Sistema de Gestão de Rede Ferroviária

- UC 42532: Base de Dados
- L. Engenharia Informática 18/19
- Trabalho Prático Final

- Grupo: P6G10
- 88886: Tiago Mendes
- 89296: Tomás Batista

universidade de aveiro

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

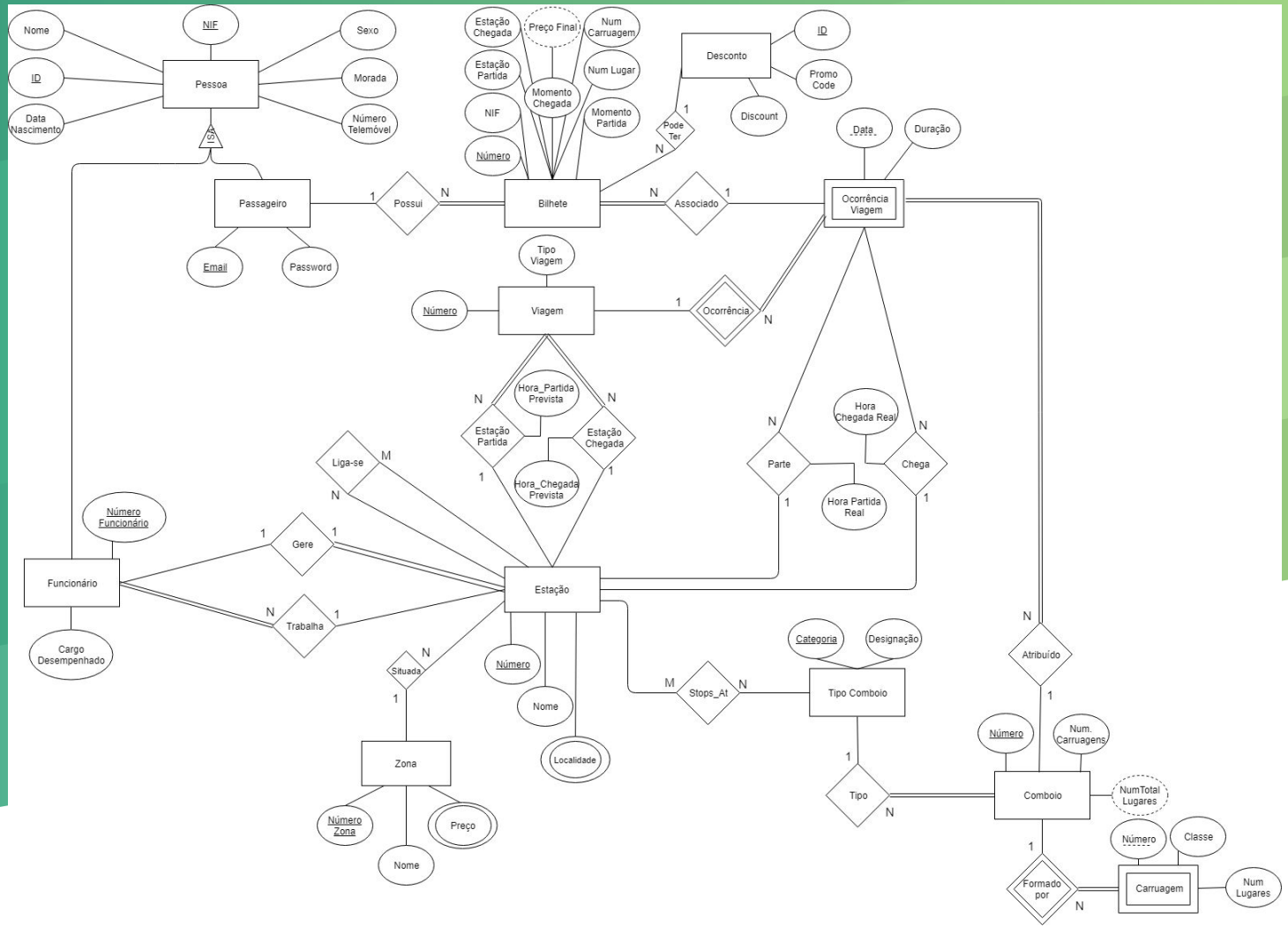
1. Tema

1. Tema

- Sistema de gestão de uma rede ferroviária
- Particular atenção para o desenho e desenvolvimento da camada da base de dados
- O principal objetivo deste sistema é permitir a que utilizadores registados no mesmo possam realizar um conjunto útil de operações, como compra de bilhetes, por exemplo.

