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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

IberoTrem – reserva de viagens

João da Cunha Coelho (a74859), José Miguel Ribeiro da Silva (a74601), Luís Miguel Moreira Fernandes (a74748), Pedro João Novais da Cunha (a73958)

Novembro, 2016

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

IberoTrem – reserva de viagens online

João da Cunha Coelho (a74859), José Miguel Ribeiro da Silva (a74601), Luís Miguel Moreira Fernandes (a74748), Pedro João Novais da Cunha (a73958)

Novembro, 2016

Resumo

Atualmente, a forte presença tecnológica associada à maior facilidade de acesso à internet veio permitir que sistemas de reservas online sejam cada vez mais frequentes e utilizados pelos mais diversos tipos de serviços. Se é verdade que estes tipos de sistemas oferecem garantias e facilidade aos diversos utilizadores no que diz respeito à aquisição de serviços, também, no que diz respeito às empresas que os disponibilizam, permitem maior facilidade de planeamento, organização e preparação.

Após a identificação do problema, numa primeira fase do trabalho, foi feita a respectiva contextualização com base em exemplos reais e bastante bem-sucedidos, como é o caso do sistema de reservas da companhia “Comboios de Portugal”.

Realizado o levantamento de requisitos, procedeu-se ao desenvolvimento do modelo conceptual que, posteriormente, foi validado pelo docente responsável pela Unidade Curricular. A etapa seguinte consistiu na passagem do modelo conceptual para o modelo lógico, seguindo as diretrizes propostas pelo livro recomendado. O mesmo processo foi utilizado para o desenvolvimento do modelo físico a partir do modelo lógico.

Ao longo do trabalho, alguns dos pontos da metodologia e a realização de algumas transações, onde assumimos o papel de utilizadores do sistema de reservas, permitiu-nos otimizar alguns dos aspectos do sistema de gestão de base de dados em questão.

De um modo geral, podemos concluir que o sistema implementado cumpre os requisitos do nosso caso de estudo.

Área de Aplicação: <<Desenho e arquitectura de Sistemas de Bases de Dados.>>

Palavras-Chave: << Bases de Dados Relacionais, Armazenamento de dados, Modelo Conceptual, Modelo Lógico, Modelo Físico, DiagramasER, Normalização.>>

Índice

Índice de Figuras

Índice de Tabelas

Lista de Siglas e Acrónimos

CC	Cartão de cidadão
Id	Identidade
SGBD	Sistema de gestão de bases de dados

1. Introdução

1.1. Contextualização

A IberoTrem é uma empresa de transporte ferroviário sediada em Portugal, que estabelece a ligação por linhas férreas entre várias cidades do Norte de Portugal e de Espanha (Galiza). Foi criada em 2001 com o intuito de se tornar a primeira companhia *low cost* a operar na região, proporcionando bilhetes para as suas viagens a um custo semelhante ao dos bilhetes com desconto por compra antecipada praticado pelas grandes companhias. Desde início o projeto foi visto como viável, dado que é relativamente frequente uma pessoa deparar-se com a necessidade de comprar um bilhete para uma viagem que não estava prevista com uma semana (ou mais) de antecedência. Por outro lado, também não é raro um indivíduo atrasar a marcação de uma viagem com receio que, até à data, algo lhe altere os planos. Porém, pode acontecer que, quando confirme a intenção de viajar, já seja tarde para usufruir dos descontos de compra antecipada. A IberoTrem oferece assim uma alternativa em conta a quem esteja em situações semelhantes (e não só).

1.2. Apresentação do Caso de Estudo

Inicialmente, os serviços de transporte da empresa eram prestados por apenas três linhas: Linha do Minho, Linha de Vigo e Linha de Chaves. Posteriormente foi levado a cabo um processo de alargamento da oferta, que acrescentou dois troços aos serviços da companhia (Linha de Miranda e Linha da Galiza), pelo que a direção decidiu estender o serviço de reservas de viagens à *web*.

Um grupo de estudantes de informática foi contratado, sendo-lhes pedido que implementassem no site da empresa, até então meramente informativo, uma secção própria para a reserva das viagens.

1.3. Motivação e Objectivos

Numa empresa de transportes ferroviários, torna-se difícil controlar os lugares disponíveis sem ajuda de um sistema informático. Com um *SGBD* (sistema de gestão de base de dados) melhora-se a eficiência de trabalho e a organização da informação. A informação fica mais fácil de gerir, e a manipulação dos dados é sempre efetuada de forma segura. Com a possibilidade de reserva online, torna-se essencial a existência da informação em formato digital, e um SGBD é a melhor opção.

Os principais objetivos na elaboração deste projeto são:

- Construir um SGBD, de forma eficiente, que contenha a informação referente às reservas de lugares, cada uma referente a um cliente e uma viagem.
- Comparar com situações realistas de forma a perceber que problemas podem aparecer e as consequências que esses mesmos problemas geram.
- Assimilar todos os conteúdos de forma a saber como construir um SGBD relacional, estando este funcional e devidamente organizado.

1.4. Estrutura do Relatório

Após a apresentação do caso de estudo e da análise dos respetivos requisitos, segue-se a construção do modelo conceptual. Estando este devidamente validado e adaptado ao problema que se coloca, passar-se-á ao modelo lógico, construído com base no anterior. Por fim, a *forward engineering* permitir-nos-á obter o modelo físico a partir do lógico, pelo que passaremos a ter uma base de dados capaz de responder a *queries*. Na próxima secção serão apresentadas as etapas de construção destes modelos, seguindo-se uma secção final com conclusões sobre o trabalho realizado, nomeadamente os seus pontos fortes e fracos.

2. Levantamento de requisitos

- A IberoTrem pretende obter uma base de dados que seja capaz de manter e manusear a informação relativa às viagens reservadas pelos seus clientes a partir do site da empresa.

- Este site deve permitir aos utilizadores registar-se, se assim o desejarem, fornecendo o nome, número de cartão de cidadão, data de nascimento e contactos - telefone e email.

- Após o registo, os utilizadores podem então efetuar a reserva online de bilhetes, ficando assim associado a cada reserva o preço, o lugar reservado no comboio e a data da viagem, bem como a hora prevista para partida e chegada.

- Foi referenciada a existência de um desconto especial de 25% para menores de 25 anos, de forma a promover este meio de transporte entre os mais jovens, e, como tal, o preço na reserva não é obrigatoriamente igual ao preço da viagem.

3. Desenvolvimento do Relatório

3.1. Construção do modelo conceptual

A modelação conceptual dos dados utiliza três conceitos chave: entidade, para aqueles elementos sobre os quais vai assentar a informação que será gerada, atributo, para os elementos que permitem caracterizar uma entidade, isto é, as características informativas específicas, e associação, que designa um relacionamento entre entidades.

Nesta secção são descritos e adaptados ao nosso caso de estudo os passos para a elaboração do modelo conceptual. Nota para o uso do *software TerraER* para o desenho do modelo *ER* (*Entity-Relationship*).

1. Identificação das entidades

Analisando os requisitos da base de dados levantados na apresentação do caso de estudo, chegou-se à seguinte tabela de entidades:

Nome da Entidade	Descrição	Sinónimo	Ação/Ocorrência
Cliente	Termo geral que descreve todas as pessoas que utilizam os serviços de viagens prestados pela companhia IberoTrem.	Utilizador	Cada cliente pode efetuar várias reservas (limitadas à marcação de apenas um lugar).
Reserva	Termo geral que descreve o serviço de marcação de um lugar numa viagem prestada pela companhia IberoTrem.	Marcação	Cada reserva está associada a apenas uma viagem, podendo vários utilizadores fazer reservas para a mesma viagem.
Viagem	Termo geral que descreve a deslocação entre uma origem e um destino, durante um dado intervalo de tempo.	Trajeto, Percurso, Deslocação	Cada viagem é realizada por um único comboio, entre um destino e uma origem, tendo associada a si várias reservas.
Comboio	Termo geral que descreve o meio de transporte utilizado pela companhia IberoTrem para prestar os seus serviços.	-	Cada comboio efetua várias viagens.
Estação	Termo geral que descreve o lugar de embarque e desembarque dos clientes da companhia IberoTrem.	Paragem	Cada estação é simultaneamente origem e destino de várias viagens.

Tabela 1 – Identificação das entidades

2. Identificação dos relacionamentos

Com base nas entidades definidas, procedemos à análise das relações que entre elas se estabelecem de modo a que o sistema de base de dados possa responder às necessidades da empresa. Daqui resultou a seguinte tabela:

Entidade	Multiplicidade	Relação	Multiplicidade	Entidade
Cliente	1…1	Efetua	0…N	Reserva
Reserva	0…N	Relativa a	1…1	Viagem
Viagem	0…N	Efetuada por	1…1	Comboio
Viagem	0…N	Tem origem na	1…1	Estação
Viagem	0…N	Tem destino na	1…1	Estação

Tabela 2 – Identificação dos relacionamentos

3. Identificação e associação de atributos com entidades ou relacionamentos e determinação do domínio dos atributos

Nesta fase, identificaremos os atributos que constarão na base de dados, justificando a sua escolha. Estes poderão estar associados a entidades ou relacionamentos, mas no nosso modelo conceptual todos são referentes a entidades.
No que ao *Cliente* diz respeito, a informação que consideramos importante preservar foi: o número do cartão de cidadão, para termos um elemento único identificativo da pessoa; o nome, para aproximar a empresa do cliente com um tratamento mais pessoal; a data de nascimento, para avaliar se está sujeito a descontos em função da idade ou não; o contacto, nomeadamente o endereço de email, usado para enviar a confirmação da reserva e, caso haja registo no site, para fazer o login, e o telefone, útil para trocar informações com o cliente, constituindo uma alternativa ao email.

