Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Transporte de Passageiros

Autores:	49487	Ricardo Duarte António Rovisco
	49504	Jorge Filipe de Medeiros Palácios da Silva
	49508	João Miguel Castanheira Mota

Relatório para a Unidade Curricular de Programação da Licenciatura em Engenharia Informática e de Computadores

Professor: Engenheira Matilde P. M. Pato

06 – 01 – 2023

<< Esta página foi intencionalmente deixada em branco >>

Resumo

No domínio da Unidade Curricular de Introdução a Sistemas de Informação, fomos encarregues de estabelecer uma ligação ao Sistema de Gerenciamento de Base de Dados, SGBD, uma interface entre vários usuários e a base de dados, utilizando Java Database Connectivity, JDBC, uma API baseada em X/Open SQL CLI que permite o acesso, em java, a base de dados relacionais e a execução em comandos SQL.

Para tal, foi utilizado o texto com os requisitos do sistema, apresentados na primeira fase do projeto, e o modelo de dados que foi implementado na segunda fase. A partir daí foi necessário desenvolver funções tais como a opção para inserir um novo condutor, ou colocar um veículo fora de serviço a partir da matrícula.

Abstract

Within the scope of the curricular unit Introduction of Information Systems, we carried out establishing a connection to the Data Base Management System, DBMS, an interface between users and the database, using Java Database Connectivity, an API based on X/Open SQL CLI that allows access, in java, the relational database and the execution in SQL commands.

To carry out this step, the text with the system requirements, presented in the first phase of the project, and the data model in the second phase were used. From then on, it was necessary to develop functions such as the option to insert a new driver or put a vehicle out of service based on the registration.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

1. Introdução

Uma empresa do setor dos transportes pretende criar um sistema informático para a gestão dos seus condutores, clientes e viagens. Para tal, é necessário desenvolver uma aplicação Java que permita realizar as seguintes operações: Opção para inserir um novo condutor; Opção para colocar um veículo fora de serviço baseado na matrícula, não podendo assim realizar mais viagens; Opção para calcular as horas totais, kilómetros e o custo total de um veículo, onde o utilizador deverá escolher o veículo dentro da lista dos existentes; Opção para apresentar a lista dos clientes com mais viagens de um ano à escolha; Opção para saber quais condutores não efetuaram viagens baseado no seu nome, id ou NIF; Opção que apresente, após inserido dados sobre o proprietário, o número de viagens obtidas pelo seu carro; Opção para a apresentação dos dados do condutor cujo total de viagens nesse ano, préviamente escolhido, teve o maior custo final acumulado, apresentando o seu nome, número de identificação e morada.

2. Update

O update serve para atualizar as tabelas, retirando os dados antigos e inserindo os dados novos colocados na devida tabela. Um exemplo é para quando um proprietário excede o limite de 20 veículos, esta tabela é atualizada, deixando apenas os top 20 veículos do proprietário. Também é usado para apagar os dados dos condutores cujo os id's são iguais aos de um proprietário, pois um proprietário não pode ser o condutor, e para transferir os veículos com mais de 5 anos para uma tabela diferente.

3. Query

Maior parte do foco da query foi buscar os elementos para adicionar à tabela veiculo_old, verificando a sua existência na 'check_veiculo_old', e, caso esta ainda não exista, esta é criada a partir da 'create_veiculo_old'.

4. RT

RT é a classe que possui as restrições todas. A primeira restrição é o limite de 20 veículos por proprietário, em que quando excede o limite, a tabela é atualizada deixando os top 20 veículos do proprietário.

```java
public void rt1() {
    try {
        String jdbcURL = "jdbc:postgresql://10.62.73.73:5432/";
        String username = "mp25";
        String password = "mp25";
        Connection conn = DriverManager.getConnection(jdbcURL, username, password);
        Statement sta = conn.createStatement();
        ResultSet rs = sta.executeQuery(queries.t1);

        while (rs.next()) {
            if (rs.getInt( columnLabel: "n_veiculos") > 20) {
                Statement stm = conn.createStatement();
                stm.executeUpdate(update.delete_veiculos(rs.getInt( columnLabel: "idpessoa")));
            }
        }

        rs.close();
        sta.close();
        conn.close();

    } catch (SQLException E) {
        E.printStackTrace();
    }
}
```

