Relatório

Beatriz Santos - 50473
Manoela Azevedo - 50034
Rodrigo Calado- 49442

Trabalho para a Unidade Curricular
Base de Dados
1º ciclo de estudos em Informática Web

Docente: Rui Cardoso

Junho de 2024

Índice

Introdução

O presente trabalho visa consolidar os conhecimentos adquiridos na unidade curricular através do desenvolvimento e implementação de uma base de dados, bem como de uma aplicação cliente para interagir com a informação nela armazenada.

Este projeto foca-se na gestão de projetos de investigação e prestação de serviços no contexto do Departamento de Investigação da Universidade da Beira Interior (DIUBI).

Através deste sistema de informação, busca-se acompanhar as atividades dos membros da DIUBI, gerenciando dados sobre projetos e contratos, suas fontes de financiamento, e o envolvimento dos investigadores.

O trabalho inclui a modelação da base de dados, implementação do modelo relacional, e desenvolvimento de uma aplicação que garanta a integridade e consistência dos dados, facilitando consultas, inserções, atualizações e remoções de dados.

Modelo de dados

Criação da Base de Dados

Dadas as requisições e definições, este é o nosso Modelo de Base de Dados final:

```
USE DIUBI;
CREATE TABLE Pessoa (
  IdPessoa INT IDENTITY(1,1) NOT NULL,
  PrimeiroNome VARCHAR(250),
  UltimoNome VARCHAR(250),
  Email VARCHAR(250) NOT NULL,
  Telefone VARCHAR(20),
  PRIMARY KEY (IdPessoa)
);

CREATE TABLE Interno (
  IdInterno INT IDENTITY(1,1) NOT NULL,
  IdPessoa INT NOT NULL,
  PRIMARY KEY (IdInterno),
  FOREIGN KEY (IdPessoa) REFERENCES Pessoa(IdPessoa)
);

CREATE TABLE Externo (
  IdExterno INT IDENTITY(1,1) NOT NULL,
  IdPessoa INT NOT NULL,
  PRIMARY KEY (IdExterno),
  FOREIGN KEY (IdPessoa) REFERENCES Pessoa(IdPessoa)
);

CREATE TABLE Membro (
  IdMembro INT IDENTITY(1,1) NOT NULL,
  IdPessoa INT NOT NULL,
  TipoMembro VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdMembro),
  FOREIGN KEY (IdPessoa) REFERENCES Pessoa(IdPessoa)
);

CREATE TABLE Projeto_Servico (
```

```
  IdProjeto_Servico INT NOT NULL,
  TipoProjeto_Servico VARCHAR(50) NOT NULL,
  PRIMARY KEY (IdProjeto_Servico, TipoProjeto_Servico)
);

CREATE TABLE DataInfo (
  IdData INT NOT NULL,
  DataInicio DATE NOT NULL,
  DataFim DATE NOT NULL,
  PRIMARY KEY (IdData)
);

CREATE TABLE Estado (
  IdEstado INT NOT NULL,
  NomeEstado VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdEstado)
);

CREATE TABLE Programa (
  IdPrograma INT NOT NULL,
  NacionalidadePrograma VARCHAR(250) NULL,
  NomePrograma VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdPrograma)
);

CREATE TABLE Instituica (
  IdInstituicao INT NOT NULL,
  NomeInstituicao VARCHAR(250) NOT NULL,
  NacionalidadeInstituicao VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdInstituicao)
);

CREATE TABLE DominioCientifico (
