universidade de aveiro

Bases de Dados
Relatório Final

Gestão de uma Pizzaria e Sistema de Encomendas

Grupo: P6G5
Leandro Silva (93446)
Mário Silva (93430)

Disciplina: Bases de Dados

Data de entrega: 12 de junho de 2020

Índice

Índice 2

Introdução 3

Análise de Requisitos 4

- **Entidades** 4
- **Funcionalidades** 5

Diagrama Entidade Relação 6

Esquema Relacional 7

SQL Data Definition Language 8

SQL Data Manipulation Language 8

Normalização 9

Índices 10

Triggers 11

Stored Procedures 12

User Defined Functions 15

Segurança 17

Conclusão 18

Notas Finais o Docente 18

Introdução

No âmbito do trabalho final da disciplina de Bases de Dados, este relatório tem como principal objetivo apresentar a análise de requisitos desenvolvida pelo grupo tendo em conta, claramente, o tema escolhido para a realização futura deste.

O tema escolhido foi a modelação de um sistema de Gestão de uma Pizzaria e Sistema de Encomendas, sendo o foco principal o desenho e desenvolvimento da camada de Base de Dados. Pretende-se assim guardar os diversos dados e registos das entidades constituintes do sistema.

Esta base de dados será manipulada e exibida através de uma aplicação com uma interface construída usando formulários .NET framework em C#. Existe uma interfaces para cada tipo de utilizador (cliente, estafeta e administrador) que permitirão dar um uso à base de dados próximo do que se veria na realidade.

Análise de Requisitos

Feita a Introdução, nesta secção irá-se abordar o objetivo concreto deste relatório, que como já foi enunciado, é a Análise de Requisitos. Nesta análise, procedeu-se à identificação das entidades presentes no Sistema, assim como as características e relações que estas estabelecem entre si. Assim, encontra-se, de seguida a lista de requisitos:

Entidades

- *Ingrediente*: É um produto. São os ingredientes que normalmente se utilizam para fazer as pizzas.
  Tem uma quantidade disponível.

- *Piza*: É um produto.
  Tem um tamanho e uma fotografia.

- *PizaIngrediente*: São os ingredientes usados na piza.
  Tem uma quantidade.

- *Bebida*: É um produto.
  Tem uma quantidade disponível.

- *Produto*: É um item. Representa um ingrediente, uma bebida ou uma piza.

- *Menu*: É um item.

- *MenuProduto*: São os produtos usados no menu.
  Tem uma quantidade.

- *MenuPiza*: São as pizzas e suas quantidades usadas no menu.
  Tem uma quantidade.

- *Encomenda*: Conjunto de produtos, e/ou menus requisitados por um cliente.
  Tem um ID, o email do cliente que fez a encomenda, e o email do estafeta responsável por levar a encomenda. um endereço físico, uma hora, um método de pagamento e um desconto, sendo este último opcional.

- *EncomendaEntregue*: Registo de encomendas já entregues.
  Igual a encomenda, mas com um dado extra a indicar qual o restaurante a encomenda pertence.

- *Item*: Representa um menu ou um produto.
  Tem um ID, um nome e um preço.

- *EncomendaItem*: São os itens contidos na encomenda.
  Tem um ID de encomenda, ID de item e uma quantidade.

- *EncEntregueItem*: São os itens contidos na encomenda entregue.
  Tem um ID de encomenda entregue, ID de item e uma quantidade.

- *Utilizador*: Entidades que vão usar o sistema.
  Tem email, password, nome e contacto.

- *Cliente*: É um utilizador do sistema.
  Tem idade, género, morada.

- *Estafeta*: É um utilizador do sistema que entrega as encomendas ao domicílio e trabalha num restaurante.

- *Admin*: É um utilizador do sistema que faz a gestão do sistema e possui um ou mais restaurantes.

- *Restaurante*: Tem uma morada, hora de abertura e de fecho, um contacto, lotação e um dono associado.

- *Desconto*: Tem um código e uma percentagem de desconto, uma data de início e outra de fim.

- *DescontoCliente*: Registo dos código dos desconto usados por Cliente.

