Travel Agency

Base de dados de uma agência de viagens

42532 Bases de Dados
Relatório do trabalho prático final

Universidade de Aveiro,
12 de junho de 2020

Docentes:
Prof. Carlos Costa,
Prof. Joaquim Sousa Pinto,
Prof. Sérgio Matos

Trabalho realizado por:
André Alves 88811
Márcia Pires 88747
P1, G9

Introdução

No âmbito da cadeira de Bases de Dados, do Mestrado Integrado em Engenharia de Computadores e Telemática, este relatório tem como objetivo descrever detalhadamente o trabalho prático final relativo ao tema proposto: **Agência de Viagens**. Este foi realizado pelo grupo 9 da turma prática 1.

O código relativo à interface encontra-se na pasta **Interface.**

De forma a poder ser testada a nossa aplicação, o nosso utilizador da base de dados deve ser alterado na função **getSGBDConnection()** em todos os Forms da interface (Form1.cs, Form2.cs, Form3.cs, Form4.cs, Form5.cs, Form6.cs, Form7.cs, Form8.cs, add_flight.cs, add_transfer.cs, edit_transfer.cs, mod_flight.cs, new_CC.cs).

As demos (apresentação na aula prática + alterações posteriores) está na pasta **Demos.**

Os scripts pedidos encontram-se na pasta **Scripts**.

Análise de Requisitos

A criação desta base de dados tem como principal objetivo permitir a um funcionário de uma agência de viagens, gerir reservas da mesma. Através dela, este será capaz, primeiramente, de inserir um novo cliente. Segundo as preferências do cliente, o funcionário poderá personalizar um pacote, selecionando a duração, data de início e fim da viagem, número de pessoas e ainda:

- consultar e selecionar um alojamento;
- consultar e selecionar um ou mais voos, tendo a opção de escolher a classe em que deseja ir no avião;
- consultar e selecionar um transfer, se desejado;
- e ainda consultar e selecionar uma promoção que esteja ativa no momento da personalização do pacote.

Após o pacote ser finalizado, será apresentada a proposta final, incluindo o valor final dependente de todas as escolhas feitas anteriormente. Nesse momento, o cliente tem a opção de fazer a reserva do pacote e, caso a faça, o funcionário pode registá-la na base de dados. A partir do momento que a mesma é paga, a reserva passa a estar concluída e o funcionário pode arquivá-la e consultá-la. O cliente tem ainda a oportunidade de deixar uma review a um pacote que tenha comprado.

Além do funcionário poder criar pacotes e fazer customização dos mesmos, ele também é capaz de gerir dados independentes da escolha do cliente, isto é:

- adicionar novos alojamentos;
- adicionar / editar / remover voos;
- adicionar / editar / remover transfers;
- adicionar / editar promoções;
- adicionar novas cidades/países;
- adicionar / editar / remover clientes.

Os dados, especialmente aqueles relativos aos clientes, alojamento, voos, transfers e promos terão dados previamente disponíveis na base de dados, no entanto, também poderão ser alterados de forma a manter o sistema de gestão da agência de viagens atualizado. É importante realçar que esta análise foi sofrendo de alterações ao longo do desenvolvimento do trabalho, de forma a suprimir novas necessidades/ambições.

Entidade

- Uma **Person** é identificada por um primeiro nome, último nome, e-mail único e número de telemóvel.
- Um **Agent** é identificado por um ID único e uma password e é uma Person. Este é o principal utilizador do sistema.
- Um **Customer** é identificado por um ID único e um NIF e é uma Person. Este é o principal beneficiário do sistema.
- Um **Package** é identificado por um ID único, título, descrição do pacote, duração, data de início, data de término, número de pessoas, preço total. Cada pacote é o resultado de diversas escolhas relativas a uma viagem.
- Uma Promo pode ser aplicada num Package, se estiver ativa.
- Um **Flight** é identificado por um ID único, hora de partida, hora de chegada, companhia aérea, tipo de classe e preço. Entidade com as informações relativas a um voo.
- Um Package tem um Flight de partida e um Flight de chegada.
- Uma **Accommodation** é identificada por um ID único, nome, descrição do alojamento, imagem do alojamento e preço. Entidade com as informações relativas a um alojamento.
- Um Package tem uma Accommodation.
- Um **Transfer** é identificado por um ID único, companhia e preço. Entidade com as informações relativas a um transfer.
- Um Package pode ter ou não ter um transfer.
- Um **City & Country** é identificado por uma cidade única e um país.
- Um **Booking** é identificado por um ID único, um indicador de que a reserva foi paga, data de reserva e detalhes. Cada reserva reúne as informações relevantes para que um pacote possa ser comprado.
- Um Agent pode registar um ou mais Bookings.
- Um Customer pode reservar um ou mais Bookings.

