ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Trabalho Prático Sistemas de Informação II

(Fase I)

Joana Campos
Nuno Cardeal
Carolina Couto

Docentes Afonso Remédios
Nuno Datia

Relatório de progresso realizado no âmbito de Sistemas de Informação II,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Inverno 2019/2020

Novembro de 2019

Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Trabalho Prático Fase I

Grupo 14

44792 Joana Filipa Alves de Campos

44863 Nuno Rafael Mourão Cardeal

44871 Carolina Alexandra Roque Couto

Docentes Afonso dos Santos Remédios

Nuno Miguel Soares Datia

Relatório de progresso realizado no âmbito de Sistemas de Informação II,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Inverno 2019/2020

Novembro de 2019

Resumo

Este trabalho foca-se na criação de um sistema de gestão de base de dados para uma empresa, *Pilim*, de gestão de mercados financeiros. Esta fase do trabalho foca-se na criação do modelo, quer físico quer concetual do sistema de informação e do desenvolvimento do *software* para a criação em base de dados do mesmo em linguagem *Transact Structured Query Language*, o mesmo *software* deve contar com algumas funcionalidades que permitam um correto e estável funcionamento do sistema por forma que numa segunda fase deste trabalho seja possível o desenvolvimento de uma aplicação para o preenchimento e a manipulação da base de dados. Sendo este um trabalho educacional não é pretendido que o mesmo fique a funcionar por completo, mas sim que os seus desenvolvedores aprendam e expliquem como este trabalho pode realmente ser bem executado, dando azo aos mesmos de poderem tirar e discutir as suas próprias interpretações sobre os problemas que vão surgindo durante o trabalho.
Neste trabalho foram realizados procedimentos para Inserir, remover e atualizar a informação de um cliente e de um mercado. Estes procedimentos continham transações para realizar os comandos SQL *insert*, *update* e *delete.*
Foi realizado um procedimento para atualizar o valor diário dos Instrumentos, com base nos triplos recebidos. Este procedimento recebe como parâmetros a data e o identificador do instrumento (ISIN) presente no triplo recebido e verifica se na tabela de registos diários já se encontra registado o dia para o ISIN recebido, se for esse o caso irá atualizar os valores de acordo com o valor recebido no triplo, caso ainda n tenha sido inserido o dia na tabela, é realizada a inserção com os valores do triplo bem como a data e o ISIN.
Para calcular a média a seis meses de um instrumento foi realizada uma função que retorna a média do valor de fecho de um determinado Instrumento dos últimos seis meses.
Foi realizada uma função para atualizar os valores fundamentais de um Instrumento. Para tal, a função irá retornar uma tabela com os valores fundamentais (variação diária, valor atual, média a seis meses, variação a seis meses, variação diária em percentagem e variação a seis meses em percentagem) de um dia específico passado como parâmetro, do instrumento cujo ISIN seja o passado como parâmetro.
Para criar um novo portefólio foi realizado um procedimento que, dado um nif, insere na tabela *portfolio* o nome do cliente cujo nif seja o dado no procedimento, caso o mesmo ainda não exista nesta tabela, mas exista na tabela de clientes.
Para atualizar o valor total de um portfolio foi realizado um procedimento que insere na tabela *Position* o nome do portefólio, a quantidade e o ISIN do instrumento em questão passados como parâmetros e atualiza a tabela *portfolio* com o valor total do mesmo que consiste no somatório do produto do valor atual dos instrumentos e da quantidade de cada um.
Foi realizada uma função para produzir a listagem de um portefólio que retorna uma tabela com o ISIN, a quantidade, o valor atual, e a percentagem de variação em relação ao dia anterior do portefólio cujo nome foi passado como parâmetro
Foi realizada uma vista, que consiste numa tabela temporária, para mostrar um resumo do portefólio de cada cliente. A vista é constituída pelo nome do portefólio, somatório do número de instrumentos e pelo somatório do valor total.
Foi ainda realizado um procedimento que atualiza o mercado diário com base nos triplos recebidos, este procedimento tem uma estrutura semelhante ao procedimento que atualiza o registo diário acima descrito, com a diferença que este procedimento atualiza um dado mercado enquanto que o anterior atualiza apenas um instrumento.
Estes dois procedimentos, bem como a função que atualiza os valores fundamentais de um instrumento, encontram-se dentro de um *trigger* que é acionado de cada vez que é inserido um novo triplo na tabela *Exttriple*.
Por fim foram realizados testes para cada procedimento e função realizados ao longo deste trabalho com a ajuda de um *Batch file* que irá criar todas as tabelas da base dados, todas as funções e todos os procedimentos. Este ficheiro irá também correr o ficheiro SQL que contem todos os *insert* iniciais desta base de dados.
Palavras-chave: funções; listagens; procedimentos; transações; *triggers*; vistas;

Abstract

This work focuses on creating of a database management system for a financial markets management company *PILIM*.
This phase of the work focuses on the creation of the model, both physical and conceptual of the information system and software development for the creation of the database in language *Transact Structured Query Language*,The same *software* must have some features that allow the correct and stable operation of the system so that in a second phase of this work it is possible to develop an application to fill and manipulate the database. As this is an educational work, it is not intended to be fully functional, but rather that its developers learn and explain how this work can really be done well, giving them the opportunity to draw and discuss their own interpretations of the problems that come up during work.
In this work procedures were created to Insert, remove and update the information of a client and a market. These procedures contained transactions to execute the commands *SQL insert, update and delete*.
A procedure was created to update the daily value of the Instruments, based on the triples received. This procedure takes as parameters the date and the instrument identifier (ISIN) present in the triple received and checks if the day record table is already registered for the received ISIN, if so it will update the values according to the value received in triple, if the day has not yet been entered in the table, is inserted with the triple values as well as the date and the ISIN.
To calculate the average of six months of an instrument was created a function that returns the average of the closing value of a given instrument of the last six months.
A function has been created to update the fundamental values of an Instrument. To do this, the function will return a table with the fundamental values (daily change, current value, six month average, six month change, daily percentage change and six month percentage change) for a specific day passed as a parameter, instrument whose ISIN is the past as a parameter.
To create a new portfolio a procedure was created which, given a NIF, inserts in the portfolio table the name of the client whose NIF is given in the procedure, if it does not already exist in this table, but exists in the *client* table.
In order to update the total value of a portfolio a procedure was created which inserts in the *Position* table the portfolio name, quantity and ISIN of the instrument in question passed as parameters and updates the portfolio table with the total value of the portfolio consisting of the sum of the portfolio. product of the present value of the instruments and the quantity of each.
A function was created to produce the listing of a portfolio that returns a table with the ISIN, the quantity, the current value, and the percentage change from the previous day of the portfolio whose name was passed as a parameter.
A view, consisting of a temporary table, was created to show a summary of each client's portfolio. The view consists of the name of the portfolio, the sum of the number of instruments and the sum of the total value.
A procedure that updates the daily market based on the triples received was also created, this procedure has a structure similar to the procedure that updates the daily record described above, except that this procedure updates a given market while the previous one updates only one instrument.
These two procedures, as well as the function that updates the fundamental values of an instrument, are within a *trigger* that is triggered each time a new triple is inserted into the *Exttriple* table.
Finally, tests were performed for each procedure and function created throughout this work with the help of a Batch file that will create all database tables, all functions and all procedures. This file will also run the SQL file that contains all the initial inserts of this database.

Keywords: functions; listings; procedures; transactions; triggers; views;

