Base de Dados:

O primeiro exercício do projeto divide-se em duas partes, o esquema e o carregamento de dados.

No esquema, começa-se por executar os comandos *drop Tablet* para dar *reset* às tabelas. De seguida, são criadas todas as tabelas de acordo com o Esquema A fornecido no final do enunciado.

No carregamento de dados, insere-se, manualmente, dados para preencher as tabelas da base de dados.

O código desta secção encontra-se no ficheiro populate.sql.

Restrições de integridade:

Neste ponto, pede-se para implementar três restrições. Foi feito um *trigger* para cada uma das restrições.

Para a primeira restrição, nomeadamente, uma Categoria não poder estar contida em si própria, efetuou-se um *trigger* onde, antes de inserir na tabela *tem_outra*, verifica se o nome da super categoria é diferente da categoria.

```
CREATE OR REPLACE FUNCTION doesnt_contain_itself_proc()
RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.nome_cat == NEW.nome_cat_super THEN
        RAISE EXCEPTION 'Uma categoria não pode estar contida em si pró-
pria'
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER doesnt_contain_itself_trigger
BEFORE INSERT ON tem_outra
FOR EACH ROW EXECUTE PROCEDURE doesnt_contain_itself_proc();
```

Para a segunda restrição, nomeadamente, o número de unidades repostas num Evento de Reposição não poder exceder o número de unidades especificado no Planograma, efetuou-se um *trigger* onde, antes de inserir na tabela *evento_reposição*, verifica se o número de unidades repostas no planograma é excedida.

```
CREATE OR REPLACE FUNCTION check_units_proc()
```

```
DECLARE uns NUMBER;
BEGIN
    SELECT unidades INTO uns
    FROM planograma
    WHERE ean := NEW.ean AND nro := NEW.nro;
    IF NEW.unidades > uns THEN
        RAISE EXCEPTION 'Excedeu o limite de unidades'
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_units_trigger
BEFORE INSERT ON evento_reposicao
FOR EACH ROW EXECUTE PROCEDURE check_units_proc();
```

Para a terceira restrição, nomeadamente, um Produto só poder ser reposto numa Prateleira que apresente (pelo menos) uma das Categorias desse produto, efetuou-se um *trigger* onde, antes de inserir na tabela *planograma*, verifica se, para a respetiva categoria de cada produto, faz-se contagem dessa categoria na prateleira e se for igual a 0, levanta o erro.

```
CREATE OR REPLACE FUNCTION is_valid_proc()
DECLARE cat_prod varchar(255), cat_prat NUMBER;
BEGIN
    SELECT categoria INTO cat_prod
    FROM produto
    WHERE ean := NEW.ean;
    SELECT COUNT(*) INTO cat_prat
    FROM prateleira
    WHERE nro := NEW.nro AND categoria := cat_prod;
    IF cat_prat == 0 THEN
        RAISE EXCEPTION 'Não existe categoria presente na prateleira'
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER is_valid_proc()
BEFORE INSERT ON planograma
FOR EACH ROW EXECUTE PROCEDURE is_valid_proc();
```

SQL:

Nesta secção, pede-se para fazer quatro consultas.

Na primeira, nome do retalhista (ou retalhistas) responsáveis pela reposição do maior número de categorias, para cada retalhista vamos fazer a contagem de categorias distintas e devolvemos o nome do(s) retalhista(s) que apresentarem a maior contagem.

```
SELECT nome, COUNT(DISTINCT nome_cat)
FROM retalhista NATURAL JOIN responsavel_por
GROUP BY tin
HAVING COUNT(DISTINCT nome_cat) >= ALL (
    SELECT COUNT(DISTINCT nome_cat)
    FROM retalhista NATURAL JOIN responsavel_por
    GROUP BY tin
);
```

Na segunda, o nome do ou dos retalhistas que são responsáveis por todas as categorias simples, faz-se a contagem de categorias na tabela *responsável_por* para cada retalhista e se for igual à contagem de categorias na tabela *categoria_simples* significa que o retalhista reposiciona todas as categorias simples.

```
SELECT nome
FROM retalhista NATURAL JOIN responsavel_por
GROUP BY nome
HAVING COUNT (DISTINCT (nome_cat)) = (SELECT COUNT(nome_cat)
FROM categoria_simples);
```

Na terceira, os produtos (ean) que nunca foram repostos, realiza-se uma simples divisão com *Except* entre a coluna *ean* de *produto* e a coluna ean de *evento_reposiçao*.

```
SELECT ean
FROM produto
EXCEPT
SELECT ean
FROM evento_reposicao;
```

