Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2018/2019

GELATUM

Adriana Meireles, Nuno Silva, Ricardo Pereira, Shahzod Yusupov

Novembro 2018

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

GELATUM

Adriana Meireles, Nuno Silva, Ricardo Pereira,

Shahzod Yusupov

Novembro 2018

Resumo

Este relatório foi elaborado no âmbito da Unidade Curricular de Base de Dados e retrata de uma forma pormenorizada a organização e execução de uma arquitetura de uma base de dados de uma gelataria com serviço de consulta e encomendas *online*, desde o modelo conceptual até à implementação do modelo físico.

Primeiramente, iremos retratar as fases iniciais do projeto, nomeadamente a contextualização, apresentação do sistema escolhido e análise de viabilidade do processo. Será ainda feito um levantamento dos requisitos onde estarão identificadas as entidades consideradas essenciais para este sistema.

Ao nível do modelo conceptual serão descritas as entidades e os relacionamentos e os respetivos atributos, sem esquecer a caracterização da associação dos mesmos.

Passando para o modelo lógico, aqui é aprofundado o estudo dos elementos referidos anteriormente, sendo descritos detalhadamente os vários tipos de relacionamentos, as várias associações e os atributos correspondentes. É feita, também, a validação deste modelo através de vários métodos.

Por fim, será feita a conversão para o modelo físico onde será usado o sistema de gestão de base de dados *MySQL*.

Este trabalho tem como objetivo a construção de um sistema de base de dados que seja capaz de gerir de uma forma correta a informação acerca de um sistema de gelataria *online*, permitindo o bom funcionamento do mesmo.

Área de Aplicação: Desenho e arquitetura de Sistemas de Bases de Dados.

Palavras-Chave: *GELATUM*, *SQL*, informação, dados, gelados, cliente, funcionário, pedido, bases de dados relacionais, análise de requisitos, entidades, atributos, relacionamentos, metodologia, modelo concetual, modelo lógico, modelo físico, sistema de gestão de base de dados, transações, gatilhos, procedimentos.

Índice

Índice de Figuras

Índice de Tabelas

1. Definição do Sistema

1.1. Contexto de aplicação do sistema

"Vai um gelado?" As referências acerca da origem desta sobremesa não são consensuais; ora apontam para os povos árabes ora para os chineses, que ao fazer a conservação de determinados alimentos, recorrendo a temperaturas baixas, perceberam que ingeri-los daquela forma, nomeadamente as frutas, era bastante agradável. Entretanto, esta base foi evoluindo e sofrendo alterações resultantes das experiências com novos ingredientes que acabaram por moldar tanto o sabor como a textura dos gelados tal como hoje os conhecemos. Atualmente, existe uma panóplia enorme de sabores de gelado que como base têm um creme de natas, açúcar e leite; temos os gelados normais, com aromas frutíferos, e outros mais arrojados, como por exemplo, os gelados com biscoitos, cerveja, carne, especiarias, queijo ou ainda gelados com tinta de lula.

O mercado dos gelados encontra-se em franca expansão e é recorrente encontramos gelatarias artesanais "ao virar da esquina". É verdade que esta sobremesa tem o seu pico de consumo no verão, pois a sua frescura contrasta com as temperaturas altas que se fazem sentir, não obstante, é uma sobremesa bastante solicitada nas restantes épocas.

Querendo atender à crescente procura desta iguaria, o senhor Oleb decidiu montar um negócio de venda de gelados na cidade de Braga, junto ao polo da Universidade do Minho. A esta gelataria, o senhor Oleb quer associar uma aplicação que permita aos seus clientes a consulta, e posteriormente, a venda dos gelados disponíveis de uma forma prática e não presencial, uma vez que a zona é maioritariamente frequentada por estudantes que facilmente a podem instalar e usar no seu *smartphone*.

1.2. Apresentação do Caso de Estudo

GELATUM, a gelataria da Universidade do Minho, terá uma aplicação à qual terá que ser associada uma base de dados. O objetivo desta aplicação será permitir aos clientes a escolha de um gelado a qualquer hora, em qualquer lugar. Para uma maior proximidade com o cliente, a aplicação vai solicitar-lhe que faça o seu registo com os seus dados pessoais. Findo o registo, o cliente pode entrar na aplicação com o seu perfil quando lhe aprouver e verá imediatamente os gelados mais solicitados pelos clientes da *GELATUM*.

A aplicação irá listar todos os gelados que a gelataria vende, tendo cada um deles a informação acerca do preço, da disponibilidade, do valor nutricional e dos ingredientes utilizados na sua confeção. Estas informações são extremamente importantes para que o cliente se sinta tão ou mais informado que o que se sentiria se estivesse fisicamente na *GELATUM*, algo que faz os clientes primarem pelo uso da aplicação, o que valorizará o que nela se investiu e facilitará o trabalho dos funcionários.

Para motivar os clientes para o uso desta aplicação o senhor Oleb pensou também em incluir-lhe uma funcionalidade onde possam visualizar todas as promoções e ofertas em vigor num determinado momento que irão incidir sobre um produto específico.

Os clientes da *GELATUM* que tiverem a aplicação instalada e a respetiva conta criada, poderão ainda usufruir de um serviço que o senhor Oleb acha que irá dinamizar a sua gelataria; as encomendas *online*. Com este serviço é pretendido que o cliente, após ter escolhido o gelado, faça imediatamente a sua compra, sendo constantemente informado do tempo que ainda falta para o pedido estar pronto.

A base de dados terá que albergar toda esta informação de modo a que a aplicação possa usufruir dos dados de que necessita de forma organizada, rápida e eficaz e que não comprometa o seu bom funcionamento.

1.3. Fundamentação da implementação da base de dados

O pedido de gelados implica sempre algum gasto de tempo por parte do cliente, quer na escolha dos pedidos que pretende quer na fila de espera para a escolha dos mesmos.

Com o desenvolvimento deste projeto pretendemos simplificar todo o processo desde o momento em que o cliente efetua o seu pedido até à entrega do produto. Os pedidos efetuam-se *online*, facilitando a visualização dos mesmos por parte do funcionário, assim como a disponibilidade dos gelados existentes em catálogo.

Assim sendo, vamos construir uma base de dados simples, que nos fará estar focados sempre na realidade e nos problemas que possam surgir aos clientes. Podemos assim, dar a entender como funciona um sistema de pedidos de gelados deste estilo, facilitando a gestão dos dados em todo o processo dos pedidos efetuados online, bem como o armazenamento de toda a informação numa base de dados.

Em suma, o cliente irá poupar um dos recursos mais valiosos, nos dias de hoje, tempo.

1.4. Estrutura do Relatório

Após termos apresentado o caso de estudo escolhido e a motivação que está por trás deste, nos seguintes capítulos deste projeto será exposta a análise e a construção do modelo conceptual da base de dados, a transição deste para o seu esquema Lógico e por último a

implementação do Modelo Físico. Estas etapas serão mostradas em pormenor nos próximos capítulos do relatório.

Na análise do caso de estudo apresentamos uma breve descrição de entidades e dos relacionamentos entre elas, bem como os atributos que as caracterizam. A forma como um utilizador irá interagir com a informação da base dados é descrita sob a forma de possíveis operações de manipulação e consulta através de exemplos práticos.

De seguida, é exposta a analise do modelo conceptual da base de dados, onde são apresentadas de forma tabular as entidades e relacionamentos envolvidos, bem como, os respetivos atributos. Para cada entidade, é apresentada a informação sobre a sua descrição e ocorrência, enquanto que, para os relacionamentos é caracterizada a sua multiplicidade entre as entidades envolvidas. Para os atributos, são identificados os tipos de dados e o seu tamanho, bem como se estes podem ou não ser nulos, multivalorados ou derivados, após isto, determina-se também o domínio de valores possíveis. Apresentam-se, para cada entidade, as chaves candidatas, onde é identificada a chave primária, bem como as chaves alternativas.

1.5. Análise de Viabilidade do processo

Este projeto consiste na implementação de um sistema de base de dados relacional (*SBDR*). Uma base de dados relacional é composta por um conjunto de tabelas e associações entre estas. A associação entre os dados é o ponto forte dos sistemas relacionais. Estas tabelas são formadas por linhas e colunas onde se encontram os dados, que numa base de dados relacional são representados como valores nas colunas das tabelas.

A elaboração de *SBDR*, garante inúmeras vantagens que tornam este projeto bastante viável e favorável. Ao contrário de um ficheiro, as tabelas são muito vantajosas na medida em que podem ter mais do que um propósito e os seus dados podem ser classificados com diferentes formas e formatos.

Em relação à implementação da base de dados podemos destacar as cinco vantagens mais relevantes:

- **Resposta rápida aos pedidos de informação**
  Um dos ganhos a nível operacional é a resposta rápida aos pedidos de informação. Como os dados estão integrados numa única estrutura, as respostas a questões complexas processam-se mais rapidamente.

- **Acesso múltiplo**
  O *software* de gestão de base de dados permite que os dados sejam acedidos de várias formas, nomeadamente, podem ser vistos através de pesquisas sobre qualquer um dos campos da tabela.

- **Flexibilidade**

  A independência entre os dados e programas faz com que qualquer modificação num desses elementos não implique alterações radicais no outro.

- **Integridade da Informação**

  Dada a obrigação de não permitir a redundância, as alterações de dados são feitas num só sítio, evitando-se assim possíveis conflitos entre as diferentes versões da mesma informação.

- **Melhor gestão da informação**

  A localização central dos dados permite saber sempre como e onde se encontra a informação. Deste modo, podemos concluir que a implementação da base de dados tem a finalidade de simplificar e favorecer a forma como toda a informação é processada e armazenada, contribuindo para uma melhor gestão da mesma.

2. Levantamento e Análise de Requisitos

2.1. Método adotado

Para uma base de dados ser bem implementada, é necessário conhecer na íntegra a aplicação e a gelataria, isto é, as suas funcionalidades e tudo aquilo com que interagem. Por esse motivo, para além das exigências do senhor Oleb, a nossa equipa decidiu frequentar alguns estabelecimentos que se dedicam à venda de gelados, com o objetivo de ficar mais ciente da realidade destes espaços.

2.2. Requisitos Levantados

2.2.1. Requisitos de Descrição

Após uma análise cuidada e criteriosa decidimos reunir toda a informação que se mostrou relevante para a posterior construção da base de dados. A abordagem desses dados será feita tendo em conta os cinco principais elementos deste sistema: o cliente, o pedido, o funcionário, o gelado e o extra do gelado.

- **Cliente**

  É o motivo pelo qual a gelataria existe! As informações do cliente são absolutamente necessárias para que se lhe possa associar um pedido. Por essa razão, decidimos que um cliente terá de se registar na aplicação e, nesse momento, terá de fornecer um **nome de utilizador** e uma ***password*** (início de sessão na aplicação; ambos sem espaçamentos), o **nome completo**, o ***email***, o **número de telemóvel,** a **data de nascimento** e o **número de identificação fiscal** (emissão de faturas fiscais).

  Para fazer a escolha do(s) gelado(s), o cliente deve iniciar sessão na aplicação, consultar o catálogo, selecioná-lo(s), adicioná-lo(s) ao pedido e efetuar o pagamento através de um dos métodos disponíveis. Decorrido o tempo estimado de preparação do pedido, este deverá ser levantado na loja física.

- **Pedido**

  O alicerce do sistema de base de dados que estamos a construir é o pedido. Todo o processo de consulta e encomenda de gelados necessitará sempre de informação contida noutras entidades, pelo que esta será a nossa entidade central.

  Concluído o início de sessão, o cliente adiciona ao pedido um ou mais gelados, com os respetivos extras, se os tiver, sendo-lhe atribuído uma **data**. Terão de ficar registados os seguintes dados: **identificador**, **número de contribuinte** e **nome** do cliente e o **nome**, o **identificador** e o **preço** do gelado. Para que o pedido seja finalizado, o cliente necessita de validar os cupões de promoções, caso os tenha, sendo aplicado um **desconto**, devendo efetuar o pagamento do **valor total**, ficando o pedido registado como **pago**.

  Findo, o pedido é atribuído a um funcionário cujo **identificador** fica registado e é também estabelecido o **tempo** previsto para a sua preparação. Para o levantamento do(s) produto(s), o cliente terá de mostrar ao funcionário o **identificador** do pedido que realizou. Assim que o tiver, o funcionário deverá marcá-lo como **recolhido**.

- **Funcionário**

  Assim que um novo pedido chega eletronicamente à loja física, é-lhe imediatamente atribuído um funcionário (previamente autenticado) que fica incumbido de o preparar. Para esta entidade, necessitamos apenas de alguma informação que nos permita identificar quem processou o pedido, caso o cliente tenha algum problema ou caso pretenda louvar o funcionário. Posto isto, para esta entidade devem ser conhecidos o **nome**, o **identificador** e uma ***password***, sendo os dois últimos os dados requeridos para a autenticação na aplicação.

- **Gelado**

  Se por um lado o cliente é o motivo pelo qual a *GELATUM* existe, os gelados são o motivo pelo qual alguém é cliente da *GELATUM*. Nesta gelataria, a diversidade de produtos é enorme, logo é impreterível a sua organização numa base de dados onde para cada um constem um **nome**, a **composição**, o **preço** e o ***stock***. Todos estes dados serão disponibilizados ao cliente num catálogo que se divide em várias secções, cada uma correspondente a um gelado que pode ou não ter extras, de acordo com a escolha do cliente e da disponibilidade de *stock* desse extra.

  Para a tornar mais rápida e eficiente, a identificação de um gelado far-se-á também com um **identificador.**

- **Extra**

  Em regra, um gelado nunca é consumido simples. Para além da diversidade de ingredientes e sabores que os caracterizam, a maioria das gelatarias, como a do caso em estudo, oferece ainda a possibilidade de adicionar ao gelado nenhum, um ou vários

extras. Estes, não são mais que um complemento que permitem que o cliente "personalize" o seu gelado da forma que mais lhe agrada. Obviamente, existe um **preço** e um ***stock*** associados a cada extra que está adicionado ao gelado, assim como um **identificador**, um **nome** e a sua **composição.**

É importante frisar que a **quantidade** de um determinado tipo de gelado com determinado(s) extra(s), que é adicionado ao pedido, deve ficar registada, assim como a **quantidade** de extras que são adicionados a um gelado.

2.2.2. Requisitos de Exploração

Um dos propósitos das bases de dados é ter acessível a informação que é pedida, nem mais, nem menos! Por exemplo, se o administrador quiser obter uma lista de nomes dos utilizadores registados, então deverá ser-lhe facultada apenas uma lista com todos os nomes, sem qualquer outra informação. Neste exemplo, teriam que ser selecionados apenas os nomes e essa informação era posteriormente enviada ao administrador, não obstante, a tarefa não seria tão simples se o administrador quisesse introduzir mais condições no seu pedido, como por exemplo, os nomes de todos os utilizadores que são clientes, que pediram gelados de baunilha e que nasceram antes do ano 2000. Esta exigência poderia muito bem ser um requisito necessário ao administrador para poder fazer análises acerca dos produtos que se devem vender. Assim, a gestão dos dados apresentados deve ser feita pelo sistema de base de dados, pois controla toda a informação existente garantindo assim uma maior eficiência.

Por conseguinte, em baixo, expomos alguns requisitos necessários à manipulação da base de dados, tendo em conta que os mais básicos, também importantes, não os vamos explicitar.

- Consulta de todos os clientes por parte do administrador;
- Consulta de todos os funcionários por parte do administrador;
- Consulta de todos os gelados por parte de qualquer utilizador;
- Qualquer utilizador pode pedir ao sistema para filtrar os gelados cujo preço é inferior/superior a outro por ele especificado;
- Qualquer utilizador pode pedir ao sistema para filtrar os gelados de acordo com a sua composição, ou seja, dado um ingrediente, todos os gelados que o tenham não aparecem, ou vice-versa.

- O administrador (senhor Oleb) pode conhecer os 3 gelados mais vendidos num dado mês e ano;

- Exibir o cliente mais velho que fez um pedido;

- Exibir nomes de todos os funcionários que processaram pedidos com gelados de morango por ordem decrescente de vendas;

- Mostrar quais os clientes que mais gelados compraram a partir de determinada data, por ordem decrescente;

- Mostrar os clientes com idade compreendida entre 18 e 23 anos e que tenham comprado gelados de chocolate com *M&M'S*®, mais que uma vez;

2.2.3. Requisitos de Controlo

Uma base de dados é, por definição, uma ferramenta que permite guardar e gerir da forma mais eficiente toda a informação que um dado sistema (informático) possui. É importante que todos os utilizadores da aplicação possam aceder aos dados que lhes dizem respeito, não obstante, é ainda mais importante garantir que esses mesmo utilizadores não acedam a informações alheias e desnecessárias ao uso da aplicação, algo que está bem patente nas mais recentes leis de proteção de dados. Por esse motivo, é imprescindível definir várias formas de monitorizar e restringir o acesso a vários dados por parte dos utilizadores. Posto isto, na aplicação foram registados quatro tipos de utilizadores: o administrador, o cliente e o funcionário.

- **Administrador (senhor Oleb)**

  Na aplicação da *GELATUM*, é necessário que haja alguém que tenha as permissões necessárias para resolver qualquer problema que surja com qualquer uma das informações guardadas na base de dados. Esse alguém deve ser uma pessoa idónea e honesta pois é aquele que tem mais permissões de acesso à base de dados. Sendo assim, o administrador deverá poder alterar, remover, inserir e consultar qualquer informação da base de dados acerca de outros utilizadores. A única operação que o senhor Oleb não pode fazer é a remoção de toda a informação de uma base de dados.

- **Cliente**

  Este utilizador deverá poder fazer qualquer operação que diga respeito aos seus dados pessoais (número de identificação fiscal, nome, *password*, nome de

utilizador, data de nascimento, *email* e número de telemóvel), com a exceção de não poder remover nada, mesmo que o utilizador queira eliminar a sua conta. Apenas poderá consultar todos os dados relativos ao seu pedido, ao gelado e ao extra, caso exista, sendo que destes só pode alterar as quantidades.

- **Funcionário**

  Todos os dados que sejam necessários para o processamento de um pedido devem ser disponibilizados ao funcionário. Posto isto, este utilizador pode consultar a informação disponível na base de dados relativa ao cliente que faz o pedido, exceto a *password*, o nome de utilizador, o *email* e o número de telemóvel. Poderá ainda consultar os seus dados, mas apenas alterar a *password*.