Índice

RESUMO ..... **IV**

ABSTRACT ..... **VII**

LISTA DE FIGURAS ..... **XIV**

LISTA DE TABELAS ..... **XVI**

1. INTRODUÇÃO **1**

1.1 Enunciado 1

1.2 Espectativas 2

1.3 Organização do documento 2

2. FORMULAÇÃO DO PROBLEMA **3**

2.1 Modelos de Dados 3

2.2 Bases de Dados Dinâmicas 3

2.3 Transações 4

2.4 Níveis de Isolamento 4

2.5 Vistas 4

2.6 Gatilhos 4

2.7 Funções 4

2.8 Importância dos Testes 4

3. SOLUÇÃO PROPOSTA **5**

3.1 Modelo EA 5

3.2 Modelo Relacional 7

3.3 Criação do modelo físico 7

3.4 Remoção da base de dados 7

3.5 Preenchimento Inicial da Base de Dados 7

3.6 Inserção, Atualização e Remoção de um Cliente 7

3.6.1 Inserção e Atualização de um Cliente *7*

3.6.2 Remoção de um Cliente *7*

3.7 Inserção, Atualização e Remoção de um Mercado 8

3.7.1 Inserção e Atualização de um Mercado *8*

3.7.2 Remoção de um Mercado *8*

3.8 Atualização dos Valores Diários de cada Instrumento 8

3.9 Cálculo da Média a 6 meses de um Instrumento 9

3.10 Atualização dos Dados Fundamentais de um Instrumento 9

3.11 Criação de um portfólio 9

3.12 Atualização do valor total de um portfólio 10
3.13 Atualização do valor total de um portfólio 10
3.13 Criar uma vista que mostre o resumo dos portfólios 10
3.14 Atualização do Daily Market 11
3.15 Criação do Trigger 11
3.16 Funções Auxiliares 12
3.16.1 Busca do Valor Atual 12
3.16.2 Busca da Variação diária em percentagem 12
4. AVALIAÇÃO EXPERIMENTAL 13
4.1 Criação do *batch* 13
4.2 Inserção, remoção e atualização da informação de um cliente 13
4.2.1 Atualização ou Inserção de um cliente 13
4.2.2 Remoção de um cliente 14
4.3 Inserção, remoção e atualização da informação de um mercado 14
4.3.1 Atualização ou Inserção de um mercado 14
4.3.2 Remoção de um mercado 14
4.4 Atualização dos valores diários de cada instrumento 14
4.5 Cálculo da média a 6 meses de um instrumento 14
4.6 Atualização dos dados fundamentais de um instrumento 15
4.7 Criação de um portfólio 15
4.8 Atualização do valor total de um portfólio 15
4.9 Listagem de um portfólio 15
4.10 Criar uma vista que mostre o resumo dos portfólios 15
4.11 Teste do *trigger* 15
4.12 Análise de resultados 15
5. CONCLUSÕES 16
REFERÊNCIAS 17
A.1 MODELO EA 18
A.2 MODELO RELACIONAL 19
A.3 CÓDIGO *SQL* 23

Lista de Figuras

Não foi encontrada nenhuma entrada do índice de ilustrações.

Lista de Tabelas

Não foi encontrada nenhuma entrada do índice de ilustrações.

1. Introdução

1.1 Enunciado

A empresa *Pilim* pretende desenvolver um sistema de informação para a gestão de mercados financeiros. Um mercado financeiro é caracterizado por um código (único), uma descrição e um nome curto. Para cada dia e para cada mercado, são registados o valor do índice/mercado (soma de todos os valores de abertura dos seus instrumentos, calculado no fim do dia), o valor de abertura (valor de índice do dia útil anterior) e a variação diária (em euros). Um mercado é constituído por um conjunto de instrumentos financeiros (e.g. ações de empresas), doravante designado simplesmente por instrumento. Um instrumento é caracterizado por um código único, designado de *International Securities Identification Numbers* (ISIN)[1] e uma descrição. A empresa recebe triplos de um sistema externo, constituídos por

< *identificador; datatempo; valor* > onde

identificador identifica um instrumento que pode ou não estar registado no sistema,

datatempo representa um instante temporal com granularidade ao segundo e

valor indica o valor do instrumento em euros.

Todos os triplos têm de ficar registados no sistema de informação da empresa de forma persistente. Para cada instrumento é mantido o registo diário do valor mínimo, valor máximo, valor de abertura e valor de fecho. Existem associados a cada instrumento dados fundamenteis, constituídos pelo valor de variação diária (diferença entre os valores máximo e mínimo), valor atual, média a 6 meses do valor de fecho de cada dia, valor variação a 6 meses, percentagem de variação diária e percentagem de variação a 6 meses. O sistema de informação mantém um registo de clientes, sendo cada um caracterizado por um número de identificação fiscal (único), um número de cartão de cidadão (único) e um nome.

Cada cliente tem um conjunto de contactos (pelo menos 1). Um contacto é caracterizado por um código (único) e por uma descrição. Para os contactos telefónicos é necessário manter o número de telefone e o indicativo. Para os e-mails apenas é necessário manter o endereço. Os clientes têm um portfólio de instrumentos, designados de posições, onde é registada a quantidade de cada instrumento (e.g. 50 ações). Um portfólio tem um nome (único para cada cliente) e um valor total, que resulta do somatório do produto do valor atual dos instrumentos e da quantidade de cada um. Deve ser possível saber o valor total de cada posição. O sistema deve garantir que sempre que o valor diário de um instrumento é alterado, é mantida coerência com o valor diário registado para o mercado.

[1] Para mais detalhes consultar https://www.isin.org/

1.2 Espectativas

É esperado que no final do trabalho consigamos realizar os seguintes pontos:

- Desenvolver um modelo de dados adequado aos requisitos, normalizado até à 3NF;
- Conceber e implementar uma solução baseada em bases de dados dinâmicas, adequadas aos requisitos;
- Utilizar corretamente controlo transacional;
- Utilizar corretamente níveis de isolamento;
- Utilizar corretamente vistas, justificando o seu uso na solução;
- Utilizar corretamente procedimentos armazenados, justificando o seu uso na solução;
- Utilizar corretamente gatilhos, justificando o seu uso na solução;
- Utilizar corretamente funções, justificando o seu uso na solução;
- Desenvolver código de teste, em T-SQL, para cada uma das funcionalidades pretendidas;
- Escrever um relatório técnico sobre o trabalho desenvolvido.

1.3 Organização do documento

O restante relatório encontra-se organizado da seguinte forma: Na secção 2 existe a formulação do problema onde é apresentado tudo o que necessitamos de usar para cumprir as expectativas deste trabalho e ainda uma justificação do porquê de ser útil usar tais ferramentas na implementação desta fase do trabalho. A secção 3 é a explicação de como usamos cada ferramenta e do porquê de a termos usado. Na secção 4 indicamos uma avaliação experimental onde corremos vários testes e avaliamos se estão a ter ou não o resultado pretendido, e em caso negativo perceber e explicar como poderia estar a suceder bem. Por fim na 5ª secção retiraremos as conclusões sobre este trabalho. Para uma melhor compreensão deste projeto, está apresentado no anexo 3, o código desenvolvido neste trabalho.

2. Formulação do Problema

Com este trabalho é pretendido que os alunos aprendam a desenvolver modelos de dados adequados a diferentes problemas e normalizados corretamente na 3NF, aprender como funcionam bases de dados dinâmicas e conceber uma de acordo com os modelos de dados realizados, utilizar o controlo transacional, os níveis de isolamento, vistas, gatilhos e funções em *T-SQL* e criar na mesma linguagem testes para o código desenvolvido. Na secção 2.1 fala acerca dos modelos de dados e da sua importância para o desenrolar deste trabalho. A secção 2.2 apresenta como são elaboradas bases de dados dinâmicas. Na secção 2.3 aborda-se a necessidade de transações. A secção 2.4 trata dos aspetos referentes aos níveis de isolamento. A secção 2.5 refere a boa utilização de vistas. Nas secções 2.6 e 2.7 abordam-se os conteúdos referentes a gatilhos e funções respetivamente. Por fim a secção 2.8 sublinha a importância da utilização de testes para o desenvolvimento do trabalho.

2.1 Modelos de Dados

Existem 2 tipos de modelos de dados, o concetual e o físico. Tal como o próprio nome indica o modelo concetual serve para uma ajuda na conceção do sistema de gestão de base de dados, dando uma ideia visual de como a base de dados está conectada, sendo usado para este trabalho o Modelo Entidade-Associação que dá uma boa representação de quais são os diversos pontos no sistema de gestão de base de dados que têm de ter uma maior atenção na altura do seu desenvolvimento. O modelo físico é uma representação mais real de como é o sistema de gestão de base de dados, sendo que para o seu entendimento tem de haver uma maior atenção do leitor visto que é um modelo mais descritivo e que tem um maior aprofundamento de relações. O modelo físico escolhido foi o Modelo Relacional que tem uma boa ambiguidade com o Modelo Entidade-Associação e as características do modelo físico.

2.2 Bases de Dados Dinâmicas

Para este trabalho é necessário usar bases de dados dinâmicas, visto serem as bases de dados mais usadas em sistemas de gestão, uma base de dados dinâmica é uma base de dados onde se pode inserir e remover valores de uma tabela por forma que a própria base de dados mantenha a coerência em outras tabelas, para isso é necessária a utilização de gatilhos e transações visto que estes servem para dar estabilidade à base de dados e vigiar quando existe inserção o remoção de dados.

2.3 Transações

As transações servem para manter a estabilidade de uma base de dados de tal maneira que se acontecer algum erro durante a execução do código a base de dados retorna ao seu último estado mais estável.

2.4 Níveis de Isolamento

Existem diversos tipos de isolamento, estes servem para conter as transações de modo que elas trabalhem só e apenas com o que é pretendido, se uma transação tentar fazer uma ação fora do seu nível de isolamento a mesma não será executada.

2.5 Vistas

Vistas são tabelas "fictícias" nas quais não se pode inserir, remover nem atualizar dados, visto que estas tabelas na verdade não existem, mas retiram informação de tabelas existentes, são bastante uteis na comparação de dados de diferentes tabelas ou na criação de listas

2.6 Gatilhos

Os gatilhos são a maneira que um programador tem de saber se a base de dados foi alterada, eles, quando criados, ficam sempre a vigiar alguma alteração à entidade que ficou de vigiar, como tabelas, vistas ou até mesmo bases de dados inteiras, se a alteração escolhida pelo programador tiver acontecido então executa o código que está dentro do gatilho.
Geralmente estes servem para manter a coerência na base de dados quando existe uma atualização, remoção ou inserção de dados.