- Uma **Promo** é identificada por um ID único, um indicador se a promoção está ativa e o valor do desconto. Informação acerca do tipo e valor do desconto que pode ser aplicado num pacote.
- Uma **Review** é uma entidade fraca de Package identificada por um ID, uma descrição e uma pontuação.
- Um User pode escrever uma Review.
- Uma **Airline** é uma entidade de Flight identificada pelo número ICAO (International Civil Aviation Organization) e o Nome.

DER - Diagrama Entidade Relação

Em anexo.

ER - Esquema Relacional

Em anexo.

SQL - Data Defining Language (DDL)

Utilizando a linguagem SQL Data Defining Language foi-nos permitido definir várias entidades da base de dados e também especificar a informação acerca de cada relação: seja o esquema de cada relação, restrições de integridade, domínio de valores associados com cada atributo, etc.

Todas as tabelas foram criadas segundo o *SCHEMA* ***TravelAgency***.

Como exemplos ilustrativos temos a entidade dos **Agent** que necessitam de um login para iniciar sessão. Uma vez que não é boa prática guardar informações confidenciais em texto simples diretamente na base de dados, optámos por guardar a palavra-passe do Agent em *Hash*, acrescentando-lhe ainda *Salt* de forma a dificultar possíveis ataques.

```
CREATE TABLE [TravelAgency].[Agent](
    [AgID] [int] NOT NULL   identity(1,1),
    [Password] [varbinary](20),
    [Salt] CHAR(25),
    [Email] [varchar](60) NOT NULL,

PRIMARY KEY CLUSTERED
([AgID] ASC) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]) ON [PRIMARY]
GO

ALTER TABLE [TravelAgency].[Agent]
WITH CHECK ADD  CONSTRAINT [FK__Agent__Email__20CCCE1C] FOREIGN KEY([Email])
REFERENCES [TravelAgency].[Person] ([Email])
GO
ALTER TABLE [TravelAgency].[Agent] CHECK CONSTRAINT [FK__Agent__Email__20CCCE1C]
GO
```

Outro exemplo é a entidade **Accommodation** que tem uma imagem, guardada em *base64* num atributo varchar(max).

```
CREATE TABLE TravelAgency.Accommodation(
    ID          INT             NOT NULL    identity(1,1),
    Name        VARCHAR(40)     NOT NULL    unique,
    Image       VARCHAR(MAX),
    Description VARCHAR(500),
    Price       SMALLMONEY      NOT NULL,
    CC_Location VARCHAR(20)     NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(CC_Location) REFERENCES TravelAgency.CC(City),
);
```

SQL - Data Manipulation Language (DML)

A linguagem SQL Data Manipulation Language é uma grande base do nosso projeto, visto que a mesma permite realizar comandos de inserção, eliminação e atualização de dados, bem como efetuar tanto consultas simples quanto avançadas.

Seguem-se o exemplo de duas queries ilustrativas - inserção de dados numa tabela e uma projeção.

```
insert into CC (City, Country) values ('New York', 'United States');
insert into CC (City, Country) values ('Cancum', 'Mexico');
insert into CC (City, Country) values ('São Paulo', 'Brazil');
insert into CC (City, Country) values ('Buenos Aires', 'Argentina');
insert into CC (City, Country) values ('Bogotá', 'Colombia');
insert into CC (City, Country) values ('Marrakesh', 'Morocco');
insert into CC (City, Country) values ('Cairo', 'Egypt');
insert into CC (City, Country) values ('Johannesburg', 'South Africa');
insert into CC (City, Country) values ('Madrid', 'Spain');
insert into CC (City, Country) values ('Barcelona', 'Spain');
```

```
INSERT INTO @tempTable (Email, Fname, Lname, PhoneNO, CustID, NIF)
SELECT Person.Email, Person.Fname, Person.Lname, Person.phoneNo, CustID, NIF
FROM TravelAgency.Customer Join TravelAgency.Person
ON TravelAgency.Customer.Email = TravelAgency.Person.Email
WHERE Fname + ' ' + Lname like '%'+@fname + ' ' + @lname +'%'
```

