Sistema de Informação para Empresa de Madeiras e Cerâmicas

3ª Entrega

2MIEIC06 - Grupo 602

Bases de Dados 2017/2018

Mestrado Integrado em Engenharia Informática e Computação

Beatriz de Henriques Martins - up201502858@fe.up.pt
Luisa Zambelli Artmann Rangel - up201710784@fe.up.pt
Vitor Miguel Medeiros Cordeiro - up201505087@fe.up.pt

27 de abril de 2018

Conteúdo

1 Descrição

O objectivo de uma *empresa* especializada em construir cozinhas é fabricar, montar e instalar.

A *empresa* mantém um registo dos seus clientes. É necessário armazenar alguns dados dos seus *clientes* como o nome, a morada, o telefone, o e-mail e o NIF (Número de Identificação Fiscal) para a faturação e uma lista de compras anteriores. Os *clientes* são *pessoas*.

Para continuar o trabalho exemplar que os seus *clientes* apreciam, a *empresa* precisa de *funcionários* especializados nas mais diversas áreas, como por exemplo marceneiros, vidreiros, pedreiros, designers, vendedores, administrativos entre tantas outras especialidades. Sobre estes, além da *especialidade*, é necessário guardar outros dados, como por exemplo, o cargo atual, o ordenado, o horário de trabalho e o agregado familiar (que representa o número de dependentes do trabalhador). Todos os que trabalham na *empresa* são *pessoas*, logo as informações básicas necessárias que esta necessita são iguais às do *cliente*. A empresa tem dois tipos de *estabelecimento*, a *fábrica* e a *loja*, cada um deles tem dados próprios, como morada, telefone, e-mail e um grupo de funcionários diversificados adaptados às necessidades de cada estabelecimento.

Associada a cada *fábrica* existe também uma lista de *fornecedores* (empresas externas que fornecem materiais). Estes partilham dados como o nome, a morada, o telefone, o e-mail e o NIF. A *empresa* guarda também informação sobre os *materiais* fornecidos por cada *fornecedor*. Sobre os *materiais* precisamos de saber o nome, o tipo, o preço, a cor e a quantidade necessária para cada produto.

A partir dos *materiais*, a *empresa* consegue fabricar seus *produtos*, as cozinhas. Cada *produto* exige *funcionários* especializados para sua construção. Cada um tem um conjunto de dados específico como o preço, a cor e as dimensões deste.

