Gestão das atividades de aventura do Clube do Paiva.

P3 G4

Joana Cunha (98189)

Diana Rocha (98524)

Análise de Requisitos

Requisitos:

Escolhemos desenvolver uma base de dados sobre as atividades de aventura do Clube do Paiva, de forma a que possa haver um gerenciamento destas atividades e do aluguer de equipamento.

Poderá ser feita a gestão de atividades que cada cliente reserva e também a gestão dos equipamentos necessários para cada atividade.

- Criar novas atividades, clientes e funcionários, nomeadamente gerentes e guias.
- Pesquisar, as atividades marcadas por cliente, e que equipamentos é que o cliente terá que levantar.
- Consultar o guia que está encarregue de ajudar o cliente.
- Consultar as atividades correspondentes a cada época, as datas em que estas se realizam e os respetivos preços de cada uma.
- Alterar os dados relativos ao equipamento de forma a haver uma gerência do stock.

Entidades:

Um **cliente** é caracterizado por um nome, nº de telemóvel, mail e ainda possui e o NIF.

Um **funcionário** tem um nº de funcionário único, NIF, nome, telefone, email e data entrada. Existem dois tipos de funcionários, guia e gerente.

Um **guia** é um funcionário que possui um horário numa atividade registada e um número de funcionário.

Um **gerente** que está encarregue de receber os clientes e gerir o funcionamento da empresa. Tem também um numero de funcionário.

Uma **atividade registada** tem um ID, um nome, nº de pessoas, preço e uma data de quando se realizará e a data da reserva, um guia e um cliente associado.

As **atividades registadas,** são todas as atividades que existem na empresa, que são caracterizadas por um ID, nome da atividade, época , numero máximo de pessoas e preço

O **equipamento disponível** tem nome, tamanho e quantidade em stock e serve para ter uma visão do stock do equipamento que não está a ser utilizado.

O **equipamento para atividades** permite guardar material para uma certa atividade, tem um ID de reserva, o ID da atividade, o nome do equipamento, a quantidade a ser reservada e o tamanho.

