**Nombre: Rauel Farley Araújo Lima Fecha: \_\_\_\_\_\_\_\_\_\_\_**

*Responda em português ou inglês.*

1. **Write a simple code to solve the following problem: Given a group of letters, print all existing anagram. For example: {a,b,c} => abc; acb; bac; bca; cab; cba**

R-> O código fonte se encontra no seguinte repositório:

**Diretório**: fujitsu\java

[**https://github.com/RauelFarley/fujitsu.git**](https://github.com/RauelFarley/fujitsu.git)

1. **Describe a situation that you would override the method equals() from Object class.**

R-> Quando for necessário comparar o conteúdo lógico de dois objetos, para garantir a igualdade dos objetos, sempre útil quando há atributos únicos, como CPF ou matrícula.

1. **Which design pattern would you use to protect your code from a third party library, which will likely to be replaced in a couple of years.**

R-> FACADE é o padrão de projeto indicado para integrar a aplicação com biblioteca terceira.

1. **Describe what you know about Angular and provide some code and applications of use examples.**

R-> Angular é um framework front-end baseado em TypeScript, é um framework para desenvolvimento Single Page Applications.

O código fonte se encontra no seguinte repositório: [**https://github.com/RauelFarley/fujitsu.git**](https://github.com/RauelFarley/fujitsu.git)

**Diretório**: fujitsu\angular

O projeto angular se encontra dentro da pasta angular, a proposta do projeto é gerar um simples anagrama após o usuário digitar uma sequência de letras.

A estrutura do projeto segue o default do angular, uma aplicação SPA, foi criado um componente chamado *HomeComponent* que é responsável por receber a sequência de letras inseridas pelo usuário e então executar a função para gerar o anagrama e imprimir o resultado em tela.

1. **In web development, how do you prevent SQL injection attacks?**

R-> Utilizar frameworks ORM como Hibernate, validação de dados de entrada, fazer uso de queries parametrizadas e evitar concatenação de string com a querie.

1. **Let’s suppose you have a batch process that communicates with database, executes some internal logic and saves a file in a FTP server. User is complaining that this process is taking a long time to execute. How would you proceed to solve the problem?**

R-> Analisar o cenário, primeiro passo é revisar o processo batch, após a análise é possível refatorar o processo e dividi-lo em tarefas para que possam ser executados simultaneamente em threads separadas, utilizar ferramentas para monitoramento e analisar logs podem ajudar a analisar todo o cenário da lentidão no processamento em batch, dependendo, pode ser que seja necessário um escalonamento vertical ou horizontal, vai depender muito do "enviroment" onde a aplicação esteja hospedada.

|  |  |
| --- | --- |
| **Salesperson** | **Customer** |
| |  |  |  |  | | --- | --- | --- | --- | | **ID** | **Name** | **Age** | **Salary** | | 1 | Abe | 61 | 140000 | | 2 | Bob | 34 | 44000 | | 5 | Chris | 34 | 40000 | | 7 | Dan | 41 | 52000 | | 8 | Ken | 57 | 115000 | | 11 | Joe | 38 | 38000 | | |  |  |  |  | | --- | --- | --- | --- | | **ID** | **Name** | **City** | **Industry Type** | | 4 | Samsonic | Pleasant | J | | 6 | Panasung | Oaktown | J | | 7 | Samony | Jackson | B | | 9 | Orange | Jackson | B | |

|  |
| --- |
| **Orders** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **ID** | **order\_date** | **customer\_id** | **salesperson\_id** | **Amount** | | 10 | 8/2/96 | 4 | 2 | 540 | | 20 | 1/30/99 | 4 | 8 | 1800 | | 30 | 7/14/95 | 9 | 1 | 460 | | 40 | 1/29/98 | 7 | 2 | 2400 | | 50 | 2/3/98 | 6 | 7 | 600 | | 60 | 3/2/98 | 6 | 7 | 720 | | 70 | 5/6/98 | 9 | 7 | 150 | |