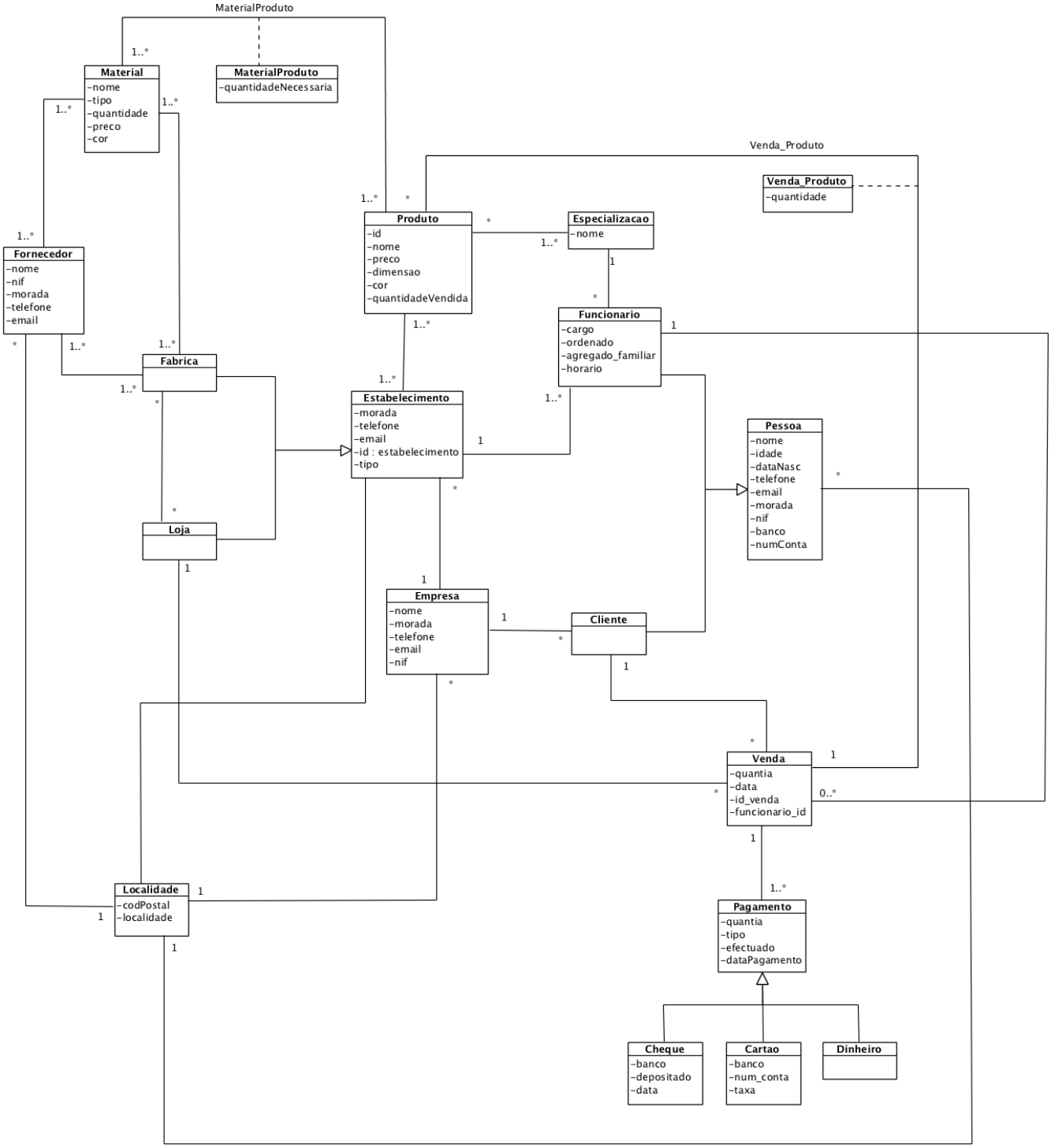
Na ocorrência de uma *venda*, que relaciona um *cliente* a um ou mais *produtos*, deve-se registar a data, a quantia final e a *loja* em que ocorreu. Sobre o *pagamento* temos de saber o estado em que se encontra e se foi realizado através de um cartão, um cheque ou dinheiro. Dependendo da forma de *pagamento*, precisamos de armazenar os dados bancários e tarifas aplicadas.

2 Classe e Atributos

Classe	Atributos
Empresa	nif nome morada telefone email empresa_codPostal
Cliente	cliente_nif empresa_nif
Venda	quantia dataVenda cliente_id funcionario_id estabelecimento_id
Dinheiro	
Cheque	banco depositado dataDeposito
Estabelecimento	morada telefone tipo email empresa_nif codPostal
Fabrica	
Especialização	nome

Classe	Atributos
Pessoa	nif nome morada telefone email dataNasc idade banco numConta codPostal
Funcionario	funcionario_nif cargo ordenado agregadoFamiliar horario especializacao_id estabelecimento_id
Pagamento	quantia tipo efetuado
Cartao	banco numConta taxa
Localidade	codPostal localidade
Loja	
Fornecedor	nif nome morada telefone email codPostal

3 Diagrama de UML



4 Modelo Relacional e Dependências Funcionais

Pessoa(nif, nome, morada, telefone, email, dataNasc, banco, numConta, codPostal->*Localidade*)

- nif → nome, morada, telefone, email, dataNasc, banco, numConta, codPostal

Cliente(nif->*Pessoa*, nif->*Empresa*)

- nif → *Pessoa*, *Empresa*

Funcionario(nif->*Pessoa*, cargo, ordenado, agregadoFamiliar, horario, idEstabelecimento->*Estabelecimento*, idEspecializacao->*Especializacao*)

- nif → *Pessoa*, cargo, ordenado, agregadoFamiliar, horario, idEstabelecimento, nomeEspecializacao

