Database OO Design Patterns

Springboard

Goals

Download Demo Code

- Refactor our Express apps to separate view logic (routing) from model logic (data)
- Compare different OO designs for interfacing with our database
- Borrow useful ideas from ORMs to build our own model layers!

Current Design

routes

```
/** get all cats: [{id, name, age}, ...] */

router.get("/", async function (req, res, next) {
  let result = await db.query("SELECT * FROM cats");
  let cats = result.rows;
  return res.json(cats)
});
```

It's ok, but it's better to get SQL out of routes

Why No SQL In Routes?

- You tend to have lots of routes
  - So lots of copy-and-paste of similar SQL
- It's nice to centralize validation, schema, etc
- Separation of concerns: routes should be about web-stuff

Object Orientation

Why do we use Object Orientation?

To help organize our code!

Abstraction

OO can offer **abstraction**, the ability to hide implementation details when they aren't needed.

- Not everyone should have to understand everything
  - Only one person has to worry about SQL, validation, etc

Encapsulation

OO can offer **encapsulation**, the ability to group functionality into larger logical pieces.

- To get in a "capsule"
  - Everything related to cat data/functionality lives in ***Cat***

Polymorphism

OO can offer **polymorphism**, the ability to implement similar functionality in different classes.

- The ability to make similar things work similarly
  - We could have other kinds of animals with same API
  - For example, dogs and cats could both have a **speak** method, even though it behaves differently for different animals ("Meow" vs "Woof")

Simple OO Model

- We can make a single class for "all cat-related functions"
- It won't hold data
- You won't ever instantiate it!
- All the methods are static (called on ***Cat***)
- Benefit: help organization, gets SQL out of routes

Getting All Cats

Cat model

```
  /** get all cats: returns [{id, name, age}, ...] */

  static async getAll() {
    const result = await db.query(
        "SELECT id, name, age FROM cats");
    return result.rows;
  }
```

(that's a method inside class ***Cat***)

routes

```
/** ((fixed)) get all cats: [{id, name, age}] */

router.get("/", async function (req, res, next) {
  let cats = await Cat.getAll();
  return res.json(cats);
});
```

Getting A Cat

Cat model

```
  /** get cat by id: returns {name, age} */

  static async getById(id) {
    const result = await db.query(
        `SELECT name, age FROM cats WHERE id = $1`,
        [id]);

    if (result.rows.length === 0) {
      throw new Error(`No such cat: ${id}`);
    }

    return result.rows[0];
  }
```

routes

```
/** get cat by id: {id, name, age} */

router.get("/:id", async function (req, res, next) {
  let cat = await Cat.getById(req.params.id);
  return res.json(cat);
});
```

Creating a Cat

Cat model

```
  /** create a cat: returns {name, age} */

  static async create(name, age) {
    const result = await db.query(
        `INSERT INTO cats (name, age)
          VALUES ($1, $2) RETURNING name, age`,
        [name, age]);

    return result.rows[0];
  }
```

routes

```
/** create cat from {name, age}: return {name, age} */

router.post("/", async function (req, res, next) {
  let cat = await Cat.create(req.body.name, req.body.age);
  return res.json(cat);
});
```

Deleting a Cat

Cat model

```
  /** delete cat with given id */

  static async remove(id) {
    const result = await db.query(
        `DELETE FROM cats WHERE id=$1 RETURNING id`,
        [id]);

    if (result.rows.length === 0) {
      throw new Error(`No such cat: ${id}`);
    }
  }
```

routes

```
/** delete cat from {id}: returns "deleted" */

router.delete("/:id", async function (req, res, next) {
  await Cat.remove(req.params.id);
  return res.json("deleted");
});
```

Aging a Cat

What if we want to do something special?

Like, age a cat by one year?