Já para a *Reserva,* os atributos escolhidos foram o lugar reservado no comboio da viagem, que é o elemento essencial na reserva de qualquer viagem e como tal não poderia deixar de fazer parte, a data da viagem reservada, importante para complementar a especificação da viagem e para controlar o período até ao qual é possível cancelar a reserva, o preço, derivado do preço da viagem dado que pode estar sujeito a desconto, e o id_reserva, que torna cada reserva única.
Quanto à *Viagem*, são quatro os atributos considerados importantes: o id_viagem, que atribui um valor único a uma viagem, o preço, em função do itinerário da viagem, a hora de partida e a hora de chegada, que juntas especificam, dentro do trajeto previsto, qual é a viagem referida.

A *Estação* foi caracterizada com recurso ao id_estação, para garantir a unicidade de cada estação, à cidade onde se localiza e ao nome da estação, que varia dada a possibilidade de existirem várias estações por cidade.
Por último, a entidade *Comboio* caracterizou-se com recurso a uma identidade (id_comboio), que garante que um comboio é único, aos números dos lugares do comboio especificado para a viagem, que é importante para saber quais os lugares que ainda podem ser reservados, e ao total de lugares do comboio.

Após estarem definidos os atributos relevantes para cada entidade, procedeu-se à análise do domínio – conjunto de valores que um atributo pode assumir - de cada um. Começando pelos atributos associados ao Cliente, o *CC* assume o valor de uma *string* de no máximo 15 caracteres. Note-se que o domínio deste atributo não poderia ser um inteiro porque, ao contrário dos *CC*'s portugueses (8 caracteres numéricos), os

documentos de identificação espanhóis, por exemplo, são constituídos por 8 números e uma letra final; a data de nascimento é do tipo Date e pode variar entre 01/01/1000 e 31/12/9999 sempre o formato yyyy-mm-dd; o nome será uma *string* com no máximo 64 caracteres; o telefone será representado por um inteiro com 15 dígitos, para cobrir possíveis contactos com os habituais 9 dígitos; o endereço de email será uma string com no máximo 64 caracteres (emailexemplo@exemplo.com).

Para os atributos da *Reserva* usamos um inteiro (1 – MAXINT), considerando o MAXINT um número suficiente para suportar o número de reservas da empresa, para o id_empresa. O preço será um DECIMAL(5,2) de valor positivo, o lugar um inteiro com valor de 1 a 10 e a data será do tipo *Date,* com o limite inferior sendo o próprio dia, o superior é 9999-12-31 e assumindo o formato yyyy-mm-aa.

Quanto à *Viagem*, o atributo id_viagem será um inteiro entre 1 - MAXINT, o preço um DECIMAL(5,2) positivo e as horas de chegada e partida do tipo Time com formato hh:mm:ss.

Para a *Estação*, teremos um id_estação inteiro entre 1 - MAXINT, uma *string* a representar o nome da cidade onde se localiza e outra *string* representando o nome da estação, ambas com tamanho máximo de 32 carateres.

Por fim, o id_comboio é representado por um inteiro entre 1 e MAXINT, os lugares, atributo multi-valor, são representados por inteiros entre 1 e 10 e o Nr_lugares é um inteiro, sem necessidade de intervalo de variação dada a reduzida e fixa capacidade dos comboios da empresa (10).

A junção das escolhas enunciadas anteriormente conduziu à seguinte tabela:

Entidade	Atributos	Descrição	Domínio	Nulls	Multi-valorado	Derivado	Valor default
Cliente	CC	Número do CC, que identifica o cliente;	String até 15 caracteres;	Não	Não	Não	Não
	Data de Nasc.	Data de Nascimento do cliente;	Data (1000/01/01-9999/12/31) com formato dd/mm/aa;	Não	Não	Não	Não
	Nome	Nome do cliente;	String até 64 caracteres;	Não	Não	Não	Não
	Contacto						
	Telefone	Contacto telefónico do cliente;	String até 15 caracteres;	Sim	Não	Não	Não
	Email	Endereço de email do cliente;	String até 64 caracteres;	Não	Não	Não	Não
Reserva	Id_reserva	Identifica a reserva (valor único);	Inteiro entre 1 e MAXINT;	Não	Não	Não	Não
	Preço	Preço da reserva do lugar numa viagem;	Decimal(5,2) (positivo);	Não	Não	Sim	Não
	Lugar	Lugar no comboio reservado;	Inteiro entre 1 e 10;	Não	Não	Não	Não
	Data	Data da viagem reservada;	Data (atualidade-9999/12/31) com formato dd/mm/aa	Não	Não	Não	Não
Viagem	Id_viagem	Identifica a viagem (valor único);	Inteiro entre 1 e MAXINT;	Não	Não	Não	Não
	Hora Partida	Hora a que o comboio da viagem parte da estação origem;	Time, formato hh:mm:ss	Não	Não	Não	Não
	Hora Chegada	Hora a que o comboio da viagem chega à estação destino;	Time, formato hh:mm:ss	Não	Não	Não	Não
	Preço	Preço da viagem;	Decimal (5,2) (positivo);	Não	Não	Não	Não

Estação	Nome	Nome da estação utilizada pelo comboio da viagem;	String até 32 caracteres;	Não	Não	Não	Não
	Cidade	Nome da cidade onde se localiza a estação;	String até 32 caracteres;	Não	Não	Não	Não
	Id_estação	Identifica a estação (valor único);	Inteiro entre 1 e MAXINT;	Não	Não	Não	Não
Comboio	Lugares	Números dos lugares do comboio;	Inteiros entre 1 e 10;	Não	Sim	Não	Não
	Nr_lugares	Identifica o comboio (valor único);	Inteiro (10)	Não	Não	Não	Não
	Id_comboio	Identifica o comboio (valor único);	Inteiro entre 1 e MAXINT;	Não	Não	Não	Não

Tabela 3 - Identificação e associação dos atributos com as entidades.

4. Determinação das chaves candidatas, primárias e suplentes das entidades

Na entidade *Cliente* podemos facilmente verificar que temos apenas 3 atributos candidatos - o *CC*, Telefone e Email – pois são os únicos a garantirem a unicidade de cada ocorrência dessa entidade. Tendo em conta ainda os critérios para a escolha da melhor chave candidata, que passará a ser a primária, notamos que tanto o atributo Telefone como o Email são atributos mais prováveis de ter os seus valores alterados do que o *CC* do Cliente, fazendo desta maneira com que este atributo seja a escolha lógica como chave primária. Telefone e Email serão então chaves suplentes.

Na entidade *Reserva*, mais uma vez, facilmente se verifica a não unicidade que os atributos Preço, Data e Lugar garantiriam. Poderíamos então pensar numa chave primária composta com Data e Lugar, mas devido ao facto de poderem ocorrer duas viagens no mesmo dia (Data igual) e em que o mesmo lugar fosse ocupado em ambas, excluímos esta hipótese, não nos restando alternativa senão a criação de um atributo Id_reserva, que garantirá a unicidade de cada ocorrência. Este atributo será apenas um inteiro, logo terá sempre o menor valor possível.

Na entidade *Viagem* temos atributos como Hora Partida e Hora Chegada, bem como Preço, atributos estes que logo à partida nunca nos garantiriam unicidade, visto o preço ser igual para várias viagens e visto serem realizadas viagens com a mesma hora de partida e chegada em dias diferentes. Portanto, achou-se por bem a criação de um atributo Id_viagem que, tal como Id_reserva, será um inteiro que tomará sempre o menor valor possível de forma a assegurarmos a unicidade das ocorrências.

Na entidade *Estação* conseguimos perceber que os atributos Nome e Cidade sozinhos não nos garantem a unicidade necessária, porém se considerarmos uma chave primária composta por ambos os atributos, verifica-se a tal unicidade requerida. No entanto, e por uma questão de eficiência, decidimos não usar esta chave composta como chave primária e criar para isso um atributo Id_estação, que, novamente, será um inteiro que tomará sempre o menor valor possível e que garanta a unicidade, sendo este um atributo que nos facilitará a comparação e nos garantirá um aumento na eficiência. Poderemos também considerar a chave primária composta (Nome, Cidade) como uma chave suplente.

Na entidade *Comboio* temos os atributos Nr_lugares e Lugar, que são logo descartados como candidatos visto que os vários comboios têm o mesmo número de lugares e temos repetição dos valores dos números identificativos do lugar, abrindo assim vaga à criação de um atributo Id_comboio que, mais uma vez, será um inteiro que irá tomar sempre o menor valor possível que garanta unicidade.