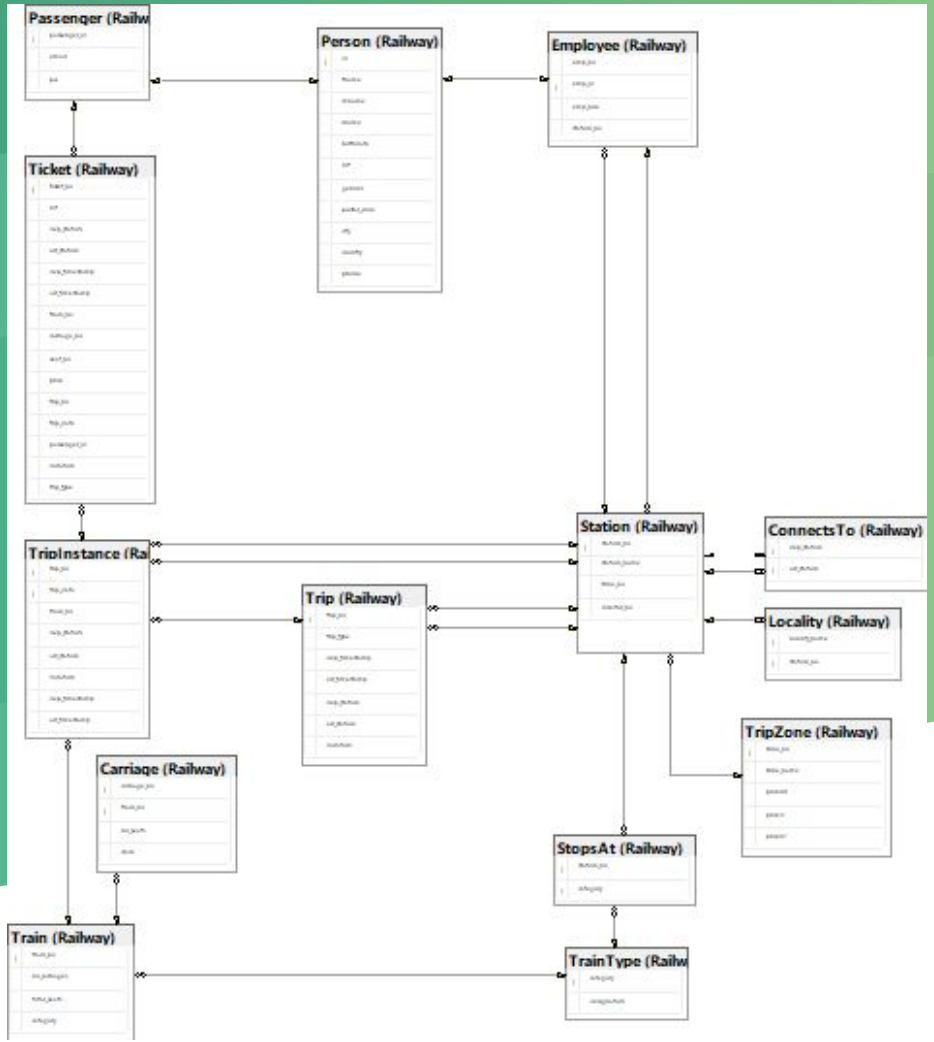
2.
Desenho Conceptual

DER


Nome
NIF
Sexo
ID
Pessoa
Morada
Data Nascimento
Número Telemóvel
IS
Passageiro
Email
Password
Possui
Bilhete
Estação Chegada
Preço Final
Num Carruagem
Estação Partida
Num Lugar
Momento Chegada
NIF
Momento Partida
Número
Desconto
ID
Promo Code
Discount
Pode Ter
Associado
Data
Duração
Ocorrência Viagem
Tipo Viagem
Número
Viagem
Ocorrência
Hora_Partida Prevista
Estação Partida
Estação Chegada
Hora_Chegada Prevista
Hora Chegada Real
Parte
Chega
Hora Partida Real
Liga-se
Número Funcionário
Gere
Funcionário
Trabalha
Cargo Desempenhado
Estação
Situada
Zona
Número Zona
Preço
Nome
Número
Nome
Localidade
Stops_At
Categoria
Designação
Tipo Comboio
Atribuído
Tipo
Número
Num. Carruagens
Comboio
NumTotal Lugares
Número
Classe
Formado por
Carruagem
Num Lugares

3.
Modelo relacional

ER


Passenger (Railw
Person (Railway)
Employee (Railway)
Ticket (Railway)
TripInstance (Ra
Station (Railway)
ConnectsTo (Railway)
Trip (Railway)
Locality (Railway)
TripZone (Railway)
Carriage (Railway)
StopsAt (Railway)
Train (Railway)
TrainType (Railw

4.
SQL - DDL

4. SQL - DDL

- Criação de tabelas
- Imposição de restrições de integridade sobre os dados

prints.sql

```sql
USE p6g10;
GO

/* Create Railway database tables */

CREATE TABLE Railway.Person (
    id              INT             IDENTITY(100000001,1)   NOT NULL,
    fname           VARCHAR(30)     NOT NULL,
    mname           VARCHAR(30),
    lname           VARCHAR(30)     NOT NULL,
    birthdate       DATE            NOT NULL,
    nif             INT             NOT NULL    CHECK(nif >= 100000000 AND nif <= 999999999),