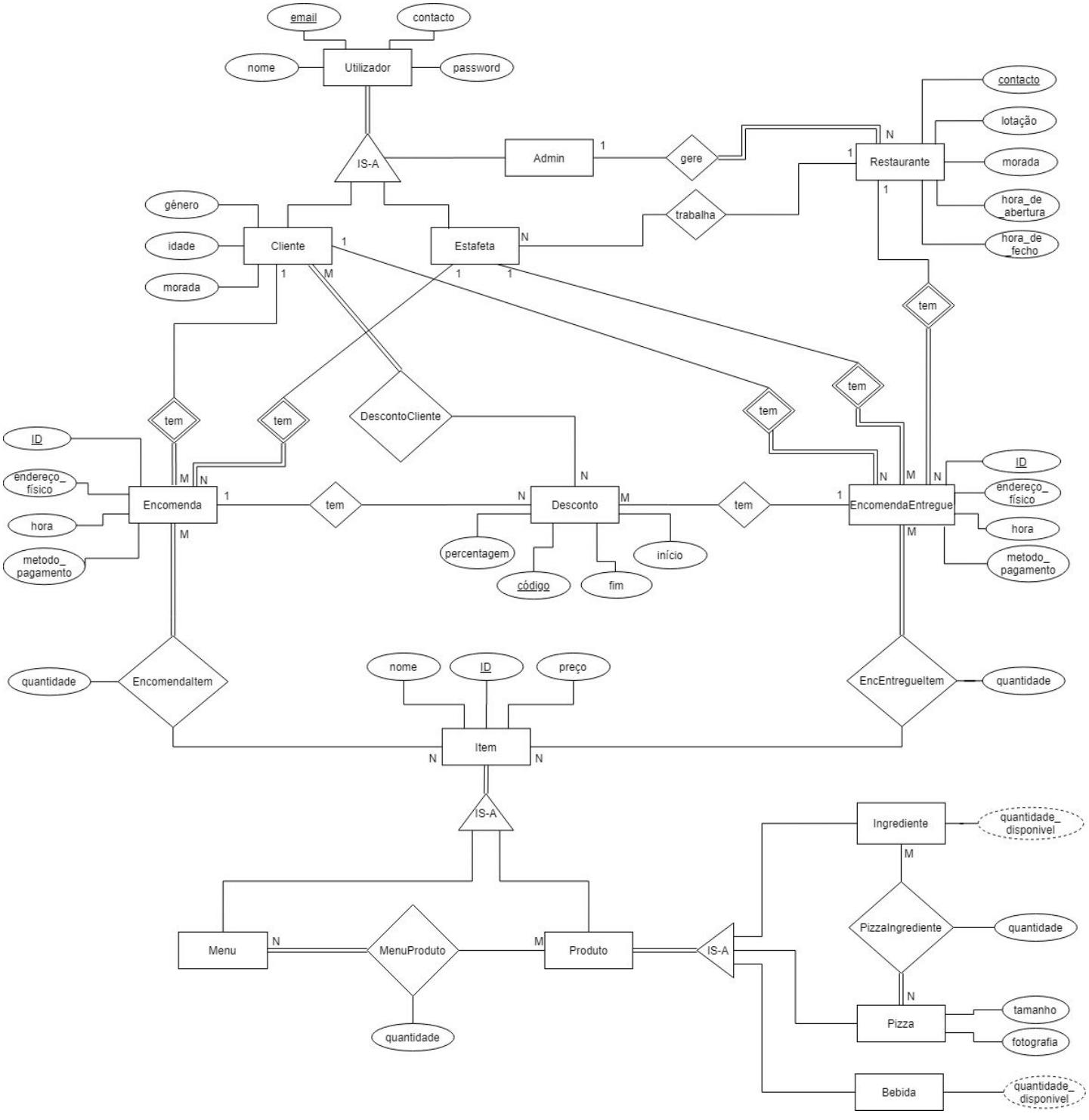
Funcionalidades

- O Cliente, o Estafeta e o Admin têm acesso ao sistema.
- O Cliente pode ver a lista de menus, de pizzas e produtos.
- O Cliente pode usar um código promocional para obter um desconto.
- O Cliente pode pedir uma encomenda.
- O Cliente não pode repetir descontos usados.
- O Cliente pode ver as suas encomendas atuais por receber e o seu histórico de encomendas já recebidas.
- O Estafeta pode ver e entregar as encomendas que tem pendentes.
- O Estafeta pode ver o seu histórico de encomendas entregues.
- O Estafeta pode ver dados sobre o restaurante para qual trabalha.
- O Estafeta pode ver dados sobre os estafetas que trabalham para o mesmo restaurante.
- O Admin pode alterar a lista de descontos.
- O Admin tem acesso a várias estatísticas do restaurante, estafetas e itens vendidos.
- O Admin pode dar restock a produtos que estiverem em falta.
- O Admin pode adicionar/atualizar restaurantes.
- O Admin pode contratar/despedir estafetas do restaurante que selecionar.