2.7 Funções

Funções são troços de código que servem maioritariamente para manipular e processar dados, uma função pode receber parâmetros e tem sempre de retornar um resultado, no final da função todas as modificações feitas são realizadas na base de dados.

2.8 Importância dos Testes

Os testes são importantes para ter a certeza que quando acabado sistema de gestão de base de dados tem todas as funcionalidade esperadas, de modo que ela possa cumprir a sua função corretamente quando for necessária, assim, se houver alguma inconsistência na base de dados é nesta parte que ela é descoberta e corrigida, por forma a ficar uma base de dados estável, coerente e dinâmica.

3. Solução Proposta

Para a realização deste trabalho começamos por desenvolver os modelos Entidade-Associação e Relacional e colocar os mesmos na 3NF por forma que fosse possível ter uma base de dados bem planeada e de fácil manipulação. De seguida começamos o desenvolvimento da base de dados, em *T-SQL*, realizando *scripts* que criassem tabelas e as apagassem, preenchessem-nas e também removessem todo o seu conteúdo. Criou-se transações para quando se inserisse removesse ou se atualizasse dados de um cliente ou de um mercado, de forma a não perder dados caso estas ações não fossem corretamente realizadas, criou-se diversas funções e procedimentos que nos auxiliassem na manipulação da base de dados e na sua procura. Por fim implementou-se também testes para o auxílio de uma melhor compreensão se tudo funcionava como suposto.

3.1 Modelo EA

Começou-se por realizar o Modelo Entidade-Associação pois este permite que se tire uma primeira interpretação do enunciado, fazer alterações, explorar mais o enunciado e tirar interpretações diferentes e mais aprofundadas, de forma a que quando se tenha um Modelo EA bem estruturado, poder-se-ia passar para o modelo físico visto que existe uma maior facilidade de criar um modelo físico olhando para um modelo concetual.

O desenvolvimento deste modelo EA foi baseado no enunciado deste trabalho que se encontra no ponto **1.1 da Introdução** deste mesmo relatório.

O nosso Modelo EA está representado no Anexo I e é constituído da seguinte forma:

A entidade EXTTRIPLES tem três atributos: *id*, *datetime* e *value*, onde *id* e *datetime* são ambos atributo chave. Esta entidade não está associada a nenhuma outra entidade deste modelo uma vez que, segundo o enunciado, estes são recebidos de uma empresa externa.

A entidade CLIENT tem três atributos: *name*, *ncc* e *nif*, onde *nif* é um atributo chave. Esta entidade tem duas associações. Uma associação de 1 para n com a entidade CONTACT onde é obrigatório que a entidade CONTACT esteja associada à entidade CLIENT. Esta associação é de 1 para n uma vez que CLIENT pode ter vários CONTACT, mas cada CONTACT só pertence a 1 CLIENT. A segunda associação de CLIENT é uma associação fraca de 1 para 1 com a entidade fraca PORTFOLIO em que ambas as entidades são obrigatórias na associação. Esta associação é de 1 para 1 uma vez que cada CLIENT só pode ter um PORTFOLIO e cada PORTFOLIO só pode pertencer a um CLIENT.

A entidade CONTACT tem dois atributos: *code* e *description*, sendo *code* atributo chave. Esta entidade tem duas associações: uma associação de 1 para n com CLIENT acima descrita e uma

generalização com EMAIL e PHONE em que CONTACT é obrigatoriamente um EMAIL ou um PHONE.

A entidade EMAIL tem um atributo, *addr* e faz parte da generalização de CONTACT acima descrita.

A entidade PHONE tem dois atributos: *areacode* e *number* e faz parte da generalização de CONTACT acima descrita.

A entidade fraca PORTFOLIO tem dois atributos: *totalval* e *name* onde *name* é um atributo chave. Esta entidade tem duas associações: uma associação fraca de 1 para 1 com CLIENT acima descrita e uma associação de n para m com INSTRUMENT onde INSTRUMENT está obrigatoriamente em pelo menos um PORTFOLIO. Esta associação tem um atributo, *quantity*. Esta associação é de n para m, uma vez que um INSTRUMENT pode estar em vários PORFOLIO e um PORTFOLIO pode ter vários INSTRUMENT.

A entidade INSTRUMENT tem sete atributos: *description*, ISIN, *currval*, *var6mperc*, *avg6m*, *dailyvarperc* e *varval6m*, onde ISIN é atributo chave e *currval*, *var6mperc*, *avg6m*, *dailyvarperc* e *varval6m* são atributos derivados. Esta entidade tem três associações: uma associação de n para m com PORTFOLIO acima descrita, uma associação fraca de 1 para n com DAILYREG onde ambas as entidades são obrigatórias na associação. Esta associação é de n para m pois um INSTRUMENT tem vários DAILYREG, mas um DAILYREG pertence apenas a um INSTRUMENT. INSTRUMENT tem ainda uma outra associação de 1 para n com MARKET onde MARKET tem obrigatoriamente INSTRUMENT. Esta associação é de 1 para n uma vez que um MARKET pode ter vários INSTRUMENT, mas INSTRUMENT só está pertence num MARKET.

A entidade fraca DAILYREG tem cinco atributos: *dailydate*, *minval*, *maxval*, *openingval* e *closingval* onde *dailydate* é atributo chave. Esta entidade tem apenas uma associação, uma associação de 1 para n com INSTRUMENT acima descrita.

A entidade MARKET tem três atributos: *code*, *description* e *name* em que *code* é atributo chave. Esta entidade tem duas associações: uma associação de 1 para n com INSTRUMENT acima descrita e uma associação fraca de 1 para n com DAILYMARKET em que as duas entidades são obrigatórias. Esta associação é de 1 para n pois um MARKET tem vários DAILYMARKET, mas cada DAILYMARKET pertence apenas a um MARKET

A entidade fraca DAILYMARKET tem quatro atributos: *date*, *idxmrkt*, *openingval* e *dailyvar*, em que *date* é atributo chave. Esta entidade tem apenas uma associação: uma associação de 1 para n com MARKET acima descrita.

3.2 Modelo Relacional

A partir do Modelo EA foi possível realizar o Modelo Relacional para assim depois se proceder à realização do modelo físico. Depois de alguns ajustes e normalizações até atingir a 3NF o nosso modelo Relacional está apresentado no Anexo II deste relatório.

3.3 Criação do modelo físico

Depois do Modelo Relacional concluído criou-se o modelo físico criando todas as tabelas na base de dados disponibilizada pelos docentes, e com os respetivos atributos, chaves primárias e chaves estrangeiras retirados do Modelo Relacional.

3.4 Remoção da base de dados

Fez-se um *script* para a remoção do modelo físico caso o mesmo exista, removendo as tabelas por ordem contrária à que foram criadas, de forma a não existir conflito das chaves estrangeiras.

3.5 Preenchimento Inicial da Base de Dados

Foi criado um terceiro *script* para o preenchimento inicial da base de dados, este é apenas auxiliar nos testes que se irão realizar mais tarde, caso seja necessária a entrega da base de dados a um cliente este *script* não deveria ser facultado e a base de dados deveria ir vazia.

3.6 Inserção, Atualização e Remoção de um Cliente

Para a inserção, atualização e remoção de um cliente na base de dados decidiu-se separar em dois procedimentos, uma para a inserção e atualização e outro para a remoção, em ambos os procedimentos utiliza-se transações com isolação de nível de leitura e escrita por forma À mesma poder ler e comparar dados na base de dados e saber se pode escrever ou não e onde pode escrever, utilizamos transações pois a nossa implementação conta com uma possível exceção e caso a mesma aconteça a base de dados deve voltar ao estado anterior.

3.6.1 Inserção e Atualização de um Cliente

Para este procedimento recebemos os dados fundamentais de um cliente *nif*, *ncc* e *name* e é verificado se o cliente já está inserido na base de dados, caso o mesmo já se encontre apenas se faz uma atualização dos seus dados, caso contrário ele é inserido pela primeira vez como um novo cliente.

3.6.2 Remoção de um Cliente

Para a implementação deste procedimento apenas é necessário receber como parâmetro a chave primária da tabela *CLIENT*, o *nif*, e durante a transação é visto se o cliente está inserido na base de dados, e caso esteja é removido da mesma, removendo o cliente com o *nif* igual ao recebido e a tabela de ligação entre o cliente e o seu portefólio, *CLIENT_PORTFOLIO*, também tem este

tuplo removido, no entanto escolheu-se não remover o portefólio da tabela *PORTFOLIO* pois se mais tarde o cliente reentrar na base de dados já tem o seu portefólio criado.

3.7 Inserção, Atualização e Remoção de um Mercado

Esta implementação tem a mesma ideia da implementação acima, usando o mesmo tipo de transações e nível de isolamento em cada uma delas, com algumas pequenas alterações relativas aos parâmetros recebidos e às comparações feitas.