    gender          CHAR            NOT NULL    CHECK(gender = 'M' OR gender = 'F'),
    postal_code     VARCHAR(50)     NOT NULL,
    city            VARCHAR(30)     NOT NULL,
    country         VARCHAR(30)     NOT NULL,
    phone           INT                         CHECK(phone >= 900000000 AND phone <= 999999999),
    PRIMARY KEY(id),
    UNIQUE(nif)
);

CREATE TABLE Railway.Passenger (
    passenger_id    INT             NOT NULL    CHECK(passenger_id > 100000000),
    email           VARCHAR(50)     NOT NULL    CHECK(email LIKE '%@%' AND email LIKE '%.%'),
    pw              VARCHAR(50)     NOT NULL    CHECK(LEN(pw) > 0),
    PRIMARY KEY(passenger_id),
    UNIQUE(email)
);
```

prints.sql

prints.sql

```sql
    zone_name           VARCHAR(30)             NOT NULL,
    priceUR             SMALLMONEY              NOT NULL    CHECK(priceUR >= 0),
    priceIC             SMALLMONEY              NOT NULL    CHECK(priceIC >= 0),
    priceAP             SMALLMONEY              NOT NULL    CHECK(priceAP >= 0),
    PRIMARY KEY (zone_no),
    UNIQUE(zone_name)
);

/* Add foreign keys to the tables */

ALTER TABLE Railway.Passenger
ADD CONSTRAINT c1
FOREIGN KEY (passenger_id) REFERENCES Railway.Person(id);

ALTER TABLE Railway.Employee
ADD CONSTRAINT c2
FOREIGN KEY (emp_id) REFERENCES Railway.Person(id);

ALTER TABLE Railway.Employee
ADD CONSTRAINT c3
FOREIGN KEY (station_no) REFERENCES Railway.Station(station_no);

ALTER TABLE Railway.Ticket
ADD CONSTRAINT c4
FOREIGN KEY (passenger_id) REFERENCES Railway.Passenger(passenger_id);
```

5.
SQL - DML

5. SQL - DML

- Utilização de diversas instruções SQL - DML em diferentes contextos do sistema

prints.sql

```
2   INSERT INTO Railway.Trip VALUES (@trip_type,@dep_timestamp,@arr_timestamp,@dep_station,@arr_station,@duration);
3
4   SELECT DISTINCT
5           @dep_zone = dep.zone_no,
6           @arr_zone = arr.zone_no
7           FROM Railway.Trip AS trip
8           JOIN Railway.Station AS dep ON trip.dep_station = dep.station_no
9           JOIN Railway.Station AS arr ON trip.arr_station = arr.station_no
10  WHERE trip.trip_type = @trip_type AND dep.station_name = @dep_station AND arr.station_name = @arr_station;
```

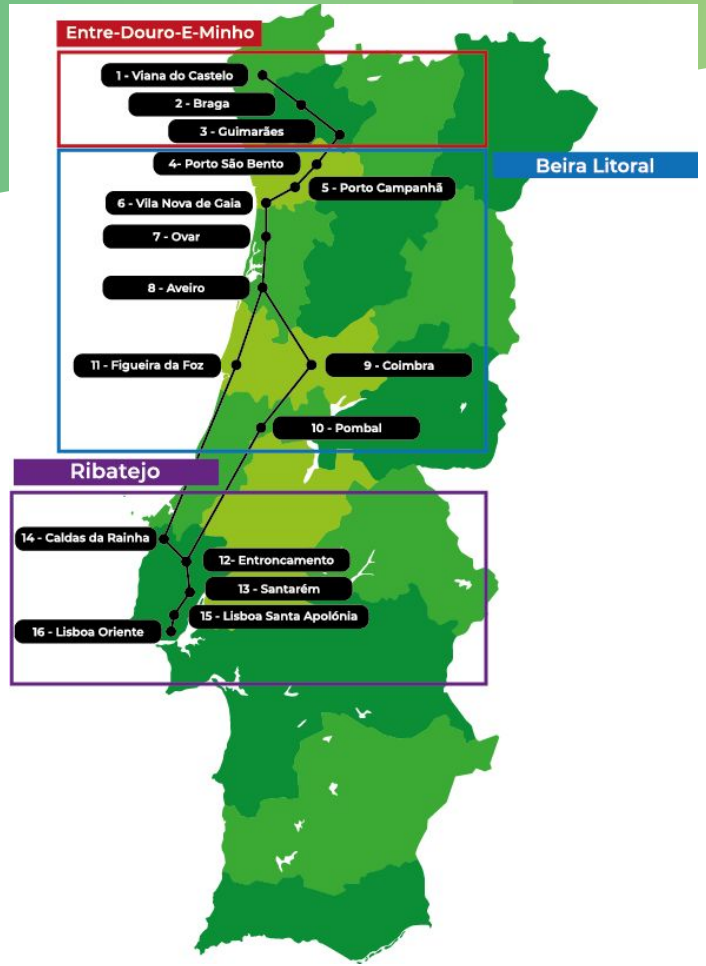
6.
População da BD

6. População da BD

- Preenchimento das tabelas da base de dados, utilizando tuplos escolhidos com critério
- Scripts de geração dos tuplos desenvolvidos em Python

Critérios escolhidos

- 100 empregados e 100 passageiros
- 16 estações ao longo de 3 regiões
- 35 comboios com diferente número de carruagens
- 4 tipos de comboio
- Diferentes preços por zona
- 1470 viagens diferentes



prints.sql

```sql
INSERT INTO Railway.Trip VALUES ('IC','15:10','23:52',15,2,NULL);
INSERT INTO Railway.TrainType VALUES ('UR','Urbano/Regional');
INSERT INTO Railway.TrainType VALUES ('IC','Intercidades');
INSERT INTO Railway.TrainType VALUES ('M','Mercadorias');
INSERT INTO Railway.TrainType VALUES ('AP','Alfa-pendular');

INSERT INTO Railway.Train VALUES (5, 286, 'UR');
INSERT INTO Railway.Train VALUES (4, 196, 'UR');
INSERT INTO Railway.Train VALUES (7, 375, 'UR');

INSERT INTO Railway.Carriage VALUES (1, 1, 32, 'C');
INSERT INTO Railway.Carriage VALUES (2, 1, 68, 'E');
INSERT INTO Railway.Carriage VALUES (3, 1, 59, 'E');
INSERT INTO Railway.Carriage VALUES (4, 1, 67, 'E');

INSERT INTO Railway.Person VALUES ('Daenerys',NULL,'Rocha','1971-06-23',100000117,'F','8540','Faro','Portugal',900000117);
INSERT INTO Railway.Person VALUES ('Beatriz',NULL,'Lopes','1979-02-27',100000118,'F','3945','Aveiro','Portugal',900000118);
INSERT INTO Railway.Person VALUES ('Matilde',NULL,'Barros','1952-03-24',100000119,'F','9147','Madeira','Portugal',900000119);


INSERT INTO Railway.TripZone VALUES ('Entre-Douro-e-Minho',3,4,5);
INSERT INTO Railway.TripZone VALUES ('Beira Litoral',4.5,5.5,6.5);
INSERT INTO Railway.TripZone VALUES ('Ribatejo',3.5,4.5,5.5);

INSERT INTO Railway.ConnectsTo VALUES (1,2);
INSERT INTO Railway.ConnectsTo VALUES (2,1);
INSERT INTO Railway.ConnectsTo VALUES (2,3);
INSERT INTO Railway.ConnectsTo VALUES (3,2);

INSERT INTO Railway.Trip VALUES ('IC','16:46','22:22',14,3,NULL);
INSERT INTO Railway.Trip VALUES ('IC','16:46','23:52',14,2,NULL);
INSERT INTO Railway.Trip VALUES ('IC','16:46','00:37',14,1,NULL);
INSERT INTO Railway.Trip VALUES ('IC','18:46','19:56',11,8,NULL);
```

7.
SQL - View

7. SQL - View

- A utilização de views foi pequena, devido às vantagens oferecidas pelas stored procedures e user defined functions.

prints.sql

```sql
CREATE VIEW Railway.TicketBasicInfo AS
SELECT TOP 1 dep_sta.station_name AS dep_station, arr_sta.station_name AS arr_station, T.dep_timestamp, T.arr_timestamp, T.trip_date, T.passenger
FROM Railway.Ticket AS T JOIN Railway.Station AS dep_sta on T.dep_station = dep_sta.station_no
JOIN Railway.Station AS arr_sta on T.arr_station = arr_sta.station_no
ORDER BY T.trip_date;
```

8. Normalização

8. Normalização

- À data desta apresentação, ainda não foi realizada a análise de qualidade do desenho desta base de dados através de processos de normalização. No entanto, irá ser incluída no relatório final.

9.
Índices e Optimização

9. Índices e Optimização

- Foram utilizados clustered indexes para atributos de chave primária pelas diferentes tabelas da base de dados, através da propriedade IDENTITY

prints.sql

```sql
CREATE TABLE Railway.Ticket (
    ticket_no           INT         IDENTITY(100,1) NOT NULL,
    nif                 INT         NOT NULL    CHECK(nif >= 100000000 AND nif < 999999999),
    dep_station         INT         NOT NULL    CHECK(dep_station > 0),
    arr_station         INT         NOT NULL    CHECK(arr_station > 0),
    dep_timestamp       TIME        NOT NULL,
    arr_timestamp       TIME        NOT NULL,
    train_no            INT         NOT NULL    CHECK(train_no > 0),
    carriage_no         INT         NOT NULL    CHECK(carriage_no > 0),
    seat_no             INT         NOT NULL    CHECK(seat_no > 0),
    price               SMALLMONEY  NOT NULL    CHECK(price > 0),
    trip_no             INT         NOT NULL    CHECK(trip_no > 0),
    trip_date           DATE        NOT NULL,
    passenger_id        INT         NOT NULL    CHECK(passenger_id > 0),
    PRIMARY KEY(ticket_no)
);

CREATE TABLE Railway.Trip (
    trip_no             INT             IDENTITY(1,1)   NOT NULL    CHECK(trip_no > 0),
    trip_type           VARCHAR(2)      NOT NULL,
    dep_timestamp       TIME            NOT NULL,
    arr_timestamp       TIME            NOT NULL,
    dep_station         INT             NOT NULL    CHECK(dep_station > 0),
    arr_station         INT             NOT NULL    CHECK(arr_station > 0),
    duration            TIME,
    PRIMARY KEY(trip_no)
);
```