Utilizamos maioritariamente SQL parametrizado, com apenas algumas pesquisas dinâmicas, de forma a tentar evitar *SQL Injection* ao máximo.

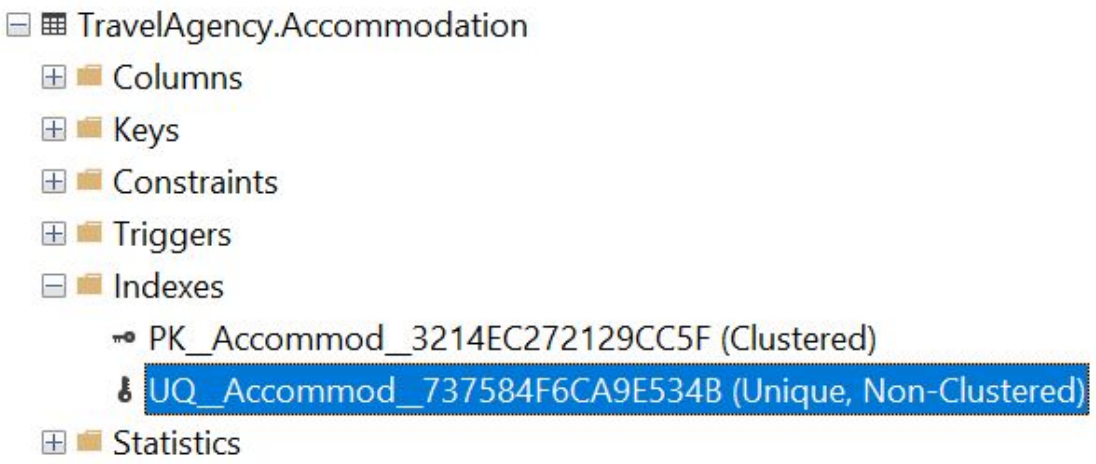
Normalização

Após uma análise às nossas tabelas, ao diagrama ER, e também a realização de vários testes, pudemos constatar que não tínhamos entidades não normalizadas. Assim, todas elas se encontram na 3ª Forma Normal e portanto podemos garantir uma menor redundância de dados, um aumento de integridade de dados e também de desempenho na nossa base de dados.

Índices e otimização

Relativamente a índices e otimização, depois de termos feito uma análise cuidada para perceber quais *queries* é que poderiam beneficiar da atribuição de índices e feito também alguns testes no SQL Server Profiler, concluímos que não precisávamos de mais nenhum índice. Isto porque, além do SQL gerar automaticamente clustered indexes para atributos com restrição *primary key,* em alguns casos, a necessidade de criação de um índice já tinha sido suprimida ao se definir um atributo como *unique* e *not null* - como acontece na entidade Accommodation, e a sua coluna Name - pois desta forma é também automaticamente gerado um *non-clustered index*.

```
CREATE UNIQUE INDEX NDX_SecurityAccounts_AccountName
    ON TravelAgency.Agent (AgID) INCLUDE (Salt, Password);
```



Stored Procedures

Foram utilizados cerca de **30 stored procedures (SP)**, com o intuito de criar um nível de abstração no acesso à escrita na base de dados, conferindo assim uma ainda maior segurança neste acesso e também uma maior eficiência em relação a *queries ad-hoc*. Assim sendo, estas foram utilizadas para operações do tipo: **adicionar / criar, editar, remover, carregar, pesquisar e filtrar**. Usamos também uma SP para o **login** do Agent.