3.7.1 Inserção e Atualização de um Mercado

Este procedimento recebe os parâmetros *description, name* e *code* que são os atributos da tabela *MARKET* e de maneira similar ao procedimento de inserção e atualização do cliente é verificado se o mercado já existe para poder apurar se cria ou atualiza o mercado na base de dados.

3.7.2 Remoção de um Mercado

Quando é necessário apagar o mercado da base de dados similarmente à remoção de um cliente vê-se se o mesmo se encontra na base de dados, visto que se não se encontrar não tem sido apagá-lo. Mas na altura de apagar este tuplo é preciso fazer muito mais remoções e verificações, visto que também é necessário remover os instrumentos associados a esse mercado, e para isso remove-se todos os tuplos de *DAILYREG* e de *POSITION* associados ao *isin* do instrumento que tenha o mesmo *mrktcode*, na tabela *INSTRUMENT*, e *code*, na tabela *MARKET* remove-se também os instrumentos e os registos das tabelas *INSTRUMENT* e *DAILYMARKET*, respetivamente que tenham os mesmos códigos de mercado. Para concluir remove-se o mercado da tabela *MARKET* que tenha o código passado como parâmetro.

3.8 Atualização dos Valores Diários de cada Instrumento

Neste procedimento começa-se por calcular os valores diários a colocar na tabela *DAILYREG* para o instrumento que tenha o mesmo *id* e *datetime*, na tabela *EXTTRIPLE*, passados como parâmetro calculando os atributos *minval*, valor mínimo na tabela *EXTTRIPLE* para aquele instrumento naquele dia, *maxval*, valor máximo na tabela *EXTTRIPLE* para aquele instrumento naquele dia, *openingval*, primeiro valor inserido na tabela *EXTTRIPLE* para aquele instrumento naquele dia, e *closingval*, último valor inserido na tabela *EXTTRIPLE* para aquele instrumento naquele dia. De seguida é feita uma verificação se já existe em *DAILYREG* e *EXTTRIPLE* um instrumento com o mesmo *id* na mesma data passados como parâmetros e caso exista faz uma atualização na tabela *DAILYREG* com os valores obtidos, se não existir na tabela *INSTRUMENT* então não é inserido na tabela *DAILYREG* se existir significa que foi a primeira inserção em *EXTTRIPLE* para aquele instrumento naquele dia, por isso procedesse à inserção dos valores obtidos na tabela *DAILYREG*.

3.9 Cálculo da Média a 6 meses de um Instrumento

Para esta função decidiu-se passar a quantidade de tempo em dias que se tem de andar para trás como parâmetro, para dar liberdade ao utilizador de escolher a quantidade de tempo caso queira uma média maior ou menor de 6 meses. A data inicial é calculada através do auxilio da função *getdate()* do *t-sql* fazendo uma diferença do dia com a quantidade de dias passados como parâmetro. Por fim retorna-se uma seleção da média de todos os *closingval* na tabela *DAILYREG* que tenham uma data, *dailydate*, mais recente que a data inicial, esta média é calculada com auxilio da função *avg()* do *t-sql*, onde é passado como parâmetro o atributo *closingval*.

3.10 Atualização dos Dados Fundamentais de um Instrumento

Esta função recebe como parâmetros o *isin* do instrumento e a *date* do registo do instrumento para aferir a partir de que data deve ser calculado os dados fundamentais, estes dados são retornados numa tabela, visto que são dados derivados não tem sentido juntar a uma tabela têm sentido juntar a uma tabela do modelo físico. Os dados *dailyvar* e *var6m*, variação diária e variação a seis meses respetivamente são os primeiros a serem calculados pois serão necessários para o cálculo de outros dados. O atributo *dailyvar* é calculado a partir da diferença dos valores mínimos e máximo registados em *DAILYREG* para o *isin* passado como parâmetro, já o *var6m* é calculado através da diferença do valor máximo nos últimos seis meses que exista na tabela *DAILYREG* e do valor mínimo da mesma tabela para a mesma quantidade de tempo, para esta operação utiliza-se o auxílio da função *getdate()* do *t-sql*. De seguida insere-se os valores na tabela retornada que são respetivamente o *dailyvar* e o *var6m*, já calculados, o *currval*, que é calculado numa função auxiliar criada, que está descrita neste relatório mais abaixo, e é denominada por *get_Currval()* recebendo como parâmetro o *isin*, o *avg6m* calculado através da função *Average()*, descrita acima que recebe a data e o *isin* passados como parâmetros, e as percentagens de variação *dailyvarperc* e *var6mperc* que são calculados a partir da divisão de *dailyvar* e *var6m* pelos seus correspondentes valores mínimos.

3.11 Criação de um portfólio

Foi criado um procedimento visto não ser necessário retornar nada pois ia apenas ser alterada uma tabela já criada na base de dados. Esse procedimento recebe o *nif* de um cliente já existente na base de dados. Começa-se por declarar o nome do portfólio que se decidiu nomear pelo *nif* do cliente e a string "_portfolio". De seguida utiliza-se um "*if not exists"* para verificar se não existe já um portfólio com aquele *nif* pois os clientes só podem ter um portfólio para isso terá de se selecionar o *nif* da tabela *CLIENT_PORTFOLIO* onde o *nif* da tabela for igual ao *nif* passado como parâmetro e seguidamente faz-se um "and" porque é necessário que o *nif* passado como

parâmetro esteja presente na tabela *CLIENT* para tal apenas fez-se um *select* do *nif* da tabela em questão onde o *nif* fosse igual ao *nif* passado como parâmetro e caso esse exista ele irá inserir um novo portfólio desse cliente, para tal fez- se um *insert* do nome anteriormente definido do portfólio e deixou-se o *totalval* a *null* porque só vai ser atualizado quando for chamada do procedimento *UpdateTotalVal.*

3.12 Atualização do valor total de um portfólio

Para a resolução deste problema decidiu-se criar um procedimento pois não se sentiu a necessidade de ter uma função pois não iria retornar nada pois apenas iria fazer a atualização de um tuplo na tabela *PORTFOLIO.*

Nesse procedimento irão se receber três parâmetros, o nome que irá representar o nome do portfólio, a quantidade para poder calcular o valor total e o *isin* para poder ser identificado o instrumento em que o valor total será atualizado.

Para começar irá se inserir esse portfolio na tabela *POSITION* simplesmente inserindo os parâmetros *quantity, name* e *isin.* De seguida irá ser feita a atualização na tabela *PORTFOLIO,* segundo o enunciado providenciado pelos docentes o valor total será calculado através do somatório do produto do valor atual dos instrumentos e da quantidade de cada um. Utilizou-se a função *SUM* e dentro desse fez-se o produto da *quantity* e *closingval* colunas que resultariam do *join* da tabela *POSITION* e *DAILYREG* onde os *isin* forem iguais onde os nomes forem iguais ao nome passado como parâmetro e o *dailydate* do *DAILYREG* for igual à data mais recente registada no *DAILYREG* onde o *isin* for igual ao *isin* passado como parâmetro. Assim irá resultar num *join* onde tivermos apenas o nome passado e a data mais recente que será o *closingval* que foi assumido como o valor atual. Este *update* apenas foi feito onde o nome no *PORTFOLIO* for igual ao nome passado como parâmetro.

3.13 Atualização do valor total de um portfólio

Foi criada uma função denominada de *Portfolio_List* quer irá receber o nome do portfólio em questão. No enunciado é pedido na lista o *isin,* a quantidade, o valor atual e a percentagem de variação em relação ao dia anterior.

Esta listagem irá retornar uma tabela com esses atributos, para isso irá ser selecionado, o *isin,* a quantidade e irá ser selecionada a função *get_Currval* para ir buscar o valor atual e *get_dailypercvar* para retornar a variação. Estes valores estão a ser buscados da tabela *POSITION* onde o nome de *POSITION* for igual ao nome passado como parâmetro.

3.13 Criar uma vista que mostre o resumo dos portfólios

O objetivo era apresentar um resumo dos portfólios de todos os clientes incluindo o somatório do número de instrumentos e pelo somatório do valor total. Para isto irá se apresentar o nome do portfolio, para apresentar o número de instrumentos utilizou-se a função *count* para contar os

isin agrupando por nome e fez-se um alias para ser mais claro na vista do que se trata aquele valor para *NoInstruments* no final para ir buscar o somatório do valor total reparou-se que era apenas o valor total que estava na tabela visto que cada cliente só pode ter um portfolio por isso o valor total desse portfolio iria ser automaticamente o somatório do valor total visto o cliente não ter mais nenhum. Por deu-se o nome desse *select* de T1 para poder fazer *join* com *PORTFOLIO* onde os nomes forem iguais e assim poder juntar a informação que vem do *PORTFOLIO* e do *count* feito.

3.14 Atualização do Daily Market

Foi criado um procedimento por forma a atualizar os dados da tabela *DAILYMARKET* que irá receber como parâmetro o código do mercado que se pretende atualizar e a data que se pretende atualizar.

