WORM

WORM is our Warmup Object Relational Mapper. The idea is to warm up by making a very simple ORM following the Active Record pattern.

The goals:

- Explore different OOP ideas
- Explore the Active Record pattern
- Explore the basics of an ORM
- Explore building an abstraction with a “subjectively nice” developer experience

I like using node-postgres:

```
npm install pg
```

Here’s a quickie to get you started:

```
import pg from "pg";

(async () => {
  const db = new pg.Client({
    connectionString: "postgres://postgres:postgres@postgres:5432/postgres",
  });
  await db.connect();

  // Now you got a database connection to work with

  await db.end();
})();
```

What to accomplish:

1. We need a way to create grade levels, schools, and students via a `create` method

```
// Should return a School object
```

```
const school = await School.create({
  school_name: "Turtle Academy",
});

// Should return a GradeLevel object
const kinderGl = await GradeLevel.create({
  grade_level_code: "K",
  grade_level_name: "Kindergarten",
});
const firstGl = await GradeLevel.create({
  grade_level_code: "1",
  grade_level_name: "1st Grade",
});

// Should return a Student object
const student = await Student.create({
    student_name: "Alice",
    school_id: school.getId(),
    grade_level_id: gradeLevel.getId(),
});
```

2. We need a way to find grade levels, schools, or students using their `id` via a `find` method

```
const school = await School.find(1);
const gradeLevel = await GradeLevel.find(1);
const student = await Student.find(1);
```

3. We need a way to find a GradeLevel using its `grade_level_code`

```
const kinder = await GradeLevel.findByCode("K");
```

4. We need a way to update grade levels, schools, and students via a `setData` and `save` method.

```
const school = await School.find(1);
school.setData({
    school_name: "Teenage Turtle Academy",
});
await school.save();
```

```
const student = await Student.find(1);
student.setData({
    student_name: "Alice A",
    grade_level_id: (await GradeLevel.findByCode("1")).getId(),
    school_id: school.getId(),
});
await student.save();

// NOTE: If the model has an updated_at field make sure it is being populated
with the latest datetime.
```

4. We need a way to get the field data on a grade level, school, or student object via a `getData` method

```
const school = await School.find(1);
// getData without a parameter returns all the data as an object
// { school_id: "", school_name: "", etc... }
console.log(school.getData());

// pass a field parameter to getData to get the data for just that field
const schoolName = school.getData("school_name");
console.log(`Welcome to ${schoolName}!! It was last updated at $
{school.getData("updated_at")}`);
```

5. We need a way to get all of the grade levels, schools, or students via a `fetchAll` method.

```
const allGrades = (await GradeLevel.fetchAll()).map((gl) =>
gl.grade_level_name);
console.log(`${allGrades.join(", ")}`);
```

6. We need to a way to update the grade levels associated with a school via the `school_grade_level_aff` table via an `updateGradeLevels` function.

```
// The function should take an array of GradeLevel model objects and
// replace the grade levels associated to the school with the ones that
// are passed in
const kinder = await GradeLevel.findByCode("K");
await school.updateGradeLevels([kinder]);
```

```
// NOTE: Make sure to update the updated_at column on the schools table once
the grade levels have changed.
```

7. We need a way to get the GradeLevel and School a student is affiliated with via the `getSchool` and `getGradeLevel` methods

```
const stu = await Student.find(1);
const schoolName = (await alice.getSchool()).school_name;
const gl = (await alice.getGradeLevel()).grade_level_name;

console.log(`${stu.student_name} goes to ${schoolName} and is in ${gl}`);
```

8. We need a way to delete grade levels, students, and schools via a `delete` function

```
const kinder = await GradeLevel.findByCode("K");
await kinder.delete();

const stu = await Student.find(1);
const school = await stu.getSchool();
await stu.delete();
await school.delete()
```

9. We need a way to soft delete and restore schools and students via the `softDelete` and `restore` methods

```
const stu = await Student.find(1);
await stu.softDelete();
console.log(`${stu.student_name} was soft deleted on ${stu.deleted_at}`);

await stu.restore();
console.log(`${stu.student_name} was restored.`);
```

Here is the database schema you can start with:

```
CREATE TABLE IF NOT EXISTS grade_levels (
```

```
  grade_level_id serial PRIMARY KEY,
  grade_level_code varchar(10) NOT NULL,
  grade_level_name varchar(255) NOT NULL,
  created_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  updated_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(grade_level_code)
);

CREATE TABLE IF NOT EXISTS schools (
  school_id serial PRIMARY KEY,
  school_name varchar(255) NOT NULL,
  created_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  updated_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  deleted_at timestamptz
);

CREATE TABLE school_grade_level_aff (
  school_id integer NOT NULL REFERENCES schools,
  grade_level_id integer NOT NULL REFERENCES grade_levels,
  UNIQUE(school_id, grade_level_id)
);

CREATE TABLE IF NOT EXISTS students (
  student_id serial PRIMARY KEY,
  student_name varchar(255) NOT NULL,
  grade_level_id integer NOT NULL REFERENCES grade_levels,
  school_id integer NOT NULL REFERENCES schools,
  created_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  updated_at timestamptz DEFAULT CURRENT_TIMESTAMP,
  deleted_at timestamptz
);

CREATE INDEX idx_students_school_grade_level ON students (school_id,
grade_level_id)
WHERE deleted_at IS NULL;
```