2.3. Análise geral dos requisitos

O levantamento de requisitos e a sua posterior análise, como já foi dito, é algo extremamente importante para estarmos bem cientes de toda a informação que precisamos guardar na base de dados. Por esse motivo, detalhámo-los o mais que pudemos e de seguida apresentamo-los ao senhor Oleb. Em determinados aspetos, o nosso cliente não estava satisfeito com o lhe que apresentamos, talvez por má compreensão da nossa parte, pelo que tivemos que nos sujeitar a algumas alterações. Entre elas encontravam-se a remoção da validade do gelado, do número de identificação fiscal, da morada, do número de telemóvel e do *email* do funcionário, a adição de uma data de nascimento ao cliente e de informação acerca do pagamento e da recolha do pedido, etc. Uma vez corrigidas as incongruências existentes nos requisitos, voltamos a apresentá-los ao senhor Oleb e este mostrou-se mais satisfeito e confiante acerca da informação que tínhamos concluído. Ora, acabados o levantamento e a análise de requisitos, ficamos finalmente aptos para iniciar a construção de um modelo concetual que terá toda a sua estrutura dependente da informação recolhida, explícita anteriormente.

3. Modelação Concetual

3.1. Apresentação da abordagem de modelação realizada

Como não se trata de um projeto muito complexo, apenas utilizamos uma vista de desenvolvimento,usando a ferramenta *BrModel*.

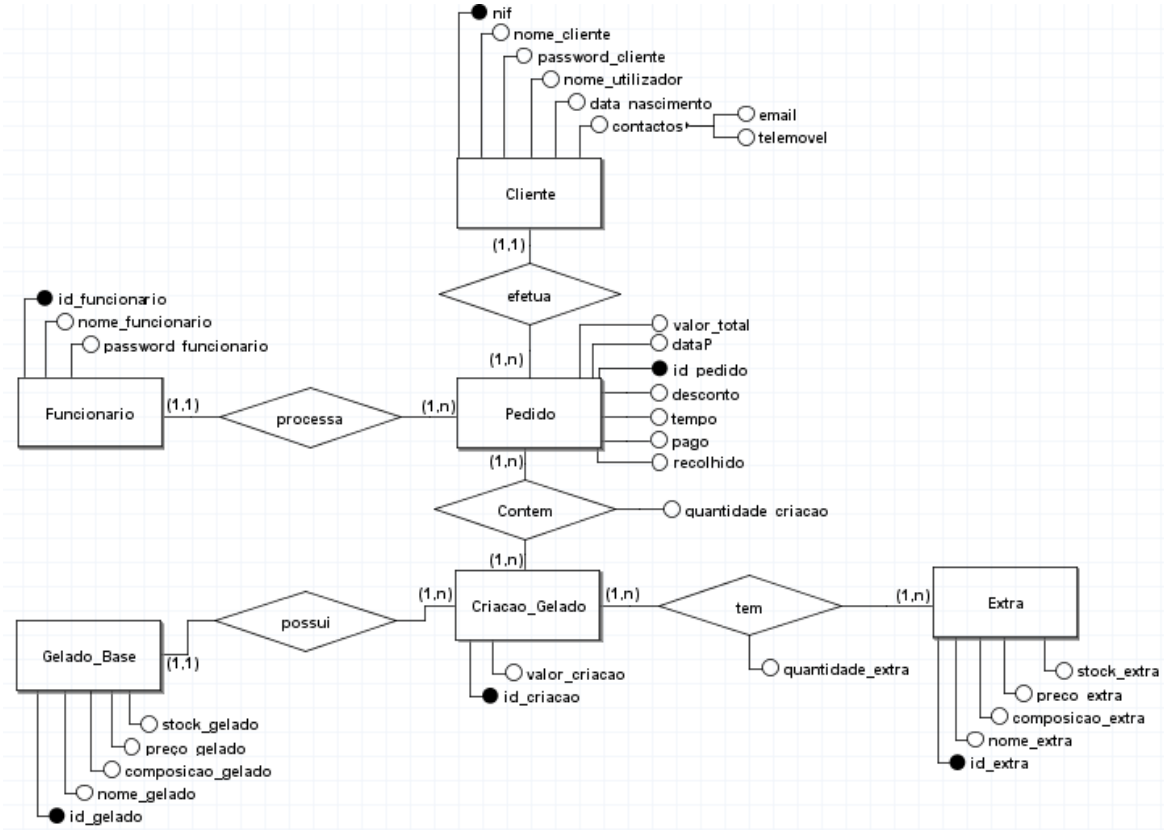

Figura 1. Esquema concetual

3.2. Identificação e caracterização das entidades

Para a elaboração do modelo conceptual começamos por identificar as entidades do problema. É essencial uma observação cuidadosa aos requisitos de modo a não fazer uma seleção incorreta de entidades, pois, por vezes, existem objetos com uma importância para o modelo, mas não são necessariamente uma entidade.

Assim sendo, iremos ilustrar as entidades escolhidas bem como o seu significado e a justificação da sua ocorrência.

- **Cliente**

  Termo geral que descreve todos os clientes do registados na base de dados. Cada cliente possui atributos próprios, tem uma existência autónoma e pode ser identificado univocamente, sendo assim uma entidade.

- **Pedido**

  Entidade que indica todos as criações de gelados com extras que o cliente escolheu. O pedido só é preparado depois do cliente pagar.

- **Funcionário**

  Entidade que representa todos os funcionários da gelataria registados na base de dados, responsável por preparar os pedidos dos diversos clientes.

- **Criacao_Gelado**

  Entidade que representa a criação de um gelado que pode ou não ter extras.

- **Gelado_Base**

  Entidade que representa os vários tipos de gelados presentes na nossa gelataria, por exemplo, gelado de morango, chocolate, baunilha, etc.

- **Extra**

  Outra entidade relevante é o extra que torna os gelados dos clientes mais saborosos e únicos com a adição de *toppings*, *chantilly*, etc., ao invés de apenas ser um gelado de um determinado sabor.

3.2.1. Dicionário de dados das entidades do modelo

Entidade	Descrição	Ocorrência
Cliente	Entidade representativa da pessoa que compra um ou mais gelados na nossa gelataria.	O Cliente é uma entidade fundamental pois é ele quem faz uso da nossa empresa.
Pedido	Entidade que contém toda a informação de um pedido que é feito por um cliente.	O pedido define a lista de gelados, com ou sem extra, escolhidos pelos nossos clientes.
Funcionario	Entidade responsável pela preparação do(s) gelado(s) dos clientes.	O funcionário é a base da preparação do(s) gelado(s) na gelataria.
Criacao_Gelado	Entidade responsável pela junção do gelado base com o(s) extra(s) se existir(em).	Criacao_Gelado é a base para a criação dos gelados de acordo com a escolha e preferência do cliente.
Gelado_Base	Entidade representativa do gelado base (morango, baunilha, chocolate, etc.) que os clientes podem escolher.	O Gelado é outra entidade muito importante pois é a razão do funcionamento da nossa gelataria.
Extra	Entidade representativa de um extra (*topping*, *chantilly*, bolacha, etc.) que os clientes podem escolher.	O Extra é outra entidade relevante que permite personalizar os gelados de acordo com a preferência do cliente.

Tabela 1. Dicionário de dados das entidades do modelo

3.3. Identificação e caracterização dos relacionamentos

- **Cliente e Pedido**

  O relacionamento entre a entidade Cliente e a entidade Pedido caracteriza-se por uma cardinalidade de 1 para N, uma vez que um cliente pode fazer vários pedidos.

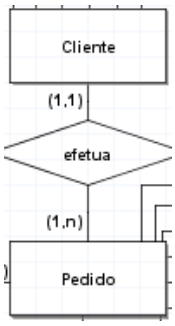

Figura 2. Relacionamento entre Cliente e Pedido

- **Funcionario e Pedido**

  O relacionamento entre a entidade Funcionário e a entidade Pedido caracteriza-se por uma cardinalidade de 1 para N, uma vez que um funcionário pode fazer vários pedidos, mas um pedido só é feito por um funcionário.

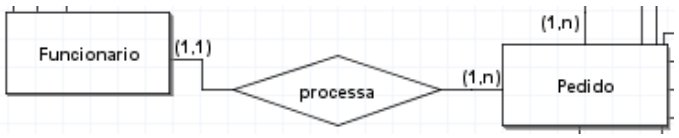

Figura 3. Relacionamento entre Funcionario e Pedido

- **Pedido e Criacao_Gelado**

  O relacionamento entre a entidade Pedido e a entidade Criacao_Gelado caracteriza-se por uma cardinalidade de N para N, porque um pedido pode ser constituído por várias criações de gelados e estas podem fazer parte de diversos pedidos.

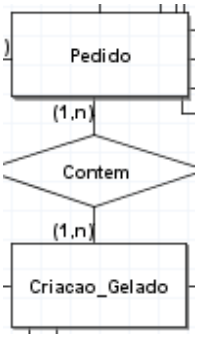

Figura 4. Relacionamento entre Pedido e Criacao_Gelado

- **Criacao_Gelado e Gelado_Base**

  O relacionamento entre a entidade Criacao_Gelado e a entidade Gelado_Base caracteriza-se por uma cardinalidade de N para 1, porque as criações dos diversos gelados têm de ter um gelado base.

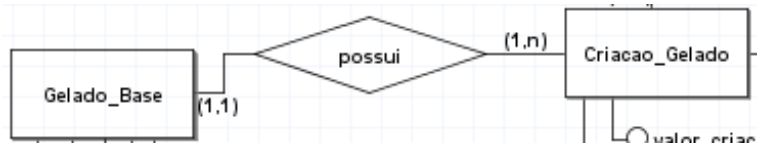

Figura 5. Relacionamento entre Criacao_Gelado e Gelado_Base

- **Criacao_Gelado e Extra**

  O relacionamento entre a entidade Criacao_Gelado e a entidade Extra caracteriza-se por uma cardinalidade de N para N, porque as criações de gelados podem ser constituídas por vários extras e um extra pode fazer parte de diversas criações.

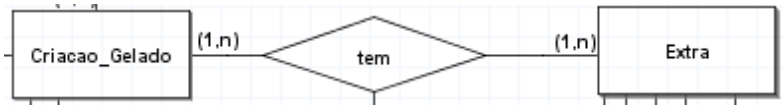

Figura 6. Relacionamento entre Criacao_Gelado e Gelado_Extra

3.3.1. Dicionário de relacionamentos do modelo

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Cliente	1..1	efetua	1..N	Pedido
Funcionario	1..1	processa	1..N	Pedido
Pedido	1..N	contem	1..N	Criacao_Gelado
Criacao_Gelado	1..N	possui	1..1	Gelado_Base
Criacao_Gelado	1..N	Tem	1..N	Extra

Tabela 2. Dicionário de relacionamentos

3.4. Identificação e caracterização da Associação dos Atributos com as Entidades e Relacionamentos

Depois de termos identificado todas as entidades envolventes no nosso problema e de as termos relacionado entre si, passamos agora ao momento de identificarmos todas as suas propriedades/atributos que consideramos relevantes para o problema em causa.

Entidade	Atributo	Descrição	Tipo de Dados e Tamanho	Null	Tipo de atributo
Cliente	nif	Número que identifica um cliente	INT	Não	Chave primária
	nome_cliente	Nome e apelido do cliente	VARCHAR (45)	Não	Simples
	data_nascimento	Data de nascimento do cliente (para efeitos de desconto)	DATE	Não	Simples
	nome_utilizador	Nome usado para se registar e iniciar sessão na aplicação	VARCHAR (20)	Não	Simples
	password_cliente	*Password* associada ao nome do utilizador para o início de sessão na aplicação	VARCHAR (20)	Não	Simples
	contactos	----------	----------	----	Composto
	email	*Email* do cliente	VARCHAR (45)	Não	Simples
	telemovel	Telemóvel do cliente	INT	Não	Simples
Pedido	id_pedido	Número que identifica um pedido	INT	Não	Chave primária
	dataP	Data do pedido	DATETIME	Não	Simples
	valor_total	Preço do pedido	DECIMAL(5,2)	Não	Simples
	tempo	Duração, em minutos, da preparação do pedido	INT	Não	Simples
	pago	Indica se o pedido foi pago ou não	BOOLEAN	Não	Simples
	recolhido	Indica se o pedido foi ou não recolhido no estabelecimento	BOOLEAN	Não	Simples
	desconto	Desconto aplicado após a junção do gelado base com os respetivos extras(o cliente também tem desconto no aniversário)	DECIMAL(3,2)	Não	Simples
Funcionário	id_funcionario	Número que identifica um funcionário	INT	Não	Chave Primária
	nome_funcionario	Nome do funcionário	VARCHAR (45)	Não	Simples
	password_funcionario	*Password* associada ao id para o início de sessão na aplicação	VARCHAR (20)	Não	Simples

Criacao_Gelado	id_criacao	Número que identifica uma criação	INT	Não	Simples
	valor_criacao	Preço de uma criação de gelado	DECIMAL(2,2)	Não	Simples
Gelado_Base	id_gelado	Número que identifica um gelado	INT	Não	Chave Primária
	nome_gelado	Nome do gelado	VARCHAR (45)	Não	Simples
	composicao_gelado	Constituição de cada gelado	VARCHAR (300)	Não	Simples
	preco_gelado	Preço do gelado	DECIMAL(2,2)	Não	Simples
	stock_gelado	Quantidade disponível de cada gelado base	INT	Não	Simples
Extra	id_extra	Número que identifica um extra	INT	Não	Chave Primária
	nome_extra	Nome de um extra	VARCHAR (45)	Não	Simples
	composicao_extra	Constituição de um extra	VARCHAR (100)	Não	Simples
	preco_extra	Preço de um extra	DECIMAL(2,2)	Não	Simples
	stock_extra	Quantidade disponível de cada extra	INT	Não	Simples

Tabela 3. Dicionário de dados dos atributos das entidades do modelo

Relacionamento	Atributo	Descrição	Tipo de Dados e Tamanho	NULL	Tipo de atributo
Contém	quantidade_criacao	Quantidade de um gelado do mesmo tipo	INT	Não	Simples
Tem	quantidade_extra	Quantidade de um extra do mesmo tipo	INT	Não	Simples

Tabela 4. Dicionário de dados dos atributos dos relacionamentos do modelo

3.5. Detalhe ou generalização de entidades

Inicialmente tínhamos 4 entidades, o cliente, o funcionário, o pedido e o produto. No entanto, reparámos que seria necessário alterar a entidade produto e detalhá-la mais, pois a

nosso ver, um produto poderia ser um gelado, mas também um extra. Ora, o produto tem muitos mais atributos que não são relevantes para a caracterização do extra, portanto, decidimos criar uma nova entidade chamada extra que herda apenas alguns atributos da entidade produto. Esta alteração só ficou completa depois de atribuirmos um novo nome à entidade produto, passando esta a chamar-se gelado base.

Após outra análise ainda mais cuidada, verificámos que o modelo continuava a fazer o que não pretendíamos, isto é, os clientes não podiam escolher os extras que desejavam. No nosso modelo anterior tínhamos de ter menus já pré-definidos e o cliente iria ter de escolher entre eles, no entanto, isso limitaria as opções do consumidor. Por isso, foi criada uma nova entidade, Criacao_gelado, que faz a junção entre o gelado base (que pode ser só de morango, só de chocolate, etc.) e ao mesmo podem ou não ser adicionados os extras, de acordo com a preferência do cliente.

3.6. Apresentação e explicação do diagrama ER

Durante o processo de criação do modelo concetual deparamo-nos com várias alternativas, que no início nos pareceram bastante aceitáveis. Contudo, de modo a garantir a melhor acessibilidade e organização dos dados, chegamos a este modelo que foi o que nos pareceu mais assertivo e adequado à base de dados em questão.

Relativamente à entidade cliente: o mesmo regista-se na aplicação da gelataria através do nome de utilizador e respetiva *password*. Vai realizar as suas escolhas para o gelado e depois de tudo escolhido, o cliente tem de pagar para o seu pedido poder ser processado (pela entidade Funcionario) sendo-lhe dado o valor do tempo que irá demorar. Posteriormente ao pagamento, o pedido vai ser preparado pelo funcionário e o cliente vai recolher o seu pedido.

Em relação, à entidade Pedido, vai possuir o valor total de todas as criações feitas pelo cliente.

A entidade Criacao_Gelado inicialmente não existia, mas para os extras poderem ser escolhidos e adicionados ao pedido, foi necessária a sua implementação. Associada a esta entidade, estão as entidades Gelado_Base e Extra. A primeira corresponde à escolha do sabor do gelado. A segunda corresponde à escolha do(s) extra(s) para poderem ser adicionados ao gelado base. O cliente pode optar por não colocar extras.

Observando a cardinalidade dos relacionamentos, podemos verificar que um cliente pode fazer tantos pedidos quantos desejar. Cada pedido vai ser processado por um funcionário e pode ter várias criações de gelados. Cada criação de gelado tem um gelado base e nenhum ou vários extras.

3.7. Revisão e validação do modelo com o utilizador

Na finalização desta fase do projeto é necessária a revisão e aprovação do modelo por parte do utilizador na medida em que este consegue reconhecer que o modelo apresentado é uma representação fiável do que foi requisitado à gelataria. Assim, após averiguação e avaliação da documentação deste modelo de dados, nomeadamente dicionário de dados de entidades, o dicionário de dados de relacionamentos, o dicionário de dados de atributos e o diagrama ER, chegou-se à conclusão que o projeto foi aprovado pelo cliente, pois todas as entidades presentes no modelo são associadas aos objetivos e necessidades do cliente, assim como as suas relações correspondem à realidade do problema em causa.

4. Modelação lógica

4.1. Construção e validação do modelo de dados lógico

Nesta fase de transição de uma modelação conceptual para uma modelação lógica é necessário a derivação de todas as relações no modelo lógico que representem as entidades, relacionamentos e atributos que foram criados e usados anteriormente no modelo conceptual.

Para que tal objetivo seja alcançado, foquemos a nossa atenção nos elementos do modelo conceptual, tendo por base a *Database Definition Language*: entidades fortes, entidades fracas, relacionamentos binários um para muitos (1-N), relacionamentos binários um para um (1-1), relacionamentos recursivos um para um (1-1), relacionamentos superclasse/subclasse, relacionamentos binários muitos para muitos (N-M), relacionamentos complexos e atributos multivalorados.