Diagrama Entidade Relação

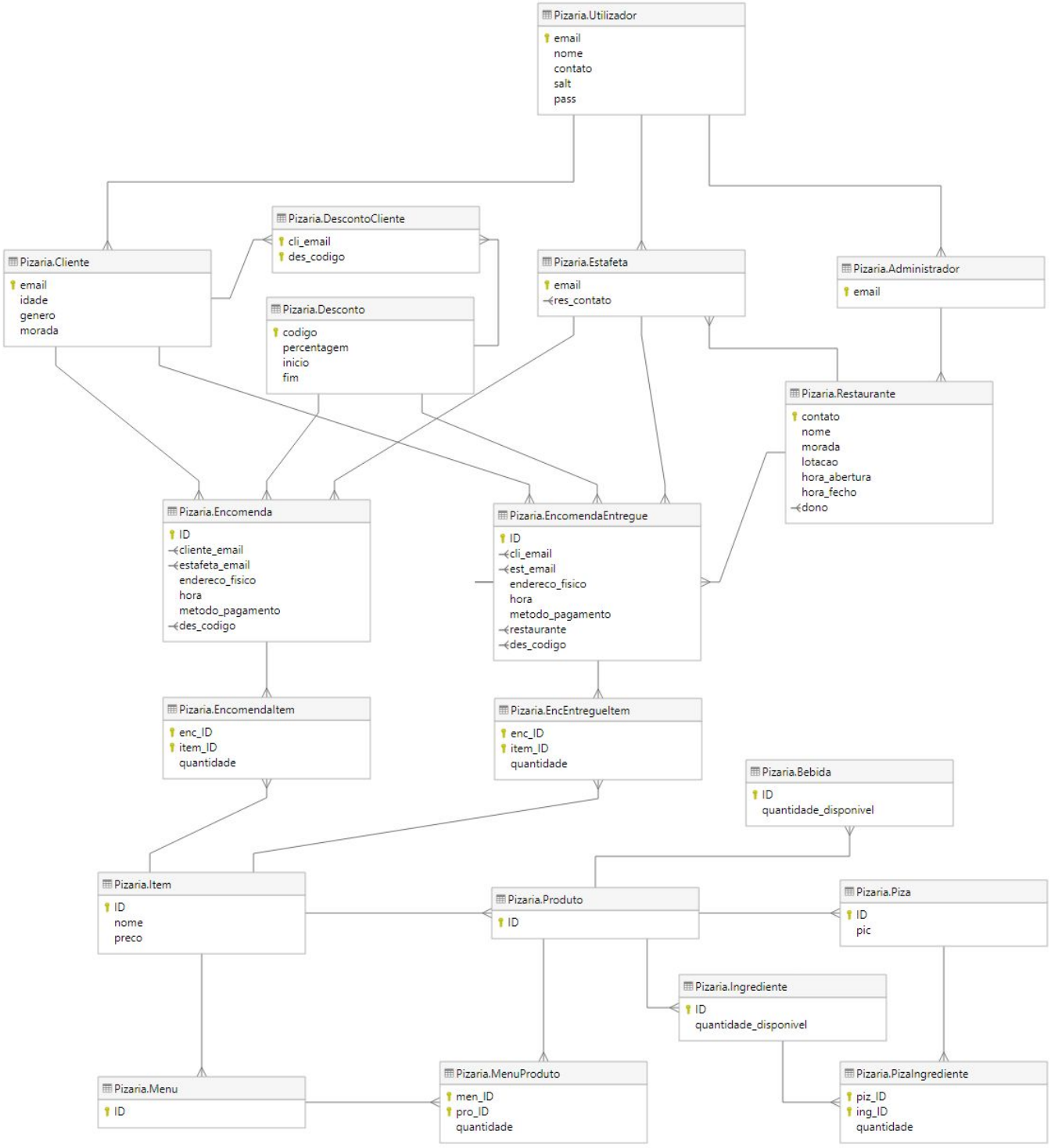
Este desenho é a última versão do diagrama entidade relação.

Algumas alterações foram feitas se comparadas com o primeiro, como a criação da tabela *EncomendaEntregue*, que foi essencial para separar as encomendas entregues das ativas e, por consequência, na criação da tabela *EncEntregueItem*, que tem a mesma função que *EncomendaItem*. Para além dos atributos em *EncomendaItem*, adicionamos *restaurante* a *EncEntregueItem*, uma vez que seria importante associar as encomendas entregues ao restaurante mesmo que o estafeta fosse despedido, isto é, perdesse a associação ao seu restaurante.