Segundo o enunciado, "Para cada dia e para cada mercado, são registados o valor do índice/mercado (soma de todos os valores de abertura dos seus instrumentos, calculado no fim do dia), o valor de abertura (valor de índice do dia útil anterior) e a variação diária (em euros)" .

Para tal, para começar por atualizar o valor de índice/mercado irá se selecionar a soma dos valores de abertura do *join* do *DAILYREG* com *INSTRUMENT* onde o *isin* fosse igual onde o *code* que está na tabela *INSTRUMENT* fosse igual ao código passado como parâmetro e a *dailydate* de *DAILYREG* fosse igual à data passada como parâmetro. De seguida para atualizar o valor de abertura, fez um *select top 1 idxmrkt* do *DAILYMARKET* para selecionar o que estaria no topo da lista sendo que estava organizado por data por ordem decrescente onde a data do *DAILYMARKET* for menor que a data passada como parâmetro e o código da mesma tabela for igual ao código passado como parâmetro. Finalmente, para definir a variação diária selecionou-se o *maxval* do *join* de *DAILYREG* com *INSTRUMENT* onde os *isin* forem iguais onde o os códigos do *join* forem iguais ao código passado como parâmetro e a *dailydate* é igual à data passada como parâmetro subtraindo ao *minval* que foi adquirido da mesma maneira que o *maxval* como foi descrito em cima. De seguida irá-se verificar se o código e data passados como parâmetro existem no *DAILYMARKET* ou não, se existir irá fazer um *update* em que irá substituir onde estiver esse código e data pelos valores declarados em cima se não existir irá fazer um *insert* dos valores declarados em cima.

3.15 Criação do Trigger

O objetivo de este *trigger* é atualizar os valores diários de um instrumento, os dados fundamentais do mesmo e os valores diários do mercado desse instrumento.

Para tal foi dito ao *trigger* para ser realizado na tabela *EXTTRIPLE* após ser feita uma inserção

nessa mesma tabela. Quando uma inserção é feita ele irá selecionar o *id* e o *datetime* dessa inserção feita e de seguida irá se buscar o código presente no *INSTRUMENT* onde o *isin* for igual ao *id* selecionado. Com isso feito irá executar os procedimentos *p_atualizaValorDiario* e do *Dailymarketupdate* e a função *FundamentalDataTable.*

3.16 Funções Auxiliares

As funções auxiliares são funções criadas para evitar repetição de código uma vez que existem valores que são necessários utilizar mais do que uma vez durante a realização do trabalho.

3.16.1 Busca do Valor Atual

A função *dbo.get_Currval* retorna o valor do valor atual em *Money*.

Para tal retorna-se o valor que está no topo da coluna *closingval* da tabela DAILYREG onde a coluna ISIN for igual à variável ISIN passada como parâmetro, estando a tabela ordenada de forma decrescente pela coluna *dailydate*, uma vez que o valor atual é determinado através do último valor de fecho registado na tabela DAILYREG.

3.16.2 Busca da Variação diária em percentagem

A função *dbo.get_dailypercvar* retorna a variação diária em percentagem, em *decimal (5,2).*

Para tal foi necessárias quatro variáveis auxiliares, todas do tipo Money. A variável *current_closingval* chama a função *dbo.get_Currval* passando como parametro o ISIN passado como parâmetro na função *dbo.get_dailypercvar*. A variável last_closing é o valor retornado pelo *select* da coluna *closingval* da tabela retornada pelo *select* dos dois valores que estão no topo da coluna *closingval* ordenada pela coluna *dailydate* de forma decrescente onde a coluna isin for igual ao isin passado como parâmetro e escolhe-se a linha que está em segundo no topo.

De seguida compara-se estas duas variáveis vendo qual delas é maior. A que for maior é guardada na variável max e a que for menor é guardada na variável min.

Por fim retorna-se a conversão para *decimal (5,2)* da divisão da subtração max- min por max.

4. Avaliação Experimental

Este capítulo servirá para mostrar uma avaliação das diferentes soluções escolhidas para a resolução dos diferentes problemas apresentados.

Para a parte da avaliação experimental decidiu-se criar um ficheiro *batch* que iria testar as funcionalidades do segundo ponto por forma as mostrar que as diferentes soluções criadas estariam a funcionar corretamente.

De seguida, desenvolveu – se um *script* em T-SQL com testes para testar a funcionalidade desde a alínea 2.c até ao 2.m que já estariam criadas através do *batch file* para uma melhor compreensão do funcionamento das diferentes alíneas. O *script* está dividido pelas diferentes alíneas apresentadas no enunciado deste trabalho para um melhor entendimento o teste a ser feito.

4.1 Criação do *batch*

Primeiro foi criado um ficheiro *batch* que irá estar responsável pela criação das tabelas, os *drops* das mesmas e a inserção dos valores iniciais. Para além disso também irá estar responsável de criar as funções, procedimentos e vistas pedidas nas alíneas seguintes.

4.2 Inserção, remoção e atualização da informação de um cliente

Para esta alínea tomou-se a decisão de dividir em dois procedimentos para um melhor funcionamento e melhor compreensão da ação a querer tomar. Para isso utilizou-se o procedimento *update_client* para fazer o *update* ou inserção do cliente e para remoção o *remove_client*.

4.2.1 Atualização ou Inserção de um cliente

Para testar o *update* de um cliente executou-se o comando *exec* que será usado para correr um procedimento passando como parâmetro um *nif* já existente na base de dados e nos restantes parâmetros nova informação a ser atualizada e ao executar e de seguida fazer um *select* da tabela *CLIENT* foi possível observar que a atualização foi feita. Seguidamente executou-se o mesmo procedimento mas com um *nif* ainda não existente na base de dados e quando tal acontece ele irá inserir na tabela *CLIENT* esse novo cliente com o resto da informação passada como parâmetro

4.2.2 Remoção de um cliente

Na remoção de um cliente apenas executou-se o procedimento criado e passou-se um *nif* já existente na base de dados e ao selecionar a tabela irá se verificar que o mesmo já não se encontra lá. Caso o *nif* passado não se encontre na tabela não irá alterar nada.

4.3 Inserção, remoção e atualização da informação de um mercado

Na realização de este código utilizou-se o mesmo raciocínio que na alínea anterior também dividindo a este código em dois procedimentos. Neste caso seria o *update_market* para o *update* ou inserção de um mercado e *remove_market* para a remoção de um mercado.

4.3.1 Atualização ou Inserção de um mercado

Assim como foi descrito na alínea anterior serão executados dois comandos *exec* chamando o mesmo procedimento em que será passado como parâmetro informação nova tanto na *description* do mercado assim como no nome do mercado sendo que a diferença de um teste para o outro é que em um é passado um código de mercado que existe e assim faz uma atualização e no outro é passado um código que não esteja registado na tabela *MARKET*.

4.3.2 Remoção de um mercado

Para a remoção apenas se passou um código de um mercado que esteja registado na base de dados e irá removê-lo e caso não exista o código passado não irá fazer nada.

4.4 Atualização dos valores diários de cada instrumento

Neste teste foram testadas três situações, o caso de o *isin* passado já esteja registada no *DAILYREG*, caso esteja registado na tabela *INSTRUMENT* mas não no *DAILYREG* e por último o caso de não estar registado no *DAILYREG* nem nos *INSTRUMENT*. Para o primeiro caso, passando um *isin* já existente e passada uma data nos registos diários irá atualizar o tuplo que contiver aquele *isin* e a data. Para o segundo, passando um *isin* estando apenas registado na tabela *INSTRUMENT* irá fazer uma inserção dos dados daquele instrumento na data passada como parâmetro.

Finalmente, para o terceiro e último caso não irá fazer nada visto o *isin* inserido não estar registada na tabela de *INSTRUMENT*.

4.5 Cálculo da média a 6 meses de um instrumento

Neste caso apenas se fez o teste em que se passa os dias equivalentes a seis meses e o *isin* correspondente do instrumento a usar para calcular a médio e o teste deverá retornar a média dos valores registados naquele instrumento até seis meses atrás.

4.6 Atualização dos dados fundamentais de um instrumento

Para este teste é chamada a função *FundamentalDataTable* passando como parâmetro o *isin* do instrumento e a data a atualizar. Ao fazer o *select* deverá mostrar uma listagem de uma tabela com a atualização dos dados fundamentais desse instrumento.

4.7 Criação de um portfólio

Neste teste apenas irá será passado o *nif* de um cliente já registado na base de dados e com essa informação irá inserir um novo tuplo na tabela PORTFOLIO, caso não exista registo desse *nif* não irá fazer nada.

4.8 Atualização do valor total de um portfólio

Para mostrar o funcionamento da resolução deste problema fez-se um *exec* do procedimento *UpdateTotalVal* e passou-se como parâmetro o nome de portfólio, quantidade de um cliente e e o *isin* do instrumento a atualizar e assim irá atualizar o seu valor total.

