Intern Task

Family Tree Project

Valdi is doing a freelance project. His client has some data about people and their relation to others. The data comes as two tables:

PERSON TABLE

	A	B	C	D	E	F
1	**person_id**	**name**	**gender**	**birth_date**	**father**	**mother**
2	<id>	<string>	<male/female>	<date>	<id>	<id>

MARRIAGE TABLE

	A	B	C	D	E
1	**marriage_id**	**husband**	**wife**	**start_date**	**end_date**
2	<id>	<person_id>	<person_id>	<date>	<date/null>

Task 1: Data Structure

MODELS

Valdi starts with a data structure to describe the entry. For **person** he considered following repository model:

```
class Person {

    constructor(repo, {id, name, gender, birth_date, fatherId, motherId}) {

    }

    getId() {
        // returns id
    }

    isMale() {
```

```
        // returns boolean
    }

    isFemale() {
        // returns boolean
    }

    getName() {
        // returns name
    }

    getBirthDate() {
        // returns date
    }

    getFather() {
        // returns Person or null if unregistered
    }

    getMother() {
        // returns Person or null if unregistered
    }
}
```

And following model for **marriage**:

```
class Marriage {

    constructor(repo, {id, husbandId, wifeId, startDate, endDate}) {

    }

    getId() {
```

```
        // returns id
    }

    getHusband() {
        // returns Person
    }

    getWife() {
        // returns Person
    }

    getMarriageDate() {
        // returns date
    }

    getDivorceDate() {
        // returns date
    }

    isEnded() {
        // returns boolean
    }
}
```

REPOSITORY

For the repository he thinks of some simple methods:

```
class Repo {

    addPerson({id, name, gender, birth_date, fatherId, motherId}) {

    }
```

```
    addMarriage({id, husbandId, wifeId, startDate, endDate}) {

    }

    getPersonById(id) {
        // returns Person
    }

    getMarriageById(id) {
        // returns Marriage
    }
}
```

TASK

Write `models.js` containing implementation for both models:

```
class Person {...}
class Marriage {...}
class Repo {...}

module.exports = {Person, Marriage, Repo};"use strict";

class Person {...}
class Marriage {...}
class Repo {...}

module.exports = {Person, Marriage, Repo};
```

Task 2: Importing Data

The client gives the data as CSV. Valdi needs to be able to work with the data. The CSV format of the person database looks like this:

```
0000590198;Budi Raharjo;male;1965-12-20;0000480251;0000480253;
```

The marriage CSV format looks like this:

```
009201;0000480251;0000480253;1964-11-23;null;
```

TASK

Write an `importer.js` with following scaffold:

```
let {Repo} = require(”./models.js”);

function importPersonCSV(filePath, repo) {
   return repo;
}

function importMarriageCSV(filePath, repo) {
   return repo;
}"use strict";

let {Repo} = require("./models.js");

function importPersonCSV(filePath, repo) {
    return repo;
}

function importMarriageCSV(filePath, repo) {
    return repo;
}
```

Task 3: Simple Relations

The client asked to check simple relations:

- Is a person married
- Is a person divorced
- Is a person a biological parent of a child
- Is a person a step parent of a child

TASK

Extend `Person` model to provide following functions:

```
person.isMarried() // returns boolean
person.isDivorced() // returns boolean
person.isParent(child) // returns boolean
person.isStepParent(child) // returns boolean
```

Task 4: Finding Related Persons

Extend `Person` to be able to get related persons:

- Spouse
- Former spouses
- Parents (biological)
- Children (biological)
- Siblings (biological brothers, sisters, including half brothers & sisters)
- Steps (parents, brothers, sisters, children)
- Uncles, Aunties (only biological)
- Grandparents (only biological)
- Grandchildren (only biological)
- Nephews (only biological)
- Cousins (only biological)

TASK

Implement following methods:

```
person.getSpouse()
person.getFormerSpouses()
person.getParents()
person.getChildren()
person.getSiblings()
person.getSisters()
person.getBrothers()
person.getStepChildren()
person.getStepSisters()
person.getStepBrothers()
```

```
person.getStepMother()
person.getStepFather()
person.getUncles()
person.getAunties()
person.getGrandFathers()
person.getGrandMothers()
person.getGrandParents()
person.getGrandChildren()
person.getCousins()
person.getNephews()
```

Task 5: Two Related Person

The client also want to know:

- Relation Root of two person
- Relation distance: defined as maximum relation steps from person to relation-root

TASK

Implement following function:

```
function getRelationRoot(person1, person2) {
    // returns {root: Person, distance: number}
    // null if no relation
}
```

Task 6: Relation in Natural Language

The client want also the system to be able to check relation in natural language such as following:

> *find brother of sister of nephew of parent of person A*

All previously defined relation shall be implemented.

TASK

Implement following functions:

```
// find relation in natural language
// return all possible results in array
person.find("brother of step sister of nephew of parent");
// get relation to a person
// return relation in natural language
person.relationTo(person2);
```

hints:

- split and join string by “of”
- for the second, look for shortest relation phrase
- Beware of circular paths