Esquema Relacional

Esta figura representa o estado final do esquema relacional.



SQL Data Definition Language

Acabada a definição do nosso esquema relacional, procedemos à criação da estrutura da nossa base de dados. Na Figura 1 temos uma amostra da utilização da DDL para a criação das tabelas com restrições de chave, restrições de domínio, integridade de vazios e integridade referencial.

```
create table Pizaria.Utilizador (
    email           nvarchar(255),
    nome            varchar(50)     not null,
    contato         int             not null    check(100000000 <= contato and contato <= 999999999),
    salt            UNIQUEIDENTIFIER            not null,
    pass            binary(64)      not null,
    primary key (email)
);
create table Pizaria.Administrador (
    email           nvarchar(255),
    primary key (email),
    foreign key (email) references Pizaria.Utilizador(email)
);
```

Figura 1 - Exemplo da utilização de comandos DDL para criar as tabelas *Utilizador* e *Administrador*.

SQL Data Manipulation Language

Os comandos DML foram amplamente utilizados em diversos aspetos, sendo os comandos mais básicos como o SELECT, INSERT, DELETE e UPDATE os mais recorridos em geral, como será mostrado mais à frente em detalhe.

A primeira utilização da DDL foi na população da nossa base de dados, como está mostrado na Figura 2.

```
INSERT INTO Pizaria.Administrador VALUES ('admin@gmail.com');

INSERT INTO Pizaria.Restaurante (contato,nome,morada,lotacao,hora_abertura,hora_fecho,dono) VALUES (256455582,'Pizaria X','8109 Or

INSERT INTO Pizaria.Estafeta (email,res_contato) VALUES ('estafeta@gmail.com', 256455582),('Maecenas.libero.est@arcu.com', 2564555

INSERT INTO Pizaria.Cliente (email,idade,genero,morada) VALUES ('tellus.justo.sit@iaculis.net',58,'M','3711 Congue St.'), ('vehicu
INSERT INTO Pizaria.Cliente (email,idade,genero,morada) VALUES ('cliente@gmail.com',18,'M','440-8601 Dolor Rd.'), ('ipsum.cursus.v
INSERT INTO Pizaria.Cliente (email,idade,genero,morada) VALUES ('malesuada.vel.convallis@enimconsequat.org',38,'M','Ap #442-6580 S

INSERT INTO Pizaria.Desconto (codigo,percentagem,inicio,fim) VALUES (1, 20, '2020-06-10', '2020-11-20'),(2, 5, '2019-10-07', '2020

set identity_insert Pizaria.Encomenda on
insert into Pizaria.Encomenda (ID,cliente_email,estafeta_email,endereco_fisico,hora,metodo_pagamento,des_codigo) values (1002,'tem
insert into Pizaria.Encomenda (ID,cliente_email,estafeta_email,endereco_fisico,hora,metodo_pagamento,des_codigo) values (1102,'per
set identity_insert Pizaria.Encomenda off
```

Figura 2 - Amostra da inserção de dados na BD.

Normalização

Para minimizar a redundância dos dados da nossa base de dados, preservando sempre a informação lá contida, procedemos à realização de testes para garantir de que estávamos perante um desenho relacional que satisfazia a 3ª Forma Normal. Para satisfazer a 3ª Forma Normal é necessário satisfazer a 2ª Forma Normal, que por sua vez é necessário satisfazer a 1ª Forma Normal.

Na primeira análise garantimos por satisfeita a 1ª Forma Normal, uma vez que não tínhamos presente atributos compostos nem multivalor. Chegamos a pensar em dividir o atributo morada em subpartes, como localidade e rua, mas depressa abandonamos esta ideia. Para a 1ª Forma Normal também é necessário remover as relações dentro de relações, o que não nos foi problema.

Como todos os atributos não pertencentes à chave primária dependem de todos os atributos da chave primária (dependência total) garantimos também a 2ª Forma Normal, não havendo necessidade de decompor nenhuma relação.

Por fim, como não encontramos dependências transitivas nas nossas tabelas entre atributos não pertencentes à chave primária, determinamos por satisfeita a 3ª Forma Normal no nosso desenho relacional.

Índices

Para a indexação usamos sempre apenas clustered index para a primary key que o SQL coloca por defeito. Decidimos não adicionar mais nenhum índice devido à nossa base de dados conseguir executar as queries em tempo útil e daí não justificar a adição de índices extras.