4.9 Listagem de um portfólio

Para testar esta função apenas é passado o nome do portfólio e irá aparecer a listagem dos objetivos pedidos neste problema

4.10 Criar uma vista que mostre o resumo dos portfólios

Para o teste de funcionamento vista foi apenas feito um *select * from* da vista criada já no batch.

4.11 Teste do *trigger*

Para testar o *trigger*, depois de o criar basta apenas fazer uma inserção na tabela *EXTTRIPLE* que o *trigger* irá atualizar o *DAILYREG* e o *DAILYMARKET*

4.12 Análise de resultados

Após correr o ficheiro *batch* foi possível confirmar o funcionamento das funcionalidades das alíneas do ponto 2 e com o teste do *script* de testes e da criação do *trigger* foi garantido, para os utilizadores da base de dados, que todas as funcionalidades desde o 2.c até ao 2.l estariam a realizar os resultados esperados, tendo sempre em conta as restrições de integridade e regras de negócio.

5. Conclusões

Neste trabalho foi pedido para implementar um sistema de informação de uma empresa de gestão de mercados financeiros de nome *Pilim*. Para tal foi desenvolvido uma base de dados utilizando o *Transact* SQL (T-SQL).

Numa fase inicial do trabalho foi desenvolvido um Modelo Entidade-Associação do sistema de informação que, à medida que se foi avançando no trabalho, foram encontradas pequenas falhas que foram retificadas. Essas falhas contribuíram para uma melhor compreensão da base de dados a ser desenvolvida.

O modelo Relacional realizado através da representação do modelo EA anteriormente realizado, foi um impulso para o início do desenvolvimento do modelo físico do sistema de informação.

Para o modelo físico, foram criadas as tabelas descritas no modelo relacional. Enquanto eram resolvidas as alíneas do trabalho e eram encontradas falhas no modelo EA, as mesmas eram retificadas nos modelos Relacional e físico, o que levou a um melhoramento do trabalho.

Este trabalho tem m alíneas de exercícios para serem resolvidos, sendo que as primeiras 3 alíneas a criação do modelo físico, a remoção do mesmo e o preenchimento inicial da base de dados.

Ao resolver as restantes alíneas, foi compreendido quando usar uma função ou um procedimento. Um procedimento é usado quando a alínea em questão não pede qualquer tipo de retorno, como por exemplo as alíneas que pedem para atualizar uma ou mais tabelas da base de dados, enquanto que uma função é usada quando é preciso retornar algo, sendo uma tabela, uma listagem ou uma variável.

Com este trabalho obteve-se uma melhor compreensão da utilização de *triggers* e transações. As transações foram utilizadas nas alíneas d) e e) em que é necessário inserir, atualizar e remover dados de tabelas. O *trigger* foi utilizado na inserção de valores na tabela de triplos para que quando é inserido um novo valor nessa tabela, o *trigger* irá desencadear os procedimentos de atualização dos registo diário e mercado diário bem como a função de atualização dos dados fundamentais de um instrumento.

Para concluir realizou-se um ficheiro de teste que testas as funcionalidades das alineas anteriormente realizadas utilizando um *Batch file* para executar os comandos SQL e um ficheiro SQL que os testa.

Referências

[1] Moodle 2019-2020 SI2 " https://1920moodle.isel.pt/course/view.php?id=4301"

A.1 Modelo EA

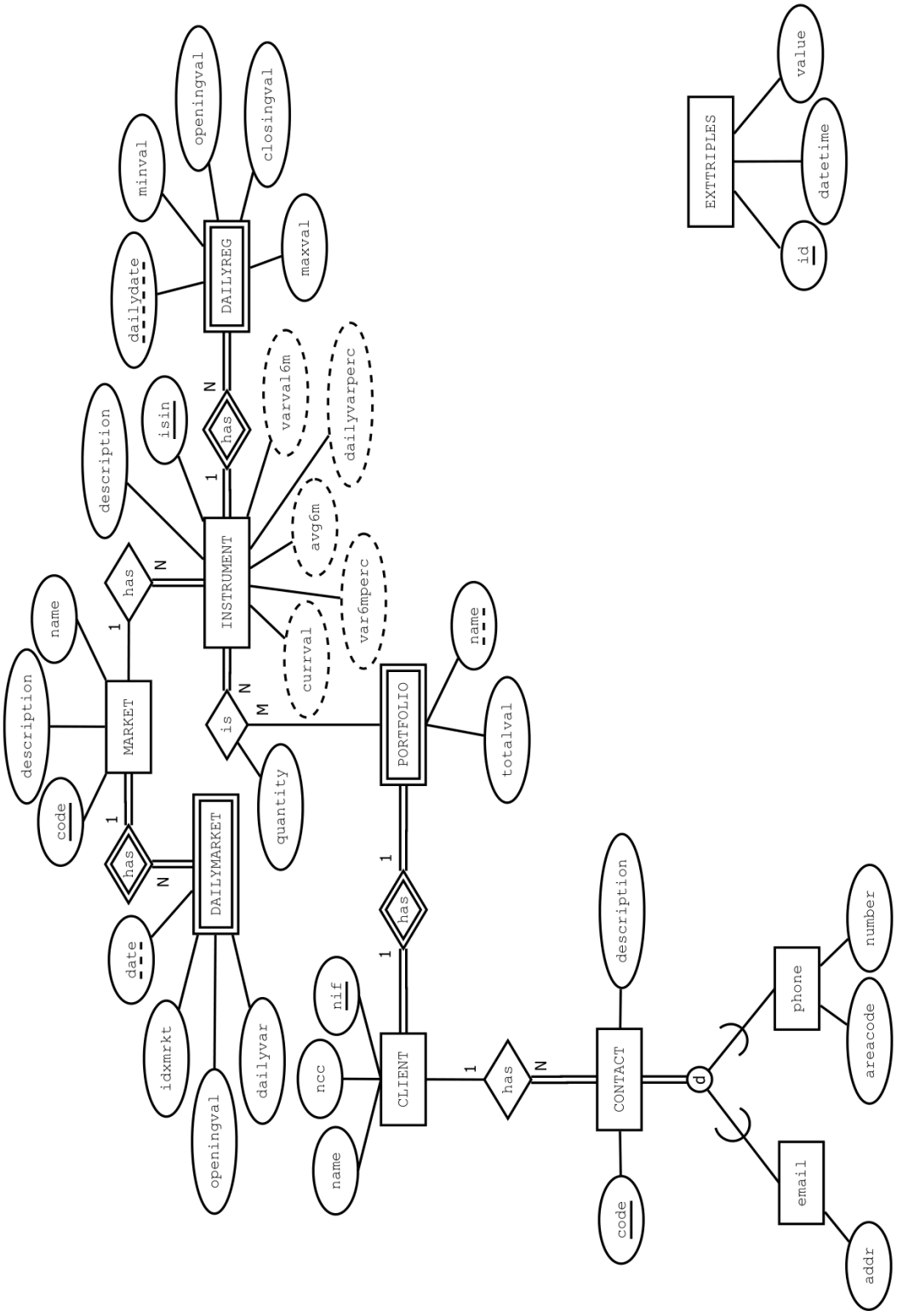

Figura 1 – Diagrama do Modelo EA.

A.2 Modelo Relacional

ExtTriples

EXTTRIPLES (value, datetime, id)

Atributo	Tipo	Restrições de Integridade
value	Money	Em Euros(€)
datetime	Datetime	
id	Char(12)	

PK{id, datetime}

F= {id, datetime} → {value}

Email

EMAIL (<u>code</u>, addr,description)

Atributo	Tipo	Restrições de Integridade
code	Int	
addr	Varchar(300)	
description	Varchar(50)	

PK: {code}

F= {code} → {addr, description}

Phone

PHONE (<u>code</u>, areacode, number, description)

Atributo	Tipo	Restrições de Integridade
code	Int	
areacode	Varchar(4)	Indicativo do país
number	Decimal(9)	
description	Varchar(300)	

PK: {code}

F= {code} → {areacode, number, description}

Client

CLIENT(nif, ncc, name)

Atributo	Tipo	Restrições de Integridade
nif	Decimal(9)	
ncc	Decimal(7)	
name	Varchar(50)	

PK: {nif}

F= {nif} → {ncc, name}

Market

MARKET(<u>code</u>, description, name)

Atributo	Tipo	Restrições de Integridade
code	Int	
description	Varchar(300)	
name	Varchar(50)	

PK: {code}

F= {code} → {description, name}

Daily Market

DAILYMARKET(<u>code</u>, date, idxmrkt, idxopeningval, dailyvar)

Atributo	Tipo	Restrições de Integridade
code	Int	
date	date	
idxmrkt	money	Em Euros(€)
idxopeningval	money	Em Euros(€)
dailyvar	money	Em Euros(€)

PK: {code, date}

FK: {code} para MARKET(code)

F= {code, date} → {idxmrkt, dailyvar, idxopeningval}

Client Portfolio

CLIENT_PORTFOLIO(nif, name)