1. **Given the tables above, write the SQL query that:**

**a. Returns the names of all Salesperson that don’t have any order with Samsonic.**

**R->**

SELECT DISTINCT s.Name

FROM Salesperson s

WHERE s.ID NOT IN (

SELECT o.salesperson\_id

FROM Orders o

JOIN Customer c ON o.customer\_id = c.ID

WHERE c.Name = 'Samsonic'

);

**b. Updates the names of Salesperson that have 2 or more orders. It’s necessary to add an ‘\*’ in the end of the name.**

**R->**

UPDATE Salesperson

SET Name = CONCAT(Name, '\*')

WHERE ID IN (

SELECT salesperson\_id

FROM Orders

GROUP BY salesperson\_id

HAVING COUNT(ID) >= 2

);

**c. Deletes all Ssalesperson that placed orders to the city of Jackson.**

**R->**

**DELETE FROM Salesperson**

**WHERE ID IN (**

**SELECT DISTINCT o.salesperson\_id**

**FROM Orders o**

**JOIN Customer c ON o.customer\_id = c.ID**

**WHERE c.City = 'Jackson'**

**);**

**d. The total sales amount for each Salesperson. If the salesperson hasn’t sold anything, show zero.**

**R->**

SELECT

s.Name AS Salesperson,

COALESCE(SUM(o.Amount), 0) AS TotalSales

FROM

Salesperson s

LEFT JOIN

Orders o

ON

s.ID = o.salesperson\_id

GROUP BY

s.Name

ORDER BY

TotalSales DESC;

1. **The customer has a system called XYZ and he wants to start some updates that will be split into 3 phases. The requirements for the first phase are the following:**

* **I want to start new data entries in the system (new information that will be used in the 2nd phase)**
* **I want to include, update, delete and search plants**
* **Plants can contain code (only number) and description (10 digits)**
* **Only code is mandatory;**
* **No plant can have the same code;**
* **Only admin users can delete plants;**

**Based on the above information, create a use case (not the diagram) related to it.**

R-> O arquivo com o caso de uso se encontra no seguinte repositório:

Nome do arquivo: Caso\_de\_Uso\_Gerenciar\_de\_Plantas.docx

[**https://github.com/RauelFarley/fujitsu.git**](https://github.com/RauelFarley/fujitsu.git)

1. **Read the following description of a system functionality:**

**User Registration**

* **Screen where you can insert, delete or update user information;**
* **User has properties “name”, “email”, “address” and “phone”. Only the first two are mandatory;**
* **No user can have the same email;**
* **Only admin users can delete other users;**

**Describe which tests you would do to guarantee that everything is working correctly. (For example: insert user with every property filled)**

R-> Segue abaixo os cenários de testes:

1° cenário de teste: Inserção de usuário válido

* Inserir um usuário com todas as informações fornecidas corretamente.
* Inserir um usuário com apenas as informações obrigatórias (nome e e-mail) e verificar se o cadastro é concluído com sucesso.

**2º cenário de teste: Atualização de dados do usuário**

* Atualizar o nome, endereço ou telefone de um usuário existente e verificar se as mudanças são persistidas corretamente no sistema.
* Tentar atualizar o e-mail de um usuário e garantir que a validação de unicidade de e-mail seja mantida.

**3° cenário de teste: Exclusão de usuário**

* Verificar se um usuário administrador consegue excluir outro usuário com sucesso.
* Garantir que usuários não administradores recebam uma mensagem de erro ao tentar excluir outro usuário.

**4° cenário de teste: Validação de Campos**

* Inserir um usuário sem o campo "nome" e garantir que o sistema bloqueie o cadastro.
* Tentar cadastrar um usuário sem um e-mail e verificar se o sistema bloqueia.
* Tentar cadastrar um e-mail em formato inválido e garantir que o sistema rejeite o cadastro.
* Cadastrar dois usuários com o mesmo e-mail e garantir que o segundo cadastro seja rejeitado.