Projecto Base de Dados:

Sistema de Booking de Reservas

Joao Fonseca Nmec:73779

Rui Dias nmec:72016

8 de Junho de 2016

Conteúdo

Listings

1 Introdução

1.1 Apresentação do Projecto

O nosso trabalho é um Sistema de Booking de Reservas de Viagens , ou seja um sistema para ser usado por um funcionário de uma loja de Viagens diariamente. Este Sistema pode ser acedido por todos os funcionários das várias lojas do grupo de Agências.

Tirando estes utilizadores apenas os administradores do sistema podem aceder ao sistema.

Neste Relatório é descrita a organização da Base de Dados , a utilização de Indices, Triggers , Udf e StoreProcedures, bem como uma listagem dos mesmos

2 Análise de Requisitos

Para perceber este traballho é necessário "colocar-se na pele do funcionário/admistrador"que vai ter de utilizar a apliacação todos os dias .

2.1 Funcionário

Num dia usual, o funcionário da Loja(ou Posto de Venda) tem de poder fazer uma nova reserva e , consultar as reservas feitas, "emitir"recibos e poder dizer a qualquer cliente quais as etapas dos itinerários disponiveis e a criação de estadias para essa reserva se o cliente o desejar.Além disso tem de efectuar a "inscrição"do cliente na base de dados.
Em outras ocasiões o funcionário necessitará de editar/apagar informação associada a um cliente, a uma reserva e a uma estadia associada, se existirem.

2.2 Admin

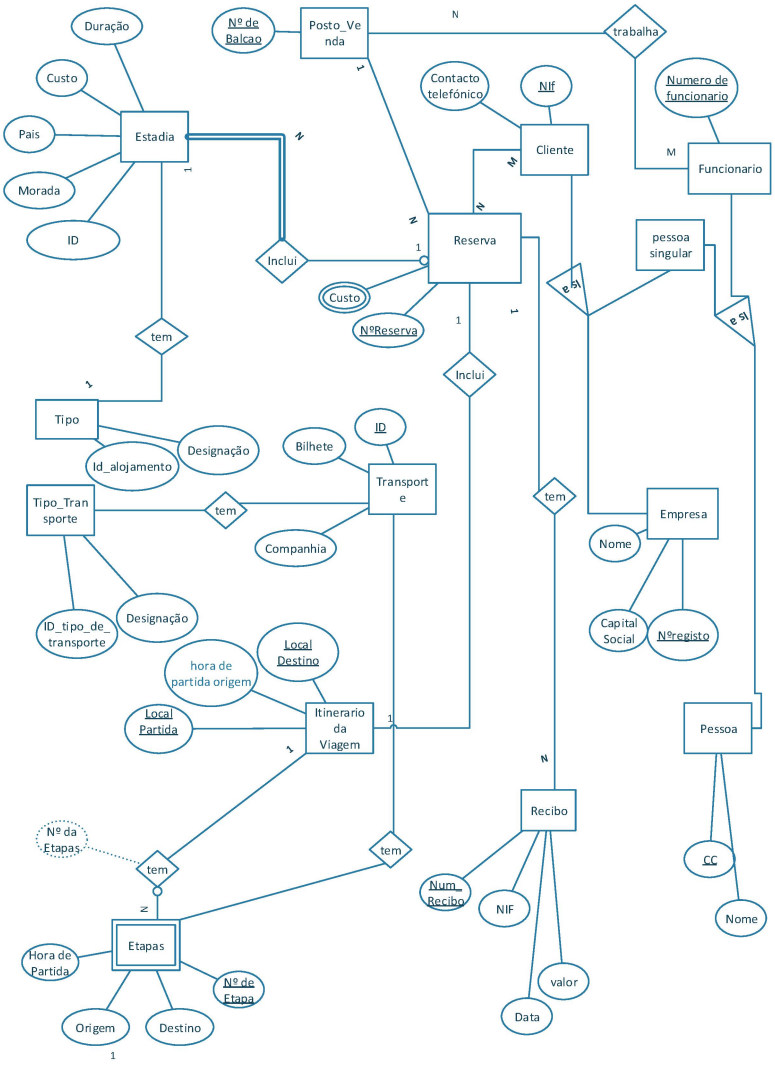
Se pensarmos agora num administrador do sistema , este deve ter permissões para aceder a qualquer tabela do Sistema, e alterar e eliminar o conteúdo das mesmas. Além disso o administrador do sistema tem funções especificas como:

- Criar Itinerários e as Etapas assocadas a esse Itinerário
- Criar Transporte associado a uma Etapa e o tipo
- Criar Postos de Venda e Funcionários

3 Diagrama Entidade Relação

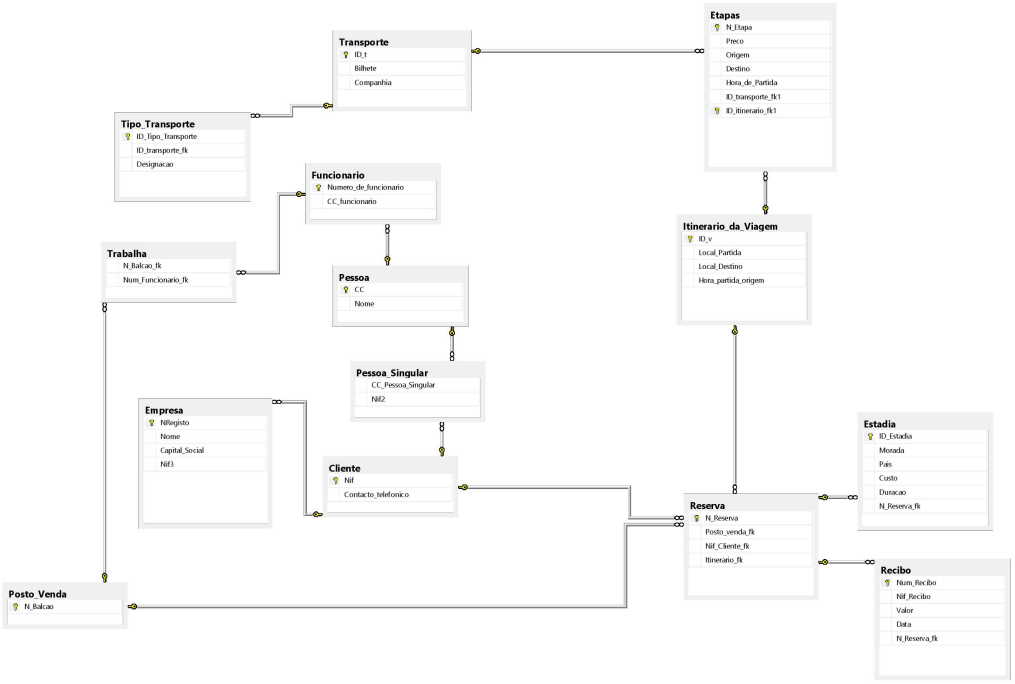
O diagrama Entidade Relação do Sistema utilizado é apresentado em seguida.

Em anexo é apresentado um pdf com uma melhor resolução da imagem.



4 Esquema Relacional

O Esquema Relacional da Base de Dados é apresentado em seguida. Em anexo é apresentado um pdf com uma melhor resolução da imagem.