Especializacao(idEspecializacao, Nome)

Produto(idProduto, nome, preco, dimensao, cor)

- idProduto → nome, preco, dimensao, cor

Estabelecimento(idEstabelecimento, morada, telefone, email, tipo, nomeEmpresa->*Empresa*, codPostal->*Localidade*)

- idEstabelecimento → morada, telefone, email, tipo, nomeEmpresa, codPostal

Fabrica(idEstabelecimento->*Estabelecimento*)

Loja(idEstabelecimento->*Estabelecimento*)

Empresa(nif, nome, morada, telefone, email, codPostal->*Localidade*)

- nif → nome, morada, telefone, email

Material(idMaterial, nome, tipo, quantidade, preco, cor)

- idMaterial → nome, tipo, quantidade, preco, cor

Fornecedor(nif, nome, morada, telefone, email, codPostal->*Localidade*)

- nif → nome, morada, telefone, email, codPostal

Localidade(codPostal, localidade)

- codPostal → localidade

Venda(idVenda, quantia, data, idCliente->*Cliente*, idEstabelecimento->*Loja*)

- idVenda → quantia, data, idCliente, idEstabelecimento

Pagamento(idPagamento, idVenda->*Venda*, quantia, tipo, efetuado, dataPagamento)

- idPagamento → idVenda, quantia, tipo, efetuado, dataPagamento

Cheque(idPagamento->*Pagamento*, banco, depositado, dataDeposito)

- idPagamento → *Pagamento*, banco, depositado, dataDeposito

Cartao(idPagamento->*Pagamento*, banco, numConta, taxa)

- idPagamento → *Pagamento*, banco, numConta, taxa

Dinheiro(idPagamento->*Pagamento*)

- idPagamento → *Pagamento*

Material_Produto(idMaterial->*Material*, idProduto->*Produto*, quantidadeNecessaria)

- idMaterial, idProduto → *Material*, *Produto*, quantidadeNecessaria

Material_Forcedor(idMaterial->*Material*, idFornecedor->*Fornecedor*)

- idMaterial, idFornecedor → *Material*, *Fornecedor*

Material_Fabrica(idMaterial->*Material*, idEstabelecimento->*Fabrica*)

- idMaterial, idEstabelecimento → *Material*, *Fabrica*

Fornecedor_Fabrica(idFornecedor->*Fornecedor*, idEstabelecimento->*Fabrica*)

- idFornecedor, idEstabelecimento → *Fornecedor*, *Fabrica*

Loja_Fabrica(idEstabelecimento->*Loja*, idEstabelecimento->*Fabrica*)

- idEstabelecimento, idEstabelecimento → *Loja*, *Fabrica*

Estabelecimento_Produto(idProduto->*Produto*, idEstabelecimento->*Estabelecimento*)

- idProduto, idEstabelecimento → *Produto*, *Estabelecimento*

Produto_Especialização(idProduto->*Produto*, idEspecialização->*Especializacao*)

- idProduto, idEspecialização → *Produto*, *Especializacao*

5 Formas Normais

Para garantir a integridade de um banco de dados e evitarmos as repetições é importante a normalização. Uma relação pode se encontrar em diferentes formas de normalização, sendo a *Terceira Forma Normal* e a *Forma Normal de Boyce-Codd* as principais e mais utilizadas.

Para uma relação estar na *Terceira Forma Normal* (3FN) é necessário atender primeiramente a *Primeira* (1FN) e a *Segunda* (2FN) Formas Normais. A 1FN consiste em uma tabela apresentar apenas atributos atômicos, fato que é conferido em todas as relações do modelo relacional apresentado. As relações também atendem à 2FN pois além de atenderem a *Primeira Forma Normal*, nenhum atributo não primo é funcionalmente dependente de algum subconjunto de uma chave candidata. Um atributo pertencente a alguma chave é denominado primo.

Além disso, para estar na 3FN, para cada dependência funcional não trivial, o lado esquerdo deve ser uma super chave ou o lado direito consistir somente em atributos primos. Em outras palavras, todos os atributos não chave são completamente dependentes dos atributos chave e são independentes entre si.