5. Procura de redundâncias no modelo

Para se verificar que não existem quaisquer evidências de redundância é necessária uma averiguação que assenta em três passos:

1) **Examinar relações 1:1**. Dado que o nosso modelo não possui nenhuma relação 1:1 podemos concluir que, no que diz respeito a este passo, não existe qualquer suspeita de redundância.

2) **Remoção de relações redundantes**. Uma das situações que levanta mais suspeitas no que diz respeito à presença de relações redundantes são os modelos conceptuais onde existe mais do que um caminho entre duas entidades. No entanto, isto não implica necessariamente redundância, como podemos verificar pelo nosso modelo. Apesar de se estabelecer duas relações entre a entidade 'Viagem' e a entidade 'Estação', estamos perante duas associações diferentes e, neste caso particular, até mesmo opostas. Podemos concluir assim que, também no que diz respeito a este passo, não existe qualquer suspeita de redundância, já que todas as relações presentes no modelo lógico vêm proporcionar informação adicional e única às relações entre entidades.

3) **Considerar a dimensão do tempo**. Neste modelo conceptual, apenas a entidade *Estação* possui dois caminhos alternativos, porém estes representam relações diferentes tal como referimos anteriormente.

6. Validação do modelo conceptual contra transações

Neste passo deveremos assegurar que o modelo conceptual suporta os possíveis requerimentos dos utilizadores da companhia IberoTrem. De seguida enumeraremos algumas das transações que na nossa opinião, caso efetuadas com sucesso, demostram a validade do modelo conceptual em questão:

- **Registar Cliente**. Para registar um cliente é necessário guardar as seguintes informações sobre o mesmo: Cartão de cidadão (ou documento de identificação equivalente, que permite distinguir o cliente de outros), nome, data de nascimento e contactos do mesmo. No modelo conceptual desenvolvido, a entidade Cliente tem como atributos: **CC, nome, data de nascimento** e **Contacto** composto por **email** e **telefone**. Pode concluir-se que a entidade Cliente tem todos os atributos necessários para a transação.
- **Registar comboio.** Para registar um comboio é necessário guardar um identificador único, que o distingue. Também é importante guardar os seus lugares e ainda o número de lugares, este último meramente informativo. No modelo conceptual desenvolvido, a entidade Comboio tem como atributos: **id_comboio, nr_lugares** e ainda, como atributo multi-valor, **lugar**. Pode concluir-se que a entidade Comboio tem todos os atributos necessários para a transação.
- **Registar Reserva.** Para registar reserva é necessário guardar as seguintes informações: identificador único para a distinguir, lugar que fica reservado, data da viagem e ainda o preço da reserva. Também é preciso saber a que viagem se refere e que cliente efetuou a reserva. No modelo conceptual desenvolvido a entidade Reserva tem como atributos o **lugar,** o **id_reserva** e a **data**. Tem ainda, como atributo derivado, o **preço.** Existem os relacionamentos **1 Cliente efetua N reservas** e **N reservas são relativas a 1 Viagem.** Podemos concluir que a entidade Reserva possui os atributos necessários e está relacionada com as entidades relevantes para esta transação.
- **Consultar lugares disponíveis em alguma viagem, numa determinada data.** Para consultar os lugares disponíveis, é necessário que sejam apresentados todos os lugares do comboio que efetua a viagem retirando os reservados para essa data. No modelo conceptual desenvolvido, cada reserva tem a data correspondente, e existem os relacionamentos **N reservas são relativas a 1 Viagem e N viagens são efetuadas por 1 comboio.** Com estes

relacionamentos, chegamos aos lugares do comboio. Retirando destes as reservas dessa viagem para o dia em questão, obtemos a informação pretendida. Podemos concluir que existem os atributos e relacionamentos necessários para esta transação.

O resultado final dos 6 passos abordados anteriormente está representado no esquema conceptual que a seguir se apresenta na figura seguinte:

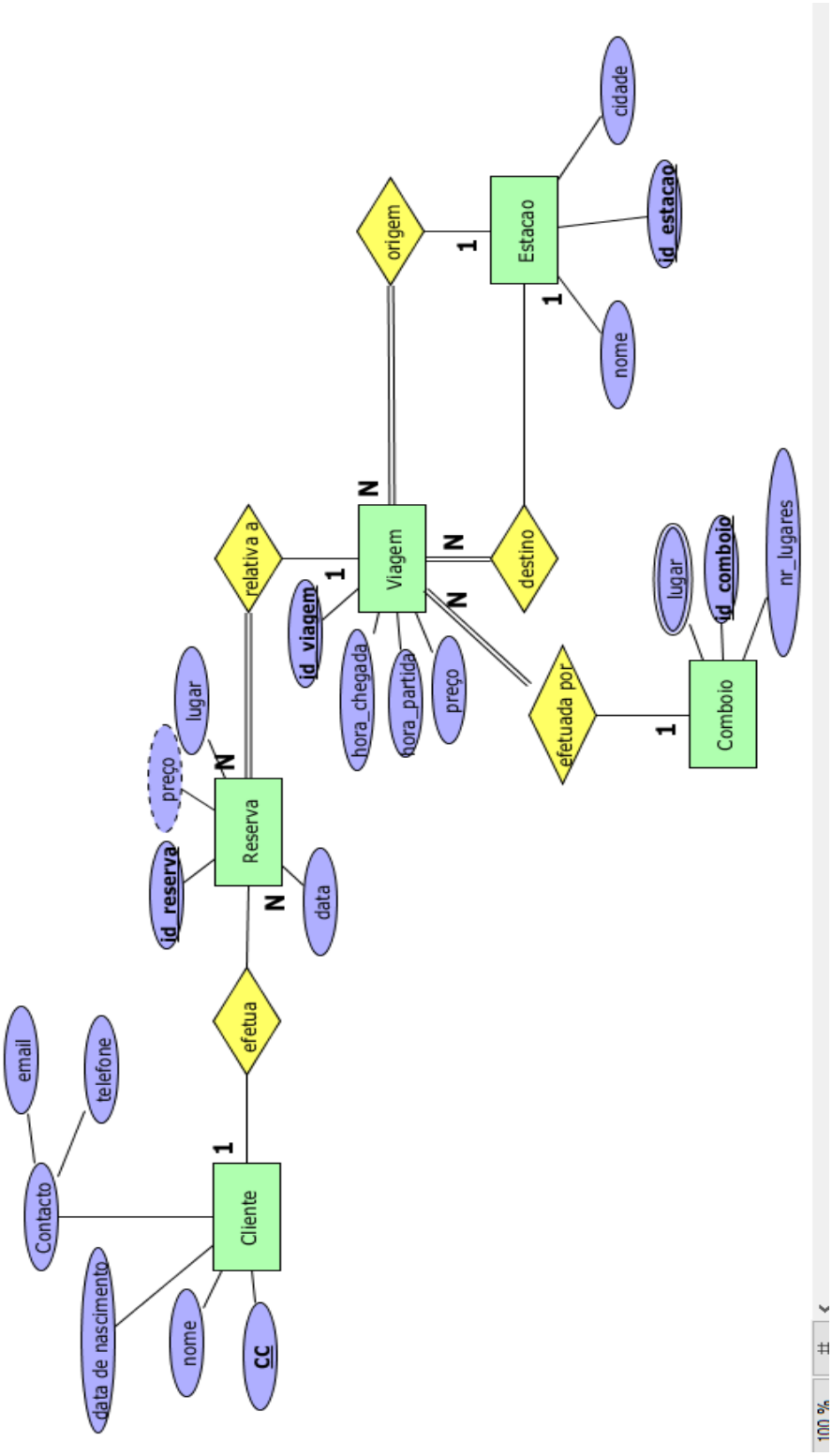

Figura 1 – Ilustração do modelo conceptual

3.2. Construção do modelo lógico

Nesta secção esclareceremos como se construiu o modelo lógico, tendo por base a conversão do modelo conceptual previamente abordado.

1. Derivar relacionamentos para o modelo lógico

i. Entidades fortes

As entidades fortes que constituem o nosso modelo conceptual são: Cliente, Reserva, Viagem, Estação e Comboio. Para cada uma destas entidades criámos uma relação que inclui os seus respetivos atributos. A representação é feita seguindo o que a metodologia recomenda.

