Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Reservas Online - Relatório da Fase 3

47192 : Alexandre Silva (47192@alunos.isel.pt)

47220 : João Nunes (47220@alunos.isel.pt)

Relatório para a Unidade Curricular de Sistemas de Informação 1
da Licenciatura em Engenharia Informática e de Computadores

Professora : Doutora Matilde Pós-de-Mina Pato

20 – 06 – 2021

Resumo

Este relatório é constituido por uma pequena introdução aos conceitos de *Application Programming Interface* e de *Java Database Connectivity*. O seu desenvolvimento consistiu numa introdução de como se criou a ligação entre a base de dados e a aplicação. Este conta com a adição de métodos de modo a garantir restrições previamente irrealizáveis. Demonstram-se também as vantagens destas implementações através de técnicas usando classes e/ou método, tudo programado na linguagem *Java*. Num final conclusivo, apresentam-se várias observações ao problema exposto, outras abordagens possíveis e futuras atualizações.

Abstract

This report consists in a short introduction to concepts of Application Programming Interface and Java Database Connectivity. We introduce the reader to the connection between the database and the application and how it was achieved. Prior to this, we implement methods to assure restrictions that were impossible to realize in a previous version. We also show the advantages of using Java and its techniques such that we reach all the objectives for a solid database. In the end, we present several observations in the matter, other possible approaches and future updates.

Índice

Lista de Tabelas

1

Introdução

1.1 Contexto

API - ***Aplication Programming Interface*** Conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da sua própria implementação, mas apenas usar seus serviços. [1]

JDBC - ***Java Database Connectivity*** *Standard* de indústria para conectividade independente de bases de dados entre a linguagem Java e uma vasta escolha de bases de dados SQL e outras fontes de dados, como por exemplo *spreadsheets* ou *flat files*. [3]

1.2 Objetivos

Desenvolvimento de uma aplicação em Java que, com interacção de um utilizador, realize determinadas funções que consistem numa ligação entre a aplicação e o modelo físico de dados através da API JDBC. Esta tem em consideração as seguintes regras:

- Sempre que se justificar deve-se utilizar mecanismos transacionais necessários para garantir a atomicidade das operações e de prevenção a ataques *SQL Injection*;

- Descrição e, se necessário, explicação da realização das funções utilizadas para o funcionamento dos exercícios propostos no enunciado;
- Criação, em Java, de um modelo de dados que mapeie as relações utilizadas para objetos em memória;
- A implementação em Java terá de incluir as restrições de integridade necessárias que não foram possíveis implementar na fase anterior em SQL;
- Deve ser possível identificar cada um dos objetivos de aprendizagem no material que for entregue;

2

Desenvolvimento

2.1 Introdução

Seguindo as normas solicitadas, foram realizadas classes e os seus respetivos métodos em Java com o intuito de resolver os problemas propostos. Seguir-se-à uma breve descrição da estrutura da aplicação e, quando necessário, procedemos com uma explicação do código implementado.

2.2 Análise

Deu-se início ao trabalho começando pela implementação da API de forma a criar a ligação entre a aplicação e a BD[1]. Prosseguiu-se à criação do método *main* e das classes responsáveis pelos objetos. Foram criadas as essenciais para a realização dos problemas propostos. Tendo as bases para a aplicação sido criadas, de seguida realizaram-se os métodos responsáveis para a resolução das funções propostas para o programa. Durante esse processo sentimos necessidade de criar a classe presente em *package.utils*, *UserInput*, responsável pela receção e organização dos dados recebidos.

[1]BD - Base de Dados

2.3 Aplicação do utilizador

A classe *App.java* representa toda a lógica de interface com o utilizador. A aplicação corre em modo consola, ou seja, mostra os conteúdos num terminal e o único periférico de interação é o teclado. Esta tem que ser instanciada para poder correr. Primeiro, é inicializada a partir do *Run()* no qual cria uma conexão com a BD indicada, mostra o menu e passa a receber *input* do utilizador. Este menu é constituído por *1+n* opções, sendo a primeira (*1. Sair*) a opção para sair da aplicação. As outras *n* opções são criadas no enumerador *Option*, cada uma com o seu devido título/descrição. O utilizador insere o número da opção e pressiona *Enter* para aceder à função correspondente.

Compilar e executar O código-fonte vem com os ficheiros anexados a este relatório. Na pasta *code* encontra-se um *README.txt* que passam a ser descritos aqui também: Se se encontra no *Windows*, para compilar basta correr o ficheiro *compilar.bat* ou, manualmente abra um terminal na pasta *code* e insira a seguinte linha:

javac -cp ".;./lib/mssql-jdbc-9.2.1.jre8.jar"jdbc/JDBC.java

Para executar a aplicação basta correr o ficheiro *run.bat* ou, manualmente abra um terminal na pasta *code* e insira a seguinte linha:

java -cp ".;./lib/mssql-jdbc-9.2.1.jre8.jar"jdbc/JDBC

É importante utilizar a nova versáo do *script insertTable.sql*, uma vez que existia uma gralha nas matrículas dos autocarros (eram todas iguais). Este *script* encontra-se em *scripts/insertTable.sql*.