Esquema relacional

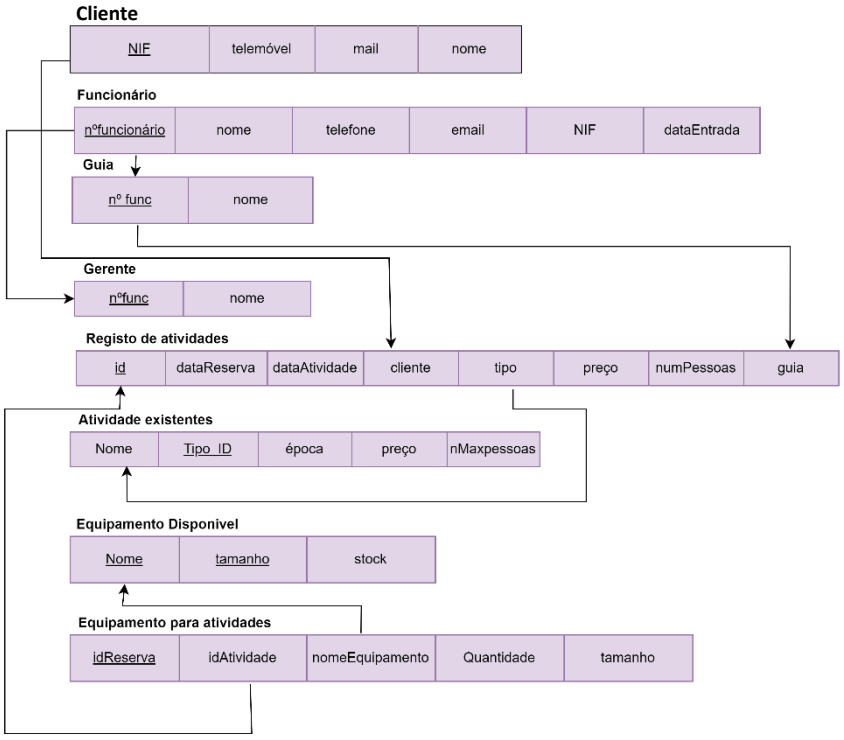
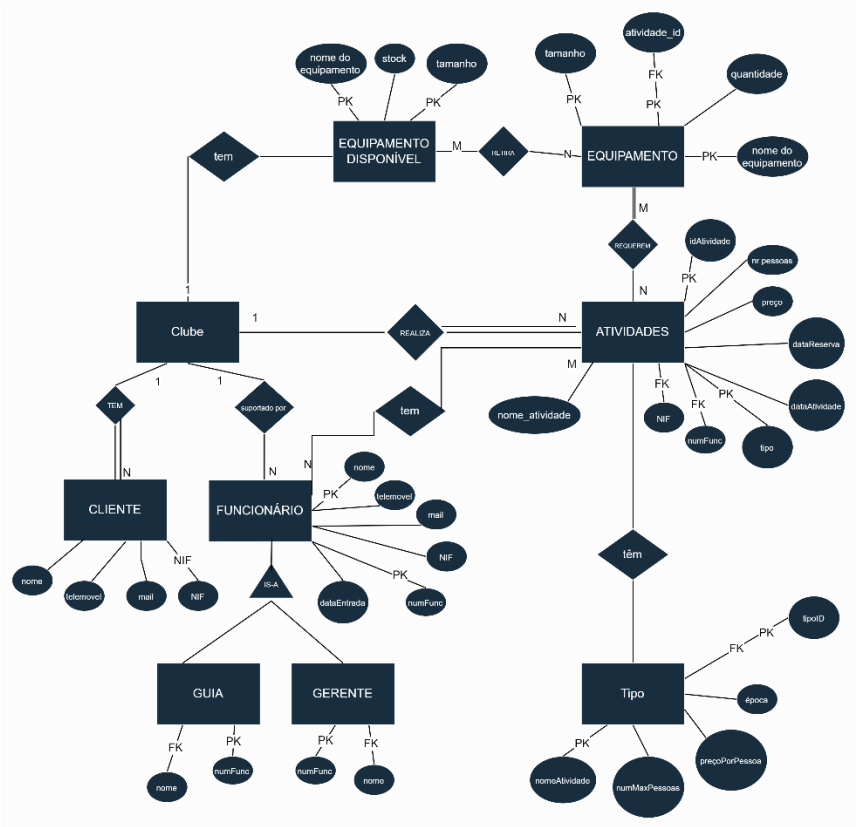


Diagrama Entidade-Relacionamento



Sql DDL

Para criar a base de dados deste projeto foram utilizadas as funcionalidades de **Create, Drop** e do comandos SQL DDL.

Exemplos:

Drop tables

```
drop table ClubePaiva.EquipamentoParaAtividades
drop table ClubePaiva.EquipamentoDisponível
drop table ClubePaiva.RegistoDeAtividades
drop table ClubePaiva.AtividadesExistentes
drop table ClubePaiva.Gerente
drop table ClubePaiva.Guia
drop table ClubePaiva.Funcionario;
drop table ClubePaiva.Cliente
```

Create tables

```
create table ClubePaiva.RegistoDeAtividades(
    idAtividade BIGINT not null identity(1,1),
    dataReserva datetime not null,  /* data em que foi feita a reserva da atividade */
    dataAtividade datetime not null, /* data da atividade */
    cliente BIGINT not null FOREIGN KEY REFERENCES ClubePaiva.Cliente(NIF),
    tipo varchar(255) not null,
    preco numeric not null,
    numPessoas int not null,
    guia bigint FOREIGN KEY REFERENCES ClubePaiva.Guia(numFunc),

    PRIMARY KEY(idAtividade)
);
```