Cat model

```
  /** age cat by 1 year, return new age */

  static async makeOlder(id) {
    const result = await db.query(
        `UPDATE cats SET age=age+1 WHERE id=$1 RETURNING age`,
        [id]);

    if (result.rows.length === 0) {
      throw new Error(`No such cat: ${id}`);
    }

    return result.rows[0].age;
  }
}
```

routes

```
/** age cat: returns new age */

router.post("/:id/age", async function (req, res, next) {
  let newAge = await Cat.makeOlder(req.params.id);
  return res.json(newAge);
});
```

Meh. Annoying to have to make special function.

We could make a special "update-data" function.

Smarter OO Model

- We can make a more traditional OO class
- You will instantiate it — once per dog!
- It will hold data specific to each dog
- It has static methods
  - To get all dogs, get a particular dog
- It has regular methods
- It's like a mini-ORM

Dogs

We'll make a "smarter model" for dogs.

Dog model

```
constructor(id, name, age) {
  this.id = id;
  this.name = name;
  this.age = age;
}
```

Getting All Dogs

Dog model

```
  /** get all dogs: returns [dog, ...] */

  static async getAll() {
    const result = await db.query(
        `SELECT id, name, age FROM dogs`);
    return result.rows.map(d => new Dog(d.id, d.name, d.age));
  }
```

routes

```
/** get all dogs: [{id, name, age}, ...] */

router.get("/", async function (req, res, next) {
  let dogs = await Dog.getAll();
  return res.json(dogs);
});
```

We get Dog instances, but Express can turn them into JSON

Getting A Dog

Dog model

```
  /** get dog by id: returns dog */

  static async getById(id) {
    const result = await db.query(
        `SELECT name, age FROM dogs WHERE id = $1`,
        [id]);

    if (result.rows.length === 0) {
      throw new Error(`No such dog: ${id}`);
    }

    let d = result.rows[0];
    return new Dog(id, d.name, d.age);
  }
```

routes

```
/** get dog by id: {id, name, age} */

router.get("/:id", async function (req, res, next) {
  let dog = await Dog.getById(req.params.id);
  return res.json(dog);
});
```

Creating a Dog

Dog model

```
  /** create a dog: returns dog */

  static async create(name, age) {
    const result = await db.query(
        `INSERT INTO dogs (name, age)
          VALUES ($1, $2) RETURNING id`,
        [name, age]);

    let { id } = result.rows[0];
    return new Dog(id, name, age);
  }
```

routes

```
/** create dog from {name, age}: return id */

router.post("/", async function (req, res, next) {
  let id = await Dog.create(req.body.name, req.body.age);
  return res.json(id);
});
```

Deleting a Dog

Dog model

```
  /** delete dog */

  async remove() {
    await db.query(
        `DELETE FROM dogs WHERE id = $1`,
        [this.id]);
  }
```

routes

```
/** delete dog from {id}: returns "deleted" */

router.delete("/:id", async function (req, res, next) {
  let dog = await Dog.getById(req.params.id);
  await dog.remove();
  return res.json("deleted");
});
```

Notice: it's just a method that acts on current dog!

Aging a Dog

Now, we don't need special functionality to age a dog

We can just age an instance and **.save()** it!

Dog model

```
  async save() {
    await db.query(
        `UPDATE dogs SET name=$1, age=$2 WHERE id = $3`,
        [this.name, this.age, this.id]);
  }
```

routes

```
/** age dog: returns new age */

router.post("/:id/age", async function (req, res, next) {
  let dog = await Dog.getById(req.params.id);
  dog.age += 1;
  await dog.save();
  return res.json(dog.age);
});
```

Which Is Better?

- "Simple class" *(no data, only static methods)*
  - Can be easier to write class
  - Fewer SQL queries may flex (compare delete between **Cat** and **Dog**
  - Doing more interesting things can be trickier
- "Smarter class" *(data, real methods)*
  - Real attributes can be handy!
  - Easier to do validation
  - Can do things like `cat.speak()` rather than `Cat.speak(id)`

Are There ORMs For JavaScript?

Yes!

There's a nice one called Sequelize

Not as popular as ORMs in other languages, though.