10.
SQL Programming

10.1
Stored Procedures

10.1 Stored procedures

- Foram utilizados cerca de 22 stored procedures, de modo a consumir os dados a partir da aplicação do cliente de uma forma mais segura e eficiente.

prints.sql

```sql
ALTER PROC Railway.pr_sign_up(
    @fname          VARCHAR(30),
    @lname          VARCHAR(30),
    @birthdate      DATE,
    @nif            INT,
    @gender         CHAR,
    @postal_code    VARCHAR(50),
    @city           VARCHAR(30),
    @country        VARCHAR(30),
    @phone          INT,
    @email VARCHAR(50), @pw VARCHAR(50))
AS
    INSERT INTO Railway.Person VALUES (@fname, NULL, @lname, @birthdate, @nif, @gender, @postal_code, @city, @country, @phone);
    DECLARE @passenger_id   AS INT;
    SELECT @passenger_id = p.id FROM Railway.Person As p WHERE p.nif = @nif;
    INSERT INTO Railway.Passenger VALUES (@passenger_id, @email, HASHBYTES('SHA1', @pw));
GO

ALTER PROC Railway.pr_get_stations
AS
    SELECT station_name FROM Railway.Station;
GO

ALTER PROC Railway.pr_insert_image(@passenger_id INT, @img_base64 VARCHAR(MAX))
AS
    IF EXISTS(SELECT * FROM Railway.ProfilePictures WHERE passenger_id = @passenger_id)
        UPDATE Railway.ProfilePictures SET img_base64 = @img_base64 WHERE passenger_id = @passenger_id;
    ELSE
        INSERT INTO Railway.ProfilePictures VALUES (@img_base64, @passenger_id);
GO
```

10.2
User Defined Functions

10.2 User Defined Functions

- Foram utilizadas cerca de 10 user defined functions de diferentes tipos, nomeadamente escalares e inline table-valued.

prints.sql

```sql
ALTER FUNCTION Railway.f_check_password (@email VARCHAR(50), @password VARCHAR(50)) RETURNS INT
AS
BEGIN
    IF EXISTS(SELECT * FROM Railway.Passenger AS p WHERE p.email = @email AND p.pw = HASHBYTES('SHA1',@password))
        RETURN 1;
    RETURN 0;
END
GO

ALTER FUNCTION Railway.f_return_login (@email VARCHAR(50), @password VARCHAR(50)) RETURNS Table
AS
    RETURN (SELECT * FROM Railway.Person AS person JOIN Railway.Passenger As passenger ON person.id = passenger.passenger_id
        WHERE passenger.email = @email AND passenger.pw = HASHBYTES('SHA1', @password))
GO

ALTER FUNCTION Railway.f_get_trips(@dep_station_name VARCHAR(30), @arr_station_name VARCHAR(30)) RETURNS @table TABLE
(trip_no INT, trip_type VARCHAR(2), dep_timestamp TIME, arr_timestamp TIME, duration TIME, price SMALLMONEY)
AS
    BEGIN
        DECLARE @dep_station AS INT;
        DECLARE @arr_station AS INT;
        INSERT @table (t.trip_no, t.trip_type, t.dep_timestamp, t.arr_timestamp, t.duration)
            SELECT t.trip_no, t.trip_type, t.dep_timestamp, t.arr_timestamp, t.duration
            FROM Railway.Trip AS t JOIN Railway.Station AS dep_station ON t.dep_station = dep_station.station_no
            JOIN Railway.Station AS arr_station ON t.arr_station = arr_station.station_no
            WHERE dep_station.station_name = @dep_station_name AND arr_station.station_name = @arr_station_name AND t.trip_type <> 'M';

...
```

prints.sql

```sql
        JOIN Railway.Station AS arr ON trip.arr_station = arr.station_no
        WHERE trip.trip_type = @trip_type AND dep.station_name = @dep_station AND arr.station_name = @arr_station;

    IF (@arr_zone < @dep_zone)
        BEGIN
            DECLARE @aux AS INT;
            SET @aux = @arr_zone;
            SET @arr_zone = @dep_zone;
            SET @dep_zone = @aux;
        END

        DECLARE @price AS SMALLMONEY;
        SET @price = 0;

        DECLARE @loop_cnt INT = @dep_zone;
        WHILE @loop_cnt <= @arr_zone
        BEGIN
            IF (@trip_type = 'UR')
                BEGIN
                    SELECT @price = (@price + priceUR)
                    FROM Railway.TripZone
                    WHERE zone_no = @loop_cnt;
                END
            ELSE IF (@trip_type = 'IC')
                BEGIN
                    SELECT @price = (@price + priceIC)
                    FROM Railway.TripZone
                    WHERE zone_no = @loop_cnt;
                END
            ELSE IF (@trip_type = 'AP')
                BEGIN
                    SELECT @price = (@price + priceAP)
                    FROM Railway.TripZone
                    WHERE zone_no = @loop_cnt;
                END

            SET @loop_cnt = @loop_cnt + 1;
        END
        RETURN @price;
END
```

10.3 Triggers

10.3 Triggers

- Foram utilizados 2 INSTEAD OF INSERT triggers
- Um de modo a calcular a duração de todas as viagens e outro com vista à atribuição de um comboio a uma determinada instância de uma viagem

prints.sql

```sql
ALTER TRIGGER Railway.InsertTripTrigger ON Railway.Trip
INSTEAD OF INSERT
AS
    BEGIN
        IF(SELECT COUNT(*) FROM inserted) = 1
            BEGIN
                DECLARE @trip_no        AS INT;
                DECLARE @trip_type      AS VARCHAR(2);
                DECLARE @dep_timestamp  AS TIME;
                DECLARE @arr_timestamp  AS TIME;
                DECLARE @dep_station    AS INT;
                DECLARE @arr_station    AS INT;
                DECLARE @duration       AS VARCHAR(30);

                -- retrieve info about the inserted tuple
                SELECT
                @trip_no = trip_no,
                @trip_type = trip_type,
                @dep_timestamp = dep_timestamp,
                @arr_timestamp = arr_timestamp,
                @dep_station = dep_station,
                @arr_station = arr_station
                FROM inserted;

                -- compute trip duration
                SET @duration = CONVERT(VARCHAR(5), DATEADD(MINUTE, DATEDIFF(MINUTE, @dep_timestamp, @arr_timestamp), 0), 114);

                -- insert the tuple in Railway.Trip table
                INSERT INTO Railway.Trip VALUES (@trip_type,@dep_timestamp,@arr_timestamp,@dep_station,@arr_station,@duration);
            END
    END
GO
```

prints.sql

```sql
ALTER TRIGGER Railway.InsertTripInstanceTrigger ON Railway.TripInstance
INSTEAD OF INSERT
AS
    BEGIN
        IF(SELECT COUNT(*) FROM inserted) = 1
            BEGIN
                DECLARE @trip_no        AS INT;
                DECLARE @trip_date      AS DATE;
                DECLARE @train_no       AS INT;
                DECLARE @dep_station    AS INT;
                DECLARE @arr_station    AS INT;
                DECLARE @duration       AS TIME;
                DECLARE @dep_timestamp  AS TIME;
                DECLARE @arr_timestamp  AS TIME;

                -- retrieve info about the inserted tuple
                SELECT
                @trip_no = trip_no,
                @trip_date = trip_date
                FROM inserted;

                SELECT
                @dep_station = dep_station,
                @arr_station = arr_station,
                @duration = duration,
                @dep_timestamp = dep_timestamp,
                @arr_timestamp = arr_timestamp
                FROM Railway.Trip WHERE trip_no = @trip_no;

                -- assign the train to the trip

                -- urbano/regional
                IF (1 <= @trip_no AND @trip_no <= 3) OR (86 <= @trip_no AND @trip_no <= 88)
                OR (165 <= @trip_no AND @trip_no <= 167) OR (250 <= @trip_no AND @trip_no <= 252)
                OR (329 <= @trip_no AND @trip_no <= 331)
                    BEGIN
                        SET @train_no = 1;
                    END

                ELSE IF (4 <= @trip_no AND @trip_no <= 6) OR (83 <= @trip_no AND @trip_no <= 85)
```

