Define a Driver as having many Cars, thus creating a foreign key in the `car` table
Driver.hasMany(Car, {
  foreignKey: 'driver_id',
  onDelete: 'CASCADE',
});

// The association can also be created from the Car side
Car.belongsTo(Driver, {
  foreignKey: 'driver_id',
});

module.exports = { Driver, License, Car };
```

Applicants:

Users
Projects

https://www.bezkoder.com/sequelize-associate-many-to-many/

https://stackoverflow.com/questions/22958683/how-to-implement-many-to-many-association-in-sequelize

```
const router = require('express').Router();
const driverRoutes = require('./driverRoutes');

router.use('/drivers', driverRoutes);

module.exports = router;
```

```
const router = require('express').Router();
const { Driver, License, Car } = require('../../models');

// GET all drivers
router.get('/', async (req, res) => {
  try {
    const driverData = await Driver.findAll({
      include: [{ model: License }, { model: Car }],
    });
    res.status(200).json(driverData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// GET a single driver
router.get('/:id', async (req, res) => {
  try {
    const driverData = await Driver.findByPk(req.params.id, {
      include: [{ model: License }, { model: Car }],
    });
    if (!driverData) {
      res.status(404).json({ message: 'No driver found with that id!' });
      return;
    }
    res.status(200).json(driverData);
  } catch (err) {
    res.status(500).json(err);
  }
});

module.exports = router;
```

Literals

```javascript
const router = require('express').Router();
const sequelize = require('../../config/connection');
const { Reader, Book, LibraryCard } = require('../../models');

// GET all readers
router.get('/', async (req, res) => {
  try {
    const readerData = await Reader.findAll({
      include: [{ model: LibraryCard }, { model: Book }],
      attributes: {
        include: [
          [
            // Use plain SQL to get a count of all short books
            sequelize.literal(
              '(SELECT COUNT(*) FROM book
                         WHERE pages BETWEEN 100 AND 300 AND book.reader_id = reader.id)'
            ),
            'shortBooks',
          ],
        ],
      },
    });
    res.status(200).json(readerData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// GET a single reader
router.get('/:id', async (req, res) => {
  try {
    const readerData = await Reader.findByPk(req.params.id, {
      include: [{ model: LibraryCard }, { model: Book }],
      attributes: {
        include: [
          [
            // Use plain SQL to get a count of all short books
            sequelize.literal(
              '(SELECT COUNT(*) FROM book
                         WHERE pages BETWEEN 100 AND 300 AND book.reader_id = reader.id)'
            ),
            'shortBooks',
          ],
        ],
      },
    });
    if (!readerData) {
      res.status(404).json({ message: 'No reader found with that id!' });
      return;
    }
    res.status(200).json(readerData);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

```json
{
  "name": "library-api",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\"
                              && exit 1",
    "start": "node server.js",
    "watch": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "mysql2": "^2.2.1",
    "sequelize": "^6.3.5"
  },
  "devDependencies": {
    "nodemon": "^2.0.4"
  }
}
```

```javascript
// CREATE a reader
router.post('/', async (req, res) => {
  try {
    const readerData = await Reader.create(req.body);
    res.status(200).json(readerData);
  } catch (err) {
    res.status(400).json(err);
  }
});

// DELETE a reader
router.delete('/:id', async (req, res) => {
  try {
    const readerData = await Reader.destroy({
      where: {
        id: req.params.id,
      },
    });
    if (!readerData) {
      res.status(404).json({ message: 'No reader found with that id!' });
      return;
    }
    res.status(200).json(readerData);
  } catch (err) {
    res.status(500).json(err);
  }
});

module.exports = router;
```

Mini-Project pt. 1

```
const Traveller = require('./Traveller');
const Location = require('./Location');
const Trip = require('./Trip');

Traveller.belongsToMany(Location, {
  // Define the third table needed to store the foreign keys
  through: {
    model: Trip,
    unique: false
  },
  // Define an alias for when data is retrieved
  as: 'planned_trips'
});

Location.belongsToMany(Traveller, {
  // Define the third table needed to store the foreign keys
  through: {
    model: Trip,
    unique: false
  },
  // Define an alias for when data is retrieved
  as: 'location_travellers'
});

module.exports = { Traveller, Location, Trip };
```

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

// create our Location model
class Location extends Model {}

// create fields/columns for Location model
Location.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true
    },
    location_name: {
      type: DataTypes.STRING,
      allowNull: false
    }
  },
  {
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'location'
  }
);

module.exports = Location;
```

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

// create our Trip model
class Trip extends Model {}

// create fields/columns for Trip model
Trip.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true
    },
    trip_budget: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: true
    },
    traveller_amount: {
      type: DataTypes.INTEGER,
      allowNull: false,
      defaultValue: 1
    },
    traveller_id: {
      type: DataTypes.INTEGER,
      references: {
        model: 'traveller',
        key: 'id',
        unique: false
      }
    },
    location_id: {
      type: DataTypes.INTEGER,
      references: {
        model: 'location',
        key: 'id',
        unique: false
      }
    }
  },
  {
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'trip'
  }
);

module.exports = Trip;
```

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('../config/connection');

// create our Traveller model
class Traveller extends Model {}

// create fields/columns for Traveller model
Traveller.init(
  {
    id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true
    },
    name: {
      type: DataTypes.STRING,
      allowNull: false
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: true
      }
    }
  },
  {
    sequelize,
    timestamps: false,
    freezeTableName: true,
    underscored: true,
    modelName: 'traveller'
  }
);

module.exports = Traveller;
```

Mini-Project pt. 2

```
const router = require('express').Router();
const travellerRoutes = require('./travellerRoutes');
const locationRoutes = require('./locationRoutes');
const tripRoutes = require('./tripRoutes');

router.use('/travellers', travellerRoutes);
router.use('/locations', locationRoutes);
router.use('/trips', tripRoutes);

module.exports = router;
```

```
const router = require('express').Router();
const apiRoutes = require('./api');

router.use('/api', apiRoutes);

module.exports = router;
```

```
const router = require('express').Router();
const { Traveller, Trip, Location } = require('../../models');