  IdDominio INT NOT NULL,
  NomeDominio VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdDominio)
);

CREATE TABLE AreaCientifica (
  IdArea INT NOT NULL,
```

```
  NomeArea VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdArea)
);

CREATE TABLE PalavraChave (
  IdPalavraChave INT NOT NULL,
  PalavraChave VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdPalavraChave)
);

CREATE TABLE Posicao (
  IdPosicao INT NOT NULL,
  NomePosicao VARCHAR(250) NOT NULL,
  PRIMARY KEY (IdPosicao)
);

CREATE TABLE Financiador (
  IdFinanciador INT IDENTITY(1,1) NOT NULL,
  TipoFinanciador VARCHAR(50) NOT NULL,
  PRIMARY KEY (IdFinanciador, TipoFinanciador)
);

CREATE TABLE Orcid (
  IdOrcid INT NOT NULL,
  IdMembro INT NOT NULL,
  PRIMARY KEY (IdOrcid),
  FOREIGN KEY (IdMembro) REFERENCES Membro(IdMembro)
);

CREATE TABLE Instituicao_Membro (
  IdMembro INT NOT NULL,
  IdInstituicao INT NOT NULL,
  PRIMARY KEY (IdMembro, IdInstituicao),
  FOREIGN KEY (IdMembro) REFERENCES Membro(IdMembro),
 FOREIGN KEY (IdInstituicao) REFERENCES Instituicao(IdInstituicao)
);

CREATE TABLE Financiamento (
  IdFinanciamento INT NOT NULL,
  Valor DECIMAL(15, 2) NOT NULL,
```

```
  TipoFinanciamento VARCHAR(250) NOT NULL,
  OrigemFinanciamento VARCHAR(250) NOT NULL,
  IdFinanciador INT NOT NULL,
  TipoFinanciador VARCHAR(50) NOT NULL,
  PRIMARY KEY (IdFinanciamento),
    FOREIGN KEY (IdFinanciador, TipoFinanciador) REFERENCES Financiador(IdFinanciador,
TipoFinanciador)
);

CREATE TABLE Projeto (
  IdProjeto INT NOT NULL,
  NomeProjeto VARCHAR(250) NOT NULL,
  Descricao VARCHAR(250) NOT NULL,
  IdData INT NOT NULL,
  IdInstituicao INT NOT NULL,
  IdEstado INT NOT NULL,
  IdArea INT NOT NULL,
  IdDominio INT NOT NULL,
  IdInterno INT NOT NULL,
  PRIMARY KEY (IdProjeto),
  FOREIGN KEY (IdData) REFERENCES DataInfo(IdData),
  FOREIGN KEY (IdInstituicao) REFERENCES Instituicao(IdInstituicao),
  FOREIGN KEY (IdEstado) REFERENCES Estado(IdEstado),
  FOREIGN KEY (IdArea) REFERENCES AreaCientifica(IdArea),
  FOREIGN KEY (IdDominio) REFERENCES DominioCientifico(IdDominio),
  FOREIGN KEY (IdInterno) REFERENCES Interno(IdInterno)
);

CREATE TABLE Equipa (
  IdEquipa INT NOT NULL,
  IdProjeto INT NOT NULL,
  PRIMARY KEY (IdEquipa),
  FOREIGN KEY (IdProjeto) REFERENCES Projeto(IdProjeto)
);

CREATE TABLE Equipa_Membro (
  IdEquipa INT NOT NULL,
  IdMembro INT NOT NULL,
  PRIMARY KEY (IdEquipa, IdMembro),
  FOREIGN KEY (IdMembro) REFERENCES Membro(IdMembro)
```

```
);

CREATE TABLE CustoElegivelEquipa (
  IdCustoElegivelEquipa INT NOT NULL,
  IdEquipa INT NOT NULL,
  CustoEquipa DECIMAL(15, 2) NOT NULL,
  IdFinanciamento INT NOT NULL,
  PRIMARY KEY (IdCustoElegivelEquipa),
  FOREIGN KEY (IdFinanciamento) REFERENCES Financiamento(IdFinanciamento),
  FOREIGN KEY (IdEquipa) REFERENCES Equipa(IdEquipa),
);

CREATE TABLE CustoElegivelProjeto (