A Forma Normal de Boyce-Codd (BCNF) é ainda um pouco mais restrita que a 3FN e garante a não existência de anomalias. As relações estão na BCNF pois todos os atributos são dependentes exclusivamente da chave primária. Dessa forma, também não apresentam nenhuma violação à Terceira Forma Normal.

6 Restrições

De acordo com o modelo de negócio, temos as seguintes restrições:

```
CREATE TABLE Especializacao (
    id INTEGER PRIMARY KEY,
    nome TEXT NOT NULL
);
```

- Cada especialização tem um número de identificação (id) único e não nulo;
- O nome da especialização não pode ser nulo;

```
CREATE TABLE Localidade (
    codPostal INTEGER PRIMARY KEY,
    localidade TEXT NOT NULL
);
```

- O código postal (CodPostal) não pode ser nulo;
- A localidade (Localidade) não pode ser nulo;

```
CREATE TABLE Material (
    id INTEGER PRIMARY KEY,
    nome TEXT NOT NULL,
    tipo TEXT NOT NULL,
    quantidade FLOAT NOT NULL,
    preco FLOAT,
    cor TEXT
);
```

- Cada material tem um número de identificação (idMaterial) único e não nulo;
- O nome do material (nome) não pode ser nulo;
- O tipo do material (tipo) não pode ser nulo;
- O stock do material (quantidade) não pode ser nulo;

```
CREATE TABLE Pessoa (
    nif INTEGER PRIMARY KEY,
    nome TEXT NOT NULL,
    morada TEXT,
    telefone TEXT NOT NULL,
    email TEXT NOT NULL,
    dataNasc DATE,
    idade INT,
    banco TEXT NOT NULL,
    numConta INT NOT NULL,
    codPostal INT,
    FOREIGN KEY (codPostal) REFERENCES Localidade(codPostal)
);
```

- O número de identificação fiscal da *Pessoa* (nif) não pode ser nulo e é único;
- O nome da *Pessoa* (nome) não pode ser nulo;
- O telefone da *Pessoa* (telefone) não pode ser nulo;
- O email da *Pessoa* (email) não pode ser nulo;
- O nome do banco da *Pessoa* (banco) não pode ser nulo;
- O número de conta da *Pessoa* (numConta) não pode ser nulo;

```
CREATE TABLE Empresa (
    nif INTEGER PRIMARY KEY,
    nome TEXT NOT NULL,
    morada TEXT NOT NULL,
    telefone TEXT NOT NULL,
    email TEXT NOT NULL,
    empresa_codPostal INT,
    FOREIGN KEY (empresa_codPostal) REFERENCES Localidade(codPostal)
);
```

- O número de identificação da *Empresa* (nif) não pode ser nulo e é único;
- O nome da *Empresa* (nome) não pode ser nulo;
- O morada da *Empresa* (morada) não pode ser nulo;
- O telefone da *Empresa* (telefone) não pode ser nulo;
- O email da *Empresa* (email) não pode ser nulo;

```
CREATE TABLE Estabelecimento (
    id INTEGER PRIMARY KEY,
    morada TEXT NOT NULL,
    telefone TEXT NOT NULL,
    tipo TEXT NOT NULL,
    email TEXT,
    empresa_nif INT,
    estabelecimento_codPostal INT,
    FOREIGN KEY (empresa_nif) REFERENCES Empresa(nif),
    FOREIGN KEY (estabelecimento_codPostal) REFERENCES Localidade(codPostal)
);
```

- Cada *Estabelecimento* tem um número de identificação (idEstabelecimento) não nulo e é único;
- A morada (morada) *Estabelecimento* do não pode ser nulo;
- O telefone de *Estabelecimento* (telefone) não pode ser nulo;
- O tipo de *Estabelecimento* (tipo) não pode ser nulo;

```
CREATE TABLE Funcionario (
    funcionario_nif INTEGER PRIMARY KEY,
    cargo TEXT NOT NULL,
    ordenado FLOAT NOT NULL,
    agregadoFamiliar INT NOT NULL,
    horario TEXT NOT NULL,
    especializacao_id INT NOT NULL,
    estabelecimento_id INT,
    FOREIGN KEY (funcionario_nif) REFERENCES Pessoa(nif),
    FOREIGN KEY (estabelecimento_id) REFERENCES Estabelecimento(id),
    FOREIGN KEY (especializacao_id) REFERENCES Especializacao(id)
);
```

