Recruitment Module – JavaScript

# 1. Schema

## 1.1 Recruitment Database

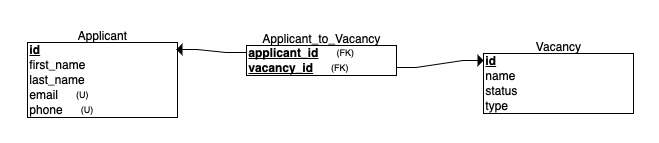
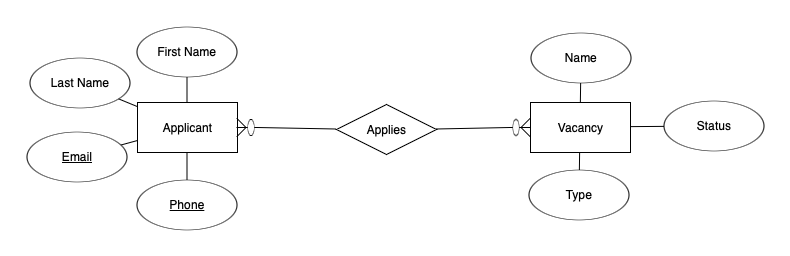
Design a database scheme of Recruitment module and attach an Entity Relationship Diagram.

The database should provide:

1. A possibility to create vacancies and applicants
2. There should be an option to create applicants related to multiple vacancies
3. There should be an ability to store a vacancy Name, Status, Type
4. There should be an ability to store applicant FirstName, LastName, Email, Phone

Solution:

[Attach completed ERD image right here:]



**CREATE TABLE Applicant (**

**id INTEGER PRIMARY KEY,**

**first\_name varchar(50) NOT NULL,**

**last\_name varchar(50) NOT NULL,**

**email varchar(50) NOT NULL,**

**phone varchar(50) NOT NULL);**

**CREATE TABLE Vacancy (**

**id INTEGER PRIMARY KEY,**

**name varchar(50) NOT NULL,**

**status varchar(10) NOT NULL,**

**type varchar(5) NOT NULL);**

**CREATE TABLE Applicant\_to\_Vacancy (**

**applicant\_id INTEGER,**

**vacancy\_id INTEGER,**

**PRIMARY KEY (applicant\_id, vacancy\_id));**

## 1.2 SQL

Based on the previously built database schema create SQL queries to retrieve records:

1. Select vacancies with "Active" Status and "HR" Type
2. Select the number of vacancies Types
3. Select applicants that applied for more than 5 vacancies

Solution:

1.

/\* Select vacancies with "Active" Status and "HR" Type \*/

**SELECT \***

**FROM Vacancy**

**WHERE status = 'Active'**

**AND type = 'HR'**

2.

/\* Select the number of vacancies Types \*/

**SELECT Count(DISTINCT type)**

**FROM Vacancy**

3.

/\* Select applicants that applied for more than 5 vacancies \*/

/\*Solution 1\*/

**WITH applicants\_five\_vacancies AS (**

**SELECT applicant\_id**

**FROM Applicant\_to\_Vacancy**

**GROUP BY applicant\_id**

**HAVING Count(vacancy\_id) > 5;**

**SELECT id, first\_name, last\_name**

**FROM Applicant**

**WHERE id IN (SELECT \* FROM applicants\_five\_vacancies)**

/\*Soluton 2\*/

**CREATE VIEW applicants\_five\_vacancies AS**

**SELECT applicant\_id**

**FROM Applicant\_to\_Vacancy**

**GROUP BY applicant \_id**

**HAVING Count(vacancy\_id) > 5;**

**SELECT id, first\_name, last\_name**

**FROM Applicant app Join applicants\_five\_vacancies afv**

**ON app.id = afv.applicant\_id;**

# 2. Object Oriented Design

## 2.1 Theory

1. What is polymorphism?
2. What Is the Prototype in JavaScript? Describe an interesting use-case from your experience when you were modifying the Prototype of an object; what was the reason? If you weren’t, where would you do it and why?
3. Have you ever used "Static Factory" OO pattern? If yes, what was the reason? If no, where would you use it and why?

Solution:

1. Polymorphism means “many forms”. So, it describes an ability of providing single interface for entities of different types. In Spite of being object-oriented principle, it came from structured programming. There are three types of polymorphism: ad-hoc, parametric and subtyping.

**An ad-hoc polymorphism** has been implemented as “overloading” of methods in many object-oriented languages.

**A parametric polymorphism** allows methods and functions to be implemented generously, so it is able to receive data of any type. It was implemented as “Generics” in many object-oriented languages.