  IdCustoElegivelProjeto INT NOT NULL,
  IdProjeto INT NOT NULL,
  CustoProjeto DECIMAL(15, 2) NOT NULL,
  IdFinanciamento INT NOT NULL,
  PRIMARY KEY (IdCustoElegivelProjeto),
  FOREIGN KEY (IdFinanciamento) REFERENCES Financiamento(IdFinanciamento),
  FOREIGN KEY (IdProjeto) REFERENCES Projeto(IdProjeto)
);

CREATE TABLE PrestacaoServico (
  IdPrestacaoServico INT NOT NULL,
  NomePrestacaoServico VARCHAR(250) NOT NULL,
  Descricao TEXT NOT NULL,
  IdInterno INT NOT NULL,
  IdData INT NOT NULL,
  IdEstado INT NOT NULL,
  PRIMARY KEY (IdPrestacaoServico),
  FOREIGN KEY (IdInterno) REFERENCES Interno(IdInterno),
  FOREIGN KEY (IdData) REFERENCES DataInfo(IdData),
  FOREIGN KEY (IdEstado) REFERENCES Estado(IdEstado)
);

CREATE TABLE Financiamento_Projeto_PrestacaoServico (
  IdFinanciamento INT IDENTITY(1,1) NOT NULL,
  IdProjeto_Servico INT NOT NULL,
  TipoProjeto_Servico VARCHAR(50) NOT NULL,
  PRIMARY KEY (IdFinanciamento, IdProjeto_Servico),
```

```
  FOREIGN KEY (IdFinanciamento) REFERENCES Financiamento(IdFinanciamento),
                FOREIGN        KEY        (IdProjeto_Servico,        TipoProjeto_Servico)        REFERENCES
Projeto_Servico(IdProjeto_Servico, TipoProjeto_Servico)
);

CREATE TABLE AssociarPalavraChave (
  IdAssociacao INT NOT NULL,
  IdProjeto INT NOT NULL,
  IdPalavraChave INT NOT NULL,
  PRIMARY KEY (IdAssociacao),
  FOREIGN KEY (IdProjeto) REFERENCES Projeto(IdProjeto),
  FOREIGN KEY (IdPalavraChave) REFERENCES PalavraChave(IdPalavraChave)
);

CREATE TABLE Atividade (
  IdAtividade INT NOT NULL,
  NomeAtividade VARCHAR(250) NOT NULL,
  TipoAtividade VARCHAR(250) NOT NULL,
  IdProjeto_Servico  INT NOT NULL,
  TipoProjeto_Servico VARCHAR(50) NOT NULL,
  PRIMARY KEY (IdAtividade),
                FOREIGN        KEY        (IdProjeto_Servico,        TipoProjeto_Servico)        REFERENCES
Projeto_Servico(IdProjeto_Servico, TipoProjeto_Servico)
);

CREATE TABLE TempoAtividade (
  IdMembro INT NOT NULL,
  TempoTrabalho TIME NOT NULL,
  IdAtividade INT NOT NULL,
  PRIMARY KEY (IdMembro, IdAtividade),
  FOREIGN KEY (IdAtividade) REFERENCES Atividade(IdAtividade)
);

CREATE TABLE PosicaoInterno (
  IdPosicao INT NOT NULL,
  IdInterno INT NOT NULL,
  IdProjeto INT NOT NULL,
  PRIMARY KEY (IdPosicao, IdInterno, IdProjeto),
  FOREIGN KEY (IdPosicao) REFERENCES Posicao(IdPosicao),
  FOREIGN KEY (IdInterno) REFERENCES Interno(IdInterno),
```

```
  FOREIGN KEY (IdProjeto) REFERENCES Projeto(IdProjeto)
);

CREATE TABLE Publicacao (
  IdPublicacao INT NOT NULL,
  DOI VARCHAR(100) NOT NULL,
  IdProjeto INT NOT NULL,
  IdInterno INT NOT NULL,
  IdData INT NOT NULL,
  PRIMARY KEY (IdPublicacao),
  FOREIGN KEY (IdProjeto) REFERENCES Projeto(IdProjeto),
  FOREIGN KEY (IdInterno) REFERENCES Interno(IdInterno),
  FOREIGN KEY (IdData) REFERENCES DataInfo(IdData)
);
```

