Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Sistema de Gestão de Veículos - F1

47206 : Tiago Alexandre Figueiredo Pardal (47206@alunos.isel.pt)

47202 : Manuel Maria Penha Gonçalves Cavaco Henriques (47202@alunos.isel.pt)

47198 : Ricardo Filipe Matos Rocha (47198@alunos.isel.pt)

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

Professor : Doutor Nuno Miguel Soares Datia

09 – 05 – 2022

Resumo

No âmbito da primeira fase do trabalho prático da cadeira, este relatório tem como propósito a elaboração de uma base de dados para o problema proposto.

Através do trabalho realizado pretendemos demonstrar conhecimento sobre transações e níveis de transação, através da forma como resolvemos os problemas propostos.

Este relatório parte do pressuposto do acesso por parte do leitor ao código desenvolvido no âmbito do mesmo, não sendo assim necessário enunciá-lo em extensão, bastando apenas mencionar trechos do mesmo.

Abstract

Under the first phase of the assignment of this subject, this report has the purpose of building a database for the proposed problem.

Through this assignment we intend to demonstrate our knowledge about transactions and transaction levels, by presenting how the proposed problems were solved.

This report comes with the assumption that the reader has access to the developed code, being unnecessary to describe it to its full extent, as code snippets should suffice.

Índice

Lista de Figuras **ix**

1 Introdução **1**

1.1 Níveis de Isolamento 1

1.2 Vistas 2

1.3 Funções 2

1.4 Procedimentos 2

1.5 Gatilhos 2

2 Chapter 2 **3**

2.1 Caso em Estudo 3

2.2 Modelo EA 3

3 Detalhes de Implementação **5**

3.1 Atualização Informação Cliente Particular 5

3.2 Tratamento de Registos 5

3.3 Geração de alarmes 6

3.4 Inserir dados numa vista 7

3.5 Remover Clientes sem os remover 7

Lista de Figuras

2.1 Diagrama EA 4

1

Introdução

1.1 Níveis de Isolamento

De forma a manter a integridade transacional, assim como a consistência da DB, foi necessário atribuir a cada função e procedimento o nível de isolamento adequado.

Assim, consoante as operações que queríamos realizar, foi atribuído um dos seguintes níveis de isolamento:

- **READ COMMITED** - nível de isolamento mais baixo, uma vez que o PostgreSQL não implementa o nível de isolamento **READ UNCOMMITED**, garante que toda a informação que será lida, é informação que já se encontra committed, deste modo evitando **DIRTY READ**;

- **REPEATABLE READ** - bloqueia a leitura, escrita e updates, por parte de outras transações, sobre todos os tuplos referenciados, deste modo impedindo a ocorrência de **NON-REPEATABLE READS**;

- **SERIALIZABLE** - garante uma execução em serie das transações.

1.2 Vistas

Podem ser definidas de certa forma como uma camada de abstração sobre as relações presentes na base de dados, podendo ser consideradas de certa forma como interfaces obtidas através do result set de uma **query**. Os tuplos de uma vista são atualizados dinamicamente.

1.3 Funções

Funções, também chamadas de **STORED PROCEDURES**, permitem realizar operações sobre a base de dados diretamente, facilitando a dinâmica dos programas e evitando a realização de multiplas **queries** para um mesmo efeito.

1.4 Procedimentos

Procedimentos, por sua vez, são utilizados como transações, em que se pode realizar funções, dando uma adaptação em que permite o uso de sintaxe específica de transações com a chamada a funções incorporada.

1.5 Gatilhos

Os gatilhos permitem que caso numa relação seja realizada uma operação, uma função será executada em relação a essa operação. Os gatilhos têm três tipos e podem ser usados nas operações de **INSERT**, **UPDATE** ou **DELETE**. Um gatilho do tipo **INSTEAD OF** deverá repor a operação realizada na relação por uma função, ou seja, no caso de ser aplicado numa operação de **DELETE**, esta operação não se realizará, enquanto que a função que está no gatilho vai se realizar. Um gatilho **BEFORE** executa uma função antes que a operação sobre a relação possa ser realizada. Por fim, um gatilho **AFTER** realiza a sua função após a operação realizada na relação tenha concluído.