Atributo	Tipo	Restrições de Integridade
nif	Decimal(9)	
name	Varchar(50)	

PK: {name, nif}

FK:{name, nif} para CLIENT(nif) e PORTFOLIO(name)

F= {name, nif} → {}

Instrument

INSTRUMENT(ISIN, mrktcode, description)

Atributo	Tipo	Restrições de Integridade
ISIN	Char(12)	
mrktcode	Int	
description	Varchar(300)	

PK: {ISIN}

FK: {mrktcode} para MARKET(code)

F= {ISIN} → {description, mrktcode}

Portfólio

PORTFOLIO(name, totalval)

Atributo	Tipo	Restrições de Integridade
name	Varchar(50)	
totalval	money	Somatório do produto do valor atual dos instrumentos e da quantidade de cada um, em Euros(€)

PK: {name}

F= {name} → {totalval}

Position

POSITION(ISIN, name, quantity)

Atributo	Tipo	Restrições de Integridade
ISIN	Char(12)	
name	Varchar(50)	
quantity	int	

PK: {ISIN, name}

FK: {ISIN, name} para PORTFOLIO(name) e INSTRUMENT(isin)

F= {ISIN, name} → {quantity}

Daily Regist

DAILYREG(ISIN, dailydate, minval, maxval, closingval, openingval)

Atributo	Tipo	Restrições de Integridade
ISIN	Char(12)	
dailydate	date	
minval	money	Em Euros (€)
maxval	money	Em Euros (€)
closingval	money	Em Euros (€)
openingval	money	Em Euros (€)

PK: {ISIN, dailydate}

FK: {ISIN} para INSTRUMENT(ISIN)