4.1.1. Entidades Fortes

Uma entidade forte é uma entidade que tem atributos suficientes para formar uma chave primária, não dependendo da existência de nenhuma outra entidade.
No nosso modelo consideramos como entidades fortes as seguintes:

- **Funcionario** (id_funcionario, nome_funcionario, password_funcionario)
  Chave primária: id_funcionario

- **Cliente** (nif, nome_cliente, password_cliente, nome_utilizador, email, telemóvel, data_nascimento)
  Chave primária: nif

- **Gelado_Base** (id_gelado, nome_gelado, preco_gelado, stock_gelado, composicao_gelado)
  Chave primária: id_gelado

- **Extra** (id_extra, nome_extra, preco_extra, composicao_extra, stock_extra)
  Chave primária: id_extra

4.1.2. Entidades Fracas

Uma entidade fraca é uma entidade cuja existência depende de outra entidade e a sua existência só, não faz sentido. No nosso modelo consideramos como entidades fracas:

- **Pedido** (id_pedido, data, valor_total, tempo, pago, recolhido, id_funcionario, nif)
  Chave primária : id_pedido

- **Criacao_Gelado** (id_Criacao, valor_criacao, id_gelado)
  Chave primária: id_Criacao

4.1.3. Relacionamentos binários um para muitos (1-N)

Nesta particularidade de relacionamento entre duas entidades, a entidade com multiplicidade N adquire um novo atributo considerado como chave estrangeira, que não é nada mais nada menos do que a chave primária da entidade de multiplicidade 1. O nosso modelo apresenta três relacionamentos deste tipo (Funcionario-Pedido, Cliente-pedido e Gelado_Base-Criacao_Gelado).

- **Pedido** (id_pedido, data, valor_total, tempo, pago, recolhido, id_funcionario, nif)
  Chave primária : id_pedido
  Chave estrangeira : nif (proveniente da relação Cliente), id_funcionario (proveniente da relação Funcionario).

- **Funcionario** (id_funcionario, nome_funcionario, password_funcionario)
  Chave primária: id_funcionario

- **Cliente** (nome_cliente, password_cliente, nome_utilizador, nif, email, telemóvel, data_nascimento)
  Chave primária: nif

- **Criacao_Gelado** (id_Criacao, valor_criacao, id_gelado)
  Chave primária: id_criacao
  Chave estrangeira: id_gelado

- **Gelado_Base** (id_gelado, nome_gelado, preco_gelado, stock_gelado, composicao_gelado)
  Chave primária: id_gelado

4.1.4. Relacionamentos binários um para um (1-1)

Este tipo particular de relacionamentos binários pode ser de participação obrigatória nos dois lados da relação, de participação obrigatória num dos lados da relação ou então de participação opcional nos dois lados da relação. No caso do nosso modelo em particular não consideramos nenhum relacionamento binário de um para um (1:1).

4.1.5. Relacionamentos recursivos de um para um (1-1)

No nosso modelo conceptual para o projeto em causa, não figura nenhum relacionamento recursivo de um para um (1:1).

4.1.6. Relacionamentos superclasse/subclasse

A nossa base de dados não inclui relacionamentos deste tipo.

4.1.7. Relacionamentos binários muitos para muitos (N-M)

Uma relação do tipo n para m, é uma relação de vários para vários, ou seja, entre duas tabelas A e B, a várias ocorrências da tabela A podem corresponder várias ocorrências da tabela B, e vice-versa.

Para toda e qualquer relação do tipo n para m, há que decompor a relação em duas do tipo 1 para n, ou seja, irá ser necessário criar uma nova tabela, onde a mesma irá conter as chaves primárias das tabelas envolvidas, chaves estas que se irão tornar numa chave composta da nova tabela.

O nosso modelo apresenta dois relacionamentos deste tipo (Criacao_Gelado-Extra e Pedido-Criacao_Gelado).

- **Extra** (<u>id_extra</u>, nome_extra, preco_extra, composicao_extra, stock_extra)
  Chave primária: id_extra
  Tabela criada a partir do relacionamento (Criacao_Gelado - Extra) :

- **Criacao_Gelado__Extra** (<u>id_Criacao</u>, <u>id_extra</u>, quantidade_extra)
  Chave primária composta : id_Criacao , id_extra
  Tabela criada a partir do relacionamento (Pedido - Criacao_Gelado):

- **Pedido__Criacao_Gelado** (<u>id_pedido</u>, <u>id_gelado</u>, quantidade_criacao)
  Chave primária composta: (id_pedido, id_criacao)

4.1.8. Relacionamentos complexos

No nosso modelo concetual para o projeto em causa, não existe nenhum relacionamento complexo.

4.1.9. Atributos multivalorados

Este tipo particular de atributo pode assumir diferentes valores para a mesma entidade. No caso do nosso modelo não existem atributos deste tipo.

4.2. Desenho do modelo lógico

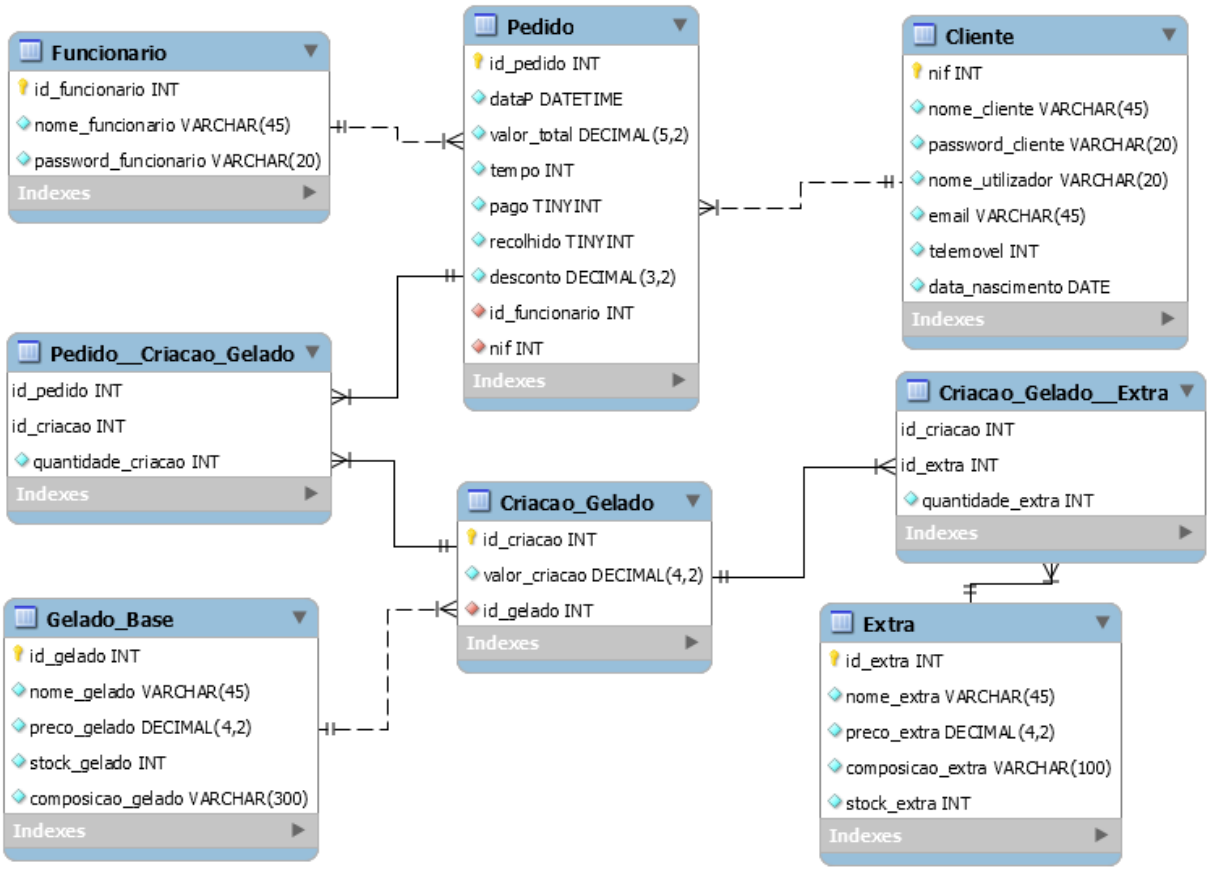

Figura 7. Esquema lógico

4.3. Validação do modelo através da normalização

A normalização de um modelo relacional tem como objetivo principal resolver problemas de atualização de bases de dados, minimizando redundâncias. Pode ser vista como o processo no qual são eliminados esquemas de relações (tabelas) não satisfatórios, decompondo-os, através da separação de seus atributos em esquemas de relações menos complexas, mas que satisfaçam as propriedades desejadas. Apesar de se poderem considerar outras fases na normalização de informação, normalmente, normalizar até à 3ª Forma Normal (3FN) é suficiente para garantir a inexistência de redundância informacional.

Para cada forma normal iremos mostrar exemplos do nosso modelo de modo a garantir que as regras são cumpridas.

4.3.1. 1FN – 1ª Forma Normal

Toda relação deve ter uma chave primária e deve-se garantir que todo atributo seja atômico.

Como podemos verificar toda a relação do nosso modelo apresenta uma chave primária e todos os atributos são atómicos, logo podemos concluir que respeita a 1ª forma normal.

4.3.2. 2ª Formal Normal

Uma relação está na 2ª Forma Normal se está na primeira e se todos os atributos não-chave dependerem da totalidade da chave-primária (e não apenas de parte dela - Dependências funcionais parciais).

As únicas chaves primárias compostas presentes no nosso modelo pertencem à relação Pedido__Criacao_Gelado e Criacao_Gelado__Extra e os atributos dependem da totalidade da chave primária, logo o modelo respeita a 2ª forma normal.

4.3.3. 3ª Forma Normal

Uma relação está na 3ª Forma Normal se está na 2ª Formal Normal e nenhum atributo não-chave é transitivamente dependente da chave primária. Assim podemos concluir que o nosso modelo respeita a 3ª forma normal. Validação do modelo com interrogações do utilizador

Um dos requisitos que é necessário para que o modelo possa ser considerado válido é responder a todas as perguntas que o utilizador possa fazer. Assim sendo foram selecionadas as consideradas pertinentes cuja viabilidade vai ser verificada através da comparação com o modelo lógico definido:

4.4. Validação do modelo com as interrogações do utilizador

Um dos requisitos que é necessário para que o modelo possa ser considerado válido é responder a todas as perguntas que o utilizador possa fazer. Assim sendo foram selecionadas as consideradas pertinentes cuja viabilidade vai ser verificada através da comparação com o modelo lógico definido:

- **Os nomes dos três gelados mais vendidos em junho de 2015**

  Para ter acesso a esta informação são necessárias as tabelas Pedido, Pedido_Criacao_Gelado, Criacao_Gelado, Gelado_Base. Após reunirmos toda a informação filtramos os pedidos cuja data está entre o primeiro e último dia do mês de

junho de 2015. Agrupando esta informação por gelados, é feita uma soma das quantidades, que são posteriormente ordenas por ordem descrescente e são retiradas da tabela resultante as primeiras três linhas.

- **Exibir todos os funcionários que processaram pedidos com gelados de morango, apresentando os seus nomes por ordem decrescente de vendas**

  Neste caso, reunindo informações das tabelas Funcionario, Pedido, Pedido__Criacao_Gelado, Criacao_Gelado e Gelado_Base. Concatenando todas estas tabelas, filtrando todas as linhas correspondentes ao gelado de morango, agrupando as linhas cujos funcionário são os mesmos e somando as respetivas quantidades, obtemos os funcionários que processaram gelados de morango assim como a respetiva quantidade. Posteriormente, é feita uma ordenação por ordem decrescente dessa quantidade.

- **Exibir o número de identificação fiscal do cliente mais velho que efetuou um pedido**

  Para ter acesso a esta informação acede-se à tabela Pedido onde tem os números de identificação fiscal dos clientes, e à tabela Cliente para aceder às datas de nascimento e ordenar por ordem crescente (quanto menor o ano de nascimento, mais idade tem a pessoa). Depois extrai-se o primeiro e vê-se a que número de identificação fiscal corresponde.

4.5. Validação do modelo com as transações estabelecidas

Ao nível do modelo lógico, validamos as transações através de mapas de transações. Desta forma, é possível verificar visualmente as interações entre as diferentes Entidades, Atributos e Relacionamentos de forma a obter o resultado pretendido.

- **Registar um Pedido**

  Para registar um pedido é necessário primeiro verificar se o gelado_base em questão e o extra associado existem. Para tal, acede-se às tabelas Gelado_base e Extra e verifica-se os respetivos stocks. Caso estejam disponíveis atualiza-se o stock e acede-se à tabela Criacao_Gelado para conseguir atualizar a quantidade de extra associado ao pedido (esta ação é feita na tabela Criacao_Gelado__Extra). Seguidamente, acede-se à tabela Pedido parar poder atualizar a quantidade da Criacao_Gelado associado ao pedido (ação efetuada na tabela

Pedido__Criacao_Gelado). Por fim regista-se o pedido adicionando o id na tabela Pedido.

- **Remover um Pedido**

  Para remover um pedido acede-se à tabela Pedido_Criacao e atualiza-se a quantidade da Criacao_Gelado presente nesta tabela. Seguidamente, acede-se à tabela Criacao_Gelado__Extra para atualizar a quantidade de Extra associado ao respetivo pedido. Após esta ação recorre-se à tabela Criacao_Gelado para se ter acesso à tabela Gelado_Base e Extra ,onde se atualizam os respetivos *stocks*. Por fim, acede-se à tabela Pedido e elimina-se o id do pedido da tabela.

4.6 Reavaliação do modelo lógico

Não foi necessário reavaliar o modelo lógico porque a nosso ver, a normalização já se encontra feita.

4.7. Revisão do modelo lógico com o utilizador

Para dar como terminada esta fase, o modelo lógico deve ser visto na perspetiva do utilizador. Este processo tem um papel muito importante pois o utilizador tem que ter a capacidade de reconhecer, que o modelo lógico idealizado é uma representação real dos requisitos da gelataria que está a ser modelada. Por isso, tivemos de avaliar toda a documentação associada ao modelo lógico desde as entidades até aos relacionamentos entre elas passando pelos atributos. Ao efetuarmos a revisão do modelo foi-nos possível identificar todas as tabelas importantes no modelo e os seus diversos relacionamentos. Ao revermos o nosso dicionário de relacionamentos (tabela de relacionamentos) conseguimos justificar que os relacionamentos utilizados são aqueles que melhor se identificam com a realidade do problema. De seguida, validamos os atributos onde o utilizador consegue verificar toda a informação de uma entidade e ter uma melhor noção dos domínios de cada atributo. Sendo assim, o nosso modelo foi aprovado pelo cliente pois ele é visto como uma solução que responde a todas as necessidades pretendidas.

5. Implementação Física

5.1. Seleção do sistema de gestão de bases de dados

Para construirmos a Base de Dados proposta utilizamos como sistema de gestão de base de dados o *MySQL*. Esta decisão deve-se ao facto deste sistema ter sido o usado nesta Unidade Curricular, o que de certa forma veio facilitar a nossa implementação da Base de Dados.

5.2. Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em *SQL*

O processo de modelação de um esquema físico de uma base de dados envolve a tradução das relações base, definidas previamente no modelo lógico, de maneira a que estas e as suas restrições sejam suportadas pelo SGDB. Assim, para concretizar corretamente este tópico é necessária a conclusão de alguns passos, nomeadamente a descrição das relações base e o desenho das restrições gerais.

5.2.1. Descrição das relações base

- **Relação Cliente**

Domínio nif	→ Inteiro
Domínio nome_cliente	→ String de tamanho variável, de tamanho 45
Domínio nome_utilizador	→ String de tamanho variável, de tamanho 20
Domínio email	→ String de tamanho variável, de tamanho 45
Domínio password_cliente	→ String de tamanho variável, de tamanho 20
Domínio telemovel	→ Inteiro

Cliente(
nif → NOT NULL,
nome_cliente → NOT NULL,
nome_utilizador → NOT NULL,
email → NOT NULL,
password_cliente → NOT NULL,
telemovel → NOT NULL,
PRIMARY KEY(nif));

- **Relação Pedido**

Domínio id_pedido → Inteiro
Domínio dataP → DATETIME
Domínio valor_total → DECIMAL(5,2)
Domínio tempo → Inteiro
Domínio pago → TINYINT
Domínio recolhido → TINYINT
Domínio desconto → DECIMAL(3,2)
Domínio id_funcionario → Inteiro
Domínio nif → Inteiro

Pedido(
id_pedido → NOT NULL,
dataP → NOT NULL,
valor_total → NOT NULL,
tempo → NOT NULL,
pago → NOT NULL,
recolhido → NOT NULL,
desconto → NOT NULL,
id_funcionario → NOT NULL,
nif → NOT NULL,
PRIMARY KEY (id_pedido));
FOREIGN KEY (id_funcionario) REFERENCES Funcionario(id_comboio);
FOREIGN KEY (nif) REFERENCES Cliente (nif)))

- **Relação Gelado_Base**

  Domínio id_gelado → Inteiro
  Domínio nome_gelado → String de tamanho variável, de tamanho 20
  Domínio preco_gelado → DECIMAL(4,2)
  Domínio stock_gelado → Inteiro
  Domínio composicao_gelado → String de tamanho variável, de tamanho 300

  Gelado_Base(
  id_gelado → NOT NULL,
  nome_gelado → NOT NULL,
  preco_gelado → NOT NULL,
  stock_gelado → NOT NULL,
  composicao_gelado → NOT NULL,
  PRIMARY KEY(id_gelado));

- **Relação Funcionario**

  Domínio id_funcionario → Inteiro
  Domínio nome_funcionario → String de tamanho variável, de tamanho 45
  Domínio password_funcionario → String de tamanho variável, de tamanho 20

  Funcionario(
  id_funcionario → NOT NULL,
  nome_funcionario → NOT NULL,
  password_funcionario → NOT NULL,
  PRIMARY KEY(id_funcionario));

- **Relação Extra**

  Domínio id_extra → Inteiro
  Domínio nome_extra → String de tamanho variável, de tamanho 20
  Domínio preco_extra → DECIMAL(4,2)
  Domínio composicao_extra → String de tamanho variável, de tamanho 100
  Domínio stock_extra → Inteiro

Extra(
id_extra → NOT NULL,
nome_extra → NOT NULL,
preco_extra → NOT NULL,
composicao_extra → NOT NULL,
stock_extra → NOT NULL,
PRIMARY KEY(id_extra));

- **Relação Criacao_Gelado**

Domínio id_criacao → Inteiro
Domínio valor_criacao → DECIMAL(4,2)
Domínio id_gelado → Inteiro

Criacao_Gelado(
id_criacao → NOT NULL,
valor_criacao → NOT NULL,
id_gelado → NOT NULL,
PRIMARY KEY(id_criacao);
FOREIGN KEY(id_gelado) REFERENCES Gelado_Base(id_gelado))

- **Relação Criacao_Gelado__Extra**

Domínio id_criacao → Inteiro
Domínio id_extra → Inteiro
Domínio quantidade_extra → Inteiro

Criacao_Gelado__Extra(
id_criacao → NOT NULL,
id_extra → NOT NULL,
quantidade_extra → NOT NULL,
PRIMARY KEY(id_criacao, id_extra))

- **Relação Pedido__Criacao_Gelado**

Domínio id_pedido → Inteiro
Domínio id_criacao → Inteiro
Domínio quantidade_criacao → Inteiro

Criacao_Gelado__Extra (
id_pedido → NOT NULL,
id_criacao → NOT NULL,
quantidade_criacao → NOT NULL,
PRIMARY KEY(id_pedido, id_criacao))

5.2.2. Desenho das restrições gerais

Nesta fase é necessário apresentar as condições restritivas gerais que dizem respeito ao problema. Assim, vão ser apresentadas todas as restrições agrupadas por relação. As restrições que vão ser apresentadas têm a sua definição no script de criação das tabelas.