Cliente (CC, Nome, Data_de_Nascimento, Email, Telefone)
Primary key CC

Reserva (Id_reserva, Preço, Lugar, Data)
Primary key Id_reserva

Viagem (Id_viagem, Hora_chegada, Hora_partida, Preco)
Primary key Id_viagem
Derived preco ((if (Data_de_Nascimento(year) < 25) then 0.75*Preço)

Estação (Id_estacao, Cidade, Nome)
Primary key Id_estacao

Comboio (Id_comboio, Nr_lugares)
Primary key Id_comboio

ii. Entidades fracas

São entidades que não possuem existência própria (a sua existência depende da existência de outra entidade) ou que para serem identificadas dependem da identificação de outra entidade. No nosso modelo conceptual não existem entidades fracas.

iii. Relacionamentos 1:*

No modelo concebido, todas as relações são do tipo 1:* . De acordo com a metodologia, para a representação destes relacionamentos copia-se a chave primária da "entidade-pai" (correspondente a 1) para a relação que representa a "entidade-filho" (correspondente a *), definindo-a como chave estrangeira. Assim obtemos:

"Entidade-pai"

Cliente (CC, Nome, Data_de_Nascimento, Email,Telefone)

Primary key CC

"Entidade-filho"

Reserva (Id_reserva, Preço, Lugar, Data, CC)

Primary key Id_reserva

Foreign key cc **references** Cliente(cc)

"Entidade-pai"

Viagem (Id_viagem, Hora_chegada, Hora_partida, Preco)

Primary key Id_viagem

"Entidade-filho"

Reserva (Id_reserva, Preço, Lugar, Data, Id_viagem)

Primary key Id_reserva

Foreign key Id_viagem **references** Viagem(Id_viagem)

“Entidade-pai”

Comboio (Id_comboio, Lugar, Nr_lugares)

Primary key Id_viagem

“Entidade-filho”

Viagem (Id_viagem, Hora_chegada, Hora_partida, Preco, Id_comboio)

Primary key Id_viagem

Foreign key Id_comboio **references** Comboio(Id_comboio)

“Entidade-pai”

Estação (Id_estacao, Cidade, Nome)

Primary key Id_estacao

“Entidade-filho”

Viagem (Id_viagem, Hora_chegada, Hora_partida, Preco, Id_estacao)

Primary key Id_viagem

Foreign key Id_estacao **references** Estação(Id_estacao)

(2x)

iv. **Relacionamentos 1:1**

No modelo concebido não existem relacionamentos 1:1.

v. Relacionamentos recursivos 1:1

No modelo concebido não existem relacionamentos recursivos 1:1.

vi. Relacionamentos superclasse/subclasse

No modelo concebido não existem relacionamentos superclasse/subclasse.

vii. Relacionamentos *:*

No modelo concebido não existem relacionamentos *:*.

viii. Relacionamentos complexos

No modelo concebido não existem relacionamentos complexos.

ix. Atributos multi-valorados

Para cada atributo multi-valorado pertencente a uma entidade deve-se criar uma relação para representar esse atributo e incluir a chave primária da entidade na nova relação, definindo-a como chave estrangeira. Assim obtemos:

Comboio (Id_comboio, Nr_lugares)

Primary key Id_comboio

Lugares (Lugar, Id_comboio)

Primary key Lugar

Foreign key Id_comboio **references** Comboio(Id_comboio)

Obtemos assim o seguinte quadro:

Cliente (CC, Nome, Data_de_Nascimento, email, telefone) **Primary key** CC
Estação (Id_estaçao, Cidade, Nome) **Primary key** Id_estacao
Comboio (Id_comboio, Nr_lugares) **Primary key** Id_comboio
Reserva (Id_reserva, Lugar, Data, CC, Id_viagem) **Primary key** Id_reserva **Foreign key** CC **references** Cliente(CC) **Foreign key** Id_viagem **references** Viagem(Id_viagem) **Derived** Preco (if (Data_de_Nascimento(year) < 25) then 0.75*Preço)
Viagem (Id_viagem, Hora_chegada, Hora_partida, Preco, Id_estacao) **Primary key** Id_viagem **Foreign key** Id_estação_origem **references** Estação(Id_estação) **Foreign key** Id_estação_destino **references** Estação(Id_estação) **Foreign key** Id_comboio **references** Comboio(Id_comboio)
Lugares (Lugar, Id_comboio) **Primary key** Lugar **Foreign key** Id_comboio **references** Comboio(Id_comboio)

Tabela 4 – Relacionamentos que representam o modelo lógico

2. Normalização

Criando os relacionamentos obtemos um esquema lógico desnormalizado, que terá agora de ser submetido a uma série de regras – normalização - até atingir o seu estado normalizado, conseguido após a 3ª Forma Normal.

i) 1ª Forma Normal

Em primeiro lugar temos de passar este esquema desnormalizado para a 1ª Forma Normal, o que acontece retirando-se os grupos de atributos que se repetem dentro de cada relacionamento.

Poderemos averiguar que o nosso esquema lógico desnormalizado se encontra já na 1ª Forma Normal, dado não haver qualquer atributo "comprometedor", isto é, um atributo que não é do tipo DATA, *boolean* ou valor numérico, visto que estes não necessitam de ser normalizados devido ao seu tamanho, repetido entre qualquer relacionamento.

ii) 2ª Forma Normal

Relacionamento	Dependência	Tipo
Cliente	CC → (Data_de_Nascimento, Nome, Telefone, Email)	Total
Reserva	Id_reserva → (Lugar, Data, CC, Id_viagem)	Total
Viagem	Id_viagem → (Hora_partida, Hora_chegada, Preço, Id_estação_origem, Id_estação_destino, Id_comboio)	Total
Estação	Id_estação → (Nome, Cidade)	Total
Comboio	Id_comboio → Nr_lugares	Total
Lugares	----------	----------

Tabela 5 – Segunda Fórmula Normal.

O esquema lógico diz-se na 2ª Forma Normal se todos os atributos *Non-Primary-Key* forem completamente dependentes da *Primary Key*, o que, analisando a tabela acima representada, podemos facilmente averiguar que acontece.

iii) 3ª Forma Normal

A 3ª Forma Normal visa eliminar dependências transitivas que possam ocorrer, e, mais uma vez, analisando a tabela acima apresentada, vemos que não há qualquer dependência transitiva em qualquer relacionamento. Podemos assim concluir que o esquema lógico se encontra na 3ª Forma Normal e, portanto, normalizado, visto que qualquer alteração visando as restantes formas normais seriam praticamente impercetíveis.

3. Validação do modelo com as transações

De seguida enumeraremos algumas das transações que na nossa opinião, caso efetuadas com sucesso, demostram a validade do modelo em questão:

Inserir Cliente:

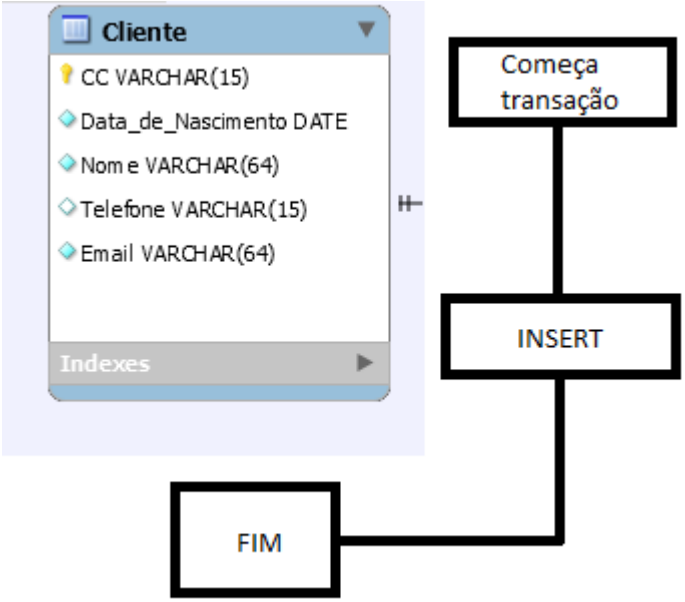

Figura 2 – Ilustração da transação gerada pela inserção de um cliente.

Inserir reserva:

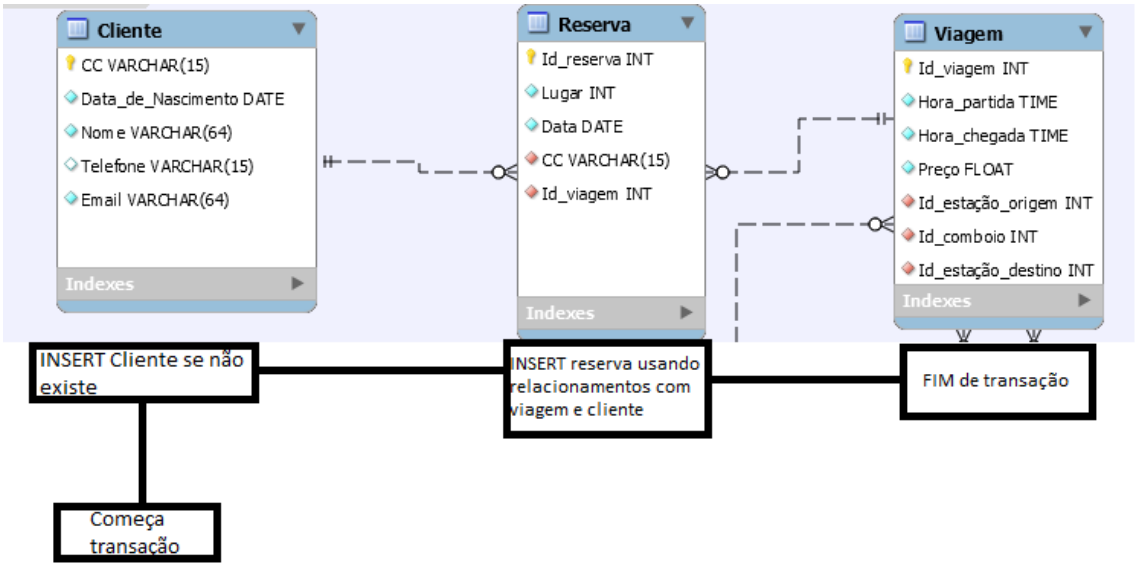

Figura 3 – Ilustração da transação gerada pela inserção de uma reserva.

