Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Projeto (Fase normal v.1.01)

Partilha de Trotinetas Elétricas

Autores: 35164 Fernando Duarte
36240 Rafael Martins
43594 Ricardo Rodrigues

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

Professor: Engº João Vitorino

03 – 06 – 2025

<< Esta página foi intencionalmente deixada em branco >>

Resumo

Pretende-se aplicar o conhecimento adquirido e lecionado ao longo do semestre de forma aplicar sobre um sistema de informação com o nome de projecto “CITES”, tratando-se de um serviço de partilha de trotinetas eléctricas numa cidade. Pretende-se o seguinte no decorrer do trabalho:

- Garantir a correta implementação das restrições de integridade e/ou lógica de negócio;
- Usar funções, procedimentos armazenados e gatilhos para implementar restrições de integridade na base de dados;
- Desenvolver uma camada de acesso a dados, que use uma implementação de JPA (Jakarta Persistence);
- Conhecer um conjunto de padrões de desenho, incluindo ValueObject DataMapper, LazyLoad, IndentityMap Repository, QueryObject e UnitOfWork, e relacionalos com o seu uso em JPA;
- Desenvolver uma aplicação em Java, que use adequadamente a camada de acesso a dados;
- Utilizar corretamente processamento transacional, através de mecanismos disponíveis no JPA;
- Garantir a correta libertação de ligações e recursos, quando estes não estejam a ser utilizados.

Índice

1. Introdução 5

2. Restrições de Integridade em PL/pgSQL 6

2.1 Restrição: Apenas uma trotineta numa doca pode ser usada para iniciar uma viagem 6

2.2 Restrição: Uma trotineta e um utilizador só podem participar numa viagem em curso de cada vez 6

3. Função de Cálculo de Ocupação 7

4. Vista RIDER com Suporte a Inserções e Atualizações 8

4.1 Definição da Vista RIDER 8

4.2 Trigger para Inserções na Vista RIDER 8

4.3 Trigger para Atualizações na Vista RIDER 8

5. Procedimento Armazenado para Iniciar Viagem 9

6. Aplicação Java 9

6.1 Modelo de Dados em Java 9

6.2 Camada de Acesso a Dados 9

6.3 Interface de Utilizador 10

6.4 Método para Criar Cliente 10

6.5 Método para Listar Docas com Ocupação 10

6.6 Método para Estacionar Trotineta 10

7. Teste de Bloqueio Otimista 10

7.1 Implementação do Bloqueio Otimista 11

7.2 Método para Estacionar Trotineta com Bloqueio Otimista 11

7.3 Teste de Conflito de Bloqueio Otimista 11

8. Conclusão 11

9. Referências 12

Lista de Figuras

Figura 1: Diagrama de Classes do Sistema ……..5

Listagens

Listagem 1 – Definição da vista RIDER ……..8

1. Introdução

O presente relatório documenta o desenvolvimento de um sistema de informação para gestão de trotinetas partilhadas, conforme solicitado no enunciado do projeto. O sistema visa permitir a gestão de utilizadores, passes, estações, docas, trotinetas e viagens, implementando diversas funcionalidades através de uma combinação de tecnologias de base de dados e programação Java.

O projeto baseia-se num modelo de dados relacional que representa o domínio de negócio de um serviço de trotinetas partilhadas, onde os utilizadores podem adquirir passes, realizar viagens e estacionar trotinetas em docas específicas. A implementação inclui restrições de integridade programáticas, funções e procedimentos armazenados, bem como uma aplicação Java que interage com a base de dados.

O modelo de dados inclui entidades como PERSON, CLIENT, EMPLOYEE, CARD, TYPEOFCARD, STATION, DOCK, SCOOTER, SCOOTERMODEL, TRAVEL, REPLACEMENTORDER, REPLACEMENT, TOPUP, SERVICECOST, que representam os diversos elementos do sistema. As relações entre estas entidades são estabelecidas através de chaves estrangeiras e restrições de integridade, garantindo a consistência dos dados.