- **Cliente**

```
-- -----------------------------------------------------
-- Table `gelataria`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `gelataria`.`Cliente` (
  `nif` INT NOT NULL,
  `nome_cliente` VARCHAR(45) NOT NULL,
  `password_cliente` VARCHAR(20) NOT NULL,
  `nome_utilizador` VARCHAR(20) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `telemovel` INT NOT NULL,
  `data_nascimento` DATE NOT NULL,
  PRIMARY KEY (`nif`))
ENGINE = InnoDB;
```

Figura 8. Criação da tabela Cliente

- **Pedido**

```
CREATE TABLE IF NOT EXISTS `GELATUM`.`Pedido` (
  `id_pedido` INT NOT NULL AUTO_INCREMENT,
  `dataP` DATETIME NOT NULL,
  `valor_total` DECIMAL(5,2) NOT NULL,
  `tempo` INT NOT NULL,
  `pago` TINYINT NOT NULL,
  `recolhido` TINYINT NOT NULL,
  `desconto` DECIMAL(3,2) NOT NULL,
  `id_funcionario` INT NOT NULL,
  `nif` INT NOT NULL,
  PRIMARY KEY (`id_pedido`),
  INDEX `fk_Pedido_Funcionario1_idx` (`id_funcionario` ASC) VISIBLE,
  INDEX `fk_Pedido_Cliente1_idx` (`nif` ASC) VISIBLE,
  CONSTRAINT `fk_Pedido_Funcionario1`
    FOREIGN KEY (`id_funcionario`)
    REFERENCES `GELATUM`.`Funcionario` (`id_funcionario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_Cliente1`
    FOREIGN KEY (`nif`)
    REFERENCES `GELATUM`.`Cliente` (`nif`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 9. Criação da tabela Pedido

- **Funcionario**

```
-- -----------------------------------------------------
Table `gelataria`.`Funcionario`
-- -----------------------------------------------------
REATE TABLE IF NOT EXISTS `gelataria`.`Funcionario` (
`id_funcionario` INT NOT NULL,
`nome_funcionario` VARCHAR(45) NOT NULL,
`password_funcionario` VARCHAR(20) NOT NULL,
PRIMARY KEY (`id_funcionario`))
NGINE = InnoDB;
```

Figura 10. Criação da tabela Funcionario

- **Gelado_base**

```
-- -----------------------------------------------------
-- Table `GELATUM`.`Gelado_Base`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Gelado_Base` (
  `id_gelado` INT NOT NULL AUTO_INCREMENT,
  `nome_gelado` VARCHAR(45) NOT NULL,
  `preco_gelado` DECIMAL(4,2) NOT NULL,
  `stock_gelado` INT NOT NULL,
  `composicao_gelado` VARCHAR(300) NOT NULL,
  PRIMARY KEY (`id_gelado`))
ENGINE = InnoDB;
```

Figura 11. Criação da tabela Gelado_Base

- **Extra**

```
-- -----------------------------------------------------
-- Table `GELATUM`.`Extra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Extra` (
  `id_extra` INT NOT NULL AUTO_INCREMENT,
  `nome_extra` VARCHAR(45) BINARY NOT NULL,
  `preco_extra` DECIMAL(4,2) NOT NULL,
  `composicao_extra` VARCHAR(100) NOT NULL,
  `stock_extra` INT NOT NULL,
  PRIMARY KEY (`id_extra`),
  UNIQUE INDEX `nome_UNIQUE` (`nome_extra` ASC) VISIBLE)
ENGINE = InnoDB;
```

Figura 12. Criação da tabela Extra

- **Criacao_Gelado**

```
-- -----------------------------------------------------
-- Table `GELATUM`.`Criacao_Gelado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Criacao_Gelado` (
  `id_criacao` INT NOT NULL AUTO_INCREMENT,
  `valor_criacao` DECIMAL(4,2) NOT NULL,
  `id_gelado` INT NOT NULL,
  PRIMARY KEY (`id_criacao`),
  INDEX `fk_Criacao_Gelado_Gelado_Base1_idx` (`id_gelado` ASC) VISIBLE,
  CONSTRAINT `fk_Criacao_Gelado_Gelado_Base1`
    FOREIGN KEY (`id_gelado`)
    REFERENCES `GELATUM`.`Gelado_Base` (`id_gelado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 13. Criação da tabela Criacao_Gelado

- **Criacao_Gelado__Extra**

```
-- -----------------------------------------------------
-- Table `GELATUM`.`Criacao_Gelado__Extra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Criacao_Gelado__Extra` (
  `id_criacao` INT NOT NULL,
  `id_extra` INT NOT NULL,
  `quantidade_extra` INT NOT NULL,
  PRIMARY KEY (`id_criacao`, `id_extra`),
  INDEX `fk_Criacao_Gelado_has_Extra_Extra1_idx` (`id_extra` ASC) VISIBLE,
  INDEX `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  CONSTRAINT `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1`
    FOREIGN KEY (`id_criacao`)
    REFERENCES `GELATUM`.`Criacao_Gelado` (`id_criacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Criacao_Gelado_has_Extra_Extra1`
    FOREIGN KEY (`id_extra`)
    REFERENCES `GELATUM`.`Extra` (`id_extra`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 14. Criação da tabela Gelado_Extra

- **Pedido__Criacao_Gelado**

```
CREATE TABLE IF NOT EXISTS `GELATUM`.`Pedido__Criacao_Gelado` (
  `id_pedido` INT NOT NULL,
  `id_criacao` INT NOT NULL,
  `quantidade_criacao` INT NOT NULL,
  PRIMARY KEY (`id_pedido`, `id_criacao`),
  INDEX `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  INDEX `fk_Pedido_has_Criacao_Gelado_Pedido1_idx` (`id_pedido` ASC) VISIBLE,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Pedido1`
    FOREIGN KEY (`id_pedido`)
    REFERENCES `GELATUM`.`Pedido` (`id_pedido`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1`
    FOREIGN KEY (`id_criacao`)
    REFERENCES `GELATUM`.`Criacao_Gelado` (`id_criacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Figura 15. Criação da tabela Pedido__Criacao_Gelado

5.3. Tradução das interrogações do utilizador para *SQL*

- **1ª *Querie***

  Os nomes dos 3 gelados base mais vendidos em junho de 2015.

```
USE GELATUM;

SELECT gb.id_gelado, gb.nome_gelado, SUM(PCG.quantidade_criacao) AS soma_quantidade
FROM Gelado_Base AS GB
    INNER JOIN Criacao_Gelado AS CG
    ON GB.id_gelado = CG.id_gelado
        INNER JOIN Pedido__Criacao_Gelado AS PCG
        ON CG.id_criacao = PCG.id_criacao
            INNER JOIN Pedido AS P
            ON PCG.id_pedido = P.id_pedido
            WHERE (P.dataP between '2015-06-01 00:00:00' AND '2015-07-01 00:00:00')
            GROUP BY id_gelado
            ORDER BY (soma_quantidade) DESC
            limit 3;
```

Figura 16. 1ª *querie* em *SQL*

- **2ª *Querie***

  Exibir o NIF e o nome do cliente mais velho que efetuou um pedido.

```
SELECT C.nif,C.nome_cliente
FROM  Pedido P
INNER JOIN Cliente C
ON C.nif = P.nif
ORDER BY C.data_nascimento ASC
LIMIT 1;
```

Figura 17. 2ª *querie* em *SQL*

- **3ª *Querie***

Exibir nomes de todos os funcionários que processaram pedidos com gelados de morango por ordem decrescente de vendas.

```
SELECT F.id_funcionario,F.nome_funcionario,SUM(PCG.quantidade_criacao)
FROM Funcionario AS F
    INNER JOIN Pedido P
    ON F.id_funcionario = P.id_funcionario
        INNER JOIN Pedido__Criacao_Gelado AS PCG
        ON P.id_pedido = PCG.id_pedido
            INNER JOIN Criacao_Gelado AS CG
            ON PCG.id_criacao = CG.id_criacao
                INNER JOIN Gelado_Base AS GB
                ON CG.id_gelado = GB.id_gelado
                WHERE (GB.id_gelado = 1 )
                GROUP BY F.id_funcionario
                ORDER BY (PCG.quantidade_criacao) DESC;
```

Figura 18. 3ª *querie* em *SQL*

5.4. Tradução das transações estabelecidas para *SQL*

Como uma possibilidade de aumentar a performance do sistema de gestão de base de dados a ser desenvolvido, tivemos que proceder, numa primeira instância, a uma análise de algumas transações que ocorrem entre as relações do nosso modelo desenvolvido.

Concluímos, portanto, que existe uma operação frequente que acontece no nosso modelo que implica um *INSERT* que corresponde ao registo de um novo pedido, e essa operação é passível de ser uma transação, pois não implica apenas operações de leitura. Consideremos então o código *MySQL* que corresponde à transação identificada:

```sql
-- Transação que regista um pedido com UMA criação de um gelado já existente
DELIMITER $$
CREATE PROCEDURE regista_pedido (IN id_pedido INT, IN dataP  DATETIME, IN valor_total DECIMAL(5,2), IN tempo INT, IN pago TINYINT,
IN recolhido TINYINT, IN desconto DECIMAL(4,2), IN id_funcionario INT, IN nif INT, IN id_criacao INT,
IN quantidade_criacao INT)

BEGIN
    DECLARE Erro BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET Erro=1;
    START TRANSACTION;

-- inserção do pedido
INSERT INTO Pedido
    (id_pedido, dataP, valor_total, tempo, pago,recolhido, desconto,  id_funcionario, nif)
VALUES
    (id_pedido, dataP, valor_total, tempo, pago, recolhido, desconto, id_funcionario, nif);

-- associação do pedido à criação
INSERT INTO Pedido__Criacao_Gelado
    (id_pedido, id_criacao, quantidade_criacao)
VALUES
    (id_pedido, id_criacao, quantidade_criacao);

-- atualização do valor total após inserção do pedido
UPDATE Pedido AS P
SET P.valor_total = valorTot(id_pedido)
WHERE(P.id_pedido = id_pedido);

-- alteração do stock fase 1
SELECT DISTINCT CG.id_gelado
    INTO @id_gelado1
    FROM Pedido__Criacao_Gelado AS PCG
        INNER JOIN Criacao_Gelado AS CG
        ON PCG.id_criacao = CG.id_criacao
        WHERE (id_criacao = CG.id_criacao);

-- alteração do stock fase 1
UPDATE Gelado_Base AS GB
SET GB.stock_gelado = GB.stock_gelado - quantidade_criacao
WHERE GB.id_gelado = @id_gelado1;

IF erro
THEN ROLLBACK;
ELSE COMMIT;
END IF;
END $$
```

Figura 19. Transação de um pedido

5.5. Escolha, definição e caracterização de índices em SQL

Depois de um pouco de pesquisa acerca deste tópico, concluímos que índices são estruturas de dados que permitem ao sistema de gestão de base de dados localizar registos num ficheiro/arquivo, de forma a minimizar o tempo de resposta a *queries* feitas pelo utilizador.

Toda a tabela *InnoDB* tem um índice especial designado de *clustered index* no qual os dados das linhas de uma tabela (os registos) são armazenados, ou seja, esse índice corresponde à chave primária da tabela.

O *InnoDB* usa índice especial para otimizar as operações de *DML* (*Data Manipulation Language)* mais comuns: *SELECT*, *INSERT*, *UPDATE* e *DELETE*.

Como foi referido, o *InnoDB* utiliza a chave primária de cada tabela como índice. Sendo assim, vamos apresentar alguns índices:

```
CREATE INDEX  ClienteNome ON Cliente(nome_cliente)

CREATE INDEX  PedidosCliente ON Pedido(id_pedido)

CREATE INDEX  CriaçãoValor ON Criacao_Gelado(valor_criacao)

CREATE INDEX  GeladoPreco ON Gelado_Base(nome_gelado)

CREATE INDEX  ExtraPreco ON Extra(nome_extra)
```

Figura 20. Criação de índices

5.6. Estimativa do espaço em disco da base de dados e taxa de crescimento anual

5.6.1. Tamanho de cada uma das tabelas da *GELATUM*, após o povoamento inicial

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Cliente	nif	INT	4 *bytes*
	nome_cliente	VARCHAR (45)	45 *bytes*
	password_cliente	VARCHAR (20)	20 *bytes*
	nome_utilizador	VARCHAR (20)	20 *bytes*
	data_nascimento	DATE	3 *bytes*
	email	VARCHAR (45)	45 *bytes*
	password_cliente	VARCHAR (20)	20 *bytes*
TOTAL			Número clientes * tamanho dos dados para um cliente = 50*157 = 7850 *bytes*

Tabela 5. Cliente

Tabela	Atributo	Tipo de dados	Espaço Ocupado
	id_pedido	INT	4 *bytes*
	dataP	DATETIME	8 *bytes*
	valor_total	DECIMAL(5,2)	5 *bytes*

Pedido	tempo	INT	4 *bytes*
	pago	TINYINT	1 *byte*
	recolhido	TINYINT	1 *byte*
	desconto	DECIMAL(3,2)	5 *bytes*
	id_funcionario	INT	4 *bytes*
	nif	INT	4 *bytes*
TOTAL			Número pedidos * tamanho dos dados para um pedido = 7*36 = 252 *bytes*

Tabela 6. Pedido

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Funcionario	id_funcionario	INT	4 *bytes*
	nome_funcionario	VARCHAR (45)	45 *bytes*
	password_funcionario	VARCHAR (20)	20 *bytes*
TOTAL			Número funcionario * tamanho dos dados para um funcionario = 4*69 = 276 *bytes*

Tabela 7. Funcionario

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Pedido__Criacao_gelado	id_pedido	INT	4 *bytes*
	id_criacao	INT	4 *bytes*
	quantidade_criacao	INT	4 *bytes*
TOTAL			Número Pedidos de criação de gelado * tamanho dos dados para um pedido de criacao = 6*12 = 72 *bytes*

Tabela 8. Pedido__Criacao_gelado

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Criacao_Gelado	Id_criacao Valor_criacao Id_gelado	INT DECIMAL(4,2) INT	4 *bytes* 5 *bytes* 4 *bytes*
TOTAL			Número criações de gelados * tamanho dos dados para uma criação = 6*13 = 78 *bytes*

Tabela 9. Criacao_gelado

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Gelado_Base	id_gelado nome_gelado preco_gelado stock_gelado composição_gelado	INT VARCHAR (45) DECIMAL(4,2) INT VARCHAR (300)	4 *bytes* 45 *bytes* 5 *bytes* 4 *bytes* 300 *bytes*
TOTAL			Número gelados base * tamanho dos dados para um gelado base = 7*358 = 2506 *bytes*

Tabela 10. Gelado_Base

Tabela	Atributo	Tipo de dados	Espaço Ocupado
Criação_Gelado__Extra	id_criacao	INT	4 *bytes*
	id_extra	INT	4 *bytes*
	quantidade_extra	INT	4 *bytes*
TOTAL			Número criações gelado extra * tamanho dos dados para uma criação gelado extra = 6*12 = 72 *bytes*

Tabela 11. Criacao_gelado__Extra

Tabela	***Atributo***	***Tipo de dados***	***Espaço Ocupado***
Extra	id_extra	INT	4 *bytes*
	nome_extra	VARCHAR (45)	45 *bytes*
	preco_extra	DECIMAL(4,2)	5 *bytes*
	composição_extra	VARCHAR (100)	100 *bytes*
	stock_Extra	INT	4 *bytes*
TOTAL			Número extras * tamanho dos dados para um extra = 7*158 = 1106 *bytes*

Tabela 12. Extra

- **Tamanho inicial da base de dados *GELATUM***

Tabela	Tamanho com os dados inseridos
Cliente	7850 *bytes*
Pedido	252 *bytes*
Funcionário	276 *bytes*
Pedido__Criacao_Gelado	72 *bytes*
Criação_Gelado	78 *bytes*
Gelado_Base	2506 *bytes*
Criacao_Gelado__Extra	72 *bytes*
Extra	1106 *bytes*
Tamanho inicial da *GELATUM*	**12 212 *bytes* (12 *kbytes*)**

Tabela 13. Tamanho inicial da base de dados *GELATUM*

5.6.2. Estimativa de crescimento anual da *GELATUM*

Hipoteticamente fizemos algumas asserções acerca do que seria um crescimento para a nossa gelataria ao fim de um ano e, consequentemente, para a base de dados que a suporta e estamos a desenvolver. Consideremos por isso que:

- O número de clientes que procuram a nossa gelataria cresce a um ritmo de 5 clientes/mês, perfazendo um aumento de 60 clientes ao fim de um ano relativamente ao número inicial.

Número de clientes ao fim de um ano	Tamanho da tabela Cliente ao fim de um ano
50 + 60 = 110	110 *7850 (tamanho dos dados para um cliente) = 863500 *bytes*

Tabela 14. Previsão do aumento do número de clientes

- A gelataria aumenta o seu leque de pedidos de forma a que, ao fim de um ano, possui mais 180(15*12) pedidos face ao número inicial.