Inserir comboio:

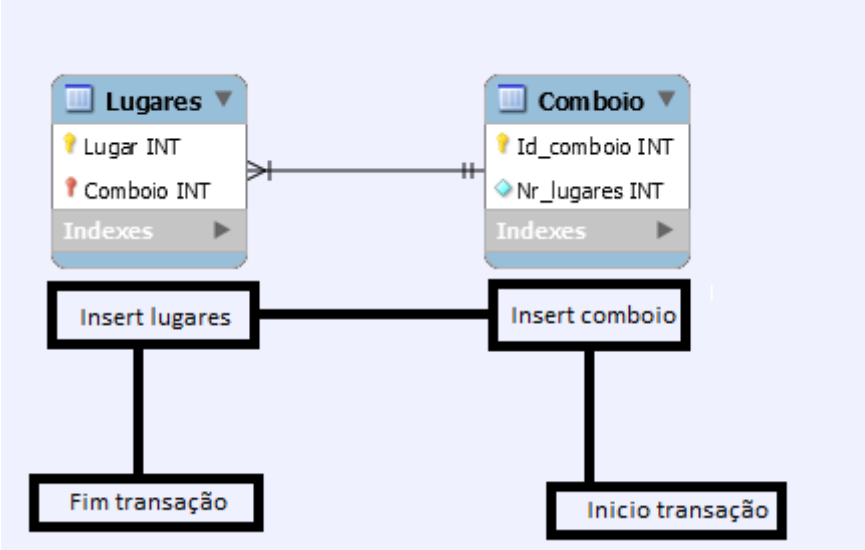

Figura 4 – Ilustração da transação gerada pela inserção de um comboio.

Consultar lugares disponíveis em viagem numa determinada data:

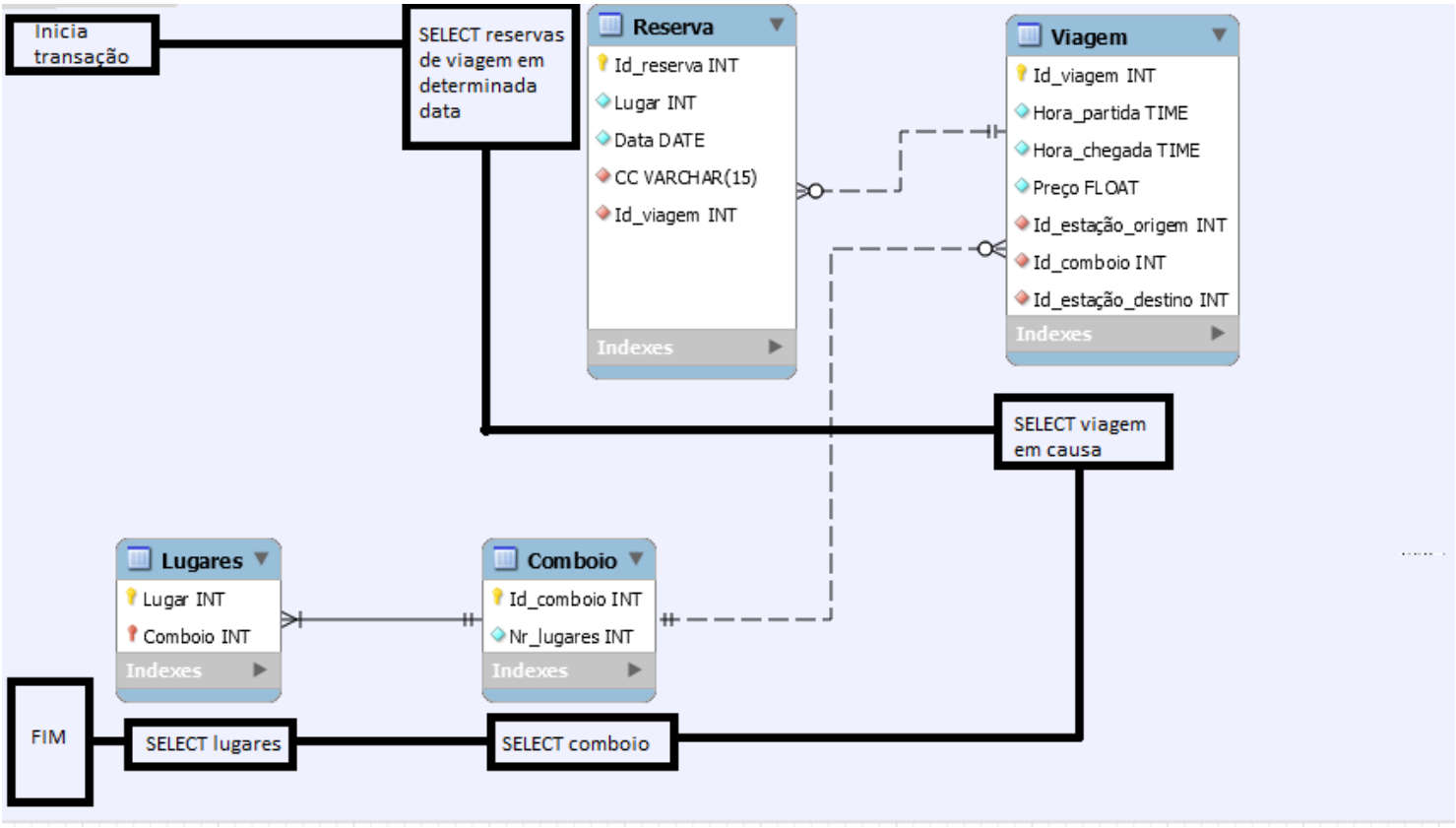

Figura 5 – Ilustração da transação gerada pela consulta dos lugares livres de uma viagem

Validação da transação com álgebra relacional:

TabelaViagemComboio ← ((Viagem (Viagem.Id_comboio = Comboio.id_comboio) ⋈ Comboio

TabelaLugaresDoComboioDaViagem ← σ (Id_viagem = inputIDViagem) (Tabela (Comboio.Id_comboio = Lugares.Comboio) ⋈ Lugares)

TabelaReservasViagemData ← σ (Id_viagem = inputIDViagem and inputdate = Reserva.Date) Reserva

π(lugar) TabelaLugaresDoComboioDaViagem – π(lugar) TabelaReservasViagemDia

Importante referir que na inserção de comboio é necessário inserir também os lugares correspondentes, o que é possível com este modelo como podemos ver na Figura 4.

4. Definição de restrições de integridade

i) Informação necessária

No nosso dicionário foi referido que apenas o telefone pode tomar o valor *NULL*. Assim, todos os atributos importantes têm sempre algum valor - por exemplo, uma viagem tem sempre horas de chegada e partida.

ii) Restrições do domínio dos atributos

Na identificação dos domínios dos atributos (Ponto 3 da construção do modelo conceptual) foi definido um domínio para cada um dos atributos.

iii) Multiplicidade

A multiplicidade representa as restrições referentes aos relacionamentos, como por exemplo uma Viagem ter N Reservas. No ponto 2 da construção do modelo conceptual foram definidos os relacionamentos entre as entidades, e tudo o que é necessário para garantir integridade foi definido neste ponto.

iv) Integridade de Entidades

As chaves primárias das entidades não podem tomar o valor *NULL*. No ponto 4 da construção do modelo conceptual foi definido que isto não nunca pode acontecer.

v) Integridade de Referências

Para garantir integridade de referências, é preciso que todas as chaves estrangeiras *NOT NULL* (nas entidades "filhas") tenham um valor que exista como chave primária numa entidade "pai". Temos o exemplo de N Reservas relativas a 1 Viagem. A chave estrangeira existente num *tuple* (linha da tabela) de Reserva tem que ter um valor que corresponda à chave primária (id_Viagem) de algum *tuple* existente de Viagem.

Em todos os relacionamentos referidos na construção do modelo conceptual, as entidades "filhas" têm que ter obrigatoriamente a chave estrangeira *NOT NULL*. Por exemplo, não faria sentido existir uma

Reserva que não se referisse a nenhuma Viagem. Noutro exemplo podemos considerar o facto de uma Viagem ter que ser efetuada obrigatoriamente por um Comboio.

Para assegurar a integridade das referências temos que definir restrições para inserções, atualizações e remoções.

Visto que só existem relacionamentos 1:N, todas as situações seguintes são referentes a esse tipo de relacionamentos.