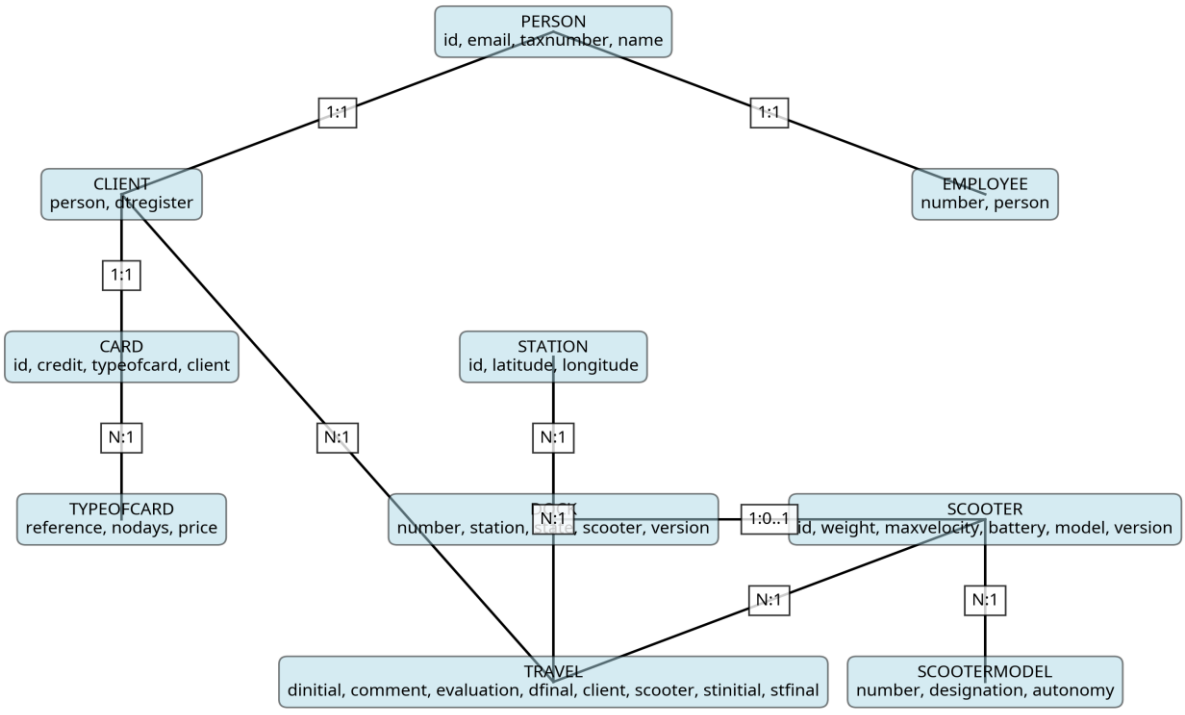

Diagrama de Classes

Figura 1: Diagrama de Classes do Sistema

O desenvolvimento do projeto seguiu uma abordagem modular, com a implementação de componentes específicos para cada requisito do enunciado. Nas secções seguintes, apresentamos detalhadamente cada componente implementado, incluindo o código desenvolvido, explicações sobre as decisões tomadas e resultados obtidos.

2. Restrições de Integridade em PL/pgSQL

2.1 Restrição: Apenas uma trotineta numa doca pode ser usada para iniciar uma viagem

Uma das restrições de integridade fundamentais do sistema é garantir que apenas uma trotineta que esteja numa doca possa ser utilizada para iniciar uma viagem. Esta restrição é crucial para manter a consistência do sistema e evitar situações em que uma trotineta que não está disponível numa doca seja erroneamente atribuída a uma viagem.

Para implementar esta restrição, foi desenvolvida uma função em PL/pgSQL que é acionada através de um trigger antes da inserção de um novo registo na tabela TRAVEL. A função verifica se a trotineta especificada está numa doca e se o estado dessa doca é "occupy", indicando que a trotineta está disponível para utilização.

A função `scooter_in_dock()` consulta a tabela DOCK para verificar o estado da doca onde a trotineta está localizada. Se a trotineta não estiver numa doca (dock_status IS NULL) ou se o estado da doca não for "occupy", a função lança uma exceção com uma mensagem de erro apropriada, impedindo a inserção do registo na tabela TRAVEL.

O trigger `trigger_scooter_in_dock` é configurado para ser executado antes de cada inserção na tabela TRAVEL, garantindo que a restrição seja verificada antes de qualquer nova viagem ser registada no sistema.

2.2 Restrição: Uma trotineta e um utilizador só podem participar numa viagem em curso de cada vez

Outra restrição importante é garantir que uma trotineta e um utilizador só possam participar numa viagem em curso de cada vez. Isto evita situações em que uma trotineta seja atribuída a múltiplas viagens simultâneas ou que um utilizador inicie uma nova viagem sem ter terminado a anterior.

Para implementar esta restrição, foi desenvolvida uma função em PL/pgSQL que é acionada através de um trigger antes da inserção de um novo registo na tabela TRAVEL. A função verifica se a trotineta ou o utilizador especificado já estão envolvidos numa viagem em curso (onde dfinal IS NULL).