Número de pedidos ao fim de um ano	Tamanho da tabela Pedido ao fim de um ano
7+180 = 187	187 *252(tamanho dos dados para um pedido) = 47124 *bytes*

Tabela 15. Previsão do aumento do número de pedidos

- A *GELATUM* adquire ao longo do ano mais 3 funcionários relativamente ao número inicial.

Número de funcionarios ao fim de um ano	Tamanho da tabela Funcionario ao fim de um ano
4+3 = 7	7 *276 (tamanho dos dados para um funcionario) = 1932 *bytes*

Tabela 16.Previsão do aumento do número de funcionários

- A *GELATUM* adquiriu mais 13 pedidos criação de gelado por ano.

Número pedidos criação gelado ao fim de um ano	Tamanho da tabela Pedido__Criacao_Gelado ao fim de um ano
6 + 13 = 19	19 *72 (tamanho dos dados para um pedido criação gelado) = 1368 *bytes*

Tabela 17. Previsão do aumento do número de relacionamentos entre pedidos e criações

- A *GELATUM* adquiriu mais 13 criações de gelados por ano.

Número de criações ao fim de um ano	Tamanho da tabela Criacao_Gelado ao fim de um ano
6 + 13 = 19	19 * 78(tamanho dos dados para uma criação gelado) = 1482 *bytes*

Tabela 18. Previsão do aumento do número de criações de gelados

- A *GELATUM* adquiriu mais 15 gelados base por mês, fazendo um aumento de 180 ao fim do ano.

Número de Gelados Base ao fim de um ano	Tamanho da tabela Gelados base ao fim de um ano
7 +15 = 22	22*358 (tamanho dos dados para um gelado base) = 7876 *bytes*

Tabela 19. Previsão do aumento do número de bases de gelado

- O número de criações de gelado extra cresceu em 13 ao longo do ano

Número de criações gelado extra aofim de um ano	Tamanho da tabela Criacao_Gelado__Extra ao fim de um ano
6 +13 = 19	19 *12 (tamanho dos dados para uma criação gelado extra) = 228 *bytes*

Tabela 20. Previsão do aumento do número de relacionamentos entre criações de gelado e extras

- O número de extras cresceu ,fazendo um aumento de 15 extras ao fim de um ano relativamente ao número inicial.

Número de extras ao fim de um ano	Tamanho da tabela Extra ao fim de um ano
7 +15 = 22	22 *158 (tamanho dos dados para um extra) = 3476 *bytes*

Tabela 21. Previsão do aumento do número de extras

Posto isto, podemos estimar qual seria o tamanho da base de dados ao fim de um ano de operacionalidade da Gelatum, somando o valor de cada uma das tabelas calculadas anteriormente. Assim, temos:

Tabela	Tamanho com os dados inseridos ao fim de um ano
Cliente	863500 *bytes*
Pedido	47124 *bytes*
Funcionário	1932 *bytes*
Pedido__Criacao_Gelado	1368 *bytes*
Criacao_Gelado	1482 *bytes*
Gelado_Base	7876 *bytes*
Criacao_Gelado__Extra	228 *bytes*
Extra	3476 *bytes*
Tamanho da GELATUM ao fim de um ano	**926 986 *bytes***

Tabela 22. Previsão do aumento do tamanho da base de dados ao fim de um ano

5.7. Definição e caracterização das vistas de utilização em *SQL*

No desenvolvimento de um SGBD um aspeto importante são as vistas de cada um dos utilizadores, ou seja, os acesso e as ação que cada um pode ter.

Uma vista pode ser considerada como uma tabela que advém da consulta de outras tabelas, sendo por vezes designada de tabela virtual. As vistas permitem simplificar e evitar escrever consultas frequentes, escondendo também certos detalhes que não têm interesse ou não dizem respeito a um dado utilizador.

Consideremos, portanto, algumas vistas do nosso modelo físico:

- **Tabela que mostra todos os gelados disponíveis**

```
CREATE VIEW Gelados_Disponiveis AS
    SELECT id_gelado, nome_gelado, preco_gelado
    FROM Gelado_Base
```

Figura 21. Código *SQL* que mostra o conteúdo de Gelados_Disponiveis

- **Tabela que mostra todos os extras disponíveis**

```
CREATE VIEW Extras_Disponiveis AS
    SELECT id_extra, nome_extra, preco_extra
    FROM Extra
```

Figura 22. Código *SQL* que mostra o conteúdo de Extras_Disponiveis

- **Tabela que, dos pedidos, mostra o funcionário que atendeu determinado cliente**

```
CREATE VIEW Funcionario_Cliente AS
   SELECT C.nif, C.nome_cliente, F.nome_funcionario
   FROM CLIENTE AS C
       INNER JOIN Pedido AS P
       ON C.nif = P.nif
           INNER JOIN Funcionario AS F
           ON P.id_funcionario = F.id_funcionario;
```

Figura 23. Código *SQL* que mostra o conteúdo de Funcionario_Cliente

- **Tabela que mostra todas as criações com o nome do gelado e dos extras (com as respetivas quantidades) que contém**

```
CREATE VIEW Criacao_Discriminada AS
    SELECT CG.id_criacao, GB.nome_gelado, GROUP_CONCAT(CONCAT(E.nome_extra,'(', CGE.quantidade_extra, ')')
                                          SEPARATOR ', ') AS nome_todos_extras, CG.valor_criacao
    FROM Gelado_Base AS GB
        INNER JOIN Criacao_Gelado AS CG
        ON GB.id_gelado = CG.id_gelado
            INNER JOIN Criacao_Gelado__Extra AS CGE
            ON CG.id_criacao = CGE.id_criacao
                INNER JOIN Extra AS E
                ON CGE.id_extra = E.id_extra
                GROUP BY CG.id_gelado, CGE.id_criacao;
```

Figura 24. Código *SQL* que mostra o conteúdo de Criacao_Discriminada

5.8. Definição e caracterização dos mecanismos de segurança em SQL

Dedicamos este capítulo para especificar os mecanismos de segurança/permissões dos três tipos de utilizadores que identificámos para a nossa base de dados: de um cliente, de um funcionário e de um administrador.

- O **administrador**, o senhor Oleb, tem permissão para realizar qualquer ação (*CREATE*, *DROP*, *SELECT*, *DELETE*, *INSERT*, *UPDATE*), por isso, temos as seguintes permissões/regras de acesso:

```
CREATE USER 'Administrador'@'localhost'
    identified by 'adminpassword';

-- da todas as permissoes ao admin excepto criar e eliminar base dados
GRANT SELECT, INSERT, DELETE, UPDATE ON *.* TO 'Administrador'@'localhost';

REVOKE DROP, CREATE
ON *.*
FROM 'Administrador'@'localhost';
```

Figura 25. Código *SQL* com as permissões do administrador

- O **cliente** deverá poder fazer qualquer operação que diga respeito aos seus dados pessoais (número de identificação fiscal, nome, *password*, nome de utilizador, data de nascimento, *email* e número de telemóvel),com a exceção de que, para além da data de nascimento, não pode remover dados específicos. Apenas poderá consultar todos os dados relativos ao seu pedido, ao gelado e ao extra, caso exista, sendo que destes só pode alterar as quantidades:

```
CREATE USER 'cliente'@'localhost'
   identified BY 'clientepassword';

GRANT SELECT ON gelataria.Gelados TO 'cliente'@'localhost';
GRANT SELECT ON gelataria.Extra To 'cliente'@'localhost';
GRANT SELECT, DELETE, UPDATE ON gelataria.Cliente[nif_cliente, nome_cliente, password_cliente, nome_utilizador_cliente, data_nascimeto_cliente, email_cliente, telemovel_cliente]
TO 'cliente'@'localhost';
GRANT SELECT, INSERT, UPDATE ON gelataria.Cliente To 'cliente'@'localhost'; --sem certeza se este insert sera para meter "novo cliente" ou seja fazer uma inscricao

REVOKE DROP, CREATE, DELETE, UPDATE, INSERT
ON *.Gelados
FROM 'cliente'@'localhost';

REVOKE DROP, CREATE, DELETE, UPDATE, INSERT
ON *.Extra
FROM 'cliente'@'localhost';

REVOKE DROP, CREATE
ON *.Cliente
FROM 'cliente'@'localhost';
```

Figura 26. Código *SQL* com as permissões do cliente

- O **funcionário** tem acesso a todos os dados que sejam necessários para o processamento de um pedido devem ser disponibilizados ao funcionário. Posto isto, este utilizador pode consultar a informação disponível na base de dados relativa ao cliente que faz o pedido, exceto a *password*, o nome de utilizador, o *email* e o número de telemóvel. Poderá ainda consultar os seus dados, mas apenas alterar a *password*:

```
CREATE USER 'Funcionario'@'localhost'
    identified by 'funcpassword';

GRANT SELECT, UPDATE gelataria.Gelados To 'Funcionario'@'localhost';
GRANT SELECT, UPDATE ON gelataria.Extra To 'Funcionario'@'localhost';
GRANT SELECT ON gelataria.Funcionario To 'Funcionario'@'localhost';
GRANT SELECT, UPDATE ON gelataria.Funcionario[password_func] To 'Funcionario'@'localhost';
GRANT SELECT ON gelataria.Pedido To 'Funcionario'@'localhost';
GRANT SELECT ON gelataria.Cliente[nome_cliente, nif_cliente] To 'Funcionario'@'localhost';
```

Figura 27. Código *SQL* com as permissões do funcionário

5.9. Revisão do sistema implementado com o utilizador

Para dar como terminada esta fase, o modelo físico deve ser visto na perspetiva do utilizador. Este processo tem um papel muito importante pois o utilizador tem que ter a capacidade de reconhecer, que o modelo físico apresentado representa a situação real da gelataria que está a ser modelada. Por isso, tivemos de avaliar os dados de modo a traduzir as relações base definidas previamente no modelo lógico, de maneira a que estas e as suas restrições sejam suportadas pelo SGDB. Deste modo, foi necessária a descrição das relações base e o desenho das restrições gerais.

Posteriormente também foram criados índices para um acesso mais eficiente e rápido aos dados, como por exemplo, ver qual o preço de um determinado extra.

Outro aspeto importante analisado foram as vistas, uma *view* é um resultado originado de uma consulta pré-definida, como podemos observar na *view* “criações com o nome do gelado e dos extras” mostra a associação do gelado base com o(s) respetivo(s) extra(s).

Por fim, foram analisadas as permissões de modo a proporcionar uma melhor segurança aos utilizadores da nossa gelataria.

Sendo assim, o nosso modelo foi aprovado pelo cliente pois é visto como uma solução que responde a todas as necessidades pretendidas.

6. Migração para Base de Dados não Relacional

6.1. Introdução

Atualmente, no mercado, cada vez mais a criação de base de dados não relacionais tem vindo a crescer surpreendentemente. A principal razão de tal acontecimento é a resolução do problema da escalabilidade das bases de dados tradicionais.

Outro motivo é a enorme quantidade de dados associados a algumas áreas de negócio e os problemas que daí advêm. Estes novos modelos, para além de permitirem a fácil manipulação dos dados, tal como nos relacionais, permitem o armazenamento de forma sustentável.

Apesar de as bases de dados representarem entre 10% a 20% do mercado mundial, esses valores correspondem a milhões de utilizadores.

A ferramenta usada para implementar a base de dados não relacional foi o *Neo4j* que é baseado em grafos. Este tipo de base de dados utiliza o modelo de grafos para representar o esquema. Podem ser observadas três componentes básicas:

- **Nodos** que representam os vértices do grafo;
- **Relacionamentos** entre nodos que são arestas do grafo;
- **Propriedades** relativas aos nodos ou aos relacionamentos entre nodos.

6.2. Bases de Dados *NoSQL*

Durante décadas, o modelo de dados predominante usado para desenvolvimento de aplicações foi o modelo usado por bases de dados relacionais, como *Oracle*, *DB2*, *SQL Server*, *MySQL* e *PostgreSQL*. Apenas em meados dos anos 2000 é que outros modelos de dados começaram a ser adotados e ter a um uso mais significativo. Para diferenciar e categorizar essas novas classes de bases e modelos de dados, o termo "*NoSQL*" foi criado.

NOSQL (originalmente referindo-se a "no *SQL*": "não *SQL*" ou "não relacional", posteriormente estendido para *Not Only SQL* - Não Apenas *SQL*) é um termo genérico que representa bases de dados não relacionais sendo *open source* e completamente distinta do

modelo relacional tradicional, isto porque as bases de dados *NoSQL* foram justamente projetadas a partir de necessidades que as bases de dados tradicionais relacionais não satisfaziam, como alta performance e capacidade de expansão.

Podemos, assim, afirmar que existem quatro tipos diferentes de base de dados *NoSQL*:

- **Grafos:** baseados na teoria dos grafos; armazena os seus dados na forma de grafo, com vértices e arestas.
  Ex: *Neo4j*, *Sesame*.

- **Documentos:** armazena os seus dados como documentos , em que cada um deles é uma coleção de pares chave-valor e permite a realização de consultas mais elaboradas, envolvendo filtros por atributos e a possibilidade de uso de índices.
  Ex: *MongoDB*, *CouchDB*.

- **Colunas:** armazena os seus dados em linhas particulares de uma tabela no disco.
  Ex: *Cassandra*, *Hibase*;

- **Chave-Valor:** armazena os seus dados num padrão chave-valor, fazendo lembrar as tabelas de *hash*.
  Ex: *MemcacheD*, *Riak*, *REDIS*.

6.2.1. Principais Características

● Alta performance e escalabilidade horizontal

Enquanto que em servidores de base de dados tradicionais possuíam a adição de um maior número de recursos ao servidor, como memória e disco, para suportar mais dados, as bases de dados *NoSQL* redistribuem-se horizontalmente, ou seja, funcionam como um sistema fracamente acoplado, o que a torna mais económica e escalável. Graças a esse mecanismo, a sua performance provém de não possuir um servidor central com muito poder computacional, mas sim, de vários servidores, que não necessitam de ser de alta performance, conectados e trabalhando em conjunto.

● Replicação

A replicação é uma estratégia que se encaixa perfeitamente na arquitetura do *NoSQL*, já que ele segue o conceito de sistema fracamente acoplado. Essa ferramenta possibilita o armazenamento de dados e backups independentemente do local físico onde os dados estão armazenados.

● **Raízes *Open Source***

Muitas bases de dados *NoSQL* tem raízes na comunidade open *source*. Talvez isso tenha sido fundamental para o rápido crescimento do seu uso e popularidade. Nota-se que as companhias que oferecem versões comerciais de base de dados *NoSQL* com uma forte estrutura de suporte e serviços. Temos o exemplo do *Facebook* que já utiliza este modelo de bases de dados para guardar todas as informações.

• **Baixo custo operacional**

Devido ao peso do open *source* no *NoSQL*, o custo para iniciar a utilização dessas bases de dados torna-se muito baixo ou zero. É comum ouvir dizer que a transição relacional -> *NoSQL* diminuiu muito os custos enquanto obteve um desempenho melhor ou igual ao anterior. As bases de dados relacionais mais elaboradas requerem computadores com excelentes especificações. Com o *NoSQL*, esse custo também diminui, pois este foi desenvolvido para trabalhar em ambientes distribuídos.

• **Ausência de esquema ou esquema flexível**

Uma das principais características de uma base de dados *NoSQL* é a ausência completa ou quase total de esquema que define a estrutura de dados. Devido a essa ausência há uma fácil aplicação da escalabilidade e também um aumento na disponibilidade. Tem uma API simples para fácil acesso aos dados.

6.2.2. *SQL* vs *NoSQL*

• **Linguagem**

As bases de dados *SQL* são estruturadas em linguagem de consulta (*SQL*) para definição e manipulação de dados. Por um lado, isso é extremamente poderoso: o *SQL* é uma das opções mais versáveis e mais utilizadas, sendo uma escolha segura e especialmente ótima para consultas complexas.

Por outro lado, ele pode ser restritivo. O *SQL* exige o uso de esquemas (arquiteturas visuais e lógicas de uma base de dados) pré-definidos para determinar a estrutura dos seus dados antes de trabalhar com eles.

• **Escalabilidade**

Na maioria das situações, as bases de dados *SQL* são verticalmente escaláveis, o que significa que se pode aumentar o carregamento num servidor melhorando coisas como *CPU*, *RAM* ou *SSD*.

As bases de dados *NoSQL*, por sua vez, são horizontalmente escaláveis. Isso quer dizer que suporta muito mais tráfego por *sharding* (particionamento de dados), ou

seja, adicionando mais servidores na sua base de dados *NoSQL*. É como se comparássemos entre adicionar mais andares no mesmo prédio e adicionar mais prédios na vizinhança. O segundo pode ficar maior e mais poderoso, tornando as bases de dados *NoSQL* a escolha de preferência para conjuntos de dados maiores ou em mudança constante.

• **Estrutura**

As bases de dados *SQL* são baseadas em tabelas, enquanto que as bases de dados *NoSQL* podem ser baseadas em documentos, pares de chave-valor, grafos ou orientados a colunas. Isso torna as bases de dados *SQL* relacionais opções melhores para aplicações que requerem transações retornando várias colunas—como um sistema de contabilidade.

• **Propriedades *ACID***

A grande maioria das bases de dados relacionais são compatíveis com *ACID*, enquanto que em *NoSQL* varia de acordo com as tecnologias, mas muitas soluções *NoSQL* sacrificam a compatibilidade *ACID* para desempenho e escalabilidade.

6.2.3. Vantagens

• Dados sempre disponiveis, pois nao necessitará da ocorrência de *loads*;
• Custo mais reduzido que uma base dados relacional;
• Base dados orientada a objectos flexíveis;
• Facilidade em introduzir novos dados;
• Excelente maneira de lidar com o problema de dados em massa;
• Performance superior na consulta de grandes ou pequenas quantidades de dados.

6.2.4. Desvantagens

• Mais recentes que as base dados relacionais, não contendo tantas alternativas;
• Não irão resolver problemas de escabilidade de um *website*;
• Situações onde o modo como os dados estão estruturados é importante, como é o caso de contas bancárias, uma base de dados relacional é muito mais adequada;
• Não é garantida a consistência dos dados presentes, estando sempre dependentes do programador e do modo com executa as inserções ou alterações nos dados;
• *Neo4J* apenas possui *hash indexes*, faltando um *index* de alcance(range indexes).

