JÖNKÖPING UNIVERSITY

School of Engineering

USING SEQUELIZE IN NODE.JS

Peter Larsson-Green

Jönköping University

Spring 2019

OBJECT-RELATIONAL MAPPING

Applications often represent data as objects, e.g.:

```
const humans = [
  {id: 0, name: "Alice"},
  {id: 1, name: "Bob"}
]
```

```
0, Alice
1, Bob
```

humans.csv

This data often needs to be stored in files, using some format.

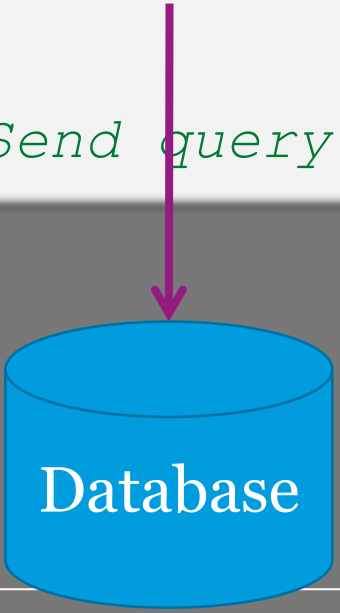
An Object-Relational Mapping framework/library do that for us.

- The programmer only needs to care about programming, not about the mapping ☺
- In practice, the objects are often mapped to SQL queries.

WITHOUT ORM

```
const human = {
  age: 10,
  name: "Alice"
}
const query = `INSERT INTO humans (age, name)
               VALUES (`+human.age+`, "`+human.name+`")`
// Send query to db...
```

Your code.



We have written code that maps the data to an INSERT query ☹

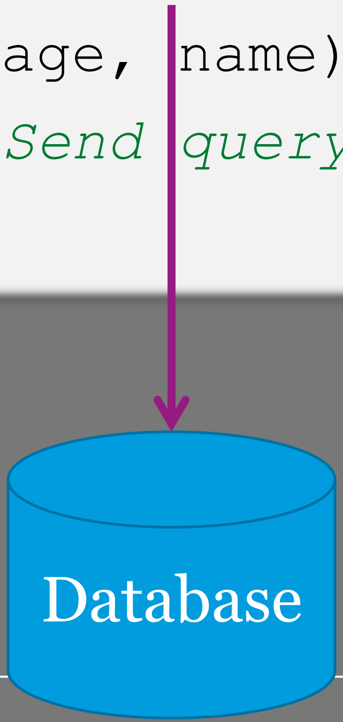
WITH ORM

Your code.

```
const human = {
  age: 10,
  name: "Alice"
}
store("humans", human)
```

Code from ORM framework/library.

```
function store(collection, entry){
  // Somehow auto-generates:
  const query = `INSERT INTO humans
    (age, name) VALUES (10, "Alice")`
  // Send query to db...
}
```



SEQUELIZE

An ORM for Node.js.

- Distributed as an npm package:
  - `npm install sequelize`
- Supports multiple different databases, including:
  - MySQL
  - MSSQL
  - PostgreSQL
  - SQLite
- Learn it: http://docs.sequelizejs.com

SEQUELIZE - BASIC USAGE

```
const Sequelize = require('sequelize')
const sequelize = new Sequelize('sqlite:my-database.db')
```

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Sequelize adds three additional fields:

- `id: {type: Sequelize.INTEGER, primaryKey: true, autoIncrement: true}`
- `createdAt: Sequelize.DATE`
- `updatedAt: Sequelize.DATE`

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

- Manually create the table `humans` in the database.
- Call `sequelize.sync()` to create the tables.

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Storing a new instance:

```
Human.create({name: "Alice", age: 12})
  .then(function(createdHuman){   })
  .catch(function(error){   })
```

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Retrieve a single instance:

```
Human.findById(1).then(function(theHuman){  })
Human.findOne({
  where: { age: 99 }
})
.then(function(theHuman){  })
```

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Retrieve multiple instances:

```
Human.findAll().then(function(allHumans){  })
Human.findAll({
  where: { age: { [Sequelize.Op.gt]: 18 }}
})
.then(function(oldHumans){  })
```

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Update instances:

```
Human.update({
  name: "The new name..."
}, {
  where: {id: 7}
}).then(function(){  })
```

SEQUELIZE - BASIC USAGE

Defining your models:

```
const Human = sequelize.define('human', {
  name: Sequelize.TEXT,
  age: Sequelize.INTEGER
})
```

Deleting instances:

```
Human.destroy({
  where: {id: 7}
}).then(function(){  })
```

SEQUELIZE - USING RELATIONS

Defining your models:

```
const User = sequelize.define('user', {
  username: Sequelize.TEXT
})
```

```
const Game = sequelize.define('game', {
  title: Sequelize.TEXT
})
```

Setup a One-To-One relationship:

```
User.belongsTo(Game) // User gets the column gameId.
User.findById(7, {include: [Game]}).then(function(user){
  console.log(user.game.title)
})
```

SEQUELIZE - USING RELATIONS

Defining your models:

```
const User = sequelize.define('user', {
  username: Sequelize.TEXT
})
```

```
const Game = sequelize.define('game', {
  title: Sequelize.TEXT
})
```

Setup a One-To-Many relationship:

```
User.hasMany(Game) // Game gets the column userId.
User.findById(7, [include: Game]).then(function(user){
  for(const game of user.games){ console.log(game.title) }
})
Game.belongsTo(User)
```

SEQUELIZE - USING RELATIONS

Defining your models:

```
const User = sequelize.define('user', {
  username: Sequelize.TEXT
})
```

```
const Game = sequelize.define('game', {
  title: Sequelize.TEXT
})
```

Setup a Many-To-Many relationship:

```
User.belongsToMany(Game, {through: "UserGame"})
Game.belongsToMany(User, {through: "UserGame"})
User.findById(7, [include: Game]).then(function(user){
  for(const game of user.games){ console.log(game.title) }
})
```