1-1

5 Normalização

Pessoa

R={CC,Nome}

F={{CC}->{Nome}}

3FN={ CC, Nome}

Funcionário

R={Numero_de_Funcionario,CC_Fk}

F={ {Numero_de_Funcionario}->{CC_Fk}}

3FN={ Numero_de_Funcionario, CC_Fk}

Cliente

R={Nif,Contacto_Telefonico}

F={{Nif}->{Contacto_Telefonico}}

3FN={ Nif, Contacto_Telefonico}

Empresa

R={NRegisto,Nome,Capital_Social,Nif_Fk}

F={{CC}->{Nome,Capital_Social,Nif_Fk}}

3FN={ CC, Nome,,Capital_Social,Nif_Fk}

Pessoa_Singular

R={CC_Fk,Nif_Fk}

F={{CC_Fk,Nif_Fk}}

3FN={ CC_Fk,Nif_Fk}

Posto_Venda

R={N_Balcao}

F={{N_Balcao}}

3FN={ N_Balcao}

Trabalha

R={N_Balcao_Fk,Numero_de_Funcionario_Fk}

F={{N_Balcao_Fk,Numero_de_Funcionario_Fk}}

3FN={ N_Balcao_Fk,Numero_de_Funcionario_Fk}

Transporte

R={ID_t,Bilhete,Companhia}

F={{ID_t}->{Bilhete,Companhia}}

3FN={ ID_t, Bilhete,Companhia}

Tipo_Transporte

R={ID_t_Pk,ID_t_Fk,Companhia}

F={{ID_t_Pk,ID_t_Fk,Companhia}}

3FN={ ID_t_Pk,ID_t_Fk,Companhia}

Tipo_Transporte

R={ID_t_PK,ID_t_Fk,Companhia}

F={{ID_t_Pk,ID_t_Fk,Companhia}}

3FN={ ID_t_Pk,ID_t_Fk,Companhia}

Itinerario_da_Viagem

R={ID_v,Local_de_Partida,Local_de_Chegada,Hora_de_Partida_Origem}

F={{ID_v}->{Local_de_Partida,Local_de_Chegada,Hora_de_Partida_Origem}}

3FN={ ID_v,Local_de_Partida,Local_de_Chegada,Hora_de_Partida_Origem}

Etapa

R={N_Etapa,Preco,Origem,Destino,Hora_de_Partida,ID_t_Fk,ID_v_Fk}

F={{ID_t}->{Preco,Origem,Destino,Hora_de_Partida,ID_t_Fk,ID_v_Fk}}

3FN={ ID_t, Preco,Origem,Destino,Hora_de_Partida,ID_t_Fk,ID_v_Fk}

Reserva

R={N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}

F={{N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}}

3FN={ N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}

Reserva

R={N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}

F={{N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}}

3FN={ N_Reserva,Posto_Venda_Fk,Nif_Fk.ID_t_Fk}

Estadia

R={ID_Estadia,Moradia,País,Custo,Duracao,N_Reserva_fk}

F={{ID_Estadia}->{Moradia,País,Custo,Duracao,N_Reserva_fk}}

3FN={ ID_Estadia,Moradia,País,Custo,Duracao,N_Reserva_fk }

Recibo

R={Num_Recibo,Nif_Recibo,Origem,Valor,Data,N_Reserva_fk}

F={{Num_Recibo}->{Nif_Recibo,Origem,Valor,Data,N_Reserva_fk}}

3FN={ Num_Recibo,Nif_Recibo,Origem,Valor,Data,N_Reserva_fk}

6 SQL Data Definition Language

A estrutura base do Sistema é constituida por 14 tabelas que são fundamentais para perceber o sistema.

Listing 1: Create Table

```
CREATE TABLE Etapas (
        N_Etapa tinyint NOT NULL IDENTITY(1,1),
        Preco decimal(10, 2) NOT NULL,
        Origem varchar(64) NOT NULL,
        Destino varchar(64) NOT NULL,
        Hora_de_Partida DATE NOT NULL,
        ID_transporte_fk1 smallint NOT NULL,
        ID_itinerario_fk1 smallint NOT NULL,
       CONSTRAINT Pk_N_Etapa PRIMARY KEY (N_Etapa,ID_itinerario_fk1),
       CONSTRAINT FK_ID_Itinerario FOREIGN KEY (ID_itinerario_fk1)
            REFERENCES Itinerario_da_Viagem(ID_v)ON DELETE CASCADE,
       CONSTRAINT FK_ID_transporte_1 FOREIGN KEY (ID_transporte_fk1)
            REFERENCES Transporte(ID_t)ON DELETE CASCADE
);
```