A função `check_ongoing_trip()` realiza duas verificações: 1. Verifica se a trotineta especificada já está em uso numa viagem em curso, contando o número de registos na

tabela TRAVEL onde o ID da trotineta corresponde ao da nova viagem e a data final (dfinal) é NULL. 2. Verifica se o utilizador especificado já tem uma viagem em curso, contando o número de registos na tabela TRAVEL onde o ID do cliente corresponde ao da nova viagem e a data final (dfinal) é NULL.

Se qualquer uma destas verificações encontrar uma viagem em curso, a função lança uma exceção com uma mensagem de erro apropriada, impedindo a inserção do registo na tabela TRAVEL.

O trigger `trigger_check_ongoing_trip` é configurado para ser executado antes de cada inserção na tabela TRAVEL, garantindo que a restrição seja verificada antes de qualquer nova viagem ser registada no sistema.

Estas restrições de integridade são fundamentais para manter a consistência dos dados no sistema, evitando situações que poderiam levar a inconsistências ou comportamentos inesperados.

3. Função de Cálculo de Ocupação

A função `fx_dock_occupancy` foi implementada para calcular e retornar o nível de ocupação de uma estação de docas como uma percentagem (um valor entre 0 e 1). Esta função é crucial para monitorizar a disponibilidade de docas nas estações e pode ser utilizada para tomar decisões sobre reposicionamento de trotinetas.

A função recebe como parâmetro o ID de uma estação (`stationid`) e realiza os seguintes passos:

1. Conta o número total de docas disponíveis na estação, excluindo aquelas que estão em manutenção (state != ‘under maintenance’).
2. Verifica se existem docas disponíveis. Se não houver, retorna 0 para evitar divisão por zero.
3. Conta o número de docas ocupadas na estação (state = ‘occupy’).
4. Calcula a taxa de ocupação dividindo o número de docas ocupadas pelo número total de docas disponíveis, convertendo os valores para o tipo NUMERIC para garantir precisão na divisão.
5. Retorna a taxa de ocupação como um valor entre 0 e 1.

Esta função é particularmente útil para: - Monitorizar a disponibilidade de docas em tempo real - Identificar estações que necessitam de reposicionamento de trotinetas - Fornecer informações aos utilizadores sobre a disponibilidade de docas - Gerar relatórios sobre a utilização das estações

A implementação considera apenas as docas que estão operacionais (excluindo as que estão em manutenção), proporcionando assim uma visão mais precisa da ocupação real da estação. O resultado é um valor normalizado entre 0 e 1, que pode ser facilmente convertido em percentagem multiplicando por 100.

4. Vista RIDER com Suporte a Inserções e Atualizações

A vista RIDER foi implementada para fornecer uma visão integrada dos dados dos clientes, incluindo informações pessoais e detalhes do cartão. Para tornar esta vista mais funcional, foram implementados triggers que permitem inserções e atualizações através da vista, propagando as alterações para as tabelas subjacentes.

4.1 Definição da Vista RIDER

```
CREATE OR REPLACE VIEW RIDER
AS
SELECT p.*,c.dtregister,cd.id AS cardid,cd.credit,cd.typeofcard
FROM CLIENT c INNER JOIN PERSON p ON (c.person=p.id)
    INNER JOIN CARD cd ON (cd.client = c.person);
```

Listagem 1 – Definição da vista RIDER

Esta vista combina informações das tabelas PERSON, CLIENT e CARD, proporcionando uma visão completa dos dados do cliente num único resultado de consulta.

4.2 Trigger para Inserções na Vista RIDER

O trigger `trg_insert_rider` é acionado em vez de uma inserção direta na vista RIDER. A função `insert_rider()` realiza as seguintes operações:

1. Insere um novo registo na tabela PERSON com os dados fornecidos (email, taxnumber, name) e obtém o ID gerado.
2. Insere um novo registo na tabela CLIENT, associando-o à pessoa recém-criada e utilizando a data de registo fornecida ou a data/hora atual se não for especificada.
3. Insere um novo registo na tabela CARD, associando-o ao cliente recém-criado e utilizando os dados de crédito e tipo de cartão fornecidos.
4. Atualiza os valores de NEW.id e NEW.cardid com os IDs gerados, para que possam ser retornados na vista.