6.3. Esquema da Base de Dados

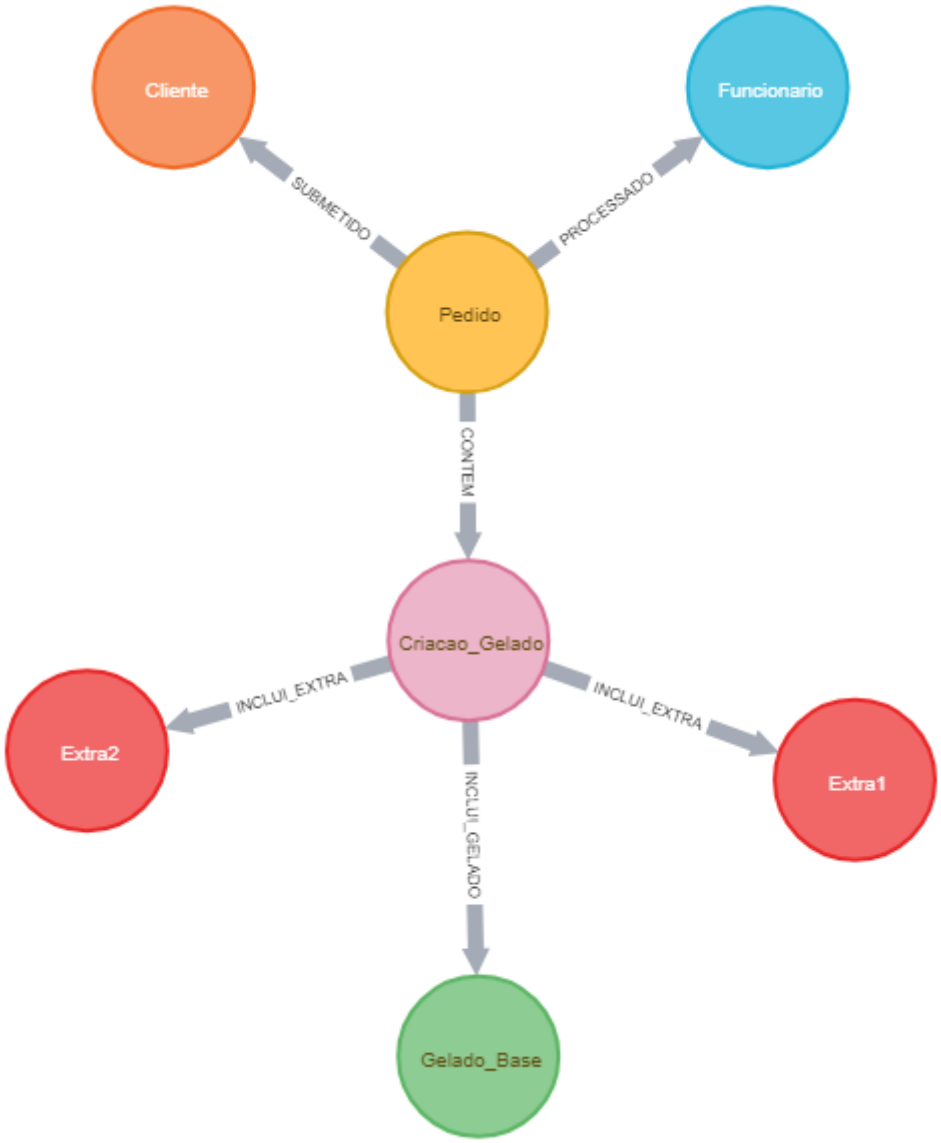

Figura 28. Esquema geral da base de dados

6.4. Migração de dados

A migração dos dados de um modelo relacional para um não relacional, nem sempre é tarefa fácil, principalmente quando estamos a tratar de bases de dados com quantidade de informação muito superior àquela que por nós foi tratada. A decisão de fazer este tipo de migrações de dados passa, entre outras razões expostas anteriormente, pela falta de flexibilidade dos modelos relacionais e é algo em que as empresas estão a apostar bastante.

Migrar dados de e para um sistema de base de dados, implica retirar a informação de uma base de dados e colocá-la noutra, algo que apesar de parecer simples, tem algumas nuances. Este movimento de dados tem que ser íntegro, o que implica que não se podem perder quaisquer informações, pelo que delinear uma estratégia para que tal não aconteça é

essencial. A nossa equipa definiu que o primeiro ponto a ter em consideração, seria a **exportação da informação** (usando um *script* em *SQL*) do sistema de base de dados onde tínhamos guardadas as informações relativas à *GELATUM* para ficheiros do tipo *csv*. De seguida, já no *Neo4j*, teríamos que **importar a informação** guardada nesses mesmos ficheiros. A importação seria feita aquando da **criação dos nodos**, algo que nos faria poupar bastante na escrita de código. Com estes, já tínhamos toda a informação necessária para de uma forma automática **criar todos os relacionamentos** que deviam existir. A última fase, seria a **remoção de dados** de determinados nodos, dados próprios de modelos relacionais que neste não faria tanto sentido que existissem. A partir da importação, todo o código foi escrito na linguagem *Cypher*, linguagem que está disponível e é no programa *Neo4j.*

6.4.1. Exportação da informação contida para ficheiros *csv*

Como já foi referido decidimos recorrer a ficherios *cvs* para guardar a informação exportada. Geramos 8 ficheiros, em que cada um deles contém toda a informação de uma determinada tabela. Neste ficheiro, cada valor (de um determinado atributo) fica delimitado por aspas (como *string*) e todos eles são separados por vírgulas. Em cada linha deste ficheiro, podemos encontrar informação de um registo, em *SQL*, correspondente a uma entrada numa tabela.

6.4.2. Importação da informação contida em ficheiros *csv*

Para termos a informação temos que a ir buscar aos ficheiros onde as guardamos anteriormente e carregá-la para algum sítio. Tal tarefa implica que guardemos a informação importada em algum local, na nova base de dados, esse local são os diferentes nodos. Assim, esta etapa é feita aquando da criação dos nodos. É necessário referir que, para determinados valores não foi possível fazer a correta conversão de *string* para o tipo de dados correspondente, no entanto, sempre que possível, fizemos essa conversão.

6.4.3. Criação dos nodos

Sabemos que, na nova base de dados, a informação tem que ficar alojada em algum local, e como tal, foi necessário proceder à criação de novos nodos. No total, este processo, obrigou-nos a criar cento e quatro novos nodos, tendo em conta que, criamos tantos tipos de nodos quantas as tabelas que tínhamos. Criamos os nodos do tipo “Pedido”, “Funcionario”, “Cliente”, “Criacao_Gelado”, “Gelado_Base”, e “Extra”.

Criamos tantos nodos de um determinado tipo, quantos os registos existentes na tabela correspondente a esse tipo, associando a cada nodo, os valores dos atributos de cada registo, agora identificados como propriedades! De um modo geral, todos os registos que existiam na nossa base de dados relacional, passaram a ser nodos na nossa base de dados não relacional. Nesta fase, se quisermos ver a

informação correspondente a uma tabela podemos simplesmente selecionar os nodos afetos a essa tabela.

6.4.4. Criação dos relacionamentos

Os relacionamentos são uma característica que tornam muito intuitiva a interação do utilizador com bases de dados não relacionais orientadas a grafos. Num modelo relacional, teríamos que que olhar para “chaves” contidas noutras tabelas, para vermos com quais é que uma dada tabela se relacionava. Neste, essa representação, visualmente, é feita com uma simples ligação acompanhada de um nome e das suas propriedades.

No total, criamos cinco tipos de relacionamentos: o relacionamento “SUBMETIDO”, “PROCESSADO”, “INCLUI_GELADO” , “INCLUI_EXTRA” e “CONTEM”.

Os relacionamentos que no modelo relacional eram 1 para n ou 1 para 1, obrigavam a que existisse uma chave estrangeira numa das tabelas. No caso dos primeiros três relacionamentos referidos, usamos a chave estrangeira que existia em determinados nodos para fazer a correta associação. Por exemplo, na anterior base de dados, um “Pedido” tinha a chave estrangeira de um “Cliente” e na atual sabemos que o valor dessa chave estrangeira ainda existe em todos os nodos do tipo “Cliente”. É esse valor que vamos usar para associar um “Cliente” aos seus “Pedidos”, tal como nos outros casos.

Os relacionamentos que no modelo relacional eram de n para m, obrigavam à existência de uma tabela. É o caso dos dois que nos restam, “INCLUI_EXTRA” e “CONTEM”. Para criar estes relacionamentos, ligamos os nodos de acordo com as chaves existentes nos nodos que correspondem a essas tabelas. Como por exemplo, no relacionamento entre “Extra” e “Criacao_Gelado”, na antiga base de dados existia uma tabela “Criacao_Gelado__Extra”. Neste momento, existe um tipo de nodos com esse nome, que tem as respetivas propriedades. Ora, o que deve ser feito, é a criação de um relacionamento entre os nodos “Criacao_Gelado” e “Extra” por cada nodo “Criacao_Gelado__Extra”, desde que este último contenha em simultâneo os identificadores de uma “Criacao_Gelado” e de um “Extra”. A cada um destes últimos relacionamentos é também associada uma propriedade que existia no modelo relacional.

6.4.5. Remoção de dados redundantes

Concretizados todos os relacionamentos, somos capazes de ver que existe informação que é redundante. Na verdade, a informação que agora dizemos redundante, foi “copiada” e disposta na base de dados não relacional de uma maneira diferente à relacional. Para nos referirmos a informação que está associada a vários nodos, simplesmente referimos esses nodos e o correspondente relacionamento, não

sendo precisa a existência de qualquer chave estrangeira para o fazer. Não obstante, as propriedades que correspondiam a chaves estrangeiras, continuam nos nodos, assim como os nodos do tipo “Criacao_Gelado__Extra” e “Pedido__Criacao_Gelado”, que só têm propriedades que agora são inúteis à base de dados.

Tendo em conta esse aspeto, decidimos eliminar todas as propriedades que correspondiam a chaves estrangeiras e todos os nodos que correspondiam a tabelas resultantes de relacionamentos n para n.

Com este último passo, concluiu-se a migração dos dados da base de dados relacional para a não relacional, passando a existir, no total, oitenta e quatro nodos e quarenta e cinco relacionamentos, sem a perda de qualquer informação, que agora é vista, acedida e manipulada de maneira diferente. Todo o código executado pode ser visto nos *scripts*.

6.5. Grafo resultante

6.5.1. Imagem do grafo resultante

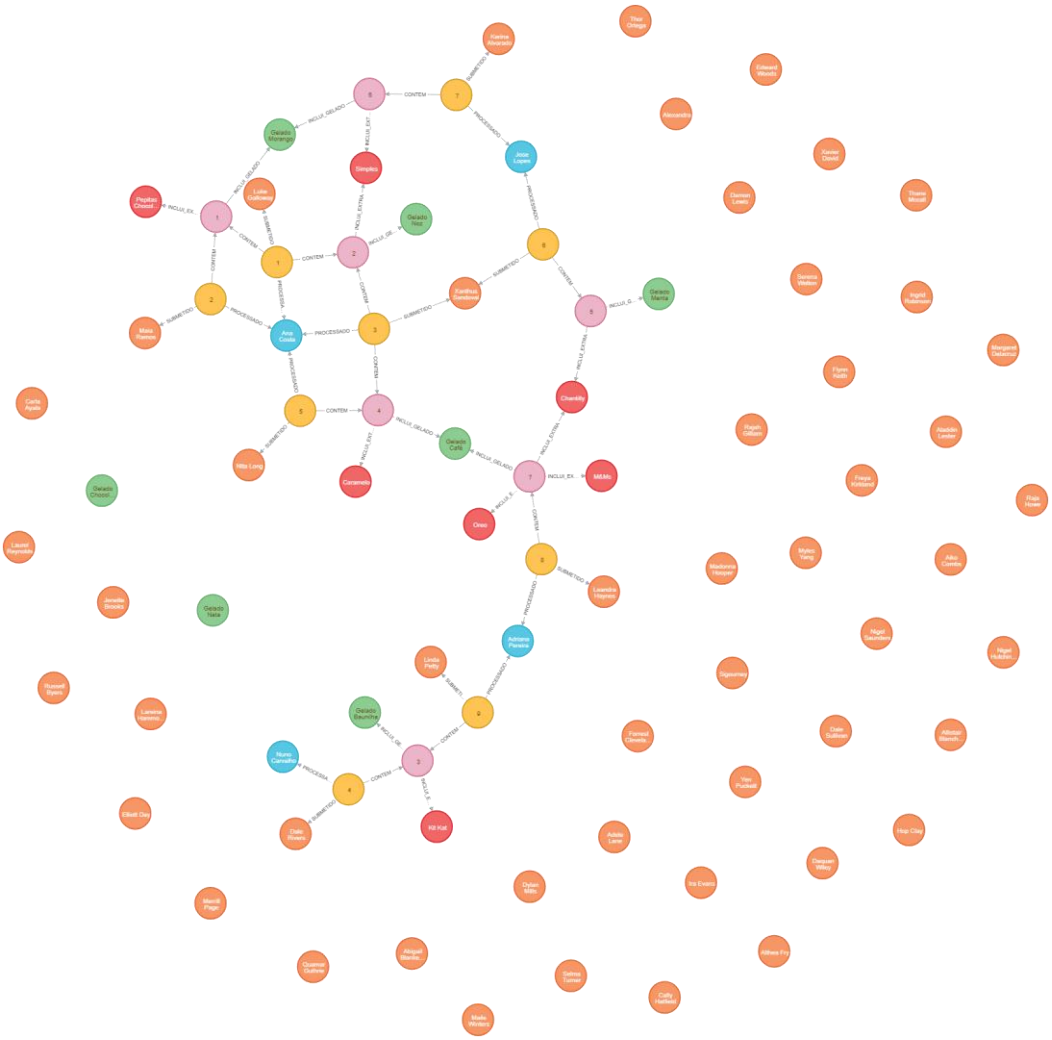

Figura 29. Grafo que representa a base de dados com todos os seus elementos

6.5.2. Imagem do grafo resultante sem os nodos não relacionados

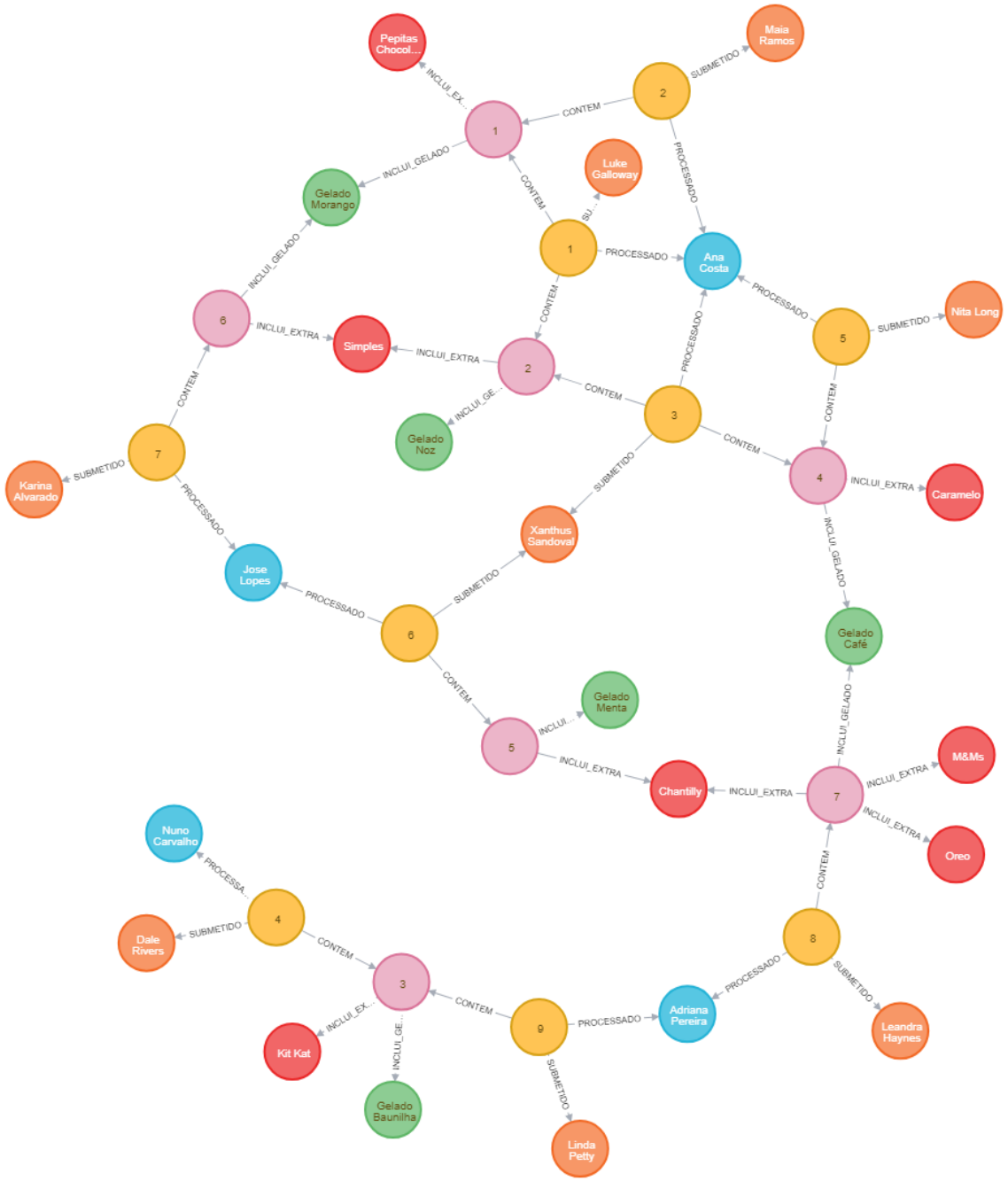

Figura 30. Grafo que representa a base de dados apenas com os elementos relacionados

6.6. Tradução das interrogações do utilizador para *Cypher*

- **1ª Querie**

  Os nomes e as quantidades dos 3 gelados base mais vendidos em junho de 2015

```
//first querie
MATCH (p:Pedido)-[co:CONTEM]->(:Criacao_Gelado)-[:INCLUI_GELADO]->
(gb:Gelado_Base)
WITH apoc.date.parse(p.Data,'s','yyyy-MM-dd HH:mm:ss') AS date,
apoc.date.parse('2015-06-01 00:00:00','s','yyyy-MM-dd HH:mm:ss') AS iDate,
apoc.date.parse('2015-07-01 00:00:00','s','yyyy-MM-dd HH:mm:ss') AS fDate,
gb.Nome AS name, co.Quantidade AS quantidade
ORDER BY quantidade DESC
WHERE date>iDate AND date<fDate
RETURN name, sum(quantidade) as quantity
LIMIT 3
```

Figura 31. 1ª *Querie* em *Cypher*

- **2ª Querie**

  Exibir o NIF e o nome do cliente mais velho que efetuou um pedido

```
//second querie
MATCH (:Pedido) -[:SUBMETIDO]-> (c:Cliente)
WITH apoc.date.parse(c.Data_Nascimento,'s','yyyy-MM-dd') as date, c.Nif as
nif, c.Nome as name
RETURN nif, name
ORDER BY date
LIMIT 1
```

Figura 32. 2ª *Querie* em *Cypher*

- **3ª Querie**

  Exibir nomes de todos os funcionários que processaram pedidos com gelados de morango

```
//third querie
MATCH (f:Funcionario)<-[:PROCESSADO]-(:Pedido)-[co:CONTEM]->(:Criacao_Gelado)-
[:INCLUI_GELADO]->(gb:Gelado_Base)
WHERE gb.Nome = 'Gelado Morango'
RETURN f.Nome AS funcionario, sum(co.Quantidade) as quantidade
ORDER BY quantidade DESC
```

Figura 33. 3ª *Querie* em *Cypher*

6.7. Revisão do sistema implementado com o utilizador

Tal como fizemos com a base de dados relacional, era necessário mostrar ao senhor Oleb o resultado final do processo de migração da base de dados. Imediatamente, obtivemos uma reação extremamente positiva da sua parte, mostrando, através da sua intuição, compreender a informação que estava a ser apresentada, ao contrário do que acontecera com a primeira base de dados.

