Sequelize

Databases & ORMs

Object Relational Mapper

- Acts as a "bridge" between your code and the RDBMS.
- Using ORM, data can be easily stored and retrieved from a database without writing SQL statements directly.

Sequelize

- Sequelize is an Object-Relational Mapper (ORM)
- Access SQL databases from Node.js
  - Using JS objects and methods instead of SQL statements
- Represents tables as "classes" and rows as objects (instances)

Without ORM

```
client.query(`select * from dogs`)

client.query(`select * from cats`)

client.query(`select * from hippos`)
```

With ORM

Dog.findAll()

Cat.findAll()

Hippo.findAll()

PostgreSQL

Tables

+

Rows

=

Sequelize

Models

+

Instances

Basic Workflow

How To

- Connecting to the database
- Defining models (tables)
- “Syncing” models
- Searching
- Creating
- Updating
- Deleting

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your Model(s)

```
const User = db.define('user', {
  name:     Sequelize.STRING,
  pictureUrl: Sequelize.STRING
});
```

- Add options to Model fields
  (validations, default values & more)

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your Model(s)

```
const User = db.define('user', {
    name: {
        type: Sequelize.STRING
        allowNull: false
    },
    pictureUrl: Sequelize.STRING
})
```

- Add options to Model fields (validations, default values & more)

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your Model(s)
  - Add options to Model fields (validations, default values & more)

```
const User = db.define('user', {
    name: {
        type: Sequelize.STRING
        allowNull: false
    },
    pictureUrl: Sequelize.STRING
})
```

- Connect/sync the Model to an ***actual*** table in the database

```
await User.sync()
```

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your Model(s)
  - Add options to Model fields (validations, default values & more)

```
const User = db.define('user', {
   name: {
      type: Sequelize.STRING
      allowNull: false
   },
   pictureUrl: Sequelize.STRING
})
```

- Connect/sync all the models to an ***actual*** table in the database

```
await db.sync()
```

How To

- Connecting to the database
- Defining models (tables)
- “Syncing” models
- Searching
- Creating
- Updating
- Deleting

Sequelize Basics: Workflow

- Use the Model (Table) to find instances (rows)

```
const pugs = await User.findAll();
```

Sequelize Basics: Workflow

- Use the Model (Table) to find instances (rows)
- Queries are formatted as objects

```
const allCodys = await User.findAll({
   where: {
      name: "Cody"
   }
})
```

Sequelize Basics: Workflow

- Use the Model (Table) to find a single instance (rows)

```
const pug = await User.findByPk(3);
```

Sequelize Basics: Workflow

- Use the Model (Table) to create instances (rows)

```
const pug = await User.create({
  name: “Cody",
  pictureUrl: "http://fillmurrary.com/10/10"
});
```

How To

- Connecting to the database
- Defining models (tables)
- “Syncing” models
- Searching
- Creating
- Updating
- Deleting

Sequelize Basics: Workflow

- Use the Instances (rows) to perform updates
  - Update is given as an object

```
console.log(pug.age) // 7
const updatedPug = await pug.update({
  age: 8
})
console.log(pug.age) // 8
```

- Use the Instances (rows) to delete

```
await pug.destroy()
// the pug is gone :(
```

Additional Model Options

- Sequelize models can be extended Hooks, Class & Instance Methods, Getter & Setters, Virtuals, etc.

Associations

Associations

- Establishes a relationship between two tables
  (using a foreign-key or a join-table)
- And more… (eager loading, etc)

Associations

```
const User = db.define("user", {…})
const Pet  = db.define(“pet", {…})

Pet.belongsTo(User)
User.hasMany(Pet)
```