- **Inserções**
  - Inserir um *tuple* de alguma entidade que seja entidade filha de alguma outra, exemplo:

    Inserir um *tuple* de Reserva. O valor em Id_Viagem tem que corresponder a um *tuple* de Viagem existente, ou seja, tem que ser chave primária do mesmo. Visto que não é permitido Id_Viagem tomar valor null, esta é a única situação possível.

  - Inserir um *tuple* de alguma entidade que seja apenas entidade pai (não há nenhum caso em que seja entidade filha), exemplo:

    Inserir um *tuple* de Cliente. Esta inserção não causa qualquer problema na integridade referencial, pois apenas se torna um Cliente que ainda não efetuou reservas. Este tipo de inserções nunca causa problemas, porque como é um novo “pai”, não existem referencias ao mesmo.

- **Remoções**
  - Apagar *tuple* de alguma entidade que apenas seja “filha”, exemplo:

    Remover um *tuple* de Reserva. Como Reserva não é entidade “pai” em nenhum relacionamento, esta situação não causa qualquer problema na integridade referencial. O que acontece é que algum *tuple* de Viagem fica com menos uma reserva e algum Cliente também perde uma Reserva. Depois desta remoção, nenhuma referencia se torna inválida.

- Apagar *tuple* de alguma entidade que seja “pai” em algum relacionamento, exemplo:

  Remover um *tuple* de Cliente. Como a entidade Cliente é a entidade “pai” no relacionamento **(Cliente)1:N(Reserva)**, a integridade referencial pode ser perdida. Isto porque, caso exista algum *tuple* da entidade “filha” deste relacionamento, num *tuple* da entidade “filha” há uma chave estrangeira referente a um *tuple* apagado. Neste caso é prevenida a remoção do elemento (NO ACTION), não é permitido remover um *tuple* que seja referenciado em algum outro *tuple*. Para o exemplo Comboio, quando é apagado um *tuple,* todos os lugares desse comboio são apagados(CASCADE). Este é o único exemplo que funciona desta forma, pois não faz sentido existir lugares sem comboio. Se o comboio efetuar alguma viagem, a remoção é prevenida na mesma.

- **Atualizações**
  - Atualizar a chave estrangeira de um *tuple* de uma entidade que é “filha” em algum dos relacionamentos. Acontece a mesma situação que no primeiro caso referido em **Inserções**.

  - Atualizar chave primária de um *tuple* de uma entidade que é “pai” em algum dos relacionamentos. Exemplo:

    Atualizar a chave primária de Cliente. Atualizar a chave primária de algum *tuple* de Cliente, pode implicar a perda da integridade referencial. Isto porque podem existir *tuples* de Reserva com chave estrangeira referente a este *tuple*. Para evitar esta perda vamos utilizar a estratégia CASCADE, ou seja, a atualização num *tuple* “pai” vai atualizar todas as referências a este.

Para garantir que acontece o que foi referido acima em todos os relacionamentos, temos que inserir, em todas as referências, o seguinte: ON DELETE NO ACTION ON UPDATE CASCADE. Visto que o descrito acima é válido para todos os relacionamentos.

vi) Restrições gerais

Será necessário garantir que:

- As chaves estrangeiras que se referem a estação destino e origem, em Viagem, nunca se refiram à mesma Estação, pois não faria sentido existir uma viagem com origem e destino iguais;
- A hora de chegada de uma viagem é superior à hora de partida;
- A viagem na qual se pretende reservar um lugar terá de ser no mínimo um dia depois do dia em que se efetua a reserva;

5. Crescimento futuro

Estimamos que ao longo dos próximos 2 anos, sejam efetuadas 10 reservas por dia, duas das quais são realizadas por novos clientes. Fazemos uma previsão que ao longo deste período também surjam dois novos trajetos entre as localidades onde a IberoTrem já opera e seja adquirido um novo comboio. Concluímos que as reservas são o principal fator de crescimento da base de dados.

De seguida mostra-se o diagrama do esquema lógico construído no *MySQL Workbench*, resultado de todo o processo enunciado nos três tópicos anteriores:

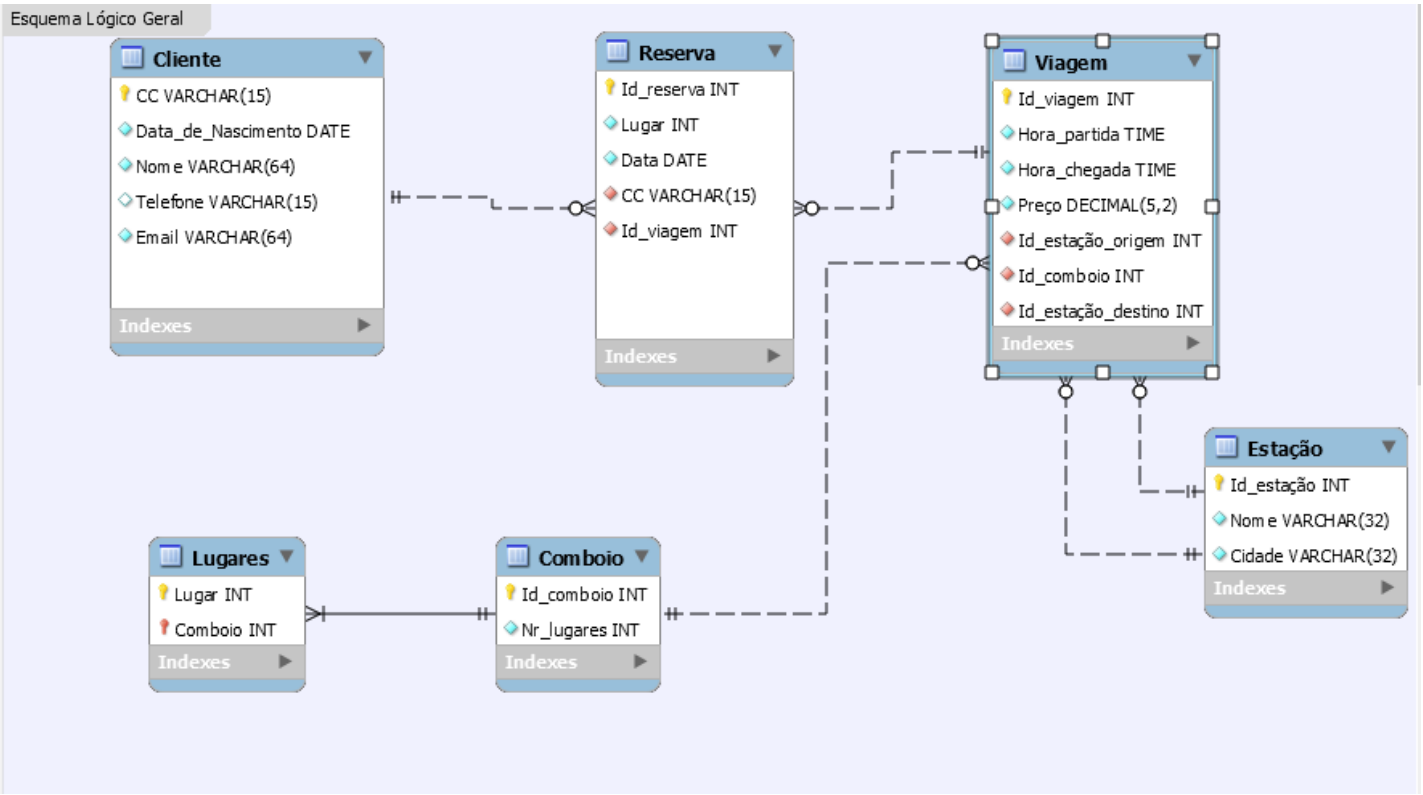

Figura 6 – Modelo lógico

3.3 Construção do modelo físico

A passagem do modelo lógico para o modelo físico (→ *Forward Engineer*…) traduz-se na conversão da descrição do que será implementado na base de dados para o modo como esta implementação será feita. Aqui é trabalhado o nível mais baixo de abstração, descrevendo-se a forma como os dados são armazenados fisicamente e os métodos de acesso aos dados.