Listagem 1 Implementação da primeira restrição

```java
public void rt2() {
    try {
        String jdbcURL = "jdbc:postgresql://10.62.73.73:5432/";
        String username = "mp25";
        String password = "mp25";
        Connection conn = DriverManager.getConnection(jdbcURL, username, password);
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(queries.t2);

        while (rs.next()) {
            Statement stm = conn.createStatement();
            stm.executeUpdate(update.delete_condutor(rs.getInt( columnLabel: "idpessoa")));
        }
        rs.close();
        st.close();
        conn.close();

    } catch (SQLException exception) {
        exception.printStackTrace();
    }
}
```

A segunda restrição é eliminar os condutores com os mesmos id's de proprietários, garantindo assim que não existe condutores proprietários. Se existir algum condutor com o id do proprietário, este é eliminado.

Listagem 2 Implementação da segunda restrição

A terceira restrição é garantir que o número da carta de condução e a matrícula de um veículo cumprem o determinado formato para as tabelas. Esta restrição foi préviamente feita na fase anterior.

A quarta restrição é a verificação da existência da tabela veiculo_old e, caso não exista, é criada uma. Verifica também se há carros com mais de 5 anos, havendo estes são adicionados à nova tabela veiculo_old como todo os seus parâmetros com o número de viagens e os kilómetros percorridos e elemina os carros com mais de 5 anos da tabela veiculo.

5. Model

O ficheiro Model é onde se encontram as funções todas que são usadas na aplicação que se deseja desenvolver.

A função CheckRestrictions verifica as restrições todas feitas anteriormente.

Na função NewCondutor a query irá receber a informação vinda do utilizador que irá fazer a inserção do condutor, seleciona o idpessoa, ncondução e dtnascimento para ir fazer verificações e cria a lista para guardar os valores. Após fazer conexão à base de dados (BD), cria um statement, executa a query para ter a lista de condutores que existem e adiciona os resultados da query às respetivas listas e os id's válidos às listas, desligando então a conexão de modo a esta não ficar muito tempo ligada. Pergunta ao utilizador os dados para criar um novo condutor, inserindo os parâmetros pedidos, e faz conexão à BD para preencher a query com os dados inseridos.

A função DesactivateVehicle é usada para desativar os veículos com a matrícula introduzida pelo utilizador. Para isso é criado um statement onde executa a query para nos dar uma lista com as matrículas de veículos ativos. Depois pergunta ao utilizador a matrícula do veículo que deseja desativar para após fazer a conexão à BD pegar nas informações recebidas e adicionar o veículo à tabela veiculo_old.

A função VEHICLE_STATS apresenta os dados do veículo, o seu id, a sua marca e o seu modelo. Apresenta também os kilómetros que o veículo percorreu, o total de horas que o veículo andou e o custo total das viagens efetuadas pelo veículo. Esta função utiliza a função auxiliar totalTime que calcula o tempo total das viagens efetuadas pelo veículo.

O objetivo da função MOST_TRAVELS é apresentar a lista de clientes como o maior número de viagens efetuadas num ano determinado pelo utilizador. Para tal é feita uma conexão à BD para nos dar uma lista com os anos onde houve viagens. Depois é pedido ao utilizador um ano válido. Após serem selecionados os resultados finais para serem apresentados ao utilizador, é aberta outra conexão à BD para nos dar os dados do condutor com mais viagens nesse ano.

Na função NEVER_TRAVELED é aberta uma conexão onde se seleciona o id, nif e nome dos condutores que nunca efetuaram viagens e executa-se a query para nos dar uma tabela com os dados dos condutores.

A função NUM_TRAVELS_PER_YEAR retorna um número total de viagens que os veículos de um proprietário, com nif introduzido pelo utilizador, fizeram num ano, também determinado por um utilizador. Para começar é selecionado o nif de todos os proprietários e inicia-se uma conexão à BD para nos dar uma tabela com os seus nif's. Depois de selecionado o nif é aberta uma conexão onde se executa a query para nos dar uma tabela com os anos em que os veículos do proprietário efetuaram viagens e

adiciona esses veículos a uma lista. Após fechada a ligação pergunta ao utilizador o ano fazendo outra conexão para apresentar então os resultados ao utilizador.

A última função, BEST_PAID_DRIVER começa por selecionar os anos em que houve viagens para serem usados depois na validação dos dados introduzidos e são criadas listas para guardar os valores. É feita uma conexão à BD em que é executada a query para nos dar uma lista com os anos em que se efetuaram viagens. Depois é perguntado ao utilizador o ano que será selecionado. Seleciona os resultados finais para serem apresentados e abrindo uma conexão, executa a query para nos dar os dados do condutor que obteve o melhor pagamento no ano inserido pelo utilizador.

Foi feita também no final do ficheiro um método que faz conexão à base de dados, poupando assim repetição de código.

6. App

O ficheiro App.java constroi a app chamando as funções presentes no Model.java. Dentro da class App, tem uma enum class Option que enumera as opções que o utilizador têm ao iniciar a app e tem um função App que escolhendo uma opção dentro da enum class, é executada a função destinada a esse propósito. Tem também uma função DisplayMenu utilizada para apresentar as opções ao utilizador.

```
private enum Option {
    Unknown,
    NewCondutor,
    DesactivateVehicle,
    VehicleStats,
    MostTravels,
    NeverTraveled,
    NumTravelsPerYear,
    BestPaidDriver,
    Exit,
}
```

Listagem 3 Enumeração das opções do utilizador

```
private App() {
    __dbMethods = new HashMap<Option, DbWorker>();
    __dbMethods.put(Option.NewCondutor, new DbWorker() {public void doWork() {Model.newCondutor();}});
    __dbMethods.put(Option.DesactivateVehicle, new DbWorker() {public void doWork() {Model.DesactivateVehicle();}});
    __dbMethods.put(Option.VehicleStats,new DbWorker(){public void doWork(){Model.VEHICLE_STATS();}});
    __dbMethods.put(Option.MostTravels,new DbWorker(){public void doWork(){Model.MOST_TRAVELS();}});
    __dbMethods.put(Option.NeverTraveled,new DbWorker(){public void doWork(){Model.NEVER_TRAVELED();}});
    __dbMethods.put(Option.NumTravelsPerYear,new DbWorker(){public void doWork(){Model.NUM_TRAVELS_PER_YEAR();}});
    __dbMethods.put(Option.BestPaidDriver,new DbWorker(){public void doWork(){Model.BEST_PAID_DRIVER();}});
}
```

Listagem 4 Função App

Listagem 5 Função utilizada para a apresentação das opções ao utilizador

```
private Option DisplayMenu() {
    Option option = Option.Unknown;
    try {
        m.CheckRestrictions();
        System.out.println("Application");
        System.out.println();
        System.out.println("1. Inserir novo condutor");
        System.out.println("2. Por veiculo fora de serviço");
        System.out.println("3. Mostrar Status do veiculo");
        System.out.println("4. Mais por ano");
        System.out.println("5. Lista de Condutores que nunca viajaram");
        System.out.println("6. Viagens por ano dos veículos dos proprietarios");
        System.out.println("7. Condutor mais pago por ano");
        System.out.println("8. Exit");
        System.out.print(">");
        Scanner s = new Scanner(System.in);
        int result = s.nextInt();
        option = Option.values()[result];
    } catch (RuntimeException ex) {
        // nothing to do.
    }
    return option;
}
```

Para o uso da aplicação é necessário usar alguns comandos por cada opção:

1- Id(inserido pelo utilizador) →ncconducção(inserido pelo utilizador)→dtnascimento(inserido pelo utilizador)
2- Matricula(selecionado pelo utilizador)
3- Id(selecionado pelo utilizador)
4- Year(selecionado pelo utilizador)
5- (é necessário apenas introduzir 5)
6- Nif(selecionado pelo utilizador) →year(selecionado pelo utilizador)
7- Year(selecionado pelo utilizador)
8- (é necessário apenas introduzir 8)

7. Conclusão

O principal erro que nos causou problema foi um erro no BuildPath, onde foi necessário adicionar uma library e alterar o gradle. O programa já desenvolvido, encontra-se sem erros e a funcionar como pedido. Foram também efetuados testes, todos eles bem sucedidos.