Na última, os produtos (ean) que foram repostos sempre pelo mesmo retalhista, para cada produto verifica-se se apenas têm um *tin* associado, através do *COUNT(DISTINCT tin) = 1*.

```
SELECT ean
FROM evento_reposicao
GROUP BY ean
HAVING COUNT(DISTINCT tin) = 1;
```

VISTAS:

Nesta secção, é-nos pedido para criar uma vista sobre vendas associada a um evento de reposição. Para fazer a vista, junta-se as tabelas *planograma, produto, categoria, IVM, ponto_de_retalho e evento_reposicao*, tendo, assim, acesso a todos os atributos pedidos. Para aceder aos atributos do instante, é utilizada a função EXTRACT.

```
CREATE OR REPLACE VIEW vendas
AS
SELECT ean, nome_cat, EXTRACT(YEAR FROM instante) AS ano,
EXTRACT(QUARTER FROM instante) AS trimestre, EXTRACT(MONTH FROM instante)
AS mes,
EXTRACT(DAY FROM instante) AS dia_mes, EXTRACT(ISODOW FROM instante) AS
dia_semana,
distrito, concelho, unidades
FROM planograma NATURAL JOIN produto
NATURAL JOIN categoria
NATURAL JOIN IVM
NATURAL JOIN ponto_de_retalho
NATURAL JOIN evento_reposicao
```

Desenvolvimento da aplicação:

Link para o website: http://web2.ist.utl.pt/ist199303/test.cgi/

O website é composto por um menu principal onde se pode selecionar 4 opções, que correspondem às funcionalidades pedidas no enunciado. As primeiras duas foram implementadas de modo semelhante e tirando inspiração do Lab 9: começa com uma página que mostra uma tabela com as categorias / retalhistas com opções para as remover individualmente da base de dados ou para adicionar novas categorias / retalhistas. Para remoção, eliminamos primeiro as instâncias das relações a que estas pertencem, assim como identidades que têm como atributo as categorias / retalhistas e, por fim, as categorias / retalhistas em si.

As duas últimas funcionalidades também foram implementadas de maneira semelhante: Escolhe-se a IVM / super categoria para ser listada e no caso da 3ª funcionalidade, é executada uma query que apresenta o ean e as quantidades (agrupada pelo ean) e no caso da 4ª é feita uma queue que guarda as sub-categorias e as sub-categorias dessas, executa-as e guarda num cursor que as compila todas para uma única tabela que é apresentada na página que aparece.

Consultas olap:

Na sexta pergunta, pede-se para analisar os artigos vendidos através da vista desenvolvida no tópico (4). Para esta análise foi utilizada a instrução Cube.

Na primeira análise, num dado período (i.e. entre duas datas), por dia da semana, por concelho e no total, verifica se o ano, mês e dia da venda está entre um certo período, organizando, depois, a instrução *cube* com os vértices *dia_semana* e *concelho*.

```
SELECT dia_semana, concelho, SUM(unidades) AS nmr_artigos_vendidos
FROM vendas AS v
WHERE v.ano = 2022 AND
v.mes BETWEEN 1 AND 5 AND
dia_mes BETWEEN 1 AND 31
GROUP BY CUBE (dia_semana, concelho)
ORDER BY dia_semana, concelho, nmr_artigos_vendidos DESC;
```

Na segunda análise, num dado distrito (i.e. “Lisboa”), por concelho, categoria, dia da semana e no total, verifica se o distrito é igual a ‘Lisboa’, organizando, depois, a instrução *cube* com os vértices *concelho, nome_cat* e *dia_semana*.

```
SELECT concelho, nome_cat, dia_semana, SUM(unidades) AS nmr_artigos_ven-
didos
FROM vendas AS v
WHERE v.distrito = 'Lisboa'
GROUP BY CUBE (concelho, nome_cat, dia_semana)
ORDER BY concelho, nome_cat, dia_semana, nmr_artigos_vendidos DESC;
```

Índices:

7.1)

CREATE INDEX melhora_responsavel_por ON responsável_por(nome_cat)

Este índice deverá ser um índice *Hash* uma vez que a filtragem se trata de uma igualdade. Este índice irá agilizar a execução do atributo nome_cat da tabela *responsavel_por*(P), e, consequentemente, será desnecessário fazer outro índice para P.tin.

7.2)

CREATE INDEX melhora_produto ON produto(desc)

Este índice deverá ser um índice *Hash* uma vez que a filtragem se trata de uma igualdade. Este índice irá agilizar a execução do atributo desc da tabela *produto*(P), e, consequentemente, será desnecessário fazer outro índice para P.cat.

CREATE INDEX melhora_tem_categoria ON tem_categoria(nome, ean)

Este índice deverá ser um índice *Hash* uma vez que a filtragem se trata de uma ordenação. Este índice irá agilizar a execução dos atributos nome e ean da tabela *tem_categoria*(T).