O senhor Oleb mencionou ainda que a existência de uma base de dados orientada a grafos permitiria aos seus utilizadores um melhor manuseamento da mesma face às complicações que tivera com a outra. Em determinada altura, achou por bem adicionar mais alguns campos informativos para o cliente preencher aquando do registo, o que se revelou uma

grande dor de cabeça para os seus técnicos, visto que teriam que reestruturar algumas entidades da base de dados. Com esta base de dados, essa particularidade, tal como outras, foi facilmente resolvida.

6.8. Análise crítica

Com a realização desta parte do projeto, podemos mostrar as principais diferenças entre uma base de dados relacional e uma não relacional ao nosso cliente.

Ao contrário do que aconteceu com a conceção da base de dados relacional, não foi necessário construir os modelos mas antes adequar os que foram realizados previamente. Deste modo, passámos por diversas etapas e tivemos de pesquisar o que era preciso para fazer essa migração.

Enquanto que na primeira fase usamos uma linguagem chamada *SQL*, nesta fase foi usada a linguagem *Cypher* que nos permitiu criar nodos, criar relacionamentos e resolver as queries, as mesmas que foram usadas no modelo relacional.

O senhor Oleb ficou a conhecer as principais características e mudanças ao utilizar o *Neo4j* entre elas: possuir grande grau de distribuição dos dados que garante que o sistema fica menos tempo não disponível. Por outro lado, o modelo relacional é mais rigoroso na perseverança das suas informações.

A comparação que fizemos entre uma base de dados relacional e não relacional permitiu-nos mostrar determinados conceitos, o modo como ambas funcionam e a sua respetiva importância. Por um lado, as bases de dados relacionais são as mais usadas no mercado de trabalho, no entanto, as não relacionais estão a ganhar cada vez mais importância.

Com esta parte do projeto, a nossa equipa adquiriu também um espírito mais crítico para possíveis situações futuras em que tenhamos de implementar uma base de dados.

7. Conclusões e Trabalho Futuro

A implementação e elaboração de um sistema de bases de dados é uma tarefa que requer diferentes cuidados ao longo da mesma. Desta maneira, é preciso seguir certos passos para o seu desenvolvimento, nomeadamente a análise dos requisitos com o cliente, onde são definidas as entidades e relacionamentos do projeto de forma a otimizar a gerência e armazenamento dos dados em causa, tendo em conta a melhor performance possível para o sistema. A base de dados em questão foi marcada como um sistema simples, mas não simplista, do pretendido, sendo flexível para adotar outras áreas de funcionamento da empresa, que o utilizador deseje adquirir.

Na tentativa de minimizar erros de organização foi seguida uma metodologia que permitiu a sistematização do processo de criação do sistema.

O grupo criador da base de dados desta empresa seguiu a metodologia sugerida na folha informativa disponibilizada pelo professor da unidade curricular. Então, numa fase inicial do projeto, foi elaborado o modelo conceptual, sendo-o sempre com conhecimento e consentimento do cliente. Numa fase secundária, foram elaborados os modelos lógico e esquema físico.

Chegou-se à conclusão de que os objetivos foram concluídos, pois conseguimos elaborar uma base de dados funcional e representativa dos requisitos da *GELATUM*. Isto foi cumprido, também, devido a manutenção regular da base de dados para evitar a diminuição da performance da mesma.

8. Referências Bibliográficas

Thomas M. Connolly, Carolyn E. Begg - Database Systems: A Practical Approach to Design, Implementation and Management - 4th Edition.

Lista de Siglas e Acrónimos

BD Base de Dados

ER Entidade-Relacionamento

SGBD Sistema de Gestão de Base de Dados

Anexos

I. *Script* de povoamento

```
USE GELATUM;

-- DROP DATABASE GELATUM;

-- POVOAMENTO DA TABELA CLIENTE
INSERT INTO Cliente
(nome_cliente, password_cliente, nome_utilizador, nif, email, telemovel, data_nascimento)
VALUES
        ('Luke Galloway','_y','LGalloway13',4715,'Kevin@Ut.us',910200300,'1963-11-13'),
        ('Merrill Page','>gW;}9J','Page1964',1117,'Kerry@parturient.net',910200308,'1964-10-26'),
        ('Dale Sullivan','_vb0D','DaleS1',5448,'Mannix@leo.edu',910200316,'1967-01-06'),
        ('Xanthus Sandoval','BFz/rei?=F','XSandy_19',1814,'Barbara@Phasellus.com',910200324,'1970-12-19'),
        ('Damon Lewis','{','DL25',8260,'Tarik@inceptos.us',910200332,'1970-12-25'),
        ('Sigourney England','K~+bB8H7p%','England_N1',3314,'Mufutau@tincidunt.org',910200340,'1972-06-01'),
        ('Aladdin Lester','eKzRV6iv','_Aladdin_',6399,'Libby@natoque.com',910200348,'1976-11-13'),
        ('Hop Clay','s#8z!;UDV','the_last_HOP(e)',5229,'Lysandra@morbi.gov',910200356,'1979-07-01'),
        ('Elliott Day','hlSJ5d35','DAY79',2248,'Willa@Cras.net',910200364,'1979-08-14'),
        ('Laurel Reynolds','@p>Dua','Reynolds',745,'Miriam@tristique.org',910200372,'1979-09-02'),
        ('Thor Ortega','Z6pkl','Thor_lightning82',9166,'Jeanette@tempus.net',910200380,'1982-01-27'),
        ('Selma Turner','(aI{ID`NfY','90Selma',2425,'Shea@montes.gov',910200388,'1990-11-16'),
        ('Dale Rivers','OKiI-k_c','RiversDale',7381,'Keaton@odio.edu',910200396,'1991-05-12'),
        ('Cally Hatfield','Xl_D~m1dT','CallyHat',2763,'Candace@lacus.us',910200404,'1991-10-15'),
        ('Adele Lane','pwq!uoFX','Adele_#1',2992,'Kadeem@Cum.net',910200412,'1994-07-27'),
        ('Maia Ramos','#H','last_Maia',1387,'Ciaran@Curae.edu',910200420,'1995-07-24'),
        ('Freya Kirkland','a','Fkirk',5785,'Felicia@dignissim.org',910200428,'1995-12-18'),
        ('Yen Puckett','~Jf Y~M!F','yenP96',3455,'Driscoll@Nam.gov',910200436,'1996-07-10'),
        ('Jenette Brooks','OzeGLBz','J_Brooks',1296,'Alice@dictum.edu',910200444,'1998-10-08'),
        ('Serena Walton',';q <o','68serena75',6875,'Ebony@ultricies.edu',910200452,'1998-12-04'),
        ('Dylan Mills','CBoE{vLR','mills02',2320,'Gary@elit.org',910200460,'2002-05-24'),
        ('Quamar Guthrie','847)','daltonGuthrie',1866,'Dalton@nostra.com',910200468,'2000-11-06'),
        ('Maile Winters','jiW','winters_is_comming',2635,'Whoopi@In.net',910200476,'2001-11-25'),
        ('Madonna Hooper','7v','queen_Madonna',2473,'Kirk@quam.gov',910200484,'2003-02-08'),
        ('Nigel Saunders','fvRh','nigelSolomon',5521,'Solomon@libero.net',910200492,'2000-12-01'),
        ('Ingrid Robinson',';DE:/-','ingridRbsn',6136,'Kevin@sem.us',910200500,'1965-06-26'),
        ('Althea Fry','v%duoe/','fry_chicken',3012,'Shelly@commodo.gov',910200508,'1966-02-01')
```

```
        ('Ingrid Robinson',';DE:/-','ingridRbsn',6136,'Kevin@sem.us',910200500,'1965-06-26'),
        ('Althea Fry','y%duoe/','fry_chicken',3012,'Shelly@commodo.gov',910200508,'1966-02-01'),
        ('Russell Byers','_','russ_byers',1272,'Xanthus@scelerisque.net',910200516,'1967-04-14'),
        ('Ira Evans',').Im~+','ira67',3451,'Armando@id.us',910200524,'1967-09-06'),
        ('Nita Long','-]','longNight',8012,'Melodie@imperdiet.net',910200532,'1968-10-08'),
        ('Nigel Hutchinson','p1&P','hutch74',5711,'Xenos@feugiat.gov',910200540,'1974-06-07'),
        ('Myles Yang','/EX?','yang_miles',4458,'Belle@faucibus.org',910200548,'1979-11-04'),
        ('Raja Howe','=jwG','raja_14_howe',6120,'Bianca@posuere.org',910200556,'1983-02-14'),
        ('Linda Petty','O','lindapetty84',7632,'Allistair@Proin.gov',910200564,'1984-10-05'),
        ('Edward Woods','W','edWood10',8048,'Nehru@orci.us',910200572,'1985-12-10'),
        ('Carla Ayala','bm`0gBg','carla1023',8921,'Tanek@Integer.gov',910200580,'1985-12-22'),
        ('Forrest Cleveland','f* IIt','forrest_not_gump',1197,'Robert@mi.com',910200588,'1988-10-23'),
        ('Rajah Gilliam','2LqcllZ','rajaG11',5330,'Howard@Cras.com',910200596,'1988-11-26'),
        ('Lareina Hammond','#<?','hamm1991',1741,'Marshall@ac.gov',910200604,'1991-03-10'),
        ('Abigail Blankenship','=1odm]','abbyBlank',2068,'Kiayada@pede.net',910200612,'1991-03-10'),
        ('Alexandra Levy','Q&6@','alexLevy',8507,'Jorden@porttitor.edu',910200620,'1993-02-20'),
        ('Daquan Wiley','HW.WUj*a-','wileynotWonka',3529,'Talon@facilisis.org',910200628,'1993-12-16'),
        ('Margaret Delacruz','r%O2}{','delacruz1994',6427,'Yvette@primis.gov',910200636,'1994-07-04'),
        ('Leandra Haynes','HqQj1g','haynesL97',93,'Mufutau@Suspendisse.us',910200644,'1997-03-24'),
        ('Thane Mccall','HNnPU','thane_the_man',6555,'Kyra@suscipit.com',910200652,'2000-11-28'),
        ('Flynn Keith','Z(\b.IKFb.','20keithFlynn02',8159,'Guinevere@adipiscing.us',910200660,'2002-09-13'),
        ('Allistair Blanchard','Uh','BlanchmyAllistair',5323,'Lawrence@primis.us',910200668,'1996-11-13'),
        ('Aiko Combs','c(Q_3','aiko_jpn',5532,'Moses@id.org',910200676,'2003-05-20'),
        ('Xavier David','Gu','professor_xavier',7138,'Kasper@tortor.us',910200684,'2003-06-17'),
        ('Karina Alvarado','Wl#b','kAlvarado21',4245,'Basil@cursus.com',910200692,'2003-08-21');
-- SELECT * FROM Cliente;



-- POVOAMENTO DA TABELA FUNCIONARIO
INSERT INTO Funcionario
(id_funcionario, nome_funcionario, password_funcionario)
VALUES
        (1,'Jose Lopes','jlopes78'),
        (id_funcionario,'Ana Costa','ac1986')
```

```
-- POVOAMENTO DA TABELA PEDIDO
INSERT INTO Pedido
(id_pedido,dataP,valor_total,tempo,pago,recolhido,desconto,id_funcionario,nif)
VALUES
        (1,'2015-06-01 12:02:00',6.6,15,1,1,1,2,4715),
        (id_pedido,'2015-06-01 14:08:00',2.4,15,1,1,1,2,1387),
        (id_pedido,'2015-06-01 02:00:00',2.1,15,1,1,1,2,1814),
        (id_pedido,'2016-06-04 00:04:00',10.5,15,1,1,1,3,7381),
        (id_pedido,'2015-06-03 20:00:00',7.5,15,1,1,1,2,8012),
        (id_pedido,'2015-06-07 00:10:00',18,15,1,1,1,1,1814),
        (id_pedido,'2017-08-28 08:08:08',2.1,10,1,1,1,1,4245),
        (id_pedido,'2018-05-22 12:09:08',3.4,5,1,1,1,4,93),
        (id_pedido,'2018-01-23 02:19:08',3,7,1,1,1,4,7632);
-- SELECT * FROM Pedido;



-- POVOAMENTO DA TABELA GELADO_BASE
INSERT INTO Gelado_Base
(id_gelado,nome_gelado, composicao_gelado, preco_gelado, stock_gelado)
VALUES
        (1,'Gelado Morango', 'LEITE magro reconstituído, morangos (31%), açúcar,
        xarope de glucose e frutose, gordura de coco, LACTOSE e proteínas lácteas
        (LEITE), LEITE magro em pó, espessantes, emulsionante, sumo de beterraba
        concentrado, aromas, sumo de limão concentrado, corante', 0.70, 100),
        (id_gelado,'Gelado Chocolate', 'LEITE magro reconstituído, NATAS (20%),
        açúcar, cacau magro em pó (6.5%), gema de OVO, LEITE magro concentrado, espessante', 0.90, 100),
        (id_gelado,'Gelado Nata', 'LEITE magro reconstituído, NATAS (20%), açúcar,
        LEITE magro em pó, emulsionante, aromas', 0.80, 100),
        (id_gelado,'Gelado Noz', 'LEITE magro reconstituído, açúcar, água, NOZES (9%),
        gordura de coco, óleo de girassol, LACTOSE e proteínas lácteas (LEITE),
        emulsionante, espessantes, aromas, corante', 0.60, 100),
        (id_gelado,'Gelado Café', 'LEITE magro reconstituído, açúcar, NATAS,água, MANTEIGA
        concentrada, LACTOSE e proteínas lácteas (LEITE), café Arábica solúvel, pasta de cacau,
        aromas, extrato de café, açúcar caramelizado, manteiga de cacau, espessantes, emulsionante', 0.50, 100)
```

```
        (id_gelado,'Gelado Café', 'LEITE magro reconstituído, açúcar, NATAS,água, MANTEIGA
        concentrada, LACTOSE e proteínas lácteas (LEITE), café Arábica solúvel, pasta de cacau,
        aromas, extrato de café, açúcar caramelizado, manteiga de cacau, espessantes, emulsionante', 0.50, 100),
        (id_gelado,'Gelado Baunilha', 'LEITE magro reconstituído, NATAS (25%), açúcar,
        LEITE magro concentrado, gema de OVO, espessante, aromas naturais, extrato de
        baunilha, vagem de baunilha', 0.50, 100),
        (id_gelado,'Gelado Menta', 'LEITE magro reconstituído, NATAS (23%), açúcar,
        LEITE magro concentrado, gema de OVO, espessante, aromas naturais, extrato de menta', 0.80, 100);
-- SELECT * FROM Gelado_Base;



-- POVOAMENTO DA TABELA EXTRA
INSERT INTO Extra
(id_extra,nome_extra, composicao_extra, preco_extra, stock_extra)
VALUES
        (1,'Pepitas Chocolate', '100% Chocolate Negro', 0.10, 30),
        (id_extra,'Chantilly', '90% Natas, 9% Açúcar, 1% Baunilha', 0.20, 30),
        (id_extra,'M&Ms', '90% Chocolate, 10% Açúcar', 0.5, 30),
        (id_extra,'Caramelo', '100% Açúcar', 0.15, 30),
        (id_extra,'Oreo', 'Farinha de trigo, açúcar, gordura vegetal, óleo vegetal,
        cacau, açúcar, sal', 0.5, 30),
        (id_extra,'Kit Kat', 'Farinha de trigo, açúcar, gordura vegetal, óleo vegetal,
        cacau, açúcar, sal', 0.5, 30),
        (id_extra,'Simples','',0,100);
-- SELECT * FROM Extra;



-- POVOAMENTO DA TABELA CRIACAO_GELADO
INSERT INTO Criacao_Gelado
(id_criacao,valor_criacao,id_gelado)
VALUES
        (1,0.8,1),
        (id_criacao,0.60,4),
        (id_criacao,1.50,6),
        (id_criacao,1.50,5),
        (id_criacao,3,7),
        (id_criacao,0.70,1),
        (id_criacao,1.70,5);
-- SELECT * FROM Criacao_Gelado;



-- POVOAMENTO CRIACAO_GELADO__EXTRA
INSERT INTO Criacao_Gelado__Extra
(id_criacao,id_extra,quantidade_extra)
VALUES
        (1,1,1),
        (2,7,1),
        (3,6,2),
        (4,4,2),
        (5,2,11),
        (6,7,1),
        (7,2,1),
        (7,3,1),
        (7,5,1);
-- SELECT * FROM Criacao_Gelado__Extra;



-- POVOAMENTO PEDIDO__CRIACAO_GELADO
INSERT INTO Pedido__Criacao_Gelado
(id_pedido,id_criacao,quantidade_criacao)
VALUES
        (1,1,6),
        (2,1,3),
        (3,2,1)
```

```sql
-- POVOAMENTO CRIACAO_GELADO__EXTRA
INSERT INTO Criacao_Gelado__Extra
(id_criacao,id_extra,quantidade_extra)
VALUES
        (1,1,1),
        (2,7,1),
        (3,6,2),
        (4,4,2),
        (5,2,11),
        (6,7,1),
        (7,2,1),
        (7,3,1),
        (7,5,1);
-- SELECT * FROM Criacao_Gelado__Extra;



-- POVOAMENTO PEDIDO__CRIACAO_GELADO
INSERT INTO Pedido__Criacao_Gelado
(id_pedido,id_criacao,quantidade_criacao)
VALUES
        (1,1,6),
        (2,1,3),
        (3,2,1),
        (4,3,7),
        (5,4,5),
        (1,2,3),
        (3,4,1),
        (6,5,6),
        (7,6,3),
        (8,7,2),
        (9,3,2);
-- SELECT * FROM Pedido__Criacao_Gelado;
```