De todas, salientaremos aquelas que consideramos terem particularidades que gostaríamos de ressaltar:

- **Create Agent:** nesta SP podemos encriptar a palavra-passe com ***Hash*** e ***Salt***, garantindo a segurança da mesma. Além disso, contém também uma ***transaction*** uma vez que queremos garantir que só queremos criar um Agent se pudermos garantir que este também é uma Person.
- **Add Package:** esta é a nossa maior SP pois é aqui que se cria um pacote e, portanto, é onde temos que garantir que quando criamos um pacote, adicionamos também as relações a ele adjacentes (accommodation, promo, flights, transfer). Com esta SP podemos fazer *insert* de values em até 3 entidades diferentes e dar *update* em outra. Por forma a também garantir a integridade da base de dados, utilizámos uma ***transaction*** para garantir que todas as operações ocorrem como esperado. Além disso, a criação deste pacote implica a execução de 3 ***triggers*** para que seja possível calcular o valor final do pacote.

- **Load Customer / Load Acc:** tanto uma SP quanto outra, permitem uma pesquisa paginada, reduzindo bastante o tempo de loading e a carga do sistema, sendo isto uma vantagem para o utilizador.
- **Delete Customer**: nesta SP, tal como o nome indica, serve para eliminarmos um Customer da nossa base de dados. Para tal, precisamos que o mesmo não só seja eliminado da entidade Customer, quanto da entidade Person e, ainda: eliminar o Booking efetuado pelo mesmo caso este ainda não tenha sido pago, ou, caso já tenha pago o Booking, alterar o ID do Customer para null, não eliminando assim esta entrada de forma a esta poder ser utilizada para efeitos estatísticos.
- **Filter Acc**: Através deste SP podemos aplicar filtros em mais do que um campo ao mesmo tempo - pesquisar pelo destino e ao mesmo tempo por preço ascendente ou descendente.

User Defined Functions

Quanto a **User Defined Functions (UDFs),** foram usadas cerca de **10**, para encapsular a base de dados, ou seja, para realizar pedidos de leitura à base de dados. Para tal, usamos 2 de 3 tipos possíveis de UDFS:

- Escalares;
- Inline table-valued.

Estamos foram então maioritariamente usadas para operações do tipo demonstração de histórico, cálculo de médias e *get* - funções que não requerem alterações na base de dados.

Gostaríamos de salientar então as seguintes:

- **Average Score:** uma **UDF escalar** com o objetivo de, dado o ID de um package, calcula a média do *score* das reviews do mesm e retornar esse valor, caso existam reviews.

```
--
ALTER FUNCTION [TravelAgency].[AverageScore] (@PackID INT) returns DECIMAL

AS
    BEGIN
            DECLARE @avgscore DECIMAL
            IF EXISTS (SELECT * FROM TravelAgency.Review WHERE Pack_ID = @PackID)
            BEGIN
                SELECT @avgscore = round(avg(cast(Score AS DECIMAL)), 1)
                FROM TravelAgency.Review
                WHERE Pack_ID = @PackID
            END

            ELSE
            BEGIN
                set @avgscore = -1
            END

            RETURN @avgscore

    END
```

- **Customer Historic:** uma **Inline table-valued UDF** cujo objetivo é, dado o ID de um Customer, retornar uma tabela com todos os *Bookings* efetuados por esse mesmo Customer.

```
ALTER FUNCTION [TravelAgency].[CustomerHistoric] (@CustID int) RETURNS TABLE

AS
    RETURN (SELECT * FROM TravelAgency.Booking
            WHERE TravelAgency.Booking.Cust_ID = @CustID)
```

Triggers

Na nossa base de dados decidimos definir três **triggers** do tipo **after insert,** sendo que o principal objetivo era calcular o valor do preço final de um package, após o preenchimento dos vários campos necessários para que este cálculo fosse possível de efetuar. Estes triggers são então despoletados aquando a criação de um package e, uma vez que a SP Add Package não só altera a entidade Package, como a entidade Contains_Flight e Contains_Transfer, é então necessário um trigger para cada uma destas entidades:

- Package

```
ALTER TRIGGER [TravelAgency].[insertPack]
   ON  [TravelAgency].[Package]
   AFTER INSERT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here
    DECLARE @acc_price SMALLMONEY
    DECLARE @duration INT

    SELECT @acc_price = Accommodation.Price, @duration = inserted.Duration
    FROM inserted JOIN TravelAgency.Accommodation ON inserted.Acomm_ID = Accommodation.ID

    UPDATE TravelAgency.Package
    SET totalPrice = @acc_price * @duration
    FROM inserted i
    WHERE Package.ID = i.ID

END
```

- Contains_Flight

```
ALTER TRIGGER [TravelAgency].[insertContFlight]
   ON  [TravelAgency].[Contains_Flight]
   AFTER INSERT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here
    DECLARE @flight_price SMALLMONEY

    SELECT @flight_price = Flight.Price
    FROM inserted JOIN TravelAgency.Flight ON inserted.Flight_ID = Flight.ID

    UPDATE TravelAgency.Package
    SET totalPrice = totalPrice + @flight_price
    FROM inserted i
    WHERE Package.ID = i.Pack_ID

END
```

- Contains_Transfer

```
ALTER TRIGGER [TravelAgency].[insertContTransfer]
   ON  [TravelAgency].[Contains_Transf]
   AFTER INSERT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here
    DECLARE @transf_price SMALLMONEY

    SELECT @transf_price = Transfer.Price
    FROM inserted JOIN TravelAgency.Transfer ON inserted.Transf_ID = Transfer.ID

    UPDATE TravelAgency.Package
    SET totalPrice = totalPrice + @transf_price
    FROM inserted i
    WHERE Package.ID = i.Pack_ID

END
```

Views

A utilização de views não foi uma recorrente ao longo do trabalho uma vez que preferimos optar pelo uso ou de stored procedures ou de user defined functions, que consideramos mais pertinentes à execução do bom trabalho da nossa base de dados. Ainda assim, temos um exemplo de uma view utilizada para apresentar os dados da entidade Accommodation.

```
ALTER VIEW [TravelAgency].[Accommodation_list]
AS
SELECT        TravelAgency.Accommodation.Image, TravelAgency.Accommodation.ID,
                TravelAgency.Accommodation.Name, TravelAgency.Accommodation.Price, TravelAgency.CC.City,
                TravelAgency.CC.Country,
                        TravelAgency.Accommodation.Description
FROM            TravelAgency.Accommodation INNER JOIN
                        TravelAgency.CC ON TravelAgency.Accommodation.CC_Location = TravelAgency.CC.City
GO
```

Notas finais

Gostaríamos de esclarecer algumas das decisões tomadas pelo grupo, antes e depois da apresentação realizada a 9 de junho de 2020.

Ao concluir este trabalho, podemos dizer que não utilizamos:

- **Cursores:** pois não nos deparamos com nenhuma situação em que fosse estritamente necessário o uso de cursores e, tendo uma vez a hipótese de utilizar álgebra relacional, fizemos preferência do uso desta última, dada a sua maior eficiência.

Após a realização da apresentação, ressaltamos as seguintes alterações efetuadas:

- Aplicação de três triggers;
- Utilização da entidade Review;
- Pequenas alterações a nível estético e aperfeiçoamento de algumas SP/UDFs.

Reforçando novamente o esforço adicional para:

- Inserção de imagens em *base64* na base de dados, e inserção através da interface, exigindo codificação e descodificação.
- Paginação de resultados nas *queries* que assim o exigiam.
- Encriptação de password em *Hash* e acrescento de *Salt*, de forma a dificultar ataques do tipo dicionário e brute-force.

Conclusão

Após a realização deste trabalho prático, podemos concluir que os nossos principais objetivos e ambições foram cumpridas: criar uma base de dados para uma agência de viagens, em que um funcionário desta pudesse ser capaz de criar e personalizar um pacote à medida das vontades do cliente. E que, além disso, fosse também capaz de manipular os dados internos à agência de viagens, ou seja, acrescentar, remover ou até editar dados como novos hotéis parceiros da agência, novos voos disponíveis, etc.

Além disso, podemos compreender que, apesar da realização dos guiões práticos da cadeira serem uma excelente maneira de assimilar os conteúdos dados, a realização de um projeto desta dimensão veio cimentar todos os conhecimentos prévios e desafiar-nos em busca de aprender mais.