Sql DML

Para popular inicialmente as nossas tabelas usamos o **Insert.**

```
INSERT INTO ClubePaiva.RegistoDeAtividades(dataReserva, dataAtividade, cliente, tipo,preco,numPessoas,guia)
VALUES
 (getdate(),'2022-06-11 12:00:00',820077185, 'canoas',254.00, 5, 1),
 (getdate(),'2022-06-20 13:00:00',183014872,'canyoning', 254.00, 5, 2),
 (getdate(),'2022-07-11 15:00:00',132400127,'rafting',254.00, 5, 3),
 (getdate(),'2022-08-10 17:00:00',145588537,'rivertracking', 254.00, 5, 1);

 SELECT SCOPE_IDENTITY() AS [Last-Inserted Identity Value];

 select * from ClubePaiva.RegistoDeAtividades;
```

Views

Exemplo de views:

```
/* QUEREMOS VER OS NIF'S DOS CLIENTES */
create view [dbo].[NIFclientes] as select NIF,nome from ClubePaiva.Cliente
go
select * from dbo.NIFclientes;

/* QUEREMOS VER OS NIF'S DOS FUNCIONÁRIOS */
create view [dbo].[NIFfuncionarios] as select NIF,nome,numFunc from ClubePaiva.Funcionario
go
select * from dbo.NIFfuncionarios;

/* QUEREMOS VER ATIVIDADES POR ORDEM DE DATA DE ATIVIDADE */
create view [dbo].[AtividadesRegistadas] as select * from ClubePaiva.RegistoDeAtividades;
go
select * from dbo.AtividadesRegistadas;


/* QUEREMOS VER O REGISTO DE ATIVIDADES ATÉ AGORA */
create view [dbo].[dataAtividades] as select idAtividade, dataAtividade from ClubePaiva.RegistoDeAtividades
go
select * from dbo.dataAtividades;

/* QUEREMOS VER O EQUIPAMENTO DISPONÍVEL NO CLUBE DO PAIVA */
create view [dbo].[stock] as select * from ClubePaiva.EquipamentoDisponível
go
select * from dbo.stock
select * from ClubePaiva.EquipamentoParaAtividades /* ver o equipamento deste stock que está a ser utilizado para verificar se não existe mais material usado do que o que temos */

/* QUEREMOS VER O EQUIPAMENTO USADO NUMA ATIVIDADE */
create view [dbo].[stockInUse] as select nomeEquipamento,quantidade,tamanho from ClubePaiva.EquipamentoParaAtividades
go
select * from dbo.stockInUse

/* QUEREMOS VER O NOME DA ATIVIDADE ASSOCIADO AO ID */
create view [dbo].[NomeAtividades] as select tipoID, nome_atividade from ClubePaiva.AtividadesExistentes
go
select * from dbo.NomeAtividades;
```

Foram usadas *Views* com o intuito de criar uma camada de abstração de forma a poupar trabalho em criar SELECT's, tanto em stored procedures, como em UDF'S. Também foram usadas para a programação da interface.

Stored Procedures

Criamos e usamos Stored Procedures para criar novas instâncias de entidades ou para editar instâncias que já existiam. Em algumas das SP's podiamos ter usado triggers, no entanto achamos mais vantajoso utilizar Stored Procedures.

As procedures que desenvolvemos para criar entidades foram:

- Criar clientes
- Criar guia
- Criar gerente
- Criar atividade
- Criar equipamento para uma reserva

As procedures que desenvolvemos para a edição e para a pesquisa foram:

- Pesquisar cliente pelo NIF
- Remover cliente
- Remover Guia
- Alterar guia em uma atividade
- Remover atividade
- Verificar o stock de um certo equipamento com um certo tamanho
- Libertar equipamento a ser utilizado
- Editar preço de uma atividade
- Verificar equipamento de uma atividade.