2

Chapter 2

2.1 Caso em Estudo

No presente trabalho é nos proposto o desenvolvimento de um sistema de gestão e registo de localização de automóveis e camiões.

Estando presentes neste as informações acerca dos clientes, dos veículos e das zonas a que os veículos se devem cingir, caso contrário deve ser gerado um alarme. Além dos próprios alarmes em si quando em situação de serem ativados.

2.2 Modelo EA

Na presente implementação foi considerado como vantajoso a separação entre veículo e condutor, de forma a facilitar a alteração do mesmo.

Para a realização desta separação foi necessário a atribuição de um id para a tabela condutor, ao que achamos que faria sentido não ser um id genérico, mas sim o seu número de cartão de cidadão cc, sendo que entendemos por cc o número de identificação civil e não o número de cartão de cidadão, visto interessar-nos apenas um identificador do cidadão e não do cartão de cidadão do mesmo.

O estado do equipamento tem como valores: {'Activo','PausaDeAlarmes','Inactivo'}.

As coordenadas GPS latitude e longitude têm graus decimais e foram assim definidas com o tipo **numeric(3,1)**, permitindo assim uma casa decimal.

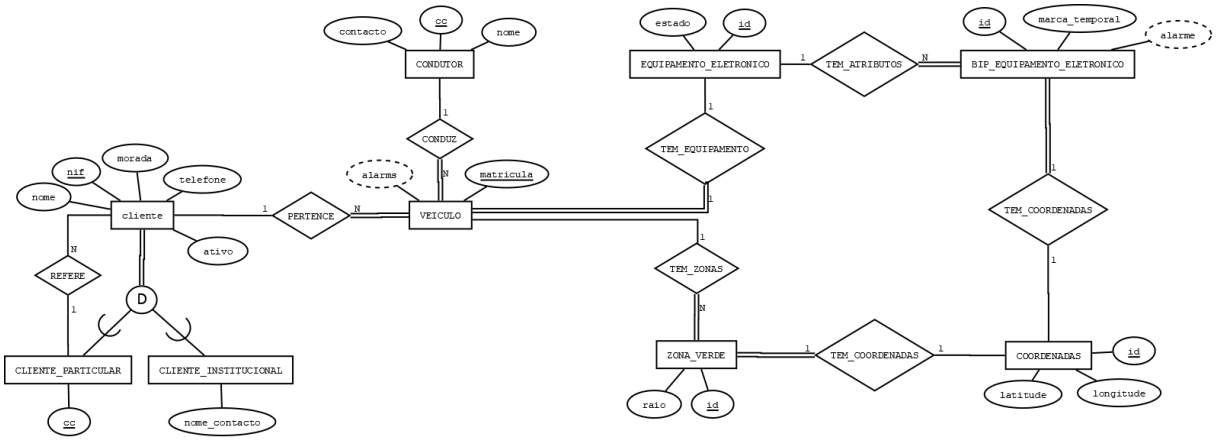

Figura 2.1: Diagrama EA

O atributo ativo é explicado em maior detalhe na secção Remover Clientes sem os remover.

O atributo alarme é um booleano que quando ativo representa a existência de alarme naquele bip.

Achamos a implementação de uma restrição em matricula poderia ser demasiado limitadora, assim não implementada nenhuma.

Existem 2 relações não presentes no modelo EA sendo estas **Requests** e **Invalid_Requests**, estas são referidas em maior detalhe na secção Tratamento de Registos.

3

Detalhes de Implementação

3.1 Atualização Informação Cliente Particular

Como já mencionado anteriormente quando falamos em cc referimo-nos a número de identificação civil e assim na implementação da atualização da informação de um cliente particular não atualizamos o cc e nif visto considerarmos que indivíduos não os podem alterar.

3.2 Tratamento de Registos

Na nossa implementação temos 3 tabelas de registos:

- **Bip_Equipamento_Eletronico** - Que corresponde aos registos processados;
- **Requests** - Que corresponde aos registos a serem processados;
- **Invalid_Requests** - Que corresponde aos Registos Inválidos.