- O cargo do *Funcionário* (cargo) não pode ser nulo;
- O valor do ordenado do *Funcionário* (ordenado) não pode ser nulo;
- O agregado familiar do *Funcionário* (agregadoFamiliar) não pode ser nulo;
- O horário do *Funcionário* (horario) não pode ser nulo;
- O nome da especialização do *Funcionário* (nomeEspecializacao) não pode ser nulo;

```
CREATE TABLE Venda (
    id INTEGER PRIMARY KEY,
    quantia FLOAT NOT NULL,
    dataVenda date NOT NULL,
    cliente_id INT,
    funcionario_id INT,
    estabelecimento_id INT,
    FOREIGN KEY (cliente_id) REFERENCES Cliente(cliente_nif),
    FOREIGN KEY (funcionario_id) REFERENCES Funcionario(funcionario_nif),
    FOREIGN KEY (estabelecimento_id) REFERENCES Loja(loja_id)
);
```

- Cada *Venda* tem um número de identificação (idVenda) único e não nulo;
- A quantia da *Venda* (quantia) não pode ser um valor nulo;
- A data da *Venda* (dataVenda) não pode ser nula;

```
CREATE TABLE Produto (
    id INTEGER PRIMARY KEY,
    nome TEXT NOT NULL,
    preco FLOAT NOT NULL,
    dimensao TEXT,
    cor TEXT,
    quantidadeVendida INT
);
```

- Cada *Produto* tem um número de identificação (idProduto) único e não nulo;
- O nome do *Produto* (nome) não pode ser nulo;
- O preço do *Produto* (preco) não pode ser nulo;

```
CREATE TABLE Pagamento (
    id INTEGER PRIMARY KEY,
    quantia FLOAT NOT NULL,
    tipo TEXT NOT NULL,
    efetuado TEXT NOT NULL,
    dataPagamento TEXT NOT NULL,
    venda_id INT,
    FOREIGN KEY (venda_id) REFERENCES Venda(id)
);
```

- Cada *Pagamento* tem um número de identificação (idPagamento) único e não nulo;
- O número de identificação de venda (idVenda) associado ao *Pagamento* não pode ser nulo;
- A quantia (quantia) do *Pagamento* não pode ser nula;
- O tipo de *Pagamento* (tipo) não pode ser nulo;
- Deve existir informação sobre o estado do *Pagamento* (efetuado) que é um valor não nulo;
- A data de *Pagamento* (dataPagamento) não pode ser nula;

```
CREATE TABLE Cheque (
    pagamento_id INTEGER PRIMARY KEY,
    banco TEXT NOT NULL,
    depositado TEXT NOT NULL,
    dataDeposito TEXT NOT NULL,
    FOREIGN KEY (pagamento_id) REFERENCES Pagamento(id)
);
```

- Cada *Pagamento* tem um número de identificação (idPagamento) único e não nulo;
- O nome do banco (banco) associado ao *Cheque* não pode ser nulo;
- Sobre cada depósito do *Cheque* é necessário saber se este já foi depoistado ou não (depositado);
- A data em que foi feito o depósito (dataDeposito) do *Cheque* não pode ser nulo;

```
CREATE TABLE Cartao (
    pagamento_id INTEGER PRIMARY KEY,
    banco TEXT NOT NULL,
    numConta INT NOT NULL,
    taxa FLOAT NOT NULL,
    FOREIGN KEY (pagamento_id) REFERENCES Pagamento(id)
);
```