Exemplo de Stored Procedure utilizada, em que usamos inserimos equipamento para uma atividade e que damos update ao stock disponivel:

```
CREATE PROCEDURE [dbo].[addEquipamento](@idReserva bigint out, @idAtividade bigint, @nomeEquipamento VARCHAR(500),@quantidade bigint,@tamanho varchar(10))
as
    begin

        declare @stock bigint;
        set @stock = (select e.stock from ClubePaiva.EquipamentoDisponível e where e.nomeEquipamento =@nomeEquipamento and e.tamanho = @tamanho)

        if @stock > @quantidade
            begin
                insert into ClubePaiva.EquipamentoParaAtividades(idAtividade,nomeEquipamento,quantidade,tamanho) values (@idAtividade,@nomeEquipamento, @quantidade,@tamanho)
                set @idReserva = SCOPE_IDENTITY()
                UPDATE ClubePaiva.EquipamentoDisponível
                set stock=stock-@quantidade where nomeEquipamento=@nomeEquipamento and tamanho=@tamanho

            end
        else
            begin
                raiserror('Não existe stock para esse equipamento!',16,1);

            end
    end
go
```

Podemos ver nesta stored procedure que sempre que se reserva um certo equipamento para uma certa atividade gere-se sempre o stock mediante a quantidade de equipamento que reservamos.

Triggers

Criamos e usamos Triggers para verificar se a informação que estamos a tentar inserir nas tabelas de clientes e funcionários com uma determinada data já existe, ou então, no caso das tabelas de atividades, garantir que não se pode inserir uma data de atividade anterior à de reserva. Desta maneira, criaram-se os seguintes triggers:

- checkCliente
- checkFuncionario
- check_dates
- checkEquipamento

No caso do trigger *checkCliente*, vamos verificar se o número de contribuinte já existe na tabela de clientes. No caso de não existir (numCc=0), então é inserido um novo cliente com a informação sobre o mesmo (nome, telefone, email e número de contribuinte). No caso de já existir (numCc=1), gera uma mensagem de erro a dizer que o cliente já existe.

```
-- Adiciona clientes na tabela Clientes -> ver nas views ou inserts

create trigger ClubePaiva.checkCliente on ClubePaiva.Cliente
instead of insert
as
    begin
            declare @nome varchar(255);
            declare @telefone varchar(100);
            declare @email varchar(255);
            declare @numCc bigint;
            select @nome=nome, @telefone=telefone, @email=email, @numCc=NIF from inserted;
                if (([dbo].[checkCliente](@numCc) = 0))
                    insert into ClubePaiva.Cliente(nome,telefone,email,NIF) values (@nome, @telefone, @email, @numCc);
                else
                    raiserror('Já existe este cliente!',16,1);
    end
go
DROP TRIGGER ClubePaiva.checkCliente
```

No caso do trigger *checkFuncionario*, vamos verificar se o número de contribuinte já existe na tabela de funcionário. No caso de não existir (numCc=0), então é inserido um novo funcionário com a informação sobre o mesmo (nome, telefone, email, número de contribuinte e data de entrada na empresa). No caso de já existir (numCc=1), gera uma mensagem de erro a dizer que o funcionário já existe.

```
-----------------------------------------------------------------
-- Adiciona funcionarios na tabela Funcionario -> ver nas views ou inserts

create trigger ClubePaiva.checkFuncionario on ClubePaiva.Funcionario
instead of insert
as
    begin
            declare @nome varchar(255);
            declare @telefone varchar(100);
            declare @email varchar(255);
            declare @numCc bigint;
            declare @dataEntrada date;
            select @nome=nome, @telefone=telefone, @email=email, @numCc=NIF, @dataEntrada=dataEntrada from inserted;
                if (([dbo].[checkFuncionario](@numCc) = 0))
                    insert into ClubePaiva.Funcionario(nome,telefone,email,NIF,dataEntrada) values (@nome, @telefone, @email, @numCc, @dataEntrada);
                else
                    raiserror('Já existe este funcionario!',16,1);
    end
go

DROP TRIGGER ClubePaiva.checkFuncionario
```