Destacamos aqui na tabela de encomendas, que através da propriedade IDENTITY fizemos inserções sequenciais (Figura 3), permitindo-nos assim recorrer a uma chave primária simples, mais pequena e auto-gerada.

```
create table Pizaria.Encomenda(
    ID                  int             identity (1200,1),
    cliente_email       nvarchar(255)   not null,
    estafeta_email      nvarchar(255)   not null,
    endereco_fisico     varchar(50)     not null,
    hora                datetime        not null,
    metodo_pagamento    varchar(30)     not null,
    des_codigo          int,
    primary key(ID),
    foreign key(estafeta_email) references Pizaria.Estafeta(email),
    foreign key(des_codigo) references Pizaria.Desconto(codigo),
    foreign key(cliente_email) references Pizaria.Cliente(email)
);
```

Figura 3 - Criação da tabela *Encomenda* com a propriedade IDENTITY no atributo ID.

Triggers

Os triggers existentes na nossa BD ajudam na manutenção e consistência dos dados, para além dos checks que fazemos na criação das tabelas e das verificações presentes nas stored procedures. Servem essencialmente para remover as dependências de chaves estrangeiras noutras tabelas, antes de proceder à remoção de uma encomenda, estafeta ou desconto, e para evitar inserir atributos a *null* ou repetidos numa tabela com essas restrições, como é o caso do *insDescontoCliente*.

O *delEncomenda*, por exemplo, é um trigger INSTEAD OF que, antes de remover uma encomenda, ou seja, quando um estafeta confirmar que esta foi entregue, seleciona os atributos da encomenda a remover e insere-os na tabela das encomendas já entregues. Depois, seleciona todas as relações da encomenda a remover na tabela *EncomendaItem* e coloca-as na tabela *EncEntregueItem*. Por fim, remove todas as linhas nas tabelas *EncomendaItem* e *Encomenda* que referem ao ID da encomenda a remover.

O *delEstafeta* também merece uma análise, uma vez que é uma das queries que faz uso de um cursor (Figura 4). Para despedir um estafeta, é atualizado o valor do atributo *res_contato* do estafeta para *null*. Se o estafeta acabado de ser despedido tiver encomendas ativas, será pois necessário distribuir aos outros estafetas as suas encomendas. Para saber a que estafeta atribuir a encomenda, criámos uma UDF, *findBestEstafeta*, que determinasse qual o melhor estafeta para receber a encomenda, mais especificamente o estafeta com menos encomendas ativas. Como não consideramos apropriado atribuir todas as encomendas a apenas um estafeta, para cada encomenda chamamos a SP e atribuímos-la ao estafeta determinado por esta, o que obriga à utilização de algo como um cursor.

O trigger utiliza um cursor STATIC, ou seja, um cursor geralmente bastante rápido que inicialmente copia os dados da tabela selecionada para uma temporária e itera apenas a tabela temporária. Decidimos usar este tipo de cursor pois é necessário fazer alterações nos dados da tabela inicial a cada iteração e este permite-nos iterar sobre a tabela temporária e operações de alteração nos dados que não são vistas pelo cursor.

```
update Pizaria.Estafeta set res_contato = null where email=@email;
--cursor para colocar o melhor estafeta em cada uma das encomendas
DECLARE c CURSOR STATIC
FOR select ID from Pizaria.Encomenda where estafeta_email=@email;

OPEN c;
FETCH NEXT FROM c into @ID;

WHILE @@FETCH_STATUS = 0
BEGIN
    update Pizaria.Encomenda set estafeta_email = (select Pizaria.FindBestEstafeta()) where ID=@ID
    FETCH NEXT FROM c into @ID;
END;
CLOSE c;
DEALLOCATE c;
```

Figura 4 - Exemplo do uso de um cursor no trigger *delEstafeta*.

Stored Procedures

Ao todo foram criadas 10 stored procedures. Como estas permitem todos os tipos de instruções DML, foram usadas para opções de filtragem, inserção ou atualização de dados, e operações em que o utilizador podia escrever o input, de forma a criar um mecanismo mais seguro contra possíveis ataques como SQL injection.

As SP também oferecem outras vantagens relativamente às UDF, pois permitem retornar dados diretamente das instruções SELECT, o que facilitava na escrita da query, e reduzem a carga do sistema, oferecendo assim uma performance maior. Também permitem o uso de blocos try-catch úteis para caso de haver algum tipo de falha e lidar com possíveis RAISERROR.