4.3 Trigger para Atualizações na Vista RIDER

O trigger `trg_update_rider` é acionado em vez de uma atualização direta na vista RIDER. A função `update_rider()` realiza as seguintes operações:

1. Atualiza o registo correspondente na tabela PERSON com os novos valores de nome e email.
2. Se um novo valor de crédito for fornecido, atualiza o registo correspondente na tabela CARD com os novos valores de crédito e tipo de cartão.

Estes triggers permitem que a vista RIDER seja utilizada como uma interface simplificada para operações de inserção e atualização de dados de clientes, ocultando a complexidade das operações nas tabelas subjacentes e proporcionando uma experiência mais intuitiva para os utilizadores do sistema.

5. Procedimento Armazenado para Iniciar Viagem

O procedimento armazenado `startTrip` foi implementado para iniciar uma viagem, realizando todas as verificações necessárias e atualizando as tabelas relevantes numa única transação atómica. Este procedimento é fundamental para garantir a consistência dos dados durante o processo de início de viagem.

O procedimento `startTrip` recebe dois parâmetros: o ID da doca (`dockid`) e o ID do cliente (`clientid`). O procedimento realiza as seguintes operações:

3. Obtém o custo de desbloqueio da tabela SERVICECOST.
4. Verifica se a doca especificada existe e está ocupada (tem uma trotineta disponível), obtendo o ID da trotineta e da estação.
5. Verifica se o cliente especificado existe e tem um cartão associado, obtendo o ID do cartão e o saldo atual.
6. Verifica se o cliente tem saldo suficiente para cobrir o custo de desbloqueio.
7. Verifica se o cliente já tem uma viagem em curso.
8. Verifica se a trotineta já está em uso numa viagem em curso.
9. Se todas as verificações forem bem-sucedidas, inicia uma transação que:
   - Atualiza o estado da doca para “free” e remove a referência à trotineta.
   - Regista uma nova viagem na tabela TRAVEL com a data/hora atual, o cliente, a trotineta e a estação inicial.
   - Deduz o custo de desbloqueio do saldo do cartão do cliente.

O procedimento inclui tratamento de exceções para garantir que, em caso de erro em qualquer uma das operações, a transação seja revertida e o erro seja propagado para o chamador.

6. Aplicação Java

A aplicação Java foi desenvolvida para interagir com a base de dados e fornecer uma interface de utilizador para as operações do sistema. A aplicação utiliza JPA (Java Persistence API) para mapeamento objeto-relacional e inclui funcionalidades para criar clientes, listar clientes, listar docas, iniciar viagens e estacionar trotinetas.

6.1 Modelo de Dados em Java

O modelo de dados em Java inclui classes que representam as entidades do sistema, como a classe `Dock` que representa uma doca.

A classe `Dock` é anotada com `@Entity` e `@Table` para indicar que é uma entidade JPA mapeada para a tabela “dock” na base de dados. A classe inclui campos para o ID, estação, estado, trotineta e versão (para bloqueio otimista), bem como métodos getter e setter para aceder e modificar estes campos.

6.2 Camada de Acesso a Dados

A camada de acesso a dados é implementada na classe `Dal`, que fornece métodos para interagir com a base de dados.

A classe `Dal` utiliza JPA para interagir com a base de dados, incluindo métodos para: - Inserir um novo cliente utilizando a vista RIDER - Listar todos os clientes com os seus detalhes - Listar todas as docas com os seus níveis de ocupação - Iniciar uma viagem utilizando o procedimento armazenado `startTrip` - Estacionar uma trotineta numa doca, utilizando bloqueio otimista para evitar conflitos

6.3 Interface de Utilizador

A interface de utilizador é implementada na classe `UI`, que fornece um menu de opções para o utilizador.

A classe `UI` utiliza um padrão Singleton para garantir que apenas uma instância da interface de utilizador existe na aplicação. A classe inclui um menu de opções e métodos para cada opção, que recolhem dados do utilizador e chamam os métodos correspondentes na classe `Dal`.

6.4 Método para Criar Cliente

O método `createCostumer` na classe `UI` permite ao utilizador criar um cliente.

Este método recolhe dados do utilizador através da consola, incluindo nome, email, NIF, tipo de cartão e crédito inicial, e chama o método `insertRider` na classe `Dal` para inserir o novo cliente na base de dados.

6.5 Método para Listar Docas com Ocupação

O método `listDocks` na classe `UI` permite ao utilizador listar todas as docas com os seus níveis de ocupação.

Este método chama o método `listDocksWithOccupancy` na classe `Dal`, que executa uma consulta SQL que utiliza a função `fx_dock_occupancy` para calcular o nível de ocupação de cada estação e apresenta os resultados na consola.

