Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Partilha de Trotinetes Elétricas

Autores: 51532 Bruno Saldanha
49504 Jorge Palácios da Silva
51283 João Nunes Chuço

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

Professor: João Vitorino

03 – 06 – 2025

<< Esta página foi intencionalmente deixada em branco >>

Resumo

No domínio da Unidade Curricular a Sistemas de Informação, fomos encarregues de desenvolver um sistema de informação para o projeto "CITES" que teve como objetivo desenvolver um sistema de gestão para o compartilhamento de trotinetes elétricas, focando na implementação de uma base de dados ativa que suporte as operações do sistema. O trabalho foi dividido em várias etapas, cada uma abordando diferentes aspectos do sistema.

Abstract

Within the scope of the Information Systems course, we were tasked with developing an information system for the "CITES" project, which aimed to create a management system for the sharing of electric scooters, focusing on the implementation of an active database to support the system's operations. The work was divided into several stages, each addressing different aspects of the system.

Índice

Listagens

1. Introdução

O projeto "CITES" teve como objetivo desenvolver um sistema de gestão para o compartilhamento de trotinetes elétricas, focando na implementação de uma base de dados ativa que suporte as operações do sistema. O trabalho foi dividido em várias etapas, cada uma abordando diferentes aspectos do sistema.

Para tal foi necessário fazer uma modelagem da base de dados, onde definimos as entidades principais, foi também necessário implementar restrições de integridade através de triggers. Houve também a criação de funções para calcular a ocupação das docas, criação de views e desenvolvimento de procedimentos.

Para terminar foi também necessário completar a aplicação Java para nos permitir criar/listar um cliente, listar as docas e a sua ocupação, iniciar uma viagem e colocar uma trotinete numa doca.

2. Modelagem da base de dados

A modelagem da base de dados é um processo fundamental no desenvolvimento de sistemas de informação, que envolve a criação de uma representação abstrata da estrutura da base de dados. O objetivo é organizar e definir como os dados serão armazenados, relacionados e acessados. Os principais componentes e etapas da modelagem da base de dados são a indentificação de entidades, definição de atributos, estabelecimento de relações, criação do diagrama Entidade-Relacionamento, Normalização, definição de restrições de integridade, e impelmentação da estrutura da base de dados no sistema de gerenciamento de banco de dados (SGBD).

3. Implementação de Restrições de Integridade

Para as restrições de integridade apenas implementamos duas, através de gatilhos, que foram dadas inicialmente pelo enunciado. A primeira dita que apenas uma trotinete que está numa doca pode ser usada no início de uma viagem. A segunda apenas permite que uma trotinete e um utilizador que não se encontrem numa viagem, possa participar de uma.

```sql
CREATE OR REPLACE FUNCTION check_scooter_in_dock()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the scooter is already in a dock
    IF NOT EXISTS (
        SELECT 1
        FROM DOCK
        WHERE scooter = NEW.scooter
            AND state = 'occupy'
    ) THEN
        RAISE EXCEPTION 'Scooter not available in any dock';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listagem 1 - Primeira Restrição

```sql
CREATE OR REPLACE FUNCTION check_unique_trip()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the scooter already has an active trip
    IF EXISTS (
        SELECT 1
        FROM TRAVEL
        WHERE scooter = NEW.scooter
            AND dfinal IS NULL
    ) THEN
        RAISE EXCEPTION 'Scooter is on an active trip';
    END IF;

    -- Check if the client is on a trip
    IF EXISTS (
        SELECT 1
        FROM TRAVEL
        WHERE client = NEW.client
            AND dfinal IS NULL
    ) THEN
        RAISE EXCEPTION 'Client already has an active trip';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listagem 2 - Segunda Restrição

4. Calcúlo da Ocupação das Docas

O cálculo da ocupação das docas é uma funcionalidade que permite monitorar a disponibilidade das docas em cada estação, facilitando a tomada de decisões e a otimização dos recursos. Para tal foi implementada uma função, que recebe como parâmetro o identificador de estação e retorna a percentagem de docas ocupadas em relação ao total de docas disponíveis.

```
CREATE OR REPLACE FUNCTION fx_dock_occupancy(stationkid integer) RETURNS NUMERIC(3,2) AS $$
DECLARE
    total_docks INTEGER;
    occupied_docks INTEGER;
BEGIN
    -- Count active docks in the station
    SELECT COUNT(*) INTO total_docks
    FROM DOCK
    WHERE station = stationkid
        AND state IN ('free', 'occupy');

    -- Return occupancy percentage
    IF total_docks = 0 THEN
        RETURN 0.00;
    END IF;

    SELECT COUNT(*) INTO occupied_docks
    FROM DOCK
    WHERE station = stationkid
    AND state = 'occupy';

    RETURN ROUND(occupied_docks::NUMERIC / total_docks, 2);
END;
$$ LANGUAGE plpgsql;
```

Listagem 3 - Cálculo da ocupação das docas

5. Criação de VIEWS

As views são objetos de banco de dados que representam consultas armazenadas, permitindo uma visualização simplificada e segura dos dados complexos armazenados em múltiplas tabelas. Neste projeto, a criação de views facilitou o acesso e a manipulação das informações relevantes, sem comprometer a integridade ou a confidencialidade dos dados.

Uma das principais views criadas foi a RIDER, que consolida informações provenientes das tabelas CLIENT, PERSON e CARD. Essa view tem como objetivo fornecer uma visão integrada dos dados dos clientes, combinando atributos pessoais, informações de registro e detalhes dos cartões associados. A implementação segue a estrutura de uma consulta SQL que realiza junções entre as tabelas relacionadas extraindo dados pessoais do cliente da tabela PERSON, data de registro do cliente da tabela CLIENT e detalhes do cartão utilizado da tabela CARD.

Na sua implementação foi necessário recorrer a gatilhos para permitir que as operações de inserção e atualização fossem refletidas nas tabelas subjacentes, garantindo que a integridade dos dados seja mantida e que as regras de negócio sejam respeitadas.

```sql
CREATE OR REPLACE FUNCTION rider_insert()
RETURNS TRIGGER AS $$
DECLARE
    new_person_id INTEGER;
    new_card_id INTEGER;
BEGIN
    INSERT INTO PERSON (name, email, taxnumber)
        VALUES (NEW.name, NEW.email, NEW.taxnumber)
        RETURNING id INTO new_person_id;

    INSERT INTO CLIENT (person, dtregister)
        VALUES (new_person_id, NEW.dtregister);

    INSERT INTO CARD (client, credit, typeofcard)
        VALUES (new_person_id, NEW.credit, NEW.typeofcard)
        RETURNING id INTO new_card_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listagem 4 - Gatilho para Inserção

```sql
CREATE OR REPLACE FUNCTION rider_update()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE PERSON
    SET name = NEW.name, email = NEW.email, taxnumber = NEW.taxnumber
    WHERE id = OLD.id;

    UPDATE CLIENT
    SET dtregister = NEW.dtregister
    WHERE person = OLD.id;

    UPDATE CARD
    SET credit = NEW.credit, typeofcard = NEW.typeofcard
    WHERE client = OLD.cardid;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listagem 5 - Gatilho para Atualização

6. Desenvolvimento de Procedimento

O procedimento armazenado startTrip é responsável por orquestrar todas as operações necessárias quando um cliente inicia uma viagem. Incluíndo verificar se a trotinete associada à doca selecionada está disponível para uso, garantir que o cliente não tenha viagens ativas antes de inciar uma nova, atualizar o estado da trotinete da doca para refletir a saída da viagem e criar um novo registo na tabela TRAVEL para registrar o início da viagem com todas as informações necessárias.

O procedimento recebe como parâmetros o id da doca em destaque e do cliente que a utilizar.

```
CREATE OR REPLACE PROCEDURE startTrip(dockid integer, clientid  integer)
LANGUAGE plpgsql AS $$
DECLARE
    v_scooter INTEGER;
    v_station INTEGER;
    v_now TIMESTAMP := NOW();
BEGIN
    -- Check if the dock exists and is occupied
    SELECT scooter, station INTO v_scooter, v_station
    FROM DOCK
    WHERE number = dockid AND state = 'occupy'
    FOR UPDATE;

    IF v_scooter IS NULL  THEN
        RAISE EXCEPTION 'Dock % not found or not occupied', dockid;
    END IF;

    -- Update the dock state to 'free' and remove the scooter
    UPDATE DOCK
    SET state = 'free', scooter = NULL
    WHERE number = dockid;

    -- Insert a new travel record
    INSERT INTO TRAVEL (dinitial, client, scooter, stinitial, stfinal, comment, evaluation, dfinal)
    VALUES (v_now, clientid, v_scooter, v_station, NULL, NULL, NULL, NULL);
    -- stfinal, comment, evaluation, dfinal are set to NULL for the start of the trip
END;
$$;
```

Listagem 6 - Procedimento startTrip

7. Aplicação Java

A aplicação Java desenvolvida é a interface principal que permite a interação dos usuários com o sistema de gestão de compartilhamento de trotinetes elétricas. Esta aplicação foi projetada para ser intuitiva e eficiente, proporcionando uma experiência de usuário fluida e responsiva.

A aplicação implementa diversas funcionalidades essenciais para a gestão do sistema de partilha de trotinetes, tais como:

- Criação de um cliente (função createCostumer);
- Listar um cliente, mostrando o tipo de passe associado (função listCostumer);
- Listar as docas e a sua ocupação (função listDocks);
- Iniciar uma viagem (função startTrip);
- Colocar uma trotinete na doca (função parkScooter).

As funções todas foram realizadas em Java onde foi necessária a criação de classes em java para as tabelas criadas em SQL. Foram também usadas as funções criadas nos pontos anteriores. Devido ao tamanho das funções, as mesmas não serão apresentadas no relatório, podendo estas ser verificadas no ficheiro (App.java) dentro do projeto em anexo.

8. Conclusão

Ao longo do projeto enfrentamos certas dificuldades, especialmente relacionadas à configuração do ambiente de desenvolvimento e à gestão de dependências no arquivo pom.xml, que estava formatado para Maven.

Infelizmente, não conseguimos resolver completamente esses problemas, o que impactou a fluidez do desenvolvimento e a integração de algumas funcionalidades, não nos sendo possível testar o código desenvolvido.

Apesar dessas dificuldades, acreditamos que o código foi elaborado de forma precisa e eficiente para a aplicação proposta. Em caso de eventuais erros, estamos confiantes de que as correções podem ser realizadas de maneira rápida e clara, permitindo que o sistema atenda às necessidades dos usuários.