Esta é uma enumeração de todas as procedures seguidas de uma breve descrição das mesmas:

- *filterItem* Filtra os itens de acordo com o seu tipo, preço e nome.
- *insDesconto* Insere um desconto, se validado.
- *insEncomenda* Insere uma encomenda.
- *insEstafeta* Insere um estafeta novo ou recontrata um estafeta.
- *insRestaurante* Insere um restaurante.
- *insUtilizador* Insere um utilizador.
- *login* Valida o login.
- *register* Insere um utilizador de qualquer tipo.
- *tranShopCart* Trata de todas as ações e validações necessárias para inserir uma encomenda.
- *updRestaurante* Atualiza um restaurante.
- *updStock* Atualiza o stock dos ingredientes e bebidas.
- statsRestaurante Devolve as estatísticas dos restaurantes.

A ideia por trás do *insEstafeta* é que ele seria usado apenas pelo administrador para inserir novos estafetas ou recontratá-los. Se o estafeta passado à procedure existir na BD e tiver o seu atributo *res_contato* a *null*, esta altera-o seu para o recebido, ou seja, recontrata o estafeta de restaurante. Se não existir, é gerada uma palavra-passe aleatoriamente (Figura 5), que pressupõe-se que seria dada ao estafeta pelo administrador, e é adicionado um novo estafeta com os atributos passado à procedure, fazendo uso do SP *register*.

```
DECLARE @pass VARCHAR(12)
DECLARE @BinaryData VARBINARY(12)
DECLARE @CharacterData VARCHAR(12)

set @BinaryData = CRYPT_GEN_RANDOM (12)

Set @CharacterData=cast ('' as xml).value ('xs:base64Binary(sql:variable("@BinaryData"))',
                'varchar (max)')

SET @pass = @CharacterData
```

Figura 5 - Geração aleatória da palavra-passe do estafeta antes da sua inserção.

O *register*, já anteriormente mencionado, insere um utilizador de qualquer tipo na BD, caso ainda não exista. Um papel importante que o *register* tem é criptografar as palavras-passes dos utilizador usando um sal criptográfico. Este sal é concatenado com a senha real e processado numa função hash criptográfica, sendo depois apenas armazenados o sal e o hash gerado pela função (Figura 6). A partir desta técnica, mesmo que os dados sejam comprometidos, o atacante não terá a senha revelada porque há apenas um hash guardado.

```
declare @salt uniqueidentifier=newid()
begin try
    insert into Pizaria.[Utilizador] (email, nome, contato, salt, pass)
    values (@email, @nome, @contato, @salt,HASHBYTES('SHA2_512', @pass + CAST(@salt AS NVARCHAR(36))))
```

Figura 6 - utilização da técnica do sal criptográfico para proteção das senhas armazenadas.

O *tranShopCart* é a nossa query mais longa e complexa por necessitar de muitas operações para garantir que a encomenda é viável.

O primeiro passo nesta procedure é determinar o estafeta a entregar a encomenda, chamando a UDF *findBestEstafeta*.

A seguir, já dentro de uma transação, é inserido o desconto utilizado na tabela *DescontoCliente*, se existente, e a encomenda na tabela *Encomenda* a partir da procedure *insEncomenda*. Esta SP é importante pois retorna o ID da encomenda inserida através da função SCOPE_IDENTITY(), uma vez que temos a propriedade IDENTITY para o ID da encomenda. Este ID será necessário mais tarde.

Nota: tivemos o cuidado de ao usarmos a função SCOPE_IDENTITY() limitarmos a que apenas uma tabela da nossa BD tivesse IDENTITY, pois devolve o último valor gerado pelo IDENTITY na sessão, se tivéssemos mais que uma tabela com IDENTITY teríamos de recorrer a outros métodos para receber o valor específico gerado nessa tabela.

O seguinte passo será um WHILE que percorre a string que contém os itens a adicionar. O CHARINDEX() separa os itens e o seus atributos por vírgulas, obtendo assim o ID. Este ID é depois utilizado como parâmetro na UDF *findItemType* que retorna o tipo de item associado a esse ID (Figura 7).

```
declare @estafeta_email nvarchar(255)
set @estafeta_email = Pizaria.findBestestafeta()

begin try
begin tran
    insert into Pizaria.DescontoCliente (cli_email,des_codigo) Values (@cliente_email,@des_codigo)

    declare @last_ID int
    Exec Pizaria.insEncomenda @cliente_email=@cliente_email, @estafeta_email=@estafeta_email, @endereco_fisico=@endereco_fisico,
    @hora=@hora, @metodo_pagamento=@metodo_pagamento, @des_codigo=@des_codigo, @last_ID=@last_ID output

    declare @pos int = 0
    declare @len int = 0
    declare @item_ID int
    declare @quantidade int
    WHILE CHARINDEX(',', @lista, @pos+1)>0
    BEGIN
        set @len = CHARINDEX(',', @lista, @pos+1) - @pos
        set @item_ID = cast(SUBSTRING(@lista, @pos, @len) as int)
        set @pos = CHARINDEX(',', @lista, @pos+@len) +1
        set @len = CHARINDEX(',', @lista, @pos+1) - @pos
        set @quantidade =cast( SUBSTRING(@lista, @pos, @len) as int)
        set @pos = CHARINDEX(',', @lista, @pos+@len) +1

        declare @itemType varchar(15)
        set @itemType=Pizaria.findItemType(@item_ID)
```