II. *Script* criação de tabelas

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema GELATUM
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema GELATUM
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `GELATUM` DEFAULT CHARACTER SET utf8 ;
USE `GELATUM` ;

-- -----------------------------------------------------
-- Table `GELATUM`.`Cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Cliente` (
  `nif` INT NOT NULL,
  `nome_cliente` VARCHAR(45) NOT NULL,
  `password_cliente` VARCHAR(20) NOT NULL,
  `nome_utilizador` VARCHAR(20) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `telemovel` INT NOT NULL,
  `data_nascimento` DATE NOT NULL,
  PRIMARY KEY (`nif`))
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- -----------------------------------------------------
-- Table `GELATUM`.`Funcionario`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Funcionario` (
  `id_funcionario` INT NOT NULL AUTO_INCREMENT,
  `nome_funcionario` VARCHAR(45) NOT NULL,
  `password_funcionario` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`id_funcionario`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Pedido`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Pedido` (
  `id_pedido` INT NOT NULL AUTO_INCREMENT,
  `dataP` DATETIME NOT NULL,
  `valor_total` DECIMAL(5,2) NOT NULL,
  `tempo` INT NOT NULL,
  `pago` TINYINT NOT NULL,
  `recolhido` TINYINT NOT NULL,
  `desconto` DECIMAL(3,2) NOT NULL,
  `id_funcionario` INT NOT NULL,
  `nif` INT NOT NULL,
  PRIMARY KEY (`id_pedido`),
  INDEX `fk_Pedido_Funcionario1_idx` (`id_funcionario` ASC) VISIBLE,
  INDEX `fk_Pedido_Cliente1_idx` (`nif` ASC) VISIBLE,
  CONSTRAINT `fk_Pedido_Funcionario1`
    FOREIGN KEY (`id_funcionario`)
    REFERENCES `GELATUM`.`Funcionario` (`id_funcionario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_Cliente1`
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_Cliente1`
    FOREIGN KEY (`nif`)
    REFERENCES `GELATUM`.`Cliente` (`nif`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Gelado_Base`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Gelado_Base` (
  `id_gelado` INT NOT NULL AUTO_INCREMENT,
  `nome_gelado` VARCHAR(45) NOT NULL,
  `preco_gelado` DECIMAL(4,2) NOT NULL,
  `stock_gelado` INT NOT NULL,
  `composicao_gelado` VARCHAR(300) NOT NULL,
  PRIMARY KEY (`id_gelado`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Extra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Extra` (
  `id_extra` INT NOT NULL AUTO_INCREMENT,
  `nome_extra` VARCHAR(45) BINARY NOT NULL,
  `preco_extra` DECIMAL(4,2) NOT NULL,
  `composicao_extra` VARCHAR(100) NOT NULL,
  `stock_extra` INT NOT NULL,
  PRIMARY KEY (`id_extra`),
  UNIQUE INDEX `nome_UNIQUE` (`nome_extra` ASC) VISIBLE)
```

```
  PRIMARY KEY (`id_extra`),
  UNIQUE INDEX `nome_UNIQUE` (`nome_extra` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Criacao_Gelado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Criacao_Gelado` (
  `id_criacao` INT NOT NULL AUTO_INCREMENT,
  `valor_criacao` DECIMAL(4,2) NOT NULL,
  `id_gelado` INT NOT NULL,
  PRIMARY KEY (`id_criacao`),
  INDEX `fk_Criacao_Gelado_Gelado_Base1_idx` (`id_gelado` ASC) VISIBLE,
  CONSTRAINT `fk_Criacao_Gelado_Gelado_Base1`
    FOREIGN KEY (`id_gelado`)
    REFERENCES `GELATUM`.`Gelado_Base` (`id_gelado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Criacao_Gelado__Extra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Criacao_Gelado__Extra` (
  `id_criacao` INT NOT NULL,
  `id_extra` INT NOT NULL,
  `quantidade_extra` INT NOT NULL,
  PRIMARY KEY (`id_criacao`, `id_extra`),
  INDEX `fk_Criacao_Gelado_has_Extra_Extra1_idx` (`id_extra` ASC) VISIBLE,
  INDEX `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  CONSTRAINT `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1`
```

```
  PRIMARY KEY (`id_criacao`, `id_extra`),
  INDEX `fk_Criacao_Gelado_has_Extra_Extra1_idx` (`id_extra` ASC) VISIBLE,
  INDEX `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  CONSTRAINT `fk_Criacao_Gelado_has_Extra_Criacao_Gelado1`
    FOREIGN KEY (`id_criacao`)
    REFERENCES `GELATUM`.`Criacao_Gelado` (`id_criacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Criacao_Gelado_has_Extra_Extra1`
    FOREIGN KEY (`id_extra`)
    REFERENCES `GELATUM`.`Extra` (`id_extra`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Pedido__Criacao_Gelado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Pedido__Criacao_Gelado` (
  `id_pedido` INT NOT NULL,
  `id_criacao` INT NOT NULL,
  `quantidade_criacao` INT NOT NULL,
  PRIMARY KEY (`id_pedido`, `id_criacao`),
  INDEX `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  INDEX `fk_Pedido_has_Criacao_Gelado_Pedido1_idx` (`id_pedido` ASC) VISIBLE,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Pedido1`
    FOREIGN KEY (`id_pedido`)
    REFERENCES `GELATUM`.`Pedido` (`id_pedido`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1`
    FOREIGN KEY (`id_criacao`)
```

```
    REFERENCES `GELATUM`.`Extra` (`id_extra`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `GELATUM`.`Pedido__Criacao_Gelado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `GELATUM`.`Pedido__Criacao_Gelado` (
  `id_pedido` INT NOT NULL,
  `id_criacao` INT NOT NULL,
  `quantidade_criacao` INT NOT NULL,
  PRIMARY KEY (`id_pedido`, `id_criacao`),
  INDEX `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1_idx` (`id_criacao` ASC) VISIBLE,
  INDEX `fk_Pedido_has_Criacao_Gelado_Pedido1_idx` (`id_pedido` ASC) VISIBLE,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Pedido1`
    FOREIGN KEY (`id_pedido`)
    REFERENCES `GELATUM`.`Pedido` (`id_pedido`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Pedido_has_Criacao_Gelado_Criacao_Gelado1`
    FOREIGN KEY (`id_criacao`)
    REFERENCES `GELATUM`.`Criacao_Gelado` (`id_criacao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

III. *Script* funções

```sql
USE GELATUM;

SET SQL_SAFE_UPDATES = 0;
SET GLOBAL log_bin_trust_function_creators = 1;

-- vai buscar o extra pretendido (caso o deseje)
DELIMITER $$
CREATE FUNCTION Gelatum.escolheExtra(nomeE VARCHAR(30)) RETURNS VARCHAR(50) NOT DETERMINISTIC
    BEGIN
        DECLARE resultado VARCHAR(50);

        SELECT nome_extra INTO resultado
        FROM Extra
        WHERE nome_extra REGEXP nomeE;

        RETURN resultado;
END $$

-- vai buscar o preco de um extra atraves do nome
DELIMITER $$
CREATE FUNCTION Gelatum.precoExtra(nomeE VARCHAR(20)) RETURNS VARCHAR(30) NOT DETERMINISTIC
    BEGIN
        DECLARE resultado VARCHAR(50);

        SELECT preco_extra INTO resultado
        FROM Extra
        WHERE nome_extra REGEXP nomeE;

        RETURN resultado;
END $$


-- vai buscar o gelado escolhido
DELIMITER $$
CREATE FUNCTION Gelatum.escolheGelado(nomeG VARCHAR(30)) RETURNS VARCHAR(50) NOT DETERMINISTIC
```

```sql
-- vai buscar o gelado escolhido
DELIMITER $$
CREATE FUNCTION Gelatum.escolheGelado(nomeG VARCHAR(30)) RETURNS VARCHAR(50) NOT DETERMINISTIC
    BEGIN
        DECLARE resultado VARCHAR(50);

        SELECT nome_gelado into resultado
        FROM Gelado_Base
        WHERE nome_gelado LIKE CONCAT('Gelado',' ', nomeG);

        RETURN resultado;
END $$

-- vai buscar o preco de um gelado com um dado nome
DELIMITER $$
CREATE FUNCTION Gelatum.getPrecoG(nomeG VARCHAR(20)) RETURNS INT NOT DETERMINISTIC
    BEGIN
        DECLARE resultado INT;

        SELECT preco_gelado into resultado
        FROM Gelado_Base
        WHERE nome_gelado LIKE CONCAT('Gelado',' ', nomeG);

        RETURN resultado;

END $$

DELIMITER $$
-- vai retornar um tempo aleatorio
CREATE FUNCTION calcTempo() RETURNS INT NOT DETERMINISTIC
    BEGIN
        DECLARE tempo INT DEFAULT FLOOR(RAND() * 10) + 1;

        RETURN tempo;
```

```sql
        RETURN tempo;
END $$

 -- vai retornar um bool indicando se o pedido foi pago ou nao
DELIMITER $$
CREATE FUNCTION setPago() RETURNS BOOL NOT DETERMINISTIC
    BEGIN
        DECLARE num INT DEFAULT FLOOR(RAND() * 2);
        DECLARE pago BOOL;

        IF num=1
            THEN SET pago=TRUE;
            ELSE SET pago=FALSE;
        END IF;

        RETURN pago;
END $$

SELECT setPago();

-- vai retornar um bool indicando se o pedido foi recolhido

DELIMITER $$
CREATE FUNCTION setRecolhido() RETURNS BOOL NOT DETERMINISTIC
    BEGIN
        DECLARE num INT DEFAULT FLOOR(RAND() * 2);
        DECLARE rec BOOL;

        IF (setPago() AND num=1)
            THEN SET rec=TRUE;
            ELSE IF (setPago() AND num=0)
                THEN SET rec=FALSE;
            ELSE
                SET rec=FALSE;
            END IF;
```

```
        DECLARE num INT DEFAULT FLOOR(RAND() * 2);
        DECLARE rec BOOL;

        IF (setPago() AND num=1)
            THEN SET rec=TRUE;
            ELSE IF (setPago() AND num=0)
                THEN SET rec=FALSE;
            ELSE
                SET rec=FALSE;
            END IF;
        END IF;

    RETURN rec;
END $$

-- Calcula o valor total de um pedido
DELIMITER $$
CREATE FUNCTION valorTot(id_pedido INT) RETURNS DECIMAL(5,2) NOT DETERMINISTIC
BEGIN
    DECLARE resultado DECIMAL(5,2);

    SELECT SUM((PCG.quantidade_criacao * CG.valor_criacao) * (1 - P.desconto))
    INTO resultado
    FROM  Pedido AS P
    INNER JOIN Pedido__Criacao_Gelado AS PCG
    ON P.id_pedido = PCG.id_pedido
        INNER JOIN Criacao_Gelado AS CG
        ON PCG.id_criacao = CG.id_criacao
        WHERE (id_pedido = PCG.id_pedido);

    RETURN resultado;
END $$
```

IV. *Script triggers*

```
  -- Verifica se no dia do pedido o cliente faz anos; caso afirmativo, o valor a pagar passa a 0
  DELIMITER $$
CREATE TRIGGER verificaAniversario BEFORE INSERT ON GELATUM.PEDIDO
FOR EACH ROW
BEGIN
    DECLARE dataN DATE;
    SET dataN = (SELECT data_nascimento FROM Gelatum.Cliente WHERE nif=NEW.nif);

    IF(dayofmonth(new.dataP)=dayofmonth(dataN) AND month(new.dataP)=month(dataN))
    THEN
        SET new.valor_total=0;
        SET new.desconto=1.0;
    END IF;
END $$
```

V. *Script* conversão de dados em ficheiros *csv*

```
USE gelatum;

SELECT *
FROM Cliente AS C
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/cliente.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Funcionario AS F
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/funcionario.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Extra AS E
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/extra.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Gelado_Base AS GB
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/gelado_base.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Criacao_Gelado__Extra AS CGE
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/criacao_gelado__extra.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Criacao_Gelado AS CG
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/criacao_gelado.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';
```

```
SELECT *
FROM Pedido AS P
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/pedido.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

SELECT *
FROM Pedido__Criacao_Gelado AS PCG
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/pedido__criacao_gelado.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';

-- SHOW VARIABLES LIKE "secure_file_priv";
```

VI. *Script* criação dos nodos (e importação de dados)

```
//cliente nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///cliente.csv" AS line
CREATE (c:Cliente { Nif: toInteger(line[0]),
Nome: (line[1]),
Password: (line[2]),
Username: (line[3]),
Email: (line[4]),
Telefone: toInteger(line[5]),
Data_Nascimento: (line[6])})
```

```
//extra nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///extra.csv" AS line
CREATE (e:Extra {Id_Extra: toInteger(line[0]),
Nome: (line[1]),
Preco: toFloat(line[2]),
Composicao: (line[3]),
Stock: toInteger(line[4])})
```

```
//gelado_base nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///gelado_base.csv" AS line
CREATE (g:Gelado_Base { Id_Gelado: toInteger(line[0]),
Nome: (line[1]),
Preco: toFloat(line[2]),
Stock: toInteger(line[3]),
Composicao: (line[4])})
```

```
//funcionario nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///funcionario.csv" AS line
CREATE (f:Funcionario {Id_Funcionario: toInteger(line[0]),
Nome: (line[1]),
Password: (line[2])})
```

```
//criacao_gelado nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///criacao_gelado.csv" AS line
CREATE (cg:Criacao_Gelado { Id_Criacao: toInteger(line[0]),
Valor: toFloat(line[1]),
Id_Gelado: toInteger(line[2])})
```

```
//criacao_gelado__extra nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///criacao_gelado__extra.csv" AS line
CREATE (cge:Criacao_Gelado__Extra {Id_Criacao: toInteger(line[0]),
Id_Extra: toInteger(line[1]),
Quantidade_Extra: toInteger(line[2])})
```

```
//pedido__criacao_gelado nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///pedido__criacao_gelado.csv" AS line
CREATE (pcg:Pedido__Criacao_Gelado { Id_Pedido: toInteger(line[0]),
Id_Criacao: toInteger(line[1]),
Quantidade: toInteger(line[2])})
```

```
//pedido nodes
USING PERIODIC COMMIT
LOAD CSV FROM "file:///pedido.csv" AS line
CREATE (p:Pedido { Id_Pedido: toInteger(line[0]),
Data: (line[1]),
Valor_Total: toFloat(line[2]),
Tempo: toInteger(line[3]),
Pago: toInteger(line[4]),
Recolhido: toInteger(line[5]),
Desconto: toInteger(line[6]),
Id_Funcionario: toInteger(line[7]),
Nif: toInteger(line[8])})
```

VII. *Script* criação dos relacionamentos

```
//cliente merge pedido
MATCH (c:Cliente)
MATCH (p:Pedido)
WHERE c.Nif = p.Nif
MERGE (p) -[:SUBMETIDO]-> (c);
```

```
//funcionario merge pedido
MATCH (f:Funcionario)
MATCH (p:Pedido)
WHERE f.Id_Funcionario = p.Id_Funcionario
MERGE (p) -[:PROCESSADO]-> (f);
```

```
//gelado_base merge criacao_gelado
MATCH (gb:Gelado_Base)
MATCH (cg:Criacao_Gelado)
WHERE gb.Id_Gelado = cg.Id_Gelado
MERGE (cg) -[:INCLUI_GELADO]-> (gb);
```

```
//extra merge criacao_gelado
MATCH (e:Extra)
MATCH (cge:Criacao_Gelado__Extra)
MATCH (cg:Criacao_Gelado)
WHERE e.Id_Extra = cge.Id_Extra AND cg.Id_Criacao = cge.Id_Criacao
MERGE (cg) -[:INCLUI_EXTRA{Quantidade_Extra: cge.Quantidade_Extra}]-> (e);
```

```
//criacao_gelado merge pedido
MATCH (cg:Criacao_Gelado)
MATCH (pcg:Pedido__Criacao_Gelado)
MATCH (p:Pedido)
WHERE cg.Id_Criacao = pcg.Id_Criacao AND pcg.Id_Pedido = p.Id_Pedido
MERGE (p) -[:CONTEM{Quantidade: pcg.Quantidade}]-> (cg);
```

VIII. *Script* criação dos *indexes*

```
//cliente index
CREATE INDEX ON :Cliente(Nif);
```

```
//extra index
CREATE INDEX ON :Extra(Id_Extra);
```

```
//gelado_base index
CREATE INDEX ON :Gelado_Base(Id_Gelado);
```

```
//funcionario index
CREATE INDEX ON :Funcionario(Id_Funcionario);
```

```
//criacao_gelado index
CREATE INDEX ON :Criacao_Gelado(Id_Criacao);
```

```
//pedido index
CREATE INDEX ON :Pedido(Id_Pedido);
```

IX. *Script* remoção dos dados (redundantes)

```
//delete criacao_gelado__extra nodes
MATCH (cge:Criacao_Gelado__Extra)
DELETE cge
```

```
//delete pedido__criacao_gelado nodes
MATCH (pcg:Pedido__Criacao_Gelado)
DELETE pcg
```

```
//remove properties Id_Funcionario and Nif from Pedidos
MATCH (p:Pedido)
REMOVE p.Id_Funcionario, p.Nif
RETURN p
```

```
//remove property Id_Gelado from Criacao_Gelado's
MATCH (cg:Criacao_Gelado)
REMOVE cg.Id_Gelado
RETURN cg
```