10.4 Cursor

10.4 Cursor

- Foi utilizado um cursor numa UDF, de modo a percorrer todos os tuplos da tabela de viagens com o objetivo de obter o custo de cada viagem.
- Este custo era dado em função do tipo de comboio atribuído à viagem e pelas zonas por qual transitava.

prints.sql

```sql
ALTER FUNCTION Railway.f_get_trips(@dep_station_name VARCHAR(30), @arr_station_name VARCHAR(30)) RETURNS @table TABLE
(trip_no INT, trip_type VARCHAR(2), dep_timestamp TIME, arr_timestamp TIME, duration TIME, price SMALLMONEY)
AS
    BEGIN
        DECLARE @dep_station AS INT;
        DECLARE @arr_station AS INT;
        INSERT @table (t.trip_no, t.trip_type, t.dep_timestamp, t.arr_timestamp, t.duration)
            SELECT t.trip_no, t.trip_type, t.dep_timestamp, t.arr_timestamp, t.duration
            FROM Railway.Trip AS t JOIN Railway.Station AS dep_station ON t.dep_station = dep_station.station_no
            JOIN Railway.Station AS arr_station ON t.arr_station = arr_station.station_no
            WHERE dep_station.station_name = @dep_station_name AND arr_station.station_name = @arr_station_name AND t.trip_type <> 'M';

        DECLARE @hello as int;
        DECLARE @tripType AS VARCHAR(2);
        DECLARE C CURSOR FAST_FORWARD
        FOR SELECT trip_type FROM @table

        OPEN C;

        WHILE @@FETCH_STATUS = 0
            BEGIN
                UPDATE @table SET price = (SELECT Railway.trip_price(@tripType, @dep_station_name, @arr_station_name)) WHERE trip_type = @tripTyp
                FETCH C INTO @tripType;
            END
        CLOSE C ;
        DEALLOCATE C;

        RETURN;
    END;
GO
```

11. Aspectos de Segurança - SQL Injection

- Os campos dos formulários do lado da aplicação do cliente foram validados antes da execução de procedimentos à base de dados.
- Não foi utilizado qualquer tipo de SQL Dinâmico.
- Não foram armazenados qualquer tipo de informações sensíveis em texto simples.
- A apresentação de erros ao cliente foi devidamente customizada.

teste.sql

```
If Not correct_password = 0 Then
    MsgBox("You inserted the correct password!", MsgBoxStyle.Exclamation)
    Return
End If

Console.WriteLine("RAILWAY LOG: correct password for " + e_mail)
sql_command.CommandText = "Railway.pr_forgot_password"
sql_command.CommandType = CommandType.StoredProcedure
sql_command.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = e_mail
sql_command.Parameters.Add("@password", SqlDbType.VarChar, 50).Value = pw
sql_command.ExecuteNonQuery()
MsgBox("Changes completed. You will be redirected to your profile page.", MsgBoxStyle.OkOnly)
```

testes.sql

```
-- Change password
    IF NOT (@new_password LIKE 'NULL')
        UPDATE Railway.Passenger SET pw = HASHBYTES('SHA1', @new_password) WHERE email = @email;
```

12.
Demo


F C P

Obrigado.
Questões?

Tiago Mendes e Tomás Batista