1. Projetar relacionamentos base

1. Cliente

Domínio:

CC:	sequência de caracteres de tamanho variável, no máximo 15
Data_de_Nascimento:	data, formato 'yyyy-mm-dd', com o intervalo de '1000-01-01' até ´9999-12-31'
Nome:	sequência de caracteres de tamanho variável, no máximo 64
Telefone:	sequência de caracteres de tamanho variável, no máximo 15
Email:	sequência de caracteres de tamanho variável, no máximo 64

Cliente (

CC	NOT NULL,
Data_de_Nacimento	NOT NULL,
Nome	NOT NULL,
Telefone	
Email	NOT NULL,
PK(CC));	

2. Reserva

Domínio:

Id_reserva:	inteiro, com o intervalo: 1 – MAX(INT)
Lugar:	inteiro, com o intervalo: 1 - 10
Data:	data, formato 'yy-mm-dd', com intervalo: atualidade - '9999-12-31'
CC:	sequência de caracteres de tamanho variável, no máximo 15
Id_viagem	inteiro, com o intervalo: 1 – MAX(INT)

Reserva (

Id_reserva NOT NULL,

Lugar NOT NULL,

Data NOT NULL,

CC NOT NULL,

Id_reserva NOT NULL,

PK (Id_reserva),

FK(Id_viagem) REFERENCES Viagem(id_viagem) ON UPDATE CASCADE ON DELETE NO ACTION

FK(CC) REFERENCES Cliente(CC) ON UPDATE CASCADE ON DELETE NO ACTION);

3. Viagem

Domínio:

Id_viagem:	inteiro, com o intervalo: 1 – MAX(INT)
Hora_chegada:	TIME, formato: ‘hh:mm:ss’
Hora_partida:	TIME, formato: ‘hh:mm:ss’
Preço:	valor monetário, com o intervalo: 0.00-MAX(DECIMAL(5,2))
Id_comboio:	inteiro, com o intervalo: 1 – MAX(INT)
Id_ estação_origem:	inteiro, com o intervalo: 1 – MAX(INT)
Id_estação_destino:	inteiro, com o intervalo: 1 – MAX(INT)

Viagem (

Id_viagem NOT NULL,

Hora_chegada NOT NULL,

Hora_partida NOT NULL,

Preço NOT NULL,

Id_comboio NOT NULL,

Id_estação_origem NOT NULL,

Id_estação_destino NOT NULL,

PK (Id_viagem),

FK(Id_comboio) REFERENCES Comboio(id_comboio) ON UPDATE CASCADE ON DELETE NO ACTION

FK(Id_estação_origem) REFERENCES Estação(Id_estação) ON UPDATE CASCADE ON DELETE NO ACTION

FK(Id_estação_destino) REFERENCES Estação(Id_estação) ON UPDATE CASCADE ON DELETE NO ACTION

4. Estação

Domínio:

Id_estação:	inteiro, com o intervalo: 1 – MAX(INT)
Nome:	sequência de caracteres de tamanho variável, no máximo 32
Cidade:	sequência de caracteres de tamanho variável, no máximo 32

Estação (

Id_estação	NOT NULL,
Nome	NOT NULL,
Cidade	NOT NULL,
PK(Id_estação));	

5. Comboio

Domínio:

Id_comboio:	inteiro, com o intervalo: 1 – MAX(INT)
Lugares	inteiro, com o intervalo: 1 – 10
Nr_lugares:	inteiro, com valor 10

Comboio (

Id_comboio	NOT NULL,
Nr_lugares	NOT NULL,
PK(Id_comboio));	

6. Lugares

Domínio:

Lugar: inteiro, com o intervalo: 1 – 10

Nr_lugares: inteiro com o valor 10

Id_comboio: inteiro, com o intervalo: 1 – MAX(INT)

Comboio (

Lugar NOT NULL,

Id_comboio NOT NULL,

PK(Lugar),

FK(Id_comboio) REFERENCES Comboio(id_comboio) ON UPDATE CASCADE ON DELETE CASCADE

2. Projetar a representação dos dados derivados

O projeto apresenta apenas um atributo derivado: o **Preço** da reserva. Este atributo pertence à entidade Reserva e corresponde ao preço da viagem multiplicado pelo desconto atribuído ao cliente em causa, caso aplicado.

Não faz parte da tabela das reservas, sendo calculado e exibido numa vista (em baixo) sob a forma de uma tabela idêntica à das reservas, mas com a coluna correspondente ao preço, para posterior consulta.

De salientar que este atributo é derivado dado que pode ser calculado a partir de outros atributos cujos valores estão já armazenados na base de dados, não havendo, portanto, necessidade de os guardar. Porém, são valores que neste caso de estudo necessitam de ser consultados, daí ser criada a vista, que torna o seu cálculo mais eficiente (a alternativa seria calcular individualmente o preço de uma reserva sempre que este fosse necessário).

```
CREATE VIEW ReservaPreço
AS SELECT *, ROUND(calculaPreço(id_viagem, cc), 2) AS Price FROM
Reserva;
```

3. Projetar as restrições gerais

1) As chaves estrangeiras que se referem a estação destino e origem, em Viagem, nunca se refiram à mesma Estação, pois não faria sentido existir uma viagem com origem e destino iguais;

//Para updates:

```
ALTER TABLE Viagem
ADD CONSTRAINT check_viagem
      CHECK (id_estação_origem != Id_estação_destino);
```

//Para inserções

```
DELIMITER $$
CREATE TRIGGER check_estacao
  BEFORE INSERT ON Viagem
    FOR EACH ROW
    BEGIN
        IF (NEW.Id_estação_origem = NEW.Id_estação_destino)
        THEN
              SIGNAL SQLSTATE '45000'
              SET MESSAGE_TEXT = 'Estação de destino não
        pode ser igual à estação de partida';
        END IF;
  END;
$$
```

2) A hora de chegada de uma viagem é superior à hora de partida;

//Para updates

```
ALTER TABLE Viagem
ADD CONSTRAINT check_horarioUp
      CHECK (Hora_chegada > Hora_partida);
```

//Para inserções

```
DELIMITER $$
CREATE TRIGGER check_horario
     BEFORE INSERT ON Viagem
     FOR EACH ROW
     BEGIN
      IF (NEW.Hora_chegada < NEW.Hora_partida)
          THEN
               SIGNAL SQLSTATE '45000'
               SET   MESSAGE_TEXT   =   'Hora   de   chegada
      inválida';
          END IF;
     END;
$$
```

3) A viagem na qual se pretende reservar um lugar terá de ser no mínimo um dia depois do dia em que se efetua a reserva;

//Para updates

```
ALTER TABLE Reserva
ADD CONSTRAINT check_ReservaUp
      CHECK (Data - CURDATE() < 1);
```

//Para inserções

```
DELIMITER $$
CREATE TRIGGER check_reserva
      BEFORE INSERT ON Reserva
       FOR EACH ROW
      BEGIN
            IF (datediff(new.data,curdate()) < 1)
                  THEN
                        SIGNAL SQLSTATE '45000'
                         SET MESSAGE_TEXT = 'Reserva
                   não pode ser efetuada';
            END IF;
      END;
$$
```

4. Povoamento da Base de Dados

Aqui expõe-se, a título de exemplo, o povoamento de algumas tabelas da base de dados:

```
-- Povoamento da tabela "Cliente"
INSERT INTO Cliente
      (CC, Data_de_Nascimento, Nome, Telefone, Email)
            VALUES
            ('15325142',   '1992-01-05',   'Rolando   Escada   Abaixo',
            '924145314', 'rea2010@sapo.pt'),
            ('76452899', '1967-11-12', 'João Bragança', '914934870',
            'joao_67_bra@iol.pt'),
            ('83509478', '1974-09-23', 'Fernanda Castro', '929896439',
            'nanda123@gmail.com'),
            ('54365476',  '1955-08-12',  'Miguel  Castro',  '93780900',
            'miguel_c@hotmail.pt'),
            ('86104398', '1996-12-25', 'Alexandre Mendes', '918626154',
            'alex_fcf_96@hotmail.pt'),
            ('55787890A', '1960-11-13', 'Diego Murillo', '917483436',
            'diego_murillo@gmail.pt');


-- Povoamento da tabela "Comboio"
INSERT INTO Comboio
      (Id_comboio, Nr_lugares)
            VALUES
            (1, 10),
            (2, 10);


-- Povoamento da tabela "Lugares"
INSERT INTO Lugares
      (Lugar, Comboio)
             VALUES
            (1, 1),
            (2, 1),
            (3, 1),
            (4, 1),
            (5, 1),
            (6, 1),
            (7, 1),
            (8, 1),
            (9, 1),
            (10, 1),
            (1, 2),
            (2, 2),
            (3, 2),
            (4, 2),
            (5, 2),
            (6, 2),
            (7, 2),
            (8, 2),
            (9, 2),
            (10, 2);
```

```
-- Povoamento da tabela "Estação"
INSERT INTO Estação
      (Id_estação, Nome, Cidade)
            VALUES
            (1, 'Campanhã', 'Porto'),
            (2, 'Santiago de Compostela ', 'Santiago de Compostela'),
            (3, 'Vigo Guixar', 'Vigo'),
            (4, 'Chaves', 'Chaves'),
            (5, 'Braga', 'Braga'),
            (6, 'Vila Real', 'Vila Real'),
            (7, 'Guimarães', 'Guimarães'),
            (8, 'Viana do Castelo', 'Viana do Castelo'),
            (9, 'A Coruña', 'Corunha'),
            (10, 'Mirandela', 'Mirandela'),
            (11, 'Ourense', 'Ourense'),
            (12, 'Duas Igrejas', 'Miranda Douro');
```

5. Análise de transações

Aqui serão expostas as transações resultantes de inserções e consulta, implementadas em *MySQL*, já abordadas nas transações de validação do modelo lógico.

