universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Licenciatura em Engenharia Informática
Bases de Dados

José Trigo 98597
Pedro Monteiro 97484
P9-G1

2020/2021

Projeto - Base de Dados para um Supermercado

Índice

Introdução

No âmbito da unidade curricular de Base de Dados foi desenvolvido um sistema para gerir um supermercado, com ênfase no desenvolvimento da sua base de dados.

A construção deste sistema foi realizada em SQL(inserção, remoção e manipulação de dados), e uma interface, em C#.

Todo o código implementado encontra-se organizado por pastas.

Na pasta SQL estão disponíveis todos os scripts SQL

- DDL
- DML
- Stored Procedures
- Indexes
- Triggers

Na pasta Interface está disponível todo o código C#.

O ficheiro data_generator.py foi usado para gerar dados a serem inseridos na base de dados.

O ficheiro demo.mp4 corresponde a uma demonstração da interface.

Análise de requisitos

Para uma boa criação da base de dados, foi necessário começar por elaborar uma análise de requisitos, sendo para isso realizadas várias pesquisas e perguntas, fazendo assim um levantamento detalhado de toda a informação que se considerou ser essencial.

Passámos, então, para o desenho conceptual, onde foram criadas as entidades, relacionamentos e restrições.

Visto que, este processo não é determinístico, houve alguns aspetos que deixaram dúvidas, sendo posteriormente alterados.

Entidades:

- Person
- Client
- Employee
- Supplier
- Type
- Warehouse
- Product
- Shoppinglist
- Invoice
- Login

Diagramas

Esquema Relacional

Após a reunião com o professor e no momento da implementação houve necessidade de fazer algumas alterações em ambos os diagramas.

A entidade *Shopping List* foi a que sofreu mais alterações, deixando de se relacionar com *Client* e *Employee*, passando a Invoice a estabelecer esta relação.

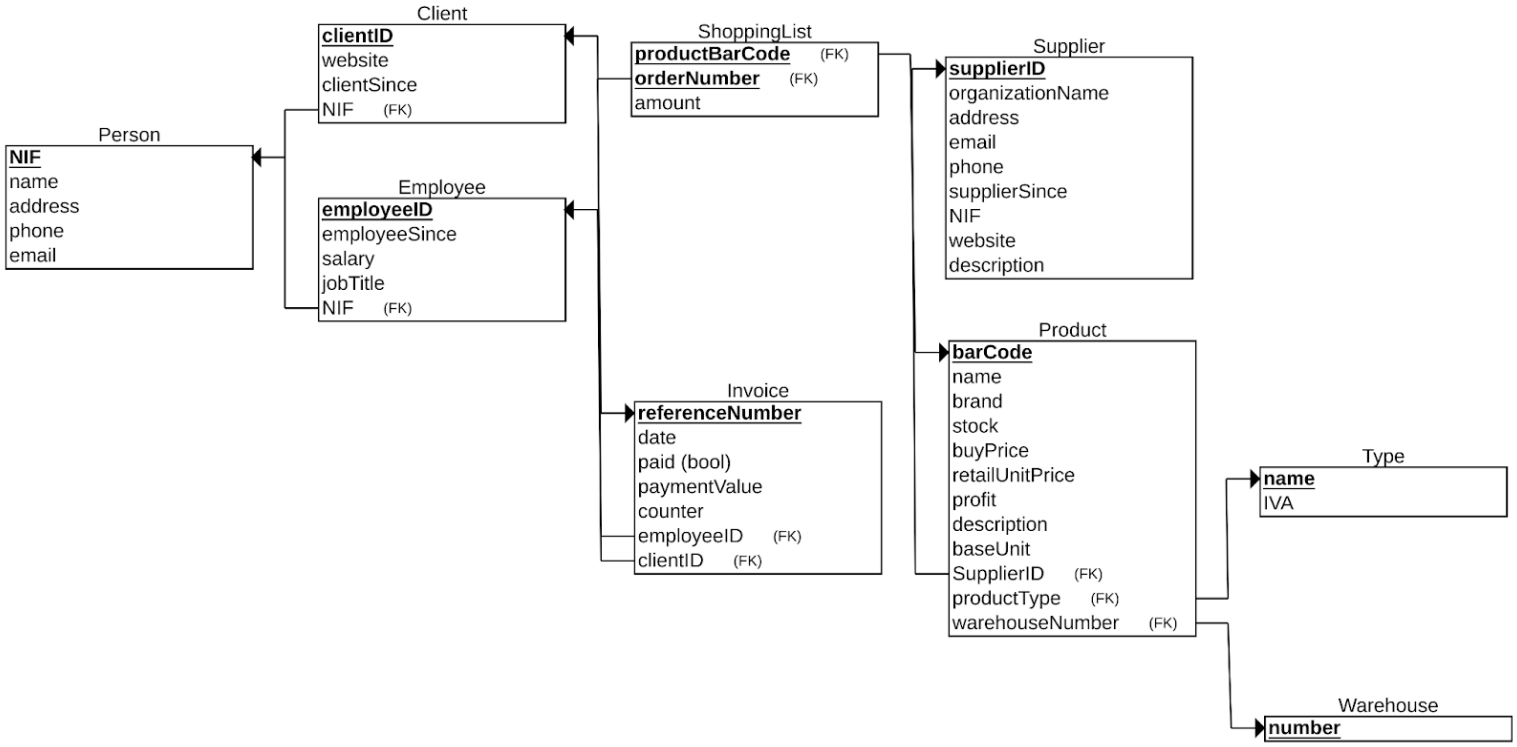
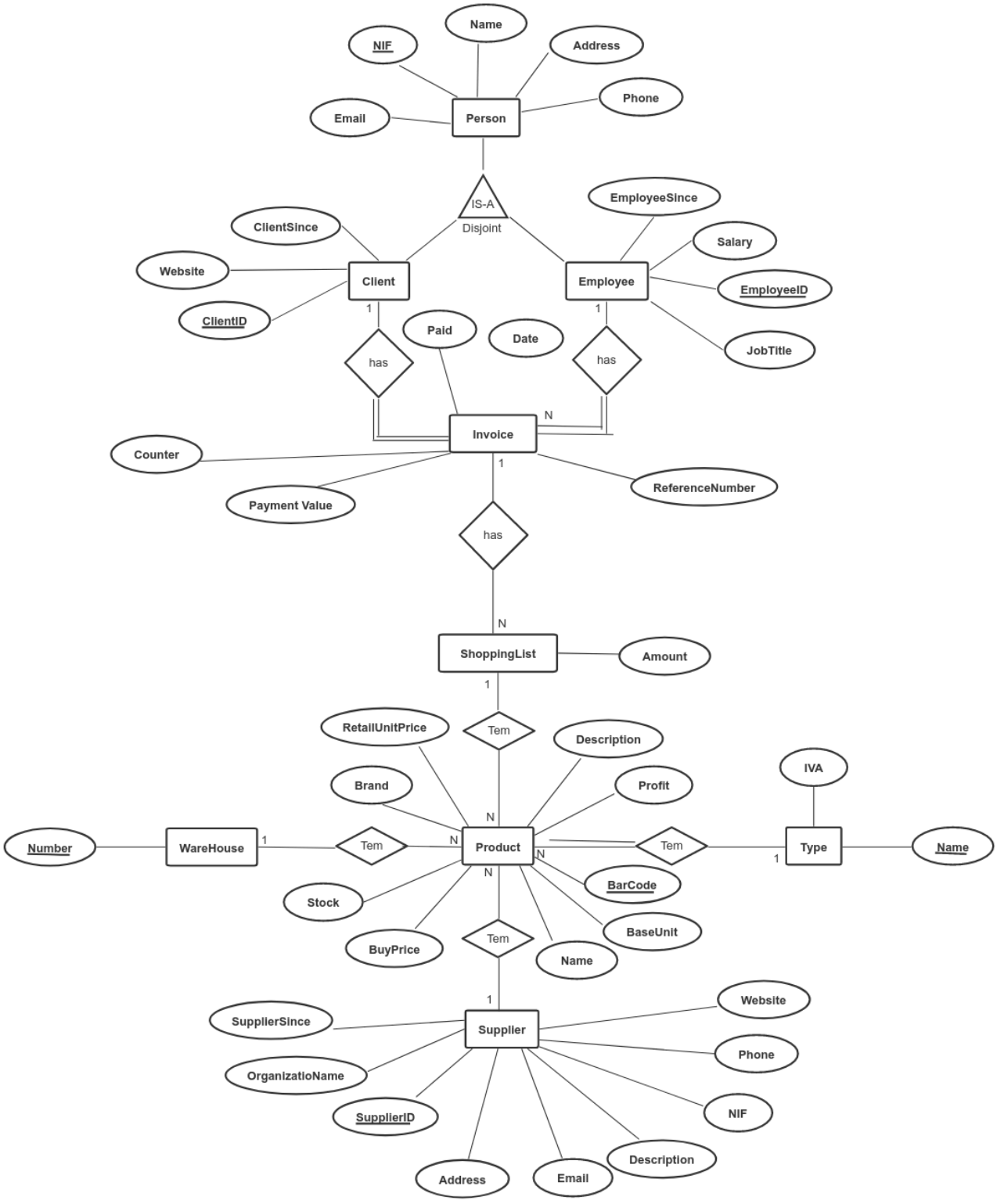


Diagrama Entidade Relação

Como já foi referido, também o Diagrama Entidade-Relação sofreu algumas alterações.



Linguagem SQL DDL

Foi criado um schema, supermarket, na base de dados utilizada durante as aulas práticas (p9g1).

```
CREATE SCHEMA supermarket;
```

Todas as tabelas foram criadas da mesma forma, seguindo, então, a regra supermarket.[tablename].

```
CREATE TABLE supermarket.person(
	NIF INT,
	[name] VARCHAR(50) NOT NULL,
	[address] VARCHAR(100),
	phone VARCHAR(15),
	email VARCHAR(30),
	PRIMARY KEY(NIF)
);
```

De notar que a tabela supermarket.login guarda um utilizador, identificado por um ID, e guarda também a palavra-passe, codificada em SHA512, levando assim a que a password nunca seja guardada diretamente na base de dados, o que garante mais segurança.

```
CREATE TABLE supermarket.login(
	username INT NOT NULL,
	password VARCHAR(128) NOT NULL,
	PRIMARY KEY (username)
);
```

Linguagem SQL DML (Data Manipulation Language)

Todos os comandos associados à manipulação e alteração de dados foram inseridos dentro de estruturas de dados como *Stored Procedures* e *Triggers*, o que garante uma abstração entre a interface e a base dados, sendo a comunicação feita através de um middleware.

Com isto, tentamos também garantir uma certa segurança relativamente a ataques, principalmente *SQL Injections*.

Stored Procedures

A maioria das chamadas feitas à base de dados são através de Stored Procedures, devido a todas as vantagens fornecidas com o uso destas estruturas de dados.

Foram criadas Stored Procedures para:

Adicionar elementos a uma tabela

```
GO
CREATE PROC dbo.addEmployee
(
    @nif INT = NULL,
    @name VARCHAR(50) = NULL,
      @address VARCHAR(100) = NULL,
      @phone VARCHAR(15) = NULL,
      @email VARCHAR(30) = NULL,
    @employeeID INT = NULL,
      @employeeSince DATE = NULL,
      @salary FLOAT(2) = NULL,
    @jobTitle VARCHAR(20) = NULL
)
AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO supermarket.person(NIF, [name], [address], phone,
email)
      VALUES(@nif, @name, @address, @phone, @email);

      INSERT INTO supermarket.employee(employeeID, employeeSince,
salary, jobtitle, NIF)
      VALUES(@employeeID, @employeeSince, @salary, @jobTitle, @nif);
END
GO
```

Remover elementos de uma tabela

```
GO
CREATE PROC dbo.deleteEmployee
(
    @nif INT
)
AS
BEGIN
      SET NOCOUNT ON;
      DELETE FROM supermarket.employee WHERE nif=@nif
      DELETE FROM supermarket.person WHERE nif=@nif
END
GO
```

Aplicar Filtros

```
GO
CREATE PROC dbo.filterClients
(
    @clientID INT = NULL,
    @name VARCHAR(50) = NULL,
      @nif INT = NULL,
      @address VARCHAR(100) = NULL,
      @phone VARCHAR(15) = NULL,
      @email VARCHAR(30) = NULL,
      @website VARCHAR(30) = NULL,
      @clientSince DATE = NULL
)
AS
BEGIN
      SET NOCOUNT ON;
    SELECT DISTINCT *
    FROM view_clients
    WHERE   (@nif IS NULL OR NIF = @nif)
            AND (@name IS NULL OR [name] LIKE @name+'%')
                  AND (@address IS NULL OR [address] LIKE @address+'%')
                  AND (@phone IS NULL OR phone LIKE @phone+'%')
            AND (@email IS NULL OR email LIKE @email+'%')
            AND (@clientID IS NULL OR clientID = @clientID)
                  AND (@clientSince IS NULL OR clientSince >= @clientSince)
                  AND (@website IS NULL OR website LIKE @website+'%')
END
GO
```

Views

Foram criadas views, visto que, como o próprio nome indica, ajudam a ter uma visão melhor sobre os dados que estão a ser modelados. Atuam como uma tabela virtual, que não existe fisicamente, pelo que podem ajudar a diminuir a complexidade de certas queries.

Exemplo de uma view utilizada para ver os clientes

```
CREATE VIEW view_clients AS
SELECT clientID, [name], person.NIF, [address], phone, email,
website, clientSince
FROM supermarket.person JOIN supermarket.client on person.NIF =
client.NIF;
```

Exemplo de uma view para ver todos os fornecedores

```
CREATE VIEW view_suppliers AS
SELECT supplierID, organizationName, [address], email, phone,
supplierSince, NIF, website, [description]
FROM supermarket.supplier
```

Triggers

Foi criado um *Trigger After* para que algumas ações sejam automaticamente despoletadas. Optou-se por este tipo de *Trigger*, visto que seria mais útil ser executado no final da ação, e não a meio, ao contrário dos *Triggers Instead of*.

Este *Trigger* está associado ao botão de *Confirm Purchase*, para que o preço seja automaticamente calculado e enviado para a base de dados, tendo em conta os produtos adicionados à *Shopping List*.

```
CREATE TRIGGER confirmPurchase ON supermarket.shoppingList
AFTER INSERT, UPDATE
AS
	SET NOCOUNT ON;
	DECLARE @barcode INT
	DECLARE @referenceNumber INT
	DECLARE @amount INT
	DECLARE @value FLOAT
	SELECT @barcode = productBarCode, @referenceNumber = orderNumber,
@amount = amount FROM inserted;
	SELECT @value = retailUnitPrice FROM supermarket.product WHERE
@barcode = barcode
	UPDATE supermarket.invoice SET paymentValue += @amount * @value
WHERE @referenceNumber = referenceNumber
GO
```

Índices

Para melhorar a performance da pesquisa na base de dados foram adicionados 4 índices.

```
-- products index
CREATE INDEX idx_product_name ON
supermarket.product(barcode, [name])

-- employees index
CREATE INDEX idx_employee ON
supermarket.employee(employeeID)

-- clients index
CREATE INDEX idx_client ON supermarket.client(clientID)

-- suppliers index
CREATE INDEX idx_supplier ON
supermarket.supplier(supplierID)
```

Demo

É possível encontrar na pasta um ficheiro demo.mp4, onde se pode ver de um modo geral o funcionamento de toda a interface.

Conclusão

Neste trabalho preocupamo-nos com a implementação de base de dados, pelo que, é de notar, que a interface pode estar menos bem concebida.

Procurámos aplicar todos os conhecimentos aprendidos ao longo de todo o semestre, sendo por isso o trabalho muitas vezes alvo de alterações.

Em suma, acreditamos que a maioria dos objetivos foi cumprida, e por isso foi possível criar uma base de dados que permite gerir de forma correta um supermercado.