// GET all travellers
router.get('/', async (req, res) => {
  try {
    const travellerData = await Traveller.findAll();
    res.status(200).json(travellerData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// GET a single traveller
router.get('/:id', async (req, res) => {
  try {
    const travellerData = await Traveller.findByPk(req.params.id, {
      // JOIN with locations, using the Trip through table
      include: [{ model: Location, through: Trip, as: 'planned_trips' }]
    });
    if (!travellerData) {
      res.status(404).json({ message: 'No traveller found with this id!' });
      return;
    }
    res.status(200).json(travellerData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// CREATE a traveller
router.post('/', async (req, res) => {
  try {
    const travellerData = await Traveller.create(req.body);
    res.status(200).json(travellerData);
  } catch (err) {
    res.status(400).json(err);
  }
});

// DELETE a traveller
router.delete('/:id', async (req, res) => {
  try {
    const travellerData = await Traveller.destroy({
      where: {
        id: req.params.id
      }
    });
    if (!travellerData) {
      res.status(404).json({ message: 'No traveller found with this id!' });
      return;
    }
    res.status(200).json(travellerData);
  } catch (err) {
    res.status(500).json(err);
  }
});

module.exports = router;
```

```
const router = require('express').Router();
const { Trip } = require('../../models');

// CREATE a trip
router.post('/', async (req, res) => {
  try {
    const tripData = await Trip.create(req.body);
    res.status(200).json(tripData);
  } catch (err) {
    res.status(400).json(err);
  }
});

// DELETE a trip
router.delete('/:id', async (req, res) => {
  try {
    const tripData = await Trip.destroy({
      where: { id: req.params.id }
    });
    if (!tripData) {
      res.status(404).json({ message: 'No trip with this id!' });
      return;
    }
    res.status(200).json(tripData);
  } catch (err) {
    res.status(500).json(err);
  }
});

module.exports = router;
```

```
const router = require('express').Router();
const { Location, Traveller, Trip } = require('../../models');

// GET all locations
router.get('/', async (req, res) => {
  try {
    const locationData = await Location.findAll();
    res.status(200).json(locationData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// GET a single location
router.get('/:id', async (req, res) => {
  try {
    const locationData = await Location.findByPk(req.params.id, {
      // JOIN with travellers, using the Trip through table
      include: [{ model: Traveller, through: Trip, as: 'location_travellers' }]
    });
    if (!locationData) {
      res.status(404).json({ message: 'No location found with this id!' });
      return;
    }
    res.status(200).json(locationData);
  } catch (err) {
    res.status(500).json(err);
  }
});

// CREATE a location
router.post('/', async (req, res) => {
  try {
    const locationData = await Location.create(req.body);
    res.status(200).json(locationData);
  } catch (err) {
    res.status(400).json(err);
  }
});
// DELETE a location
router.delete('/:id', async (req, res) => {
  try {
    const locationData = await Location.destroy({
      where: {
        id: req.params.id
      }
    });
    if (!locationData) {
      res.status(404).json({ message: 'No location found with this id!' });
      return;
    }
    res.status(200).json(locationData);
  } catch (err) {
    res.status(500).json(err);
  }
});

module.exports = router;
```

Mini-Project pt. 3

```
const sequelize = require('../config/connection');
const { Traveller, Location, Trip } = require('../models');

const travellerSeedData = require('./travellerSeedData.json');
const locationSeedData = require('./locationSeedData.json');

const seedDatabase = async () => {
  await sequelize.sync({ force: true });

  const travellers = await Traveller.bulkCreate(travellerSeedData);

  const locations = await Location.bulkCreate(locationSeedData);

  // Create trips at random
  for (let i = 0; i < 10; i++) {
    // Get a random traveller's `id`
    const { id: randomTravellerId } = travellers[
      Math.floor(Math.random() * travellers.length)
    ];

    // Get a random location's `id`
    const { id: randomLocationId } = locations[
      Math.floor(Math.random() * locations.length)
    ];

    // Create a new trip with random `trip_budget` and `traveller_amount` values,
          but with ids selected above
    await Trip.create({
      trip_budget: (Math.random() * 10000 + 1000).toFixed(2),
      traveller_amount: Math.floor(Math.random() * 10) + 1,
      traveller_id: randomTravellerId,
      location_id: randomLocationId
    }).catch((err) => {

      // If there's an error, such as the same random pairing of `traveller.id` and `location.id`
          occurring and we get a constraint error, don't quit the Node process
      console.log(err);
    });
  }
  process.exit(0);
};

seedDatabase();
```

```
const express = require('express');
const routes = require('./routes');

const sequelize = require('./config/connection');

const app = express();
const PORT = process.env.PORT || 3001;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// turn on routes
app.use(routes);

// turn on connection to db and server
sequelize.sync({ force: false }).then(() => {
  app.listen(PORT, () => console.log('Now listening'));
});
```

```
[
  {
    "location_name": "London"
  },
  {
    "location_name": "Paris"
  },
  {
    "location_name": "Pismo Beach"
  },
  {
    "location_name": "Miami"
  },
  {
    "location_name": "Austin"
  }
]
```

```
[
  {
    "name": "Sal",
    "email": "sal@hotmail.com"
  },
  {
    "name": "Lernantino",
    "email": "lernantino@gmail.com"
  },
  {
    "name": "Amiko",
    "email": "amiko2k20@aol.com"
  }
]
```