**A subtyping polymorphism** is based on inheritance.

2. We can use prototype objects to extend behavior of standard objects like: Object, Number, String, etc. Using a prototype object and a concept of “prototype chain” we can use some kind of inheritance of objects in JavaScript. For example:

const Animal = function(options) {

this.name = options.name;

this.color = options.color;

this.voice = function() {

console.log("Gav", this.name);

}

}

Animal.prototype.run = function(){

console.log(this.name, "running!");

}

const Cat = function(options) {

Animal.apply(this, arguments);

this.hasTail = options.hasTail;

this.type = 'cat';

}

Cat.prototype = Object.create(Animal.prototype);

Cat.prototype.constructor = Cat;

Here is an interesting line: Animal.prototype.run = function(){. Because the ‘run’ method in ‘prototype’ we are able to overwrite this method for the class Cat:

Cat.prototype.voice = function() {

console.log(this.name + ' meow');

}

But since ES6+ we have got a keyword “class”. The bunch of code below is absolutely equal to what we have already seen above:

class Animal {

construcor(options){

this.name = options.name;

this.color = options.color;

}

voice() {

console.log('Gav', this.name);

}

}

class Cat extends Animal {

constructor(options) {

super(options);

this.hasTail = options.hasTail;

}

voice() {

console.log(this.name + ' meow');

}

}

3. Yes, I did have experience of implementing custom “Static factory” pattern. Mostly there were methods with long names, like “getStudentWithNameAndAge”, because I needed test data with predefined states, but constructors have lack of descriptions for predefined values. Another example is coding Singleton pattern for my pet-projects.

There are many classes with static factory methods in the JDK like: java.land.String, java.util.Collections java.util.Arrays. They contain a ton of static methods with basic operations over strings, arrays and collections.

## 2.2 Practical Tasks

**Vacancy Approval Process**

**Business Overview**

The application recruitment module requires an additional service to execute actions that the head office users perform during a vacancy approval process. There is a defined list of action types that a product owner requested to implement. **Potentially, there will be more requested in future.**

**Action Types**

* Submit for Approval
* Approve
* Reject

**Functional requirements**

* Implement a service (JavaScript class) that is able to execute actions (Submit for Approval, Approve, Reject) applied to a record
* **Keep in mind that, potentially, in future the service should be able to execute other type of actions that are not related to approval process or vacancy module. Moreover, it can happen that based on a Record type a behaviour of an action may differ.**
* An action should log a message to the console (replace record\_id placeholder with an actual record Id)
* Service should have a method which can execute either one action or multiple. Order also makes sense and should be taken into consideration. Possible solution can be found here <https://www.freecodecamp.org/news/pipe-and-compose-in-javascript-5b04004ac937/>   
  If you came up with your own solution-it ls also acceptable.

Assumtions:

* Action can be either function or object. Up to implementation.

*Additional:*

*Let’s assume that action is executed asynchronously, provide an example of this kind of implementation a well*

Submit for Approval: Submitted for approval Vacancy: {record\_id}

Approve: Approved Record: {record\_id}

Reject: Reject Record: {record\_id}

**Testing solution**

The solution should provide RecruitmentSolution.js file with the logic to test a service. Do not create new files, defined your classes inside the received template. As a result of the RecruitmentSolution.js file execution the console should contain next lines:

Expected console output:

Submitted for approval Vacancy: SoftwareEngineer0001  
Approved Record: SoftwareEngineer0001  
Submitted for approval Vacancy: FrontEndEngineer0020  
Rejected Record: FrontEndEngineer0020

[[Download solution template right here]](https://ctdevcorp.sharepoint.com/:u:/s/L&D/ESA8xZ71_ydEuKcg_q-YhUcBlxjpVV4o9YGa9VnbUbrZtA?e=BYL0f1)

Solution:

[Send back the completed solution file]

# 3. Algorithms

## 3.1 Difference between list of Integers

Find difference between two lists of Integers [1, 3, 3, 4, 6, 5, 4] & [6, 3, 5, 2, 2]

**Functional Requirements**

Implement a logic that finds difference between "first" and "second" lists and prints the result to the console. If you found multiple solutions implement All.

Expected console output:

[1, 2, 4]

Solution class template:

[[Download solution template right here](https://ctdevcorp.sharepoint.com/:u:/s/L&D/Eb09J_ovt3dOsQOrzxyYgDEBuuPDFwQeMD0ik_Vs3_rlKw?e=UVPMVT)]

Solution:

[Send back the completed solution file]

# 4. Front-End

## 4.1 Todo App

Given the TodoSolution.html page with a basic design of the Todo application.

**Functional Requirements**

Implement a logic inside the HTML file

*Required:*

* The user should be able to add todo items with an entered text title**- DONE**
* The user should be able to remove a particular todo item**- DONE**
* The user should see an alert message if they try to create a todo item without entering a text**- DONE**
* The user should be able to change the title of a particular todo item**- DONE**

*Optional:*

* The user should be able to see previously created todo items even after refreshing a page**- DONE**

Solution HTML page template:

[[Download solution template right here](https://ctdevcorp.sharepoint.com/:u:/s/L&D/EaDYiBcOv35JlQxNVVkrzR8BF79-_WSnhEKKkQi0sIgBxg?e=qfc2ds)]

Solution:

[Send back the completed solution file]

# 5. Summary

Why have you chosen to be a software developer? Describe your main objectives and hopes about your future carrier.

Answer:

I have chosen to be a software developer because almost everything in the world is being controlled by software, like: sales, bank systems, public transportation, broadcasting, healthcare, navigation, etc. It really assists the humanity to develop economy, rise standards of living. And there’s going to be even more software in the future. Software developers have been creating programs for decades and do they have professional knowledges and experience to be able to understand how it works.

I think, this understanding gives me a key to comprehension of technogenic world.