O tratamento da informação dá-se do seguinte modo:

Os bips de equipamento eletrónico são registados a cada 10 segundos pelos equipamentos eletrónicos respetivos, que se encontram dentro dos veículos.

Cada bip é inicialmente registado nos **Requests**.

Os registos recebidos, são processados a cada 5 minutos em lote através da chamada do procedimento **processRequests()** que procede ao processamento de todos os registos guardados procedendo à sua validação através do procedimento **validateRequest()**.

Este processo tenta validar os vários parâmetros, caso sejam válidos, são movidos para a relação Bip Equipamento Eletrónico, caso contrário, move-los para a relação **Invalid Requests**.

Aquando da validação de um registo, verifica-se se as coordenados do mesmo existem na relação coordenadas e caso não existam são acrescentadas à mesma.

Na relação **Invalid Requests** encontram-se armazenados todos os registos inválidos para serem mantidos durante 15 dias e após este período serem removidos.

Na presente implementação o processamento de registos não se encontra automatizado sendo necessário chamar o procedimento **processRequests()** a cada 5 minutos com a seguinte instrução:

```
1 CALL processRequests();
```

À semelhança do **processRequests()** a remoção dos **Invalid Requests** após 15 dias também não se encontra automatizada devendo ser realizada diariamente com a seguinte instrução:

```
1 CALL deleteInvalids();
```

3.3 Geração de alarmes

Sempre que um novo registo é movido para a relação bip_equipamento_eletronico, o gatilho **checkAlarm** é chamado e verifica se as coordenadas fornecidas no registo se encontram numa zona verde.

Caso o equipamento se encontre fora do grupo das suas zonas verdes, o alarme do veículo é ativado, sendo adicionada uma nova entrada na relação alarmes.

Por sua vez, a inserção de um novo alarme ativa o gatilho **alarmAdded** que irá incrementar o contador de alarmes do veículo correspondente.

3.4 Inserir dados numa vista

A inserção de dados em vistas em postgres SQL originalmente não era possível por as vistas não serem mutáveis, no entanto mais tarde foi implementada apenas para vistas simplificadas.

No nosso caso de estudo a vista não poderia ser simples, devido a necessitar de obter informações de mais do que uma relação, e assim para o caso específico em que necessitamos de executar uma inserção de dados numa vista a mesma é para todos os efeitos imutável.

A inserção numa vista não passa de uma inserção nas relações por trás desta.

Assim implementamos uma inserção através de um gatilho do tipo **INSTEAD OF INSERT**, que em vez de tentar inserir dados na vista, corre uma função onde são efetuadas as inserções nas relações que dão informação à vista.

Como o que é pedido com este **INSERT** é apenas que se adicione um alarme e o respetivo registo tratado, consideramos que caso o veículo e condutor não existam já na base de dados não os inserimos e damos origem a uma exceção, o mesmo se sucede se o condutor, não for o condutor do veículo especifico passado como parâmetro.

Assim, tendo já realizado as verificações previamente referidas, procedemos à obtenção do id do equipamento através duma **query** com a matrícula e o condutor e a verificação das coordenadas, procurando verificar se elas já existem na base de dados, se existirem retornamos o id em que se encontram caso não existam inserimo-las.

Procedemos à inserção no Bip_Equipamento_Eletronico de:

- equipamento obtido a partir da **query**;
- marca temporal passada como parâmetro;
- id de coordenadas obtidas ou geradas.

E concluímos com a inserção em Alarmes do id do bip em que acabamos de inserir.

3.5 Remover Clientes sem os remover

Foi implementado a remoção de clientes sem os remover da base de dados através da criação de um atributo ativo do tipo booleano, que quando a **True** são considerados os clientes como ativos e quando a **False** são considerados os clientes como inativos.

Ou seja quando o atributo ativo se encontra a **False**, no contexto da nossa implementação consideramos que esse cliente foi removido da DB, e os seus veículos não devem ser considerados.

Assim a remoção foi alterado através de um gatilho **BEFORE DELETE** que executa a função **delete_clientes()**, que por sua vez altera a variável ativo para **False**.