F= {ISIN, dailydate} → {minval, maxval, openingval, openingval}

A.3 Código *SQL*

```
create table MARKET(
   code      int,
   description    varchar(300),
   name      varchar(50),
   constraint pkmarket primary key(code)
);

create table DAILYMARKET(
   idxmrkt          money,
   dailyvar      money,
   idxopeningval  money,
   code            int,
   date            date,
   constraint fkdailymrkt foreign key (code) references MARKET(code),
   constraint pkdailymrkt primary key(date, code)
);

create table CLIENT(
   nif       decimal(9),
   ncc       decimal(7),
   name   varchar(50),
   constraint pkclient primary key(nif)
);

create table PORTFOLIO(
   name      varchar(50),
   totalval   money,
   constraint pkportfolio primary key(name)
);

create table CLIENT_PORTFOLIO(
   name varchar(50),
   nif decimal(9),
   constraint pkCLIENT_PORTFOLIO primary key(name, nif),
   constraint fkCLIENT foreign key(nif) references CLIENT(nif),
   constraint fkPORTFOLIO foreign key(name) references PORTFOLIO(name)
);

create table INSTRUMENT(
   isin         char(12),
   description       varchar(300),
   mrktcode        int,
   constraint pkinstrument primary key(isin),
   constraint fkinstrument foreign key (mrktcode) references MARKET(code)
);

create table POSITION(
   quantity      int,
   name         varchar(50),
   isin         char(12),
   constraint pkpositions primary key(isin, name),
   constraint fkportfolio_pos foreign key(name) references PORTFOLIO(name),
   constraint fkinstrument_pos foreign key(isin) references INSTRUMENT(isin)
);
```

```
create table EMAIL(
   code      int,
   description    varchar(300),
   addr      varchar(50),
   constraint pkemail primary key(code)
);

create table PHONE(
   code      int,
   description    varchar(300),
   areacode   varchar(4),
   number    decimal(9),
   constraint pkphone primary key(code)
);
create table EXTTRIPLE(
   value     money not null,
   datetime   datetime not null,
   id       char(12) not null,
    constraint pkexttriple primary key(id, datetime)

);

create table DAILYREG(
   isin            char(12),
   minval          money,
   openingval       money,
   maxval          money,
   closingval       money,
   dailydate        date,
   constraint fkinstrument_reg foreign key(isin) references INSTRUMENT(isin),
    constraint pkdailyreg primary key(isin, dailydate)

);




drop table if exists DAILYREG;
drop table if exists EXTTRIPLE;
drop table if exists PHONE;
drop table if exists EMAIL;
drop table if exists POSITION;
drop table if exists INSTRUMENT;
drop table if exists CLIENT_PORTFOLIO;
drop table if exists PORTFOLIO;
drop table if exists CLIENT;
drop table if exists DAILYMARKET;
drop table if exists MARKET;


insert into MARKET values (111, 'Description1', 'Market1');
insert into MARKET values (222, 'Description2', 'Market2');

insert into    DAILYMARKET values (111, 1111, 11111, 111,'2019-11-01');
insert into    DAILYMARKET values (222, 2222, 22222, 222,'2019-11-03');

insert into CLIENT values (111111111, 1111111, 'Client1');
insert into CLIENT values (222222222, 2222222, 'Client2');
```

```
insert into CLIENT values (444444444, 4444444, 'Client4');
insert into CLIENT values (555555555, 5555555, 'Client5');

insert into PORTFOLIO values ('111111111_portfolio', 10000);
insert into PORTFOLIO values ('444444444_portfolio', 40000);

insert into CLIENT_PORTFOLIO values('111111111_portfolio', 111111111);
insert into CLIENT_PORTFOLIO values('444444444_portfolio', 444444444);

insert into INSTRUMENT values (111111111111, 'Description1', 111);
insert into INSTRUMENT values (222222222222, 'Description2', 222);
insert into INSTRUMENT values (333333333333, 'Description3', 333);
insert into INSTRUMENT values (555555555555, 'Description5', 111)

insert into POSITION values (1, '111111111_portfolio', 111111111111);
insert into POSITION values (2, '444444444_portfolio', 222222222222);
insert into POSITION values (3, '111111111_portfolio', 333333333333);

insert into EMAIL values (1,'Nuno mail','ncardeal@pilim.com');
insert into EMAIL values (2,'Carol mail','carolct@pilim.com');
insert into EMAIL values (3,'Juju mail','jujucmps@pilim.com');

insert into PHONE values (1,'Nuno phone', '+351', 961222333);
insert into PHONE values (2,'Carol phone', '+351', 911222333);
insert into PHONE values (3,'Juju phone', '+351', 921222333);

insert into EXTTRIPLE values (1111, '2019-11-01 11:11:11', 111111111111);
insert into EXTTRIPLE values (2222, '2019-11-01 12:12:12', 111111111111);
insert into EXTTRIPLE values (3333, '2019-11-01 13:13:13', 111111111111);
insert into EXTTRIPLE values (4444, '2019-10-02 11:11:11', 222222222222);
insert into EXTTRIPLE values (5555, '2019-10-02 12:12:12', 222222222222);
insert into EXTTRIPLE values (6666, '2019-10-02 13:13:13', 222222222222);
insert into EXTTRIPLE values (7777, '2019-12-14 12:55:08', 444444444444);
insert into EXTTRIPLE values (8888, '2019-09-10 12:55:08', 555555555555);
insert into EXTTRIPLE values (9999, '2019-09-10 13:55:08', 555555555555);
insert into EXTTRIPLE values (1000, '2019-10-10 14:55:08', 555555555555);

insert into DAILYREG values (111111111111, 11, 11111, 1111, 111, '2019-11-
01');
insert into DAILYREG values (111111111111, 22, 22222, 2222, 222, '2019-08-
02');
insert into DAILYREG values (111111111111, 33, 33333, 3333, 333, '2019-08-
03');
insert into DAILYREG values (222222222222, 44, 44444, 4444, 444, '2019-09-
04');



create procedure update_client
    @ncc decimal(7),
    @nif decimal(9),
    @name varchar(50)
as
set transaction isolation level read committed
begin transaction
    IF EXISTS(select nif from CLIENT where nif = @nif)
        begin
            update dbo.CLIENT
            set ncc  = @ncc,
            name = @name
```

```
            where @nif = dbo.CLIENT.nif
        end
    ELSE
        begin
            insert into dbo.CLIENT values
            (
             @nif,
             @ncc,
             @name
            )
        end
commit


create procedure remove_client
    @nif decimal(9)
as
set transaction isolation level read committed
begin tran
    if exists(select nif from CLIENT where @nif = nif)
        begin
            delete from CLIENT where nif = @nif
            delete from CLIENT_PORTFOLIO where nif = @nif
        end
commit


create procedure update_market
    @description varchar(300),
    @name varchar(50),
    @code int
as
set transaction isolation level read committed
begin tran
    if exists(select code from MARKET where code = @code)
        begin
            update dbo.MARKET
            set description = @description,
                name = @name
            where @code = dbo.MARKET.code
        end
    else
        begin
            insert into MARKET values
            (
            @code,
            @description,
            @name
            )
        end
commit


create procedure remove_market
    @code int
as
set transaction isolation level read committed
begin tran
```

```
    if exists(select code from MARKET where @code = code)
        begin
            delete from DAILYREG where isin = (select isin from INSTRUMENT
where mrktcode = @code)
            delete from POSITION where isin = (select isin from INSTRUMENT
where mrktcode = @code)
            delete from INSTRUMENT where mrktcode = @code
            delete from DAILYMARKET where code = @code
            delete from MARKET where code = @code
        end
commit




create procedure dbo.p_actualizaValorDiario @id char(12), @date datetime
as
begin
    declare @minval money, @maxval money, @openingval money, @closingval
money
    set @minval = (select top 1 value from dbo.EXTTRIPLE where @id= id and
convert(date, datetime) = convert(date, @date) order by value)
    set @maxval = (select top 1 value from dbo.EXTTRIPLE where @id = id and
convert(date, datetime) = convert(date, @date) order by value desc)
    set @openingval = (select top 1 value from dbo.EXTTRIPLE where @id = id
and convert(date, datetime) = convert(date, @date) order by datetime)
    set @closingval = (select top 1 value from dbo.EXTTRIPLE where @id = id
and convert(date, datetime) = convert(date, @date) order by datetime desc)
    if exists(select distinct isin, dailydate from dbo.DAILYREG join
dbo.EXTTRIPLE on isin = id where convert(date,@date) = dailydate and isin =
@id)
        begin
            update dbo.DAILYREG
            set minval     = @minval,
                maxval     = @maxval,
                openingval = @openingval,
                closingval = @closingval
            where isin = @id and dailydate = CONVERT(date,@date)
        end;
    if exists(select isin from INSTRUMENT where isin = @id)
        begin
    if not exists((select isin from DAILYREG where dailydate = CONVERT(date,
@date)))
        begin
            insert into dbo.DAILYREG
            values (@id,
                    @minval,
                    @openingval,
                    @maxval,
                    @closingval,
                    CONVERT(date, @date))
        end
    end
end

go
```

```
create function dbo.Average(@days int, @isin char(12))
    returns money
as
begin
    declare @initdate date
    set @initdate = getdate() - @days
    declare @average money
    set @average = (select AVG(closingval) from dbo.DAILYREG where
DAILYREG.isin = @isin and dailydate >= @initdate)
    return @average
end
Go
```

```
create function dbo.FundamentalDataTable(@isin char(12), @date date)
    returns @ret table
                (
                dailyvar     money,
                currval      money,
                avg6m        money,
                var6m        money,
                dailyvarperc decimal(5, 2),
                var6mperc    decimal(5, 2)
                )
as
begin
    declare @dailyvar money
    set @dailyvar = (select maxval from dbo.DAILYREG where isin = @isin and
dailydate = @date) -
                    (select minval from dbo.DAILYREG where isin = @isin and
dailydate = @date)
    declare @var6m money
    set @var6m = (select top 1 maxval from dbo.DAILYREG where dailydate >=
(getdate() - 180) order by maxval desc) -
                 (select top 1 minval from dbo.DAILYREG where dailydate >=
(getdate() - 180) order by minval)
    insert into @ret
    values (@dailyvar,
            dbo.get_Currval(@isin),
            dbo.Average(180, @isin),
            @var6m,
            @dailyvar / (select minval from dbo.DAILYREG where isin = @isin
and dailydate = @date),
            @var6m / (select top 1 minval from dbo.DAILYREG where dailydate
>= (getdate() - 180) order by minval))
    return
end
GO
```

```
create procedure dbo.createPortfolio @nif decimal(9)
as
begin
    declare @name varchar(50)
    set @name = concat(@nif, '_portfolio')
```

```
    if not exists(select nif from dbo.CLIENT_PORTFOLIO where nif = @nif) and
       exists(select nif from dbo.CLIENT where nif = @nif)
        begin
            insert into dbo.PORTFOLIO(name)
            values (@name)
        end
end
GO



create procedure dbo.UpdateTotalVal @name varchar(50),
                                    @quantity int,
                                    @isin char(12)
as
begin
    insert into dbo.POSITION
    values (@quantity,
            @name,
            @isin)
    update dbo.PORTFOLIO
    set totalval = (select sum(quantity * closingval)
                    from dbo.POSITION
                             join dbo.DAILYREG on POSITION.isin =
dbo.DAILYREG.isin
                    where name = @name
                      and dailydate = (select
                                       top 1
                                       dailydate
                                       from dbo.DAILYREG
                                       where isin = dbo.POSITION.isin
                                       order by dailydate desc))
    where name = @name
end
go


create function dbo.Portfolio_List(@name varchar(50))
    returns table
    as
    return
        (
        select isin,
               quantity,
               (select dbo.get_Currval(isin))      as CurrVal,
               (select dbo.get_dailypercvar(isin)) as Dailyvarperc
        from POSITION
        where name = @name
        )
go



create view PORTFOLIO_SUMMARY as
select PORTFOLIO.name, T1.NoInstruments, PORTFOLIO.totalval from (select
name, count(isin) as NoInstruments from POSITION group by name) T1
```

```
JOIN
PORTFOLIO
on T1.name = PORTFOLIO.name
go


create procedure dbo.DailyMarketUpdate @code int, @date date
    as
    begin
     declare   @idxmrkt money, @dailyvar money, @idxopeningval money
     set @idxmrkt = (select sum(openingval) from DAILYREG join INSTRUMENT on
DAILYREG.isin = INSTRUMENT.isin where mrktcode = @code and dailydate = @date)
     set @idxopeningval = (select top 1 idxmrkt from DAILYMARKET where date <
@date and code = @code order by date desc)
     set @dailyvar = (select maxval from dbo.DAILYREG join INSTRUMENT on
DAILYREG.isin = INSTRUMENT.isin where mrktcode = @code and dailydate = @date)
-
                    (select minval from dbo.DAILYREG join INSTRUMENT on
DAILYREG.isin = INSTRUMENT.isin where mrktcode = @code and dailydate = @date)
     if exists(select date from DAILYMARKET where code = @code and date =
@date)
         begin
            update dbo.DAILYMARKET
            set
                idxmrkt = @idxmrkt,
                idxopeningval = @idxopeningval,
                dailyvar = @dailyvar
            where code = @code and date = @date
    end

    if not exists(select date from DAILYMARKET where code = @code and date =
@date)
        begin
            insert into dbo.DAILYMARKET
            values (
                    @idxmrkt,
                    @dailyvar,
                    @idxopeningval,
                    @code,
                    @date)
        end
    end




create function dbo.get_Currval(@isin char(12))
    returns money
as
begin
    return (select top 1 closingval from dbo.DAILYREG where isin = @isin
order by dailydate desc)
end
go
create function dbo.get_dailypercvar(@isin char(12))
    returns decimal(5,2)
as
begin
```

```
    declare @current_closingval money
    set @current_closingval = dbo.get_Currval(@isin)
    declare @min money
    declare @max money
    declare @last_closingval money
    set @last_closingval = (select closingval from
                                (select top 2 closingval,
                                 ROW_NUMBER() over (order by dailydate desc)
as rn from DAILYREG
                                 where isin = @isin) as cte
                            where rn = 2)
    if @current_closingval < @last_closingval
        begin
            set @min = @current_closingval
            set @max = @last_closingval
        end
    else
        begin
            set @min = @last_closingval
            set @max = @current_closingval
        end
    return CONVERT(decimal(5,2), ((@max - @min)/@max) * 100)
end
go



create trigger dbo.EXTTRIPLE_trigger
    on dbo.EXTTRIPLE
    after insert
    as
begin
    declare @id char(12),
        @date datetime
    select @id = id, @date = datetime from inserted
    declare  @code int
    set @code = (select mrktcode from INSTRUMENT where isin = @id)
    exec dbo.p_actualizaValorDiario @id, @date
    select * from dbo.FundamentalDataTable(@id, (select distinct
convert(date, @date)))
    exec dbo.Dailymarketupdate @code,@date
end
go


--d)
exec update_client @ncc = 1111112, @nif = 111111111, @name = 'Samsung'
exec update_client @ncc = 3333333, @nif = 333333333, @name =  'Client3'
exec remove_client @nif = 222222222

--e)
exec update_market @description = 'This is the new description', @name =
'IphoneMarket', @code = '111'
exec update_market @description = 'Description3', @name = 'Market3', @code =
'333'
exec remove_market @code = 222

--f)
exec p_actualizaValorDiario @id = 111111111111, @date ='2019-11-01 13:13:13'
```

```
exec p_actualizaValorDiario @id = 222222222222, @date = '2019-10-02 13:13:13'
exec p_actualizaValorDiario @id = 444444444444, @date= '2019-12-14 12:55:08'
--g)
select dbo.Average(180, 111111111111)

--h)
select * from dbo.FundamentalDataTable(111111111111, '2019-11-01')

--i)
exec createPortfolio @nif = 555555555

--j)
exec UpdateTotalVal @name = '555555555_portfolio', @quantity = 50000, @isin =
111111111111

--k)
select * from Portfolio_List('111111111_portfolio')

--l)
select * from PORTFOLIO_SUMMARY
```