Triggers

A seguinte seção cria um *procedure* "InserirPessoa" para inserir dados de uma pessoa, distinguindo entre membros internos e externos e registrando essa informação na tabela "Membro" por meio de um *trigger*:

```
CREATE PROCEDURE InserirPessoa (
    @PrimeiroNome VARCHAR(250),
    @UltimoNome VARCHAR(250),
    @Email VARCHAR(250),
    @Telefone VARCHAR(20),
    @TipoMembro VARCHAR(250)
)
AS
BEGIN
    DECLARE @newIdPessoa INT;
   INSERT INTO Pessoa (PrimeiroNome, UltimoNome, Email, Telefone)
    VALUES (@PrimeiroNome, @UltimoNome, @Email, @Telefone);
    SET @newIdPessoa = SCOPE_IDENTITY();
    IF @TipoMembro = 'Interno'
    BEGIN
        INSERT INTO Interno (IdPessoa) VALUES (@newIdPessoa);
    END
    ELSE IF @TipoMembro = 'Externo'
    BEGIN
        INSERT INTO Externo (IdPessoa) VALUES (@newIdPessoa);
```

```
    END
END
GO
CREATE TRIGGER after_interno_insert
ON Interno
AFTER INSERT
AS
BEGIN
    DECLARE @newIdInterno INT;
    DECLARE @newIdPessoa INT;
    SELECT @newIdInterno = IdInterno, @newIdPessoa = IdPessoa FROM inserted;
    INSERT INTO Membro (IdPessoa, TipoMembro)
    VALUES (@newIdPessoa, 'Interno');


END
GO
CREATE TRIGGER after_externo_insert
ON Externo
AFTER INSERT
AS
BEGIN
    DECLARE @newIdExterno INT;
    DECLARE @newIdPessoa INT;
    SELECT @newIdExterno = IdExterno, @newIdPessoa = IdPessoa FROM inserted;
    INSERT INTO Membro (IdPessoa, TipoMembro)
    VALUES (@newIdPessoa, 'Externo');
END
GO
```

Essa seção cria triggers para inserções e exclusões nas tabelas relacionadas a programas e instituições. Após uma inserção, eles adicionam informações à tabela "Financiador" automaticamente, indicando se o financiador é um programa ou uma instituição. Também existe uma secção para tratar os dados caso um programa ou instituição seja excluída:

```
CREATE TRIGGER after_programa_insert
ON Programa
AFTER INSERT
AS
BEGIN
    DECLARE @newId INT;


    SELECT @newId = IdPrograma FROM inserted;
```

```
    INSERT INTO Financiador (IdFinanciador, TipoFinanciador)
    VALUES (@newId, 'Programa');
END
GO

CREATE TRIGGER after_instituicao_insert
ON Instituicao
AFTER INSERT
AS
BEGIN
    DECLARE @newId INT;

    SELECT @newId = IdInstituicao FROM inserted;

    INSERT INTO Financiador (IdFinanciador, TipoFinanciador)
    VALUES (@newId, 'Instituicao');
END
GO

CREATE TRIGGER delete_financiador_on_programa_delete
ON Programa
AFTER DELETE
AS
BEGIN
    DELETE FROM Financiador WHERE IdFinanciador IN (SELECT IdPrograma FROM deleted);
END
GO

CREATE TRIGGER delete_financiador_on_instituicao_delete
ON Instituicao
AFTER DELETE
AS
BEGIN
    DELETE FROM Financiador WHERE IdFinanciador IN (SELECT IdInstituicao FROM deleted);
END
GO
```

Esses *triggers* monitoram inserções e exclusões nas tabelas "Projeto" e "PrestacaoServico". Após uma inserção, eles adicionam dados à tabela

"Projeto_Servico" indicando o tipo de projeto ou prestação de serviço e adicionando o mesmo Id antes definido. Após excluir um "Projeto" ou "PrestacaoServico", eles removem as entradas correspondentes da tabela "Projeto_Servico":

```
CREATE TRIGGER InsertProjetoServico
ON Projeto
AFTