INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

LEIC & LMATE

Introdução aos Sistemas de Informação

Semestre de Inverno 2024-2025

Partilha de Trotinetas Eléctricas — Electric Scooter Sharing

Projecto
(Fase 2)

Ana Beire, Matilde Pato e Nuno Datia

Novembro, 2024

Planeamento — Planning

As datas importantes a recordar são:

- Lançamento do enunciado: **23 de Setembro 2024**
- Entrega intermédia (Fase 1): **16 de Outubro de 2024**
- Entrega intermédia (Fase 2): **18 de Dezembro de 2024**

[PT]

Cada entrega intermédia deve apresentar o relatório e código (se houver) referentes **exclusivamente** a essa fase. O relatório deve seguir obrigatoriamente um dos modelos fornecidos, sob pena de penalização. **Este deve ser conciso e apresentar a justificação de todas as decisões tomadas** (ver Critérios de Avaliação). A capa do relatório deve indicar a composição do grupo, a unidade curricular e a fase do projecto que relata. Caso tenha adendas e/ou correcções a fazer a modelos já entregues, deve indicá-las de forma explícita no relatório seguinte.

O pdf (e, o zip) gerado deve seguir o nome da seguinte forma: 'Proj**N**ISI-2425-**MM**.ext' (N representa um dígito correspondente ao número da fase do projecto, MM representa os dígitos do número do grupo, e 'ext' a extensão do ficheiro), e.g.: Proj1ISI-2425-01.pdf.

[EN]

Each intermediate delivery must present the report and code (if any) referring **exclusively** *to that phase. The report must obligatorily follow one of the templates provided, or penalties may ensue.* **This must be concise and present the justification for all decisions made** *(see Assessment Criteria). The report's cover must indicate the group composition, the curricular unit and the phase of the project being reported. If you have additions and corrections to deliver to models previously submitted, tell them explicitly in the following report.*

*The generated pdf (and zip) must follow the name as follows: 'Proj***N***ISI-2425-***MM***.ext' (N represents a digit corresponding to the project phase number, MM represents the digits of your group number, and 'ext' the file extension), e.g.: Proj1ISI-2425-01.pdf.*

14 de Novembro de 2024, Ana Beire, Matilde Pato e Nuno Datia

Objectivos de aprendizagem

No final da **segunda fase do projecto**, os alunos devem ser capazes de:

- Utilizar correctamente SQL/DDL para criar as tabelas num sistema de gestão de bases de dados (SGDB);
- Garantir as restrições de integridade identificadas enumerando aquelas que têm de ser garantidas pelas aplicações;
- Inserir dados em lote através da cláusula SQL `INSERT`, garantindo que as restrições de integridade são cumpridas;
- Garantir a atomicidade de instruções, utilizando processamento transaccional;
- Utilizar correctamente os operadores da teoria dos conjuntos em PostgreSQL;
- Utilizar correctamente as cláusulas `INNER JOIN` e `OUTER JOIN` (`LEFT` e/ou `RIGHT`);
- Utilizar correctamente sub-interrogações correlacionadas;
- Utilizar correctamente funções de agregação;
- Utilizar correctamente a cláusula `HAVING`, `ORDER BY`, `DISTINCT`, e os predicados `IN` e `EXISTS` (na pesquisa);
- Estabelecer uma ligação ao SGBD pretendido, correctamente parametrizada, utilizando JDBC;
- Utilizar correctamente comandos parametrizados para executar operações em JDBC (*prepared statement*);
- Utilizar correctamente transações para garantir atomicidade nas operações, utilizando JDBC;
- Gerir correctamente o tempo de vida das ligações JDBC;
- Garantir a libertação de recursos, quando estes não estejam a ser utilizados;
- Utilizar correctamente o tipo `ResultSet`;
- Implementar **todas as restrições de integridade** aplicacionais que não foram possíveis de definir de forma declarativa em PostgreSQL.

Após a realização da 1ª fase do projecto, segue-se a implementação do modelo físico do sistema, i.e. deverá ser construído em PostgreSQL contemplando todas as restrições que consigam garantir na forma declarativa. Os alunos terão agora a oportunidade de utilizar uma API JDBC para, através de uma aplicação Java, acederem e manipularem os dados existentes no modelo físico criado. Para a **criação** do modelo físico, deverão seguir todas as informações disponíveis no **final deste documento** ("Adenda"). Os nomes, tipos de dados e as restrições deverão ser implementadas conforme explicitadas neste documento.

As alterações/actualizações/consultas irão ser feitas a nível da aplicação e não directamente na base de dados (BD), i.e. **não deve fazer qualquer alteração à BD já construída**.

Sempre que se justificar devem ser usados os mecanismos transacionais necessários para garantir a atomicidade das operações. Também devem utilizar mecanismos que evitem ataques de "*SQL injection*", bem como que evitar problemas de formatações, e.g. campos de datas.

Nota: Deverão preencher a BD com a informação necessária que permita, em seguida, realizar interrogações que apresentem resultados pertinentes. Na etapa de preenchimento da BD, os alunos deverão ter particular atenção ao cumprimento das **restrições de integridade, utilizando de forma adequada o controlo transaccional (a atomicidade)**.

Resultados pretendidos

Tendo em conta os objectivos de aprendizagem, deverão ser produzidos os seguintes resultados:

1. Construção do **modelo físico** do sistema (disponível no final deste documento, "Adenda"), contemplando todas as restrições de integridade passíveis de ser garantidas declarativamente, assim como a **atomicidade** nas operações. O código PostgreSQL que permite:

   (a) Criar o modelo físico (1 *script* autónomo): "`createTable.sql`";

   (b) Prencher a BD com valores (1 *script* autónomo): "`insertData.sql`". Os dados introduzidos devem permitir validar todas as interrogações pedidas nesta fase do projecto;

   (c) Apagar todos os dados existentes (1 *script* autónomo): "`deleteData.sql`".

   (d) Remover todas as estruturas de dados criadas (1 *script* autónomo): "`removeData.sql`".

2. Uma **aplicação Java** (executado independente do ambiente de desenvolvimento), como ilustrado na Figura 1, que permita realizar as seguintes operações :

   (a) Opção para introduzir novos clientes na app. Os valores a inserir deverão preencher todas as tabelas onde consta informação sobre os mesmos;

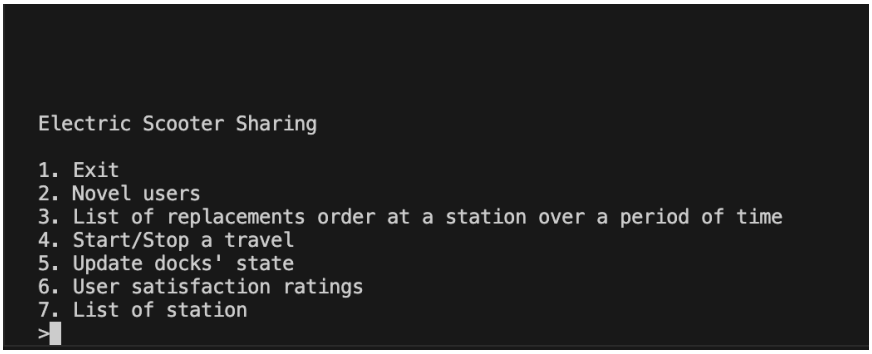

Figura 1: Screenshot da aplicação Partilha de Trotinetas Eléctricas

(b) Opção para listar as ordens de reposição numa dada estação, num período de tempo. Os valores a inserir pelo utilizador deverão ser (não necessáriamente por esta ordem): a estação, a data de início e de fim da pesquisa;

(c) Opção para iniciar/parar uma viagem. O utilizador da app poderá optar por (1) iniciar (START) ou (2) parar (STOP) a viagem. Depois, este deverá inserir todos os restantes campos que considere necessários. Deverão assegurar que (1) uma viagem tem início, se a trotineta está disponível na estação indicada pelo utilizador, e se o cliente não está a realizar uma viagem nesse momento; (2) é debitado no saldo do passe o custo do desbloqueio. Depois de concluída a viagem, é debitado o custo por minuto de utilização no saldo do passe;

(d) Opção para actualizar o estado das docas. Após o início/fim de uma viagem, o estado das docas não fica automaticamente actualizado. Outra situação, tem haver com a colocação/remoção de novas trotinetas numa dada doca pelo empregado, tornando-as disponíveis/indisponíveis. Esta opção irá assegurar essa actualização;

(e) Opção para analisar o grau de satisfação do utilzador para cada modelo de trotineta. O resultado, deve apresentar a média do *rating*, o número total de viagens, e a percentagem do grau de satisfação (*ratings* $\geqslant 4$), apresentando por odem descrescente da média do *rating*;

(f) Opção para listar as três estações com maior taxa de ocupação.

Todas as instruções devem vir indicadas (e, explicadas) no "relatório" que dá suporte a este trabalho. Se considerar necessário, pode a título ilustrativo, mostrar tabela(s) de resultados.

3. Recomendações:

   - Devem garantir **atomicidade** nas escrita (inserções, actualizações e remoções);
   - Todas as simplificações e optimizações realizadas ao modelo devem ser indicadas e justificadas;
   - Sugere-se que consulte o manual do SGDB para obter informação sobre as funções de **manipulação de datas** (`https://www.postgresql.org`);

- Na resolução não poderá recorrer aos operadores: `function`, `procedure`, `trigger`, `with`, `limit`, `cross join`;
- Deve garantir a **correcta** implementação de todas as funcionalidades, incluindo o acesso a dados;
- Deve criar em Java um modelo de dados que mapeie as relações utilizadas para **objectos em memória**;
- A lógica de interface com o utilizador deve estar em **classes separadas** da lógica de acesso a dados;
- Deve ser possível aferir cada um dos objectivos de aprendizagem no material que entregar.

Todo o código entregue tem de ser executado independente do ambiente de desenvolvimento, em linha de comandos. Os alunos têm de fornecer as instruções de execução, assumindo como único pré-requisito a existência da **máquina virtual Java 17**[1].

Data limite para entrega: 18 de Dezembro de 2024 até às 23:59.

A entrega deve incluir um relatório (em formato **`PDF`**, devidamente identificado, com o número do grupo, os números de aluno e respectivos nomes), os scripts de PostgreSQL (com extensão .sql), bem como os scripts da API (com extensão .java). Os ficheiros produzidos deverão ser comprimidos, num único arquivo `ZIP` ou `TAR.GZ`) e enviados de forma electrónica através do Moodle.

Nota: Deve ser possível aferir cada um dos objectivos de aprendizagem no material que entregar.

[1] Versões superiores serão penalizadas.

Learning Objectives

At the end of **the second step** of the project, students should be able to:

- Use SQL/DDL effectively to create tables in a database management system (DBMS);
- Ensure all the integrity restrictions identified by listing those that must be guaranteed by the applications;
- Insert batch data using the SQL `INSERT` statement, ensuring all integrity constraints are met;
- Ensure atomicity of operations by using transaction processing;
- Use set theory operators correctly in PostgreSQL;
- Correctly apply `INNER JOIN` and `OUTER JOIN` (`LEFT` and/or `RIGHT`) clauses;
- Use correlated subqueries correctly;
- Apply aggregate functions properly;
- Use the `HAVING`, `ORDER BY`, `DISTINCT` clauses, and `IN` and `EXISTS` predicates correctly;
- Establish a properly parametrized connection to the desired DBMS using JDBC;
- Correctly use parametrized commands to execute operations in JDBC ("prepared statements");
- Use transactions effectively to ensure atomicity in operations with JDBC;
- Manage the lifecycle of JDBC connections properly;
- Ensure resources are released when no longer needed;
- Use the `ResultSet` type correctly;
- Implement all application-level integrity constraints that PostgreSQL cannot enforce in a declarative manner.

After completing the first phase of the project, the next step is to implement the physical model of the system, e.g. it should be built in PostgreSQL, incorporating all the constraints that can be guaranteed in a declarative manner. Students will now have the opportunity to use a JDBC API to access and manipulate the data in the physical model created through a Java application.

For the creation of the **physical model**, all information available at the **end of this document** (Appendix) should be followed. The names, data types, and constraints should be implemented as outlined in this document.

The changes/updates/queries will be made at the application level, not directly in the database (DB), i.e., **no changes should be made to the already built DB**.

Whenever necessary, the appropriate transactional mechanisms should be used to ensure the atomicity of operations. Mechanisms to prevent "SQL injection" attacks should also be used, as well as those to avoid formatting issues, such as date fields.

Note: You should populate the DB with the necessary information that will later allow you to perform queries that return relevant results. During the DB population step, students should pay particular attention to adhering to **integrity constraints, using transactional control (atomicity) appropriately**.

Expected results

Taking into account the learning objectives, the following results should be produced:

1. Development of the system's physical model (available at the end of this document, Appendix), taking into account all integrity restrictions that can be guaranteed in a declarative way, as well as **atomicity** in operations. The PostgreSQL code that allows:
   (a) Create the physical model (1 autonomous *script*): "`createTable.sql`";
   (b) Populate the physical model: "`insertData.sql`". The data must allow validating all queries requested at this stage of the work;
   (c) Create a SQL script to erase all data in the tables (1 autonomous *script*): "`deleteData.sql`";
   (d) Remove all data structures (1 autonomous *script*): "`removeData.sql`".
2. A Java application (which runs independently of the development environment) as depicted in Figura 1 enables users to carry out the following operations **(it must not make any changes to the already established database)**:

(a) Add customers: Option to enter new customers into the app. All tables containing information about them must be filled in with the values to be entered;

(b) View replacement orders: Provide an option to list replacement orders for a specified station over a chosen date range. The user should input the station, start date, and end date (in any order);

(c) Start/Stop journey: Enable users to start (START) or stop (STOP) a journey. Users should input any additional necessary details and ensure that: (1) the journey has started; (2) the selected scooter is available at the specified station, and (3) the customer is not already on a journey. Also, when a journey starts, charge the unlocking fee, and when the journey ends, deduct the cost per minute from the pass balance;

(d) Update dock status: Add an option to update dock status after a journey starts or ends, as this does not update automatically. Also, employees can mark scooters as available/unavailable when placing or removing them at a dock;

(e) Analyse user satisfaction ratings for each scooter model showing. The results must be the average rating per model, the number of total trips, and the percentage of high satisfaction trips (ratings $\geqslant 4$). Sort results by the average rating in descending order. This analysis will help identify which scooter models are performing best in terms of user satisfaction;

(f) Top 3 stations by occupancy: Create an option to list the three stations with the highest occupancy rates.

Just so you know, all simplifications and optimisations made to the model must be indicated and justified. If you consider it necessary, you can show table(s) of results for illustrative purposes.

3. Recommendations:

- You must guarantee **atomicity** during updates;
- You'll find all the info you need on date manipulation in the SGDB manual (`https://www.postgresql.org`);
- You can't use the operators: `function`, `procedure`, `trigger`, `with`, `limit`, and `cross join` in the resolution.
- You must guarantee the correct implementation of all functionalities, including data access;
- You must create a data model in Java that maps the relationships used to objects in memory;
- The user interface logic should be in separated classes from the data access logic;
- It should be possible to assess each learning objective in the material delivered.

All the code delivered must be executed independently of the development environment, on the command line. Students must provide the execution instructions, with the only prerequisite being the existence of the **Java 17 virtual machine**[2].

Deadline 18 de Dezembro de 2024 until 23:59.

The delivery must include a report (in **PDF** format, duly labelled with the group number, student numbers, and their names), the PostgreSQL scripts (with .sql extension), as well as the API scripts (with .java extension). The files produced should be compressed into a single archive (`ZIP` or `TAR.GZ`) and sent electronically via Moodle.

Note: It should be possible to assess each of the learning objectives in the material you hand in.

[2]Higher versions will be penalised

Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

LEIC — LMATE

Modelo Relacional a usar no projecto (fase 2)
Relational Model to be used in the second assignment (2nd phase)

Introdução a Sistemas de Informação

Semestre de Inverno 2024/2025

Versão 1.1

Modelo Relacional

PT:

Todos os atributos são obrigatórios nas relações, excepto quando indicado o contrário. As relações são apresentadas por ordem alfabética. **NOTA:** O modelo apresentado não representa exactamente uma possível solução para a Fase I. Foram feitas algumas alterações (devidamente identificadas). As restrições que não conseguirem implementar no modelo físico serão asseguradas no desenvolvimento da aplicação. Os nomes e tipos de atributos devem seguir o que está descrito neste documento.

EN:

All attributes are mandatory in relationships unless otherwise indicated. The relations are presented in alphabetical order. **Note:** The model presented does not accurately reflect a potential solution for Phase I. Some changes have been made, as identified in the report. The restrictions you cannot implement in the physical model will be ensured in the application. The name and attributes type must remain as described in the document.

CARD

CARD(id, credit, typeof, client).

Attribute	TypeOf	Restrictions
id	serial	Must be a sequence number.
credit	numeric(4,2)	The value must be positive.
typeof	char(10)	FK references of `TYPEOF.{reference}`
client	integer	FK references of `CLIENT.{person}`

The acquisition date is removed because we assume is the same as the user's date register. The credit corresponds to the card balance, it must be updated whenever requested.

CLIENT

CLIENT(person, dtregister).

Attribute	TypeOf	Restrictions
person	integer	FK references of `PERSON.{id}`.

continued on the next page

Attribute	TypeOf	Restrictions
dtregister	timestamp	This date shows when the client was registered. It has the format "yyyy-mm-dd hh:mm:ss".

DOCK

DOCK(number, station, state, scooter).

Attribute	TypeOf	Restrictions
number	serial	Must be a sequence number.
station	integer	FK references of `STATION.{id}`.
state	varchar(30)	You can take only one of the following values: *"free"*, *"occupy"*, *"under maintenance"*.
scooter	integer	FK references of `SCOOTER.{id}`. Can be NULL. The state cannot be occupy if this field is NULL.

EMPLOYEE

EMPLOYEE(number, person).

Attribute	TypeOf	Restrictions
number	serial	Must be a sequence number. Alternate Key.
person	integer	FK references of `PERSON.{id}`.

PERSON

PERSON(id, email, taxnumber, name).

Attribute	TypeOf	Restrictions
id	serial	Must be a sequence number. This attribute was added to the DB to make it simpler. All persons should be CLIENT and/or EMPLOYEE.
email	varchar(40)	Must include "@". Alternate key.
taxnumber	integer	tax ID number. Alternate key.

continued on the next page

Attribute	TypeOf	Restrictions
name	varchar(50)	

REPLACEMENT

REPLACEMENT(number, dtreplacement, action, dtreporder, repstation, employee).

Attribute	TypeOf	Restrictions
number	serial	Sequential number for each replacement.
dtreplacement	timestamp	Corresponds to the replacement' date. It has the format "yyyy-mm-dd hh:mm:ss". This value must be higher than `dtreporder`.
action	char(8)	The attribute can take one of the values in the set {*"inplace"*, *"remove"*}.
dtreporder	timestamp	Corresponds to the replacement request date. It has the format "yyyy-mm-dd hh:mm:ss". Part of the FK referencing `REPLACEMENTORDER.{dtorder}`.
repstation	integer	Part of the FK referencing `REPLACEMENTORDER.{station}`.
employee	integer	FK references of `EMPLOYEE.{person}`.

REPLACEMENTORDER

REPLACEMENTORDER(dtorder, dtreplacement, roccupation, station).

Attribute	TypeOf	Restrictions
dtorder	timestamp	Corresponds to the replacement request date. It has the format "yyyy-mm-dd hh:mm:ss".
dtreplacement	timestamp	Corresponds to the replacement' date. It has the format "yyyy-mm-dd hh:mm:ss". This value must be higher than `dtorder`. Can be NULL.
roccupation	integer	Corresponds to the occupation rate. The attribute has integer values in the range [0 . . . 100], representing a percentage.
station	integer	FK references of `STATION.{id}`.

SCOOTER

SCOOTER(id, weight, maxvelocity, battery, model).

Attribute	TypeOf	Restrictions
id	serial	Must be a sequence.
weight	numeric(4,2)	In grams. Must be positive.
maxvelocity	numeric(4,2)	In Km/h. Must be positive.
battery	integer	In Km. Must be positive. The value should be equal or lower than model's autonomy.
model	integer	FK references of SCOOTERMODEL.{number}

SCOOTERMODEL

SCOOTERMODEL(number, designation, autonomy).

Attribute	TypeOf	Restrictions
number	serial	Must be a sequence. This attribute was added to the DB to make it simpler.
designation	varchar(30)	
autonomy	integer	In Km. Must be positive.

STATION

STATION(id, latitude, longitude).

Attribute	TypeOf	Restrictions
id	serial	Must be a sequential number.
latitude	numeric(6,4)	Corresponds to the latitude in a GPS coordinate in decimal degree.
longitude	numeric(6,4)	Corresponds to the longitude in a GPS coordinate in decimal degree.

TOPUP

TOPUP(dttopup, card, value).

Attribute	TypeOf	Restrictions
dttopup	timestamp	This date shows when the pass was loaded. It has the format "yyyy-mm-dd hh:mm:ss".
card	integer	FK references of `CARD.{id}`.
value	numeric(4,2)	Corresponds to the card's euro value. This value must be positive.

TRAVEL

TRAVEL(dtinitial, comment, evaluation, dtfinal, client, scooter, stinitial, stfinal).

Attribute	TypeOf	Restrictions
dtinitial	timestamp	This date shows when the travel was started. It has the format "yyyy-mm-dd hh:mm:ss".
comment	varchar(100)	Can be NULL. Cannot have a value without evaluation.
evaluation	integer	The attribute takes integer values in the interval [1...5]. Can be NULL.
dtfinal	timestamp	This date shows when the travel is concluded. It must be higher than `dtinitial`. It has the format "yyyy-mm-dd hh:mm:ss". Can be NULL.
client	integer	FK references of `CLIENT.{person}`.
scooter	integer	FK references of `SCOOTER.{id}`. Part of an alternate key.
stinitial	integer	The value represents the departure station for the trip. FK references of `STATION.{id}`.
stfinal	integer	The value represents the arrival station for the trip. This value has been added for storing the destination station of a trip. FK references of `STATION.{id}`.

TYPEOF

TYPEOF(reference, nodays, price).

Attribute	TypeOf	Restrictions
reference	char(10)	Could be "resident" or "tourist". Alternate key.

continued on the next page

Attribute	TypeOf	Restrictions
nodays	integer	Must be positive.
price	numeric(4,2)	Corresponds to the price of the type of card, in euros. This value must be positive.

SERVICECOST

SERVICECOST(unlock, usable).

Attribute	TypeOf	Restrictions
unlock	numeric(3,2)	Corresponds the value for unlocking the scooter (equal to 1.00€).
usable	numeric(3,2)	Corresponds the value of use per minute of the scooter (equal to 0.15€).

This table was added, to the system, to calculate the cost of the sharing service. It has two attributes with one value.

22/11/2024, Ana Beire, Matilde Pato e Nuno Datia