Figura 7 - Iteração pelos itens da encomenda usando o CHARINDEX() e SUBSTRING().

Ao sabermos o tipo de item, verificamos se existe quantidade disponível em stock do item em questão. Se não houver é feito um ROLLBACK TRAN que irá desfazer todas as operações feitas até então. Caso contrário, a quantidade requerida é reduzida no stock, o item é inserido na tabela *EncomendaItem* (fazendo uso do ID provido pela SP *insEncomenda*) e o WHILE passa para o próximo item da encomenda. Para o tipo de item "Menu", a complexidade destas operações é maior, pois precisa de fazer o mesmo trabalho para todos os itens dentro do menu, e para todos os itens da pizza (Figura 8).

```
    if (@itemType='Menu')
    begin
        IF EXISTS (select top 1 men_ID from Pizaria.MenuProduto left outer join Pizaria.Piza on pro_ID=Piza.ID
        left outer join Pizaria.Bebida on Bebida.ID=pro_ID
        left outer join Pizaria.PizaIngrediente on piz_ID=Piza.ID
        left outer join Pizaria.Ingrediente on ing_ID=Ingrediente.ID
        where Ingrediente.quantidade_disponivel - PizaIngrediente.quantidade*MenuProduto.quantidade*@quantidade < 0
        or Bebida.quantidade_disponivel - MenuProduto.quantidade*@quantidade < 0
        and men_ID=@item_ID
        )
        begin
            rollback tran
            set @response='Number of Products not Available'
            return
        end

        update Pizaria.Ingrediente
        set quantidade_disponivel = quantidade_disponivel - PizaIngrediente.quantidade*MenuProduto.quantidade*@quantidade
        from Pizaria.MenuProduto join Pizaria.Piza on pro_ID=Piza.ID
        join Pizaria.PizaIngrediente on piz_ID=Piza.ID
        join Pizaria.Ingrediente on ing_ID=Ingrediente.ID
        where men_ID=@item_ID

        update Pizaria.Bebida
        set quantidade_disponivel = quantidade_disponivel - MenuProduto.quantidade*@quantidade
        from Pizaria.MenuProduto join Pizaria.Bebida on Bebida.ID=pro_ID
        where men_ID=@item_ID
    end
```

Figura 8 - Verificação da disponibilidade do stock e dedução dos itens requeridos ao stock para um item do tipo “Menu”.

User Defined Functions

Inicialmente criamos views para algumas tabelas, mas por recomendação do docente, achamos que estas seriam menos eficientes e que o uso de UDFs seria mais benéfico.

Criamos as UDF *avail*, que nos permite garantir que trabalhamos sempre com produtos que estiverem disponíveis, por exemplo, se uma piza necessitar de 2 unidades de queijo, garantimos que existe esse produto nessa quantidade em stock.

De modo geral, utilizamos as UDF para criar funções, umas bastantes simples outras mais complexas, e retornar ou uma tabela - UDF inline ou multi-statement table value (por ex.: mostrar o histórico de encomendas) com dados ou um certo valor - UDF escalar (por ex.: verificar validade de um desconto).

É importante referir a UDF *findBestEstafeta* porque a sua função é retornar o estafeta com menos encomendas ativas e é geralmente usada para saber a que estafeta atribuir a próxima encomenda requisitada. Não é necessário verificar se o estafeta está empregado, pois o trigger *delEstafeta* trata de remover as encomendas ativas deste, e por isso estar a verificar seria redundante. Depois de contar o número de encomendas de cada estafeta, é ordenado ascendentemente e selecionado o email da primeira linha (Figura 9).

```
declare @email nvarchar(255);
select top 1 @email = email
from Pizaria.Encomenda join Pizaria.Estafeta on estafeta_email=email
where estafeta_email = email
group by email, res_contato
HAVING res_contato is not null and COUNT (estafeta_email) = (
    select MIN(num_enc) from (
            select count(estafeta_email) as num_enc
            from Pizaria.Encomenda join Pizaria.Estafeta on estafeta_email=email
            group by estafeta_email, res_contato
            HAVING res_contato is not null
    ) as EstafetaNumEncomendas
)
```

Figura 9 - Seleção de um estafeta com menos encomendas ativas.