No caso do trigger *check_dates*, garantimos que nenhuma atividade é registada com uma data posterior à data em que o cliente quer realizar a atividade. No caso de se inserir uma data de registo posterior, então gera uma mensagem de erro sobre a data de atividade ter de ser depois da de registo.

```
-------------------------------------------------------------
create trigger check_dates on ClubePaiva.RegistoDeAtividades
instead of insert
as
    begin
            declare @dataReserva datetime;
            declare @dataAtividade datetime;

            select @dataReserva=dataReserva, @dataAtividade=dataAtividade from inserted;

            if @dataReserva > @dataAtividade
                begin
                    raiserror ('A data de atividade tem de ser depois da data de reserva!',16,1);
                    rollback tran;
                end
    end
go

drop table check_dates;
```

No caso do trigger *checkEquipamento*, garantimos que não se pode comprometer numa atividade equipamento que não esteja disponível no stock. No caso de inserir uma quantidade superior, então gera uma mensagem de erro de que não existe aquela quantidade de equipamento disponível.

```
create trigger ClubePaiva.checkEquipamento on ClubePaiva.EquipamentoParaAtividades
instead of insert
as
    begin
            declare @idAtividade bigint;
            declare @nomeEquipamento varchar(500);
            declare @quantidade bigint;
            declare @tamanho varchar(10);
            select @idAtividade=idAtividade, @nomeEquipamento=nomeEquipamento, @quantidade=quantidade, @tamanho=tamanho from inserted;
                if (([dbo].[checkEquipamento](@nomeEquipamento, @quantidade, @tamanho) = 1))
                    insert into ClubePaiva.EquipamentoParaAtividades(idAtividade,nomeEquipamento,quantidade,tamanho) values (@idAtividade, @nomeEquipamento, @quantidade, @tamanho);
                else
                    raiserror('Não existe equipamento disponível para esta atividade!',16,1);
    end
go
DROP TRIGGER ClubePaiva.checkEquipamento


--INSERT INTO ClubePaiva.EquipamentoParaAtividades(idAtividade,nomeEquipamento,quantidade,tamanho) VALUES  (3,'colete',100,'S')
```

UDF’s

Criamos UDF’S que aceitassem parâmetros e que retorne um inteiro (0 ou 1) que dita a existência (1) ou não (0) de um valor numa tabela quando o vamos a inserir. Desta maneira, foi-nos útil para a construção dos triggers. Desta maneira, criaram-se as seguintes UDF's:

- checkCliente
- checkFuncionario
- checkEquipamento
- checkEquipamentoSelected
- ClienteAtividades
- checkAtividadeAndGuia
- EquipamentoInStock
- checkGuia
- checkGerente

UDF que recebe como parâmetro um inteiro que representa o número de contribuinte e retorna 1 se este NIF já existe na tabela de clientes ou 0 se não.

```
-- Verificar se o cliente já existe ou não

create function [dbo].checkCliente(@NIF bigint) returns int
as
    begin
            if exists(select * from ClubePaiva.Cliente as c where c.NIF= @NIF )
                return 1;
            return 0;
    end
go
```

UDF que recebe como parâmetro um inteiro que representa o número de contribuinte e retorna 1 se este NIF já existe na tabela de funcionários ou 0 se não.

```
-- Verificar se o funcionario já existe ou não

create function [dbo].checkFuncionario(@NIF bigint) returns int
as
    begin
            if exists(select * from ClubePaiva.Funcionario as f where f.NIF= @NIF)
                return 1;
            return 0;
    end
go
```

UDF que recebe como parâmetro uma data e horário que representa a data das atividades e retorna uma coluna com o id da atividade nessa data e hora, juntamente da data, do tipo de atividade em si e o guia responsável por ela.

```
-- Verificar se naquele horário existe alguma atividade, em que tipo de atividade consiste, o guia responsável e o id da atividade

create function [dbo].checkAtividadeAndGuia(@dataAtividade datetime) returns table
as
    return(select idAtividade,dataAtividade,tipo,guia
            from ClubePaiva.RegistoDeAtividades as d
            where d.dataAtividade = @dataAtividade)
go

--select * from dbo.checkAtividadeAndGuia('2022-08-10 17:00:00')
```