6.6 Método para Estacionar Trotineta

O método `parkScooter` na classe `UI` permite ao utilizador estacionar uma trotineta numa doca.

Este método recolhe o ID do cliente e o ID da doca através da consola e chama o método `parkScooter` na classe `Dal` para estacionar a trotineta na doca especificada. O método `parkScooter` na classe `Dal` utiliza bloqueio otimista para evitar conflitos quando múltiplos utilizadores tentam estacionar trotinetas na mesma doca simultaneamente.

7. Teste de Bloqueio Otimista

O bloqueio otimista é uma técnica de controlo de concorrência que permite que múltiplos utilizadores acedam e modifiquem dados simultaneamente, sem bloquear o acesso aos dados. Em vez disso, o sistema verifica se os dados foram modificados por

outro utilizador antes de confirmar uma alteração, utilizando um campo de versão para detetar conflitos.

7.1 Implementação do Bloqueio Otimista

No nosso sistema, o bloqueio otimista é implementado na classe `Dock` através do campo `version`, que é anotado com `@Version`.

Este campo é automaticamente atualizado pelo JPA sempre que um registo é modificado, e é utilizado para detetar conflitos quando múltiplos utilizadores tentam modificar o mesmo registo simultaneamente.

7.2 Método para Estacionar Trotineta com Bloqueio Otimista

O método `parkScooter` na classe `Dal` utiliza bloqueio otimista para evitar conflitos quando múltiplos utilizadores tentam estacionar trotinetas na mesma doca.

Este método realiza as seguintes operações: 1. Encontra a viagem em curso do cliente especificado. 2. Obtém a doca especificada com bloqueio otimista (`LockModeType.OPTIMISTIC`). 3. Verifica se a doca está livre. 4. Atualiza a doca para incluir a trotineta e mudar o estado para “occupy”. 5. Atualiza a viagem para incluir a data/hora final e a estação final.

Se outro utilizador modificar a doca entre o momento em que é lida e o momento em que é atualizada, o JPA lançará uma `OptimisticLockException`, que é capturada e tratada pelo método.

7.3 Teste de Conflito de Bloqueio Otimista

Para testar o bloqueio otimista, foram criados cenários em que múltiplos utilizadores tentam estacionar trotinetas na mesma doca simultaneamente. Estes testes foram realizados utilizando a classe `OptimisticLockConflictTest`.

Os testes simulam o seguinte cenário: 1. Dois utilizadores (Cliente 1 e Cliente 2) tentam estacionar trotinetas na mesma doca simultaneamente. 2. Ambos os utilizadores leem o estado atual da doca (que está livre). 3. O Cliente 1 atualiza a doca para incluir a sua trotineta e muda o estado para “occupy”. 4. O Cliente 2 tenta atualizar a doca com a mesma versão que leu inicialmente, mas a versão já foi incrementada pela atualização do Cliente 1. 5. O JPA lança uma `OptimisticLockException` para o Cliente 2, indicando um conflito de concorrência.

Este teste demonstra como o bloqueio otimista protege a integridade dos dados, evitando que múltiplos utilizadores modifiquem o mesmo registo simultaneamente de forma inconsistente.

8. Conclusão

O projeto desenvolvido implementa um sistema de informação para gestão de trotinetas partilhadas, incluindo funcionalidades para gestão de utilizadores, passes,

estações, docas, trotinetas e viagens. O sistema utiliza uma combinação de tecnologias de base de dados e programação Java para fornecer uma solução completa e robusta.

As principais funcionalidades implementadas incluem: - Restrições de integridade programáticas em PL/pgSQL - Função para cálculo de ocupação de docas - Vista RIDER com suporte a inserções e atualizações - Procedimento armazenado para iniciar viagem - Aplicação Java para interação com a base de dados - Bloqueio otimista para evitar conflitos de concorrência

O sistema demonstra a aplicação de conceitos avançados de bases de dados, incluindo triggers, procedimentos armazenados, vistas atualizáveis e controlo de concorrência, bem como a integração com uma aplicação Java utilizando JPA para mapeamento objeto-relacional.

Os testes realizados confirmam a robustez do sistema, especialmente no que diz respeito ao controlo de concorrência através do bloqueio otimista, garantindo a integridade dos dados mesmo em cenários de acesso simultâneo.

9. Referências

1. Documentação do PostgreSQL: https://www.postgresql.org/docs/
2. Documentação do JPA: https://jakarta.ee/specifications/persistence/
3. Enunciado do Projeto de Sistemas de Informação, ISEL-DEETC, 2024-2025