Transações:	**Registar cliente**				**Registar reserva**				**Registar comboio**				**Consulta de lugares livres em viagem (data)**			
	I	**R**	**U**	**D**	**I**	**R**	**U**	**D**	**I**	**R**	**U**	**D**	**I**	**R**	**U**	**D**
Viagem	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	-
Reserva	-	-	-	-	X	-	-	-	-	-	-	-	-	X	-	-
Comboio	-	-	-	-	-	-	-	-	X	-	-	-	-	X	-	-
Cliente	X	-	-	-	X	X	-	-	-	-	-	-	-	-	-	-
Estação	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lugares	-	-	-	-	-	-	-	-	X	-	-	-	-	X	-	-

I – Insert, R – Read, U – Update, D – Delete

Tabela 6 – Identificação das transações em tabela

i) Inserção de uma reserva

Para responder a esta transição, criou-se o seguinte procedimento:

```
DELIMITER $$
CREATE PROCEDURE addReserva (IN nome VARCHAR(32), IN cc VARCHAR(9), IN dob
DATE, IN tel VARCHAR(9), IN ee VARCHAR(32), IN viagem INT, IN dia DATE, IN lugar
INT)
BEGIN
      IF NOT EXISTS
            (SELECT CC FROM Cliente where cc = CC.Cliente)
      BEGIN
      START TRANSACTION;
      INSERT INTO Cliente
            (CC, Data_de_Nascimento, Nome, Telefone, Email)
            VALUES
                  (cc, dob, nome, tel, ee);
      END
      -- Declaraçãoo de um handler para tratamento de erros.
      DECLARE ErroTransacao BOOL DEFAULT 0;
      DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET ErroTransacao = 1;
      -- Início da transação
      START TRANSACTION;
      INSERT INTO Reserva
            (Id_reserva, Lugar, Data, CC, Id_viagem)
            VALUES
                  (SELECT COUNT(*) FROM Reserva + 1, lugar, dia, cc,
                  viagem);
      -- Verificação da ocorrência de um erro.
      IF ErroTransacao THEN
      -- Desfazer as operações realizadas.
      ROLLBACK;
      ELSE
      -- Confirmar as operações realizadas.
      COMMIT;
      END IF;
END
$$
```

ii) Inserção de um cliente

Já no que diz respeito à inserção de um cliente, este foi o procedimento criado:

```
DELIMITER $$
CREATE PROCEDURE addCliente (IN nome VARCHAR(32), IN cc VARCHAR(9), IN dob
DATE, IN tel VARCHAR(9), IN ee VARCHAR(32))
```

```
BEGIN
      INSERT INTO Cliente (CC, Data_de_Nascimento, Nome, Telefone, Email)
            VALUES (cc, dob, nome, tel, ee);
END $$
```

iii) Inserção de um comboio

Procedimento respeitante às inserções de um comboio:

```
DELIMITER $$
CREATE PROCEDURE addComboio (IN lugares INT)
BEGIN
      -- Declaração de um handler para tratamento de erros.
      DECLARE IdComb INT DEFAULT (SELECT COUNT(*) FROM Comboio) + 1;
      DECLARE i INT DEFAULT 1;
      DECLARE ErroTransacao BOOL DEFAULT 0;
      DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET ErroTransacao = 1;

      -- Início da transação
      START TRANSACTION;
      INSERT INTO Comboio
            (Id_comboio, Nr_lugares)
            VALUES
                  (IdComb, lugares);
      WHILE (i <= lugares) DO
            INSERT INTO Lugares
                  (Lugar, Comboio)
                  VALUES
                        (i, IdComb);
      SET i = i + 1;
      END WHILE;
      -- Verificação da ocorrência de um erro.
      IF ErroTransacao THEN
      -- Desfazer as operações realizadas.
            ROLLBACK;
      ELSE
      -- Confirmar as operações realizadas.
            COMMIT;
      END IF;
END$$
```

iv) Consulta dos lugares livres

Esta foi o procedimento criado para consultar os lugares livres de um comboio numa dada viagem:

```
DELIMITER $$
CREATE PROCEDURE LugaresLivres (IN viagem INT, IN data Date)
```

```
BEGIN
      SELECT DISTINCT Lugares.Lugar FROM Lugares
      LEFT JOIN
            (SELECT Reserva.Lugar from Reserva
            inner join Viagem on Viagem.Id_viagem = Reserva.Id_viagem
                  where Viagem.Id_viagem = viagem and
                        Reserva.Data = data) AS T
      on Lugares.Lugar = T.Lugar where T.Lugar is NULL
      ORDER BY Lugares.Lugar asc;
END$$
```

6. Escolha de índices

Considerou-se que os índices para as chaves primárias e estrangeiras são suficientes para garantir alguma eficiência nas *queries*.

7. Estimativa dos requisitos de espaço em disco

7.1 Tamanho inicial

Cliente

Para cada linha da tabela Cliente:

- CC: VARCHAR(15) → 15 bytes
- Nome: VARCHAR(64) → 64 bytes
- Telefone: VARCHAR(15) → 15 bytes
- Email: VARCHAR(64) → 64 bytes
- Data_de_Nascimento: DATE →8 bytes

Assim, como inicialmente existem 6 clientes, a tabela Cliente ocupa (166 bytes * 6) **996 bytes.**

Reserva

Para cada linha da tabela reserva:

- Id_reserva: INT → 4 bytes
- Data: DATE → 8 bytes
- Lugar: INT → 4 bytes
- CC: VARCHAR(15) → 15 bytes
- Id_viagem: INT → 4 bytes

Assim, como inicialmente existem 5 reservas, a tabela Reserva ocupa (5 * 35) **175 bytes.**

Viagem

Para cada linha da tabela viagem:

- Id_Viagem: INT → 4 bytes
- Hora_partida: TIME → 8 bytes
- Hora_chegada: TIME → 8 bytes
- Preco: DECIMAL(5,2) → 5 bytes
- Id_comboio: INT → 4 bytes
- Id_estacao_origem: INT → 4 bytes
- Id_estacao_destino: INT → 4 bytes

Assim, como inicialmente existem 10 viagens, a tabela Viagem ocupa(37 bytes * 10) **370 bytes.**

Estacao

Para cada linha da tabela estacao:

- Id_estacao: INT → 4 bytes
- Nome: VARCHAR(32) → 32 bytes
- Cidade: VARCHAR(32) → 32 bytes

Assim, como inicialmente existem 12 estações, a tabela Estacao ocupará (68 bytes * 12) **816 bytes.**

Comboio

Para cada linha da tabela comboio:

- Id_comboio: INT → 4 bytes
- Nr_lugares: INT → 4 bytes

Assim, como inicialmente existem 2 comboios, a tabela Comboio ocupa (8 bytes * 2) **16 bytes**.

Lugares

Para cada linha da tabela Lugares:

- Lugar: INT → 4 bytes
- Comboio: INT → 4 bytes

Assim, como inicialmente existem 2 comboios com 10 lugares cada, a tabela lugares ocupa (20 * 8) **160 bytes.**

A base de dados necessita de **2.533 kbytes.**

7.2 Crescimento futuro

Como referido na secção 5 da construção do modelo lógico, esperam-se 10 reservas por dia nos próximos 2 anos, ou seja, mais 365 000 reservas. Dessas reservas, duas são feitas por clientes novos, ou seja, mais 730 clientes. Foi também referido que existiria mais 1 comboio e, por consequência, mais 10 lugares. Esperam-se também que sejam introduzidas duas novas viagens até ao final dos dois anos referidos. Ao fim destes prevê-se então um aumento em: 3650 * 2 * 35 (Reservas) + 730 * 166 (Clientes) + 8 (Comboio) + 10 (Lugares) + 2 * 37 (Viagens) = **381 752 bytes,** aprox. **0.381678 MB.**

4. Conclusões e Trabalho Futuro

De uma forma geral, fazemos uma avaliação positiva do trabalho realizado. Consideramos que o trabalho se encontra bem estruturado, o que em grande parte se deve ao seguimento da metodologia apresentada no livro recomendado.

As principais dificuldades surgiram na fase inicial da realização do projeto. Na formulação dos requisitos e na concepção do modelo conceptual, a maior dificuldade passou por definir as entidades e respetivos relacionamentos capazes de suportar de uma forma viável o sistema de reservas. Contudo, apesar da reduzida complexidade, achamos que conseguimos chegar a um modelo capaz de cobrir os pontos base de qualquer sistema de reservas associado a um serviço de transporte ferroviário. No que diz respeito às passagens do modelo conceptual para o lógico e do modelo lógico para o físico, não surgiram grandes dificuldades devido não só à metodologia mas também aos conhecimentos que já tínhamos adquirido nas aulas práticas e teóricas da Unidade Curricular.

Grande parte das diretrizes propostas pela metodologia foram cumpridas, à excepção de alguns pontos relacionados sobretudo com o modelo físico. Questões relacionadas com mecanismos de segurança, por exemplo, poderiam no futuro ser analisadas e implementadas. Para além disto, um aumento da complexidade no que diz respeito às funcionalidades que a base de dados consegue suportar poderia ser igualmente um dos pontos a ter em atenção no futuro.

Concluindo, a realização do trabalho foi essencial para a consolidação dos conhecimentos adquiridos ao longo da Unidade Curricular.

Referências

Connolly, T. and Begg, C., 1995. *Database Systems - A Practical Approach to Design, Implementation, and Management.*