Na tabela apresentada, pode-se vizualizar o código SQL que permite criar uma tabela. Nele se pode ver que os atributos "mais importantes"encontram-se identificados com o valor **Constraint**. Estas variaveis têem normalmente associadas dois atributos :

- Primary Keys
- Foreign Keys

As **Primary Keys** são os atributos que identificam uma tabela em especifico e são por isso um campo de preenchimento obrigatório.
As **Foreign Keys** são o par das Primary Keys e como tal funcionam como ponteiros para Primary Key de uma tabela em especifico.
Pode-se ainda verificar, o uso da propriedade **Identity** que permite que o utilizador não tenha de inserir o valor do número da **Etapa**.Sempre que quiser adicionar uma etapa, o número é adicionado de cada vez ao valor base 1, neste caso em concreto. Para finalizar é de salientar que todas os atributos não podem ser inicializados com NULL devido à condição de **NOT NULL** que devido a um qualquer erro as tabelas sejam preenchidas a NULL.
Em anexo seguem 2 ficheiros de DDL na pasta DB , com os nomes: 1_DropTables.sql e 2_CreateTables.sql

7 SQL Data Manipulation Language

7.1 Indices

Foi lecionado nas aulas teóricas que as chaves primárias (**Primary Keys**) de uma tabela têem sempre indices do tipo: **CLUSTERED UNIQUE** sendo elas de tipo simples ou compostos.
Assim aquando da criação dos indices associados às tabelas foi tomada em atenção esse fator e só foram criados indices **NONCLUSTERED** apenas quando necessários, para facilitar a pesquisa por nome de Pessoa/Empresa entre outras pesquisas.

Listing 2: Indices Nomes Clientes

```
CREATE NONCLUSTERED INDEX idxPessoa ON Pessoa(CC) INCLUDE (Nome);
CREATE NONCLUSTERED INDEX idxNomeEmpresa ON Empresa(NRegisto)
    INCLUDE (Nome);
```

Para além de indices para iterar sobre os nomes dos clientes foi criado um indice para iterar sobre os Locais de Partida das Viagens disponiveis, para minimizar o tempo de resposta da Base de Dados em relação a uma query sobre os Locais de Partida

Listing 3: Indice Local de Partida

```
CREATE NONCLUSTERED INDEX idxPartida ON Itinerario_da_Viagem(ID_V)
    INCLUDE(Local_Partida);
```

A utilização dos indices nestes casos não deu para verificar a melhoria de desempenho pois foram inseridos um número diminuto de valores . Os Indices do da Base de Dados encontram-se na pasta DB no ficheiro :3_Indices.sql

7.2 Triggers

Os triggers são micro-programas embutidos na base de dados que começam a funcionar assim que um evento ocorre. O nosso grupo decidiu a criação de apenas um trigger que é ativado assim que o Administrador tenta apagar um determinado cliente.

Listing 4: Trigger Delete Cliente

```
GO
DROP TRIGGER DeleteCliente;
GO
CREATE TRIGGER DeleteCliente ON Cliente
INSTEAD OF DELETE
AS
BEGIN
        BEGIN TRANSACTION tranDelete
        BEGIN TRY
                DECLARE @Nif AS INT;
                SELECT @Nif=Nif FROM deleted;

                DELETE FROM Pessoa_Singular WHERE Nif2=@Nif;
                DELETE FROM Empresa WHERE Nif3=@Nif;
                DELETE FROM Reserva WHERE Nif_Cliente_fk=@Nif;

                DELETE FROM Cliente WHERE Nif=@Nif;
                COMMIT TRANSACTION tranDelete;
        END TRY
        BEGIN CATCH
                ROLLBACK TRANSACTION;
                THROW;
        END CATCH
END
```

O trigger utilizado para processo de **DELETE** do Cliente, antes do efetuar tenta apagar todas linhas das tabelas que dependam do Cliente e que contenham um NIF igual ao NIF do Cliente a apagar. Se estas forem apagadas o Cliente é apagado senão, o Cliente continua na Tabela

7.3 User defined Functions

Uma UDF são uma funções que o database owner providencia na base de dados de forma que sejam incorporadas no sistema e usadas como funções de retorno de valores. Estes podem ser de 2 tipos:

- Table Valued
- Scalar-Valued

Um Exemplo de função table Valued é apresentada de seguida (Listing 5). A função utiliza um LEFT JOIN para juntar as tabelas(Etapas,Transporte,Tipo_Transporte) e dado Um Local de Partida(previamente definido pelo administrador), Uma Etapa da Viagem e uma Data retorna o Transporte ou transportes utilizado(s) na Etapas e identifica a Companhia(s) da qual este(s) faz(em) parte.

Listing 5: Udf Table-Valued

```
DROP FUNCTION udf_getTransporteporLocaiseHora;
GO
CREATE FUNCTION udf_getTransporteporLocaiseHora(@LocalPartida varchar(40),
@LocalDestino varchar(40),@Hora DATE) RETURNS TABLE AS

RETURN (SELECT Designacao,Companhia FROM Itinerario_da_Viagem
    LEFT JOIN Etapas ON ID_itinerario_fk1=ID_v
        LEFT JOIN Transporte ON ID_transporte_fk1=ID_t
        LEFT JOIN Tipo_Transporte ON ID_transporte_fk=ID_transporte_fk1
        WHERE Hora_partida_origem=@Hora AND Local_Partida=@LocalPartida
        AND Local_Destino=@LocalDestino);
```

Por outro lado as Scalar-Valued User Defined Functions apenas retornam um valor básico : INT , FLOAT ,DATE , ETC;

Na página em baixo é apresentada uma Scalar Valued Udf(Listing 6) que retorna o valor a pagar por um Cliente que tenha feito uma reserva, pedindo ao funcionário apenas o número de reserva do Cliente.

Listing 6: Udf Scalar-Valued

```
DROP FUNCTION udf_getPrecoReservasemEstadia ;
GO
CREATE FUNCTION udf_getPrecoReservasemEstadia (@NReserva int) RETURNS INT AS
BEGIN
DECLARE @Preco INT;
        SELECT @Preco=SUM(Preco) FROM Etapas
        INNER JOIN Itinerario_da_Viagem ON
        Etapas.ID_itinerario_fk1=Itinerario_da_Viagem.ID_v
        INNER JOIN Reserva ON Itinerario_da_Viagem.ID_v=Reserva.Itinerario_fk
        WHERE N_Reserva=@NReserva
        RETURN @Preco;
END
```

7.4 Storage Procedures

Os Storages Procedures são utilizados fundamentalmente por manterem a integridade de dados aquando da utilização dos mesmos sobre a Base de Dados. Existem vários tipos de Stored Procedures(Sp's) , que permitem a um user que tenha acesso a "trabalhar sobre a base de Dados", sendo que normalmente são só 3:

- Sp's de Inserção
- Sp's de Update
- Sp's de Delete

No que toca aos Sp's de Inserção , normalmente têem argumentos e retornam um erro caso o valor não seja inserido na Base de Dados . Em seguida é apresentado um Sp de Inserção:

Listing 7: SP de Insert

```
GO
DROP PROCEDURE sp_insertPessoaCliente
GO
CREATE PROCEDURE sp_insertPessoaCliente
        @CC int,
        @nome varchar(40),
        @Nif int,
        @Contacto int
WITH EXECUTE AS OWNER AS
BEGIN
        IF EXISTS(SELECT CC FROM Pessoa WHERE CC=@CC)
                RAISERROR('O Cidadao ja existe',16,1);
        IF EXISTS(SELECT Nif FROM Cliente WHERE Nif=@Nif)
                RAISERROR('O Cliente em questao ja existe existe',16,1);
        ELSE
                INSERT INTO Pessoa(CC,Nome) VALUES (@CC,@nome)
                INSERT INTO      Cliente (Nif,Contacto_telefonico) VALUES (@Nif,
                INSERT INTO Pessoa_Singular(CC_Pessoa_Singular,Nif2) VALUES (@C
END
```

Falando agora de Storage Procedures de Update, estes SP's servem para atua-

lizar certas células da Base de Dados. Acontece frequentemente um funcionário enganar-se no nome de uma pessoa , e necessitar de o alterar:

Listing 8: SP de Update

```
GO
DROP PROCEDURE sp_updatePessoa
GO
CREATE PROCEDURE sp_updatePessoa
        @cc_pessoa int,
        @Nome varchar(40)
WITH EXECUTE AS OWNER AS
BEGIN
        IF NOT EXISTS (SELECT CC FROM Pessoa WHERE CC=@cc_pessoa)
                RAISERROR('A pessoa em questao nao existe',16,1)

        ELSE
                UPDATE Pessoa SET Nome=@Nome WHERE CC=@cc_pessoa
END
```

Para finalizar os Stored Procedures temos os de falar nos Sp's de Delete que servem para apagar elementos da Base de Dados , se existirem. O caso mais comum é o Cliente desistir ou da reserva ou da estadia e ser necessário removê-lo da Base de Dados:

Listing 9: SP de Delete

```
DROP PROCEDURE sp_deleteReserva
GO
CREATE PROCEDURE sp_deleteReserva
        @NReserva int
WITH EXECUTE AS OWNER AS
BEGIN
        IF NOT EXISTS (SELECT N_Reserva FROM Reserva
        WHERE N_Reserva=@NReserva)
                RAISERROR('A_Reserva_nao_existe',16,1);
        ELSE
                DELETE FROM Reserva WHERE N_Reserva=@NReserva;
END
```

8 Security

A nivel da Segurança da Base de dados foram criados 2 tipos de utilizador:

- Random_owner (utilização por parte do administradores)
- Random_func (utilização por parte dos Funcionários)

As permissões associadas ao login com o utilizador random_owner são as de um security admin normal.

Listing 10: AddUser

```
EXECUTE sp_droplogin 'func';
GO
EXECUTE sp_addlogin 'func', 'basededados2','Sistema','English'
GO
DROP USER random_func;
GO
CREATE USER random_func FOR LOGIN func;
GO
```

No caso do funcionário foi criado um "ROLE"especifico com permissões de EXECUTE, SELECT E INSERT :

Listing 11: Add Role

```
EXECUTE sp_droprole 'MyRole2';
GO
EXECUTE sp_addrole 'MyRole2'
GO
GRANT EXECUTE,SELECT,INSERT TO MyRole2;
```

Por fim falta apenas adicionar o Perfil MyRole2 ao user dos funcionários:

Listing 12: Add Role

```
GO
EXECUTE sp_droprolemember 'MyRole2', 'random_func'
GO
EXECUTE sp_addrolemember 'MyRole2', 'random_func'
GO
```

9 Integração da Base de dados com a Aplicação

10 Conclusão

No final deste trabalho percebemos melhoramos a nossa aptidão cim as ferramentas que o **SQL SERVER** nos disponibiliza . Além disso retivemos o *Know How* da experência de aplicação de conhecimentos como Indices,Udf's, Triggers , Stored Procedures,Etc.

No trabalho de concepção da Base de Dados tentamos minimizar falhas de segurança e além disso tentamos que fosse dificil a inserção incorreta de dados.