UDF que recebe como parâmetro um inteiro que representa o número de contribuinte de um cliente e retorna o número de contribuinte do mesmo, o nome, o id dessa atividade, o tipo, a data da atividade, o número de pessoas e o guia responsável por essa atividade apenas no caso de o NIF deste cliente estiver associado a uma atividade registada.

```
----------VER CLIENTES----------------------

-- VER AS INFORMAÇÕES SOBRE A ATIVIDADE ASSOCIADO AO CLIENTE
create function [dbo].ClienteAtividades(@cliente bigint) returns table
as
    return(select c.NIF, c.nome, rA.idAtividade, rA.tipo ,rA.dataAtividade, rA.numPessoas,rA.preco
            from ClubePaiva.Cliente as c
            join ClubePaiva.RegistoDeAtividades as rA on c.NIF= rA.cliente
            where cliente = @cliente)
go
```

UDF que recebe como parâmetro um varchar que representa o nome de equipamento, , um inteiro para a quantidade de equipamento que precisa, tendo em atenção que este valor não pode superar a quantidade em stock, e, por fim, um varchar que

representa os tipos de tamanho que necessita, e retorna 1 se tem disponível este equipamento ou 0 se não.

```
-- Verificar se aquele equipamento, naquele tamanho, está disponível

create function [dbo].checkEquipamento(@nomeEquipamento varchar(500), @quantidade bigint, @tamanho varchar(10)) returns int
as
    begin
            if exists(select * from ClubePaiva.EquipamentoDisponível as e where e.nomeEquipamento = @nomeEquipamento and e.stock >= @quantidade and e.tamanho = @tamanho )
                return 1;
            return 0;
    end
go
```

UDF que recebe como parâmetro um inteiro que representa o nome de equipamento que queremos pesquisar de uma atividade e retorna todos os tamanhos e o stock do mesmo equipamento. Isto dá-nos uma noção de manipulação de quantidades.

```
-- Verificar o equipamento que está disponível no stock

create function [dbo].EquipamentoInStock(@nomeEquipamento varchar(500)) returns table
as
    return(select nomeEquipamento,stock, tamanho
            from ClubePaiva.EquipamentoDisponível as e
            where e.nomeEquipamento = @nomeEquipamento)
go

--select * from dbo.EquipamentoInStock('colete')
```

UDF que recebe como parâmetro um inteiro que representa o id de uma atividade e retorna 1 se este id tem equipamento associado, ou 0 se não.

```
-- Verificar se já há equipamento reservado para uma atividade

create function [dbo].checkEquipamentoSelected(@idAtividade bigint) returns int
as
    begin
            if exists(select * from ClubePaiva.EquipamentoParaAtividades as eA where eA.idAtividade = @idAtividade )
                return 1;
            return 0;
    end
go
```

Transactions

```
create procedure [dbo].[createGerente](@nome varchar(255), @telefone varchar(100), @email varchar(255), @NIF bi
as
        begin Transaction
        declare @idFunc as bigint;
        exec dbo.insertFuncionario @nome=@nome, @telefone=@telefone, @email=@email, @NIF=@NIF, @numFunc=@idFunc
        insert into ClubePaiva.Gerente(nome,numFunc) values (@nome,@idFunc)
        set @numFunc = @idFunc;
        if @@ERROR !=0
        rollback tran
    else
        commit tran
go
```

Foram também utilizadas *Transactions,* para evitar erros ou *crashes* na execução de código sobre a base de dados. Seguimos o formato demonstrado acima.

Interface

Ver vídeo da demostração da aplicação no projeto/apresentaçãoFinal.pptx.