- O número de idetificação do *Pagamento* (idPagamento) não pode ser nulo;
- O nome do banco (banco) do *Cartao* não pode ser nulo;
- O número de conta (numConta) associado ao *Cartao* não pode ser nulo;
- A taxa cobrada por cada transação de multibanco (taxa) não pode ser um valor nulo;

```
CREATE TABLE Fornecedor (
    nif INTEGER PRIMARY KEY,
    nome TEXT NOT NULL UNIQUE,
    morada TEXT NOT NULL,
    telefone TEXT NOT NULL,
    email TEXT NOT NULL,
    fornecedor_codPostal INT,
    FOREIGN KEY (fornecedor_codPostal) REFERENCES Localidade(codPostal)
);
```

- Cada *Fornecedor* tem um número de identificação, o número de identificação fiscal, (nif) único e não nulo;
- O nome de cada *Fornecedor* (nome) é único e não pode ser nulo;
- A morada de cada *Fornecedor* (nome) é única e não pode ser nula;
- O telefone de cada *Fornecedor* (nome) é único e não pode ser nulo;
- O email de cada *Fornecedor* (nome) é único e não pode ser nulo;

```
CREATE TABLE Material_Produto (
    material_id INT,
    produto_id INT,
    quantidadeNecessaria INT NOT NULL,
    CONSTRAINT chaveComposta PRIMARY KEY (material_id, produto_id),
    FOREIGN KEY (material_id) REFERENCES Material(id),
    FOREIGN KEY (produto_id) REFERENCES Produto(id)
);
```

- A quantidade que cada *Produto* necessita de um determinado *Material* não pode ser nula;

```
CREATE TABLE Venda_Produto(
    venda_id INT,
    produto_id INT,
    quantidade INT NOT NULL,
    CONSTRAINT chaveComposta PRIMARY KEY (venda_id, produto_id),
    FOREIGN KEY (venda_id) REFERENCES Venda(id),
    FOREIGN KEY (produto_id) REFERENCES Produto(id)
);
```

- A quantidade de um determinado *Produto* numa *Venda* não pode ser nula.

7 Interrogações

1. Quantos pagamentos foram realizados em dinheiro ou cheque ou cartão?

   ```
   SELECT tipo, count(*) AS quantidade
   FROM pagamento
   GROUP BY tipo;
   ```

2. Quantos clientes existem?

   ```
   SELECT count(*) AS NumClientes
   FROM cliente;
   ```

3. Qual o valor médio do ordenado dos funcionários?

   ```
   SELECT avg(ordenado) AS MédiaOrdenado
   FROM funcionario;
   ```

4. Qual o valor total dos pagamentos pendentes?

   ```
   SELECT sum(quantia) AS ValorTotal
   FROM pagamento where efetuado='n';
   ```

5. Qual a especialidade que é mais utilizada na confecção dos produtos?

   ```
   SELECT nome, max(numVezes)
   FROM (SELECT nome, count(nome) AS numVezes
         FROM (SELECT produto_id, especializacao_id, nome
               FROM produto_especializacao
               LEFT OUTER JOIN especializacao ON
                   produto_especializacao.especializacao_id = especializacao.id))
   GROUP BY nome;
   ```

6. Qual o nome do funcionário que realizou a venda de valor mais alto?

   ```
   SELECT nome
   FROM (SELECT funcionario_id, max(quantia) AS maxQuantia
         FROM venda) AS A INNER JOIN pessoa ON A.funcionario_id = pessoa.nif;
   ```

7. Quais são os clientes que compraram o produto mais cara da empresa? Qual é o valor desse produto?

```
CREATE VIEW info_venda AS
SELECT venda_id, cliente_id, funcionario_id, produto_id, nome, qtd, preco
FROM (SELECT venda_id, cliente_id, funcionario_id, produto_id, quantidade AS qtd
      FROM venda_produto LEFT OUTER JOIN venda ON
       venda_produto.venda_id= venda.id) AS A LEFT OUTER JOIN
            produto ON A.produto_id = produto.id;

SELECT nome, preco
FROM (SELECT cliente_id, max(preco) AS preco
      FROM info_venda) AS A INNER JOIN pessoa ON cliente_id=nif;
```

8. De todos os produtos vendidos, qual foi o tipo de material mais utilizado?

```
CREATE VIEW qntdMaterial AS
SELECT material_id, sum(quantidadeNecessaria) AS total
FROM (SELECT *
      FROM material_produto AS A JOIN venda_produto AS B
      ON A.produto_id=B.produto_id)
GROUP BY material_id;

SELECT nome
FROM (SELECT material_id, max(total)
      FROM qntdMaterial) AS A INNER JOIN material ON A.material_id=material.id;
```

9. Quem são os clientes da empresa e quem foi o(s) funcionário(s) responsável(s) pelas vendas?

```
SELECT DISTINCT NomeCliente, nomeFuncionario
FROM (SELECT nome AS NomeCliente, funcionario_id AS id
      FROM (SELECT DISTINCT cliente_id, funcionario_id FROM venda) AS
      A INNER JOIN pessoa ON A.cliente_id=pessoa.nif) AS
      A JOIN (SELECT nome AS nomeFuncionario, funcionario_id AS id
              FROM (SELECT funcionario_id FROM venda) AS
              A INNER JOIN pessoa ON A.funcionario_id=pessoa.nif)
              AS B ON A.id = B.id;
```

10. Quais os produtos que foram vendidos em compras acima de 800 euros?

```
CREATE VIEW produto_valorVenda AS
SELECT venda_id, produto_id, quantia
FROM venda_produto JOIN venda ON venda_produto.venda_id=venda.id
ORDER BY quantia DESC;

SELECT DISTINCT nome
FROM (SELECT *
      FROM produto_valorVenda
      WHERE quantia >= 800) INNER JOIN produto ON produto_id=id;
```

8 Triggers

- Incrementar a quantidade vendida de um *Produto*

```
CREATE TRIGGER atualizaQntdVendida
AFTER INSERT ON venda_produto
FOR EACH ROW
BEGIN
  UPDATE Produto SET quantidadeVendida = quantidadeVendida + NEW.quantidade
  WHERE Produto.id = NEW.produto_id;
END;

SELECT * FROM produto;
INSERT INTO venda_produto(venda_id, produto_id, quantidade) VALUES(445, 111, 1);
SELECT * FROM produto;

DROP TRIGGER atualizaQntdVendida;
```

- Verificar se quantidade disponível de um *Material* é suficiente para atender a demanda de uma venda

```
CREATE TRIGGER IF NOT EXISTS checaMaterial
BEFORE INSERT ON Venda_Produto
FOR EACH ROW
BEGIN
SELECT CASE
WHEN ((SELECT COUNT(*) FROM
 Venda, Venda_Produto, Material_Produto, Material
 WHERE Venda.id = Venda_Produto.venda_id
 AND Venda_Produto.produto_id = Material_Produto.produto_id
 AND Material.id = Material_Produto.material_id

 AND Venda.id = NEW.venda_id
 AND Material.quantidade >= NEW.quantidade * (SELECT quantidadeNecessaria
     FROM Material_Produto
     WHERE Material_Produto.produto_id = NEW.produto_id)) = 0)
THEN RAISE(abort, 'Quantidade de material insuficiente em stock. Ação não realizada.')
END;
END;

SELECT * FROM venda_produto;
INSERT INTO venda_produto(venda_id, produto_id, quantidade) VALUES(444, 111, 8);
SELECT * FROM venda_produto;

DROP TRIGGER checaMaterial;
```

- Total de uma venda

```
CREATE VIEW IF NOT EXISTS venda_valorTotal AS
SELECT venda_id, produto_id, quantidade, preco AS precoUnitario,
SUM(preco*quantidade) AS valorTotal
FROM venda_produto AS vp JOIN produto AS p ON vp.produto_id=p.id
GROUP BY venda_id;

CREATE TRIGGER if not exists atualizaValorTotal
AFTER INSERT ON venda_produto
FOR EACH ROW
BEGIN
  UPDATE Venda SET quantia = (SELECT valorTotal FROM venda_valorTotal as vt
  WHERE vt.venda_id = NEW.venda_id)
  WHERE Venda.id = NEW.venda_id;
END;

SELECT * FROM venda;
INSERT INTO venda_produto(venda_id, produto_id, quantidade) VALUES(445, 121, 2);
SELECT * FROM venda;

DROP TRIGGER atualizaValorTotal;
```

9 Referências

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems. 3rd Edition.