2.4 Classes de modelo (*package.model*)

Esta *package* é composta por várias classes que representam as tabelas da BD utilizada. É possível usar estes modelos num conjunto totalmente diferente de exercícios envolvendo a mesma BD. Internamente, cada classe contêm os parâmetros (colunas), um construtor vazio, um construtor que recebe todos os parâmetros, *setters* e *getters*. A criação destes objetos facilita o desenvolvimento e leitura do código, estando ao mesmo tempo, a par de uma arquitetura de *software* mais correta.

2.5 Acesso a dados (*package.jdbc*)

Esta classe representa a lógica de acesso a dados na BD. Tudo o que seja pedido no enunciado deste trabalho está incluído nesta classe. A função *main* cria a instância da aplicação e envia a *String* de ligação da BD para realizar a conexão. Uma vez conectados, passamos a vez ao utilizador e a decisão de escolha no menu apresentado é sua.

Em qualquer situação de acesso à BD, se necessário, utiliza-se um *PreparedStatement*. A *query string* é criada com zonas que são possíveis modificar o valor (indicado por um '?'). Usam-se *setters* (i.e.: *PreparedStatement.setInt(1, 2)*) em cada um desses valores, indicando o índice e o seu valor. Por fim, executa-se o *statement*. Outro modo é usar somente um *Statement*. Este recebe uma *query string* já com todos os valores predefinidos no programa.

Outra técnica segura que se deve praticar é o uso de *Connection.commit* e *Connection.rollback*. Isto serve para tornar uma ou várias alterações/consultas/atualizações à BD numa operação atómica. Exemplo: na falha de ligação durante um *update*, antes de se realizar o *commit*, todas as alterações são revertidas com o *rollback*.

2.5.1 Restrições de integridade

Ao inicializar a aplicação JDBC, a base de dados será avaliada no sentido de verificar se cumpre todos os requisitos solicitados. Estes são:

- Todas as viagens não ultrapassam o máximo de lugares.
- O tempo de chegada depende da distância e da velocidade máxima do transporte.
- Existe um número de telefone associado a cada reserva com pagamento efetuado através de MBWAY.

A restrição que refere "o atributo *ncarruagens* de COMBOIO é 6 quando o seu tipo é AP" já foi cumprida na fase 2 do trabalho prático, através de uma *constraint* na criação das tabelas.

AreTicketsAvailableFor Verifica se o autocarro que recebe como parâmetro *transporte* tem lugares (bilhetes) disponíveis. Retorna *true* caso esteja pelo menos um disponível. Caso contrário, mostra uma mensagem ao utilizador e retorna *false*.

ident	dataviag	horapartida	horachegada	distancia	estpartida	estchegada
1	2021-12-24	15:20:20	22:44:00	325	Cais do Sodré	Porto

Tabela 2.1: Viagem 1 antes da atualização

ident	dataviag	horapartida	horachegada	distancia	estpartida	estchegada
1	2021-12-26	15:20:20	22:44:00	325	Lisboa Sete Rios	Porto

Tabela 2.2: Viagem 1 depois de atualização

SetArrivalTimes Calcula a hora de chegada de uma viagem através da distância e a velocidade máxima do transporte utilizado; também foi criada a função *SetArrivalTime* que atualiza a hora de chegada para a viagem especificada.

CreateMBWayPaymentIfNeeded Verifica se todas as reservas têm um número associado, se não pede ao utilizador que registe as mesmas através do método *RegisterPhoneNumber*, que *a posteriori* terá mais uso quando for efetuado a adição de reservas.

UpdateBusOnDeleteCascade Este método realiza uma alteração numa *constraint* da tabela AUTOCARRO. A chave estrangeira com o nome *FK_AUTOCARRO_TRANSPORTE* não tinha a restrição *on delete cascade* na versão anterior. Com esta modificação passa a ser eliminado todas as referências ao autocarro na tabela TRANSPORTE.

2.5.2 Métodos de modificação da BD

AddReservation Pede ao utilizador que insira os dados da reserva a adicionar, verifica se a viagem não existe e caso exista adiciona assim a mesma, cumprindo sempre a restrição do pagamento MBWay.

UpdateTrip Encarregue de atualizar a viagem através do ID, apresenta os dados atuais de modo a conferir se é a viagem pretendida e de seguida recebe os dados e atualiza os mesmos na *database*. **Exemplo:** é feita uma atualização à viagem com identificador 1 (viagem 2.1). Pretendemos mudar a estação de partida e o dia da viagem. Ao inserir o comando *2021-12-26,15:20:20,22:44:00,325,Lisboa Sete Rios,Porto* a viagem é atualizada e a sua representação encontra-se na tabela 2.2

OutofServiceBus Este método é delegado para remover um autocarro de serviço, isto é, remover quaisquer viagens, bilhetes e lugares atribuídos ao mesmo, algo que é efetuado através de um simples *delete* do transporte, é possível visto que qualquer atributo

dependente do transporte é removido através da *constraint "delete on cascade"*, que automaticamente elimina quaisquer independências que a mesma tenha, e de seguida é atualizado esse mesmo autocarro para uma nova tabela, AUTOCARROPARADO, guardando o registo de todos os autocarros fora de serviço.

2.5.3 Métodos de *query*

Estes métodos tem implementação somente com o objetivo proposto no enunciado do trabalho. As suas implementações são muito parecidas, logo encontram-se nesta subsecção de modo generalizado. Estes métodos têm, por norma, a busca de informação à base de dados pedida pelo utilizador (só se realizam *queries*) e um retorno ao utilizador da resposta. Os métodos que cumprem estes pontos são:

- Método *GetBusMileage* - calcula a quilometragem total de um autocarro (a partir da matrícula) escolhido.
- Método *EmptySeatsBusFrom* - mostra quantos lugares vazios têm os autocarros que partem de uma localidade escolhida.
- Método *PriceSumByCategory* - indica a soma dos preços dos bilhetes por categoria escolhida.
- Método *AgeAverageByReservation* - apresenta a média de idades dos passageiros por reserva escolhida.
- Método *RunningTrips* - mostra todas as viagens realizadas de uma localidade de partida e de chegada escolhidas dentro de um intervalo de tempo escolhido.
- Método *AgeAverageByPayment* - calcula a média de idades pelo tipo de pagamento escolhido.

Estes métodos efetuam o seguinte fio de execução:

1. Obter *input* do utilizador (i.e.: data, identificador, tipo de pagamento).
2. Criar a *query* a partir de um *PreparedStatement* e uma *string*.
3. Obter *ResultSet* ao executar a *query*.
4. Iterar sobre o resultado, calcular valores pedidos, mostrar tabelas e, na ausência, indicar um erro.

2.6 Classes utilitárias (*package.util*)

DBTablePrinter Esta classe imprime na consola os resultados de um *ResultSet* de forma parecida a uma tabela de *Excel/SQL*. [2]

UserInput Esta classe serve como forma de abstrair e dividir a lógica de interface do utilizador da lógica de acesso a dados da BD. Assim, um *Scanner* lê cada vez que um objeto do tipo *UserInput* é criado.

3

Conclusão

A API realizada permite-nos realizar operações sobre a BD de forma mais interactiva e apelativa a quem poderá até mesmo ter conhecimento prévio de como a mesma funciona. A facilidade de alterar dados e fazer listagens que serão depois mostrados com um *feedback* simples e conciso, embora pré-programado convém ao utilizador um sistema mais proveniente ao seu uso.

3.1 Observações

No inicio do desenvolvimento da API para a BD foi requisitado que esta tivesse certas funções, no entanto, talvez pudesse ter sido melhor que houvesse abertura para criar outras, que pudessem ser mais interessantes ou complexas, para termos a hipótese de explorar mais esta ligação entre Java e SQL. Dito isto, as listagens que são pedidas poderiam ter sido em menor quantidade e trocadas por funções novas para promover o desenvolvimento de código novo ao invés de reutilizarmos algo que teríamos feito anteriormente.

3.2 Conhecimento

O estudo da ligação entre a BD e a API ajudou na compreensão de como estas poderiam funcionar no mundo profissional. Comparando este projecto com algo que

possamos ver na Internet, algo tão simples como uma filtragem de itens num site pode ser agora correlacionado com o objectivo do projecto que desenvolvemos. Ao longo do desenvolvimento da API foi de certa forma interessante perceber a relação de SQL com Java, uma vez que estamos a usar dois tipos de linguagens diferentes. O maior desafio que o grupo teve foi começar a construção da API e a correcção de erros que poderiam ocorrer por parte do utilizador ou por parte do código escrito em SQL.

Referências

[1] (Jun. de 2021). "Significado de api", URL: https://pt.wikipedia.org/wiki/Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es.

[2] (Jun. de 2021). "Dbtableprinter", URL: https://github.com/htorun/dbtableprinter.

[3] (Jun. de 2021). "Significado de jdbc", URL: https://www.oracle.com/java/technologies/javase/javase-tech-database.html.