O *getEncPrice* também se revela importante para calcular o preço de uma encomenda. Este recebe o ID da encomenda a qual pretendemos saber o preço, verifica se a encomenda se localiza na tabela *Encomenda* ou *EncomendaEntregue*, e calcula a soma do preço de todos os itens nela contidos, considerando o preço e a quantidade. O desconto, se existente, é aplicado sobre o valor da soma calculado anteriormente (Figura 10).

```
if (exists(select top 1 ID from Pizaria.Encomenda where ID = @ID))
    set @val = (select sum(preco*quantidade) as total
            from Pizaria.Encomenda
            join Pizaria.EncomendaItem on Encomenda.ID=enc_ID
            join Pizaria.Item on item_ID=Item.ID
            where enc_ID = @ID) * Pizaria.getDesconto(@ID)
```

Figura 10 - Soma do valor dos itens de uma encomenda ativa.

O *showEstafeta* é uma UDF do tipo multi-statement table-valued function que retorna os atributos dos estafetas colegas (associados ao mesmo restaurante) de um determinado estafeta mais os seus números de encomendas entregues e por entregar. Para tal, para cada estafeta é necessário selecionar as suas encomendas ativas e contar o número de linhas selecionadas, fazendo o mesmo para as encomendas entregues. O SELECT final está representado na Figura 11.

```
select E.email, [Utilizador].nome,
(select count(*) as count_enc from Pizaria.Estafeta
join Pizaria.EncomendaEntregue on email=est_email where email=E.email)
as delivered,
(select count(*) as count_enc from Pizaria.Estafeta
join Pizaria.Encomenda on email=estafeta_email where email=E.email)
as to_deliver
from Pizaria.Estafeta as E join Pizaria.[Utilizador] on E.email=Utilizador.email
join Pizaria.Restaurante as R on res_contato=R.contato
where R.contato = (select res_contato from Pizaria.Estafeta where email=@email)
```

Figura 11 - Tabela retornada pela UDF *showEstafeta*.

Segurança

Uma das principais preocupações na realização deste projeto foi os aspetos de segurança. Assim, com o intuito de aumentar ao máximo a robustez da nossa base de dados, adotamos as seguintes medidas:

- Utilizar de preferência SQL parametrizado ao invés de usar o SQL dinâmico.

  Exemplo comando sql não dinâmico :

  ```
  var command = new SqlCommand("select * from Encomenda where id = @someId");
  command.Parameters.Add(new SqlParameter("@someId", idValue));
  ```

  Exemplo comando sql dinâmico :

  ```
  var command = new SqlCommand("select * from Encomenda where id = " + idValue);
  ```

- Utilizar a codificação de dados para armazenar dados sensíveis. Foi referido um exemplo disso na procedure *register*, onde utilizamos processos de hash com salt para armazenar as palavras-passes.
- Não confiar nos dados inseridos nos campos de preenchimento da interface, fazendo sempre que considerado necessário a respetiva validação.
- Apresentação de mensagens de erro customizadas, de forma a diminuir a informação de erros passada ao utilizador.
- Uso de triggers, que são úteis para aumentar a segurança e evitar inconsistências.
- Uso de Stored Procedures para a implementação de regras de segurança e para algumas validações de dados recebidos por utilizadores.

Conclusão

Este projeto foi de grande ajuda para tecer as capacidades necessárias à realização de uma pequena aplicação assente numa base de dados robusta. Para tal, aprendemos todo o processo para chegar ao ponto atual, desde a planificação da base de dados a partir de uma análise detalhada e consoante os requisitos da visão final desejada, até à ponderação das melhores estratégias a seguir para a construção e manipulação da mesma. Nesta última situação tivemos muita preocupação com a integridade, consistência e recuperação inteligente dos dados, não esquecendo também da eficiência no acesso aos dados e da segurança.

Por fim, consideramos que este trabalho prático foi sem dúvida fundamental e útil para o nosso percurso não só devido à experiência que ganhamos mas também porque nos apercebemos da importância de ter uma base de dados organizada e confiável.

Notas Finais para o Docente

Para mudar o utilizador da base de dados na SQLConnection, basta aceder ao ficheiro Program.cs dentro da pasta Interface.

O script de instalação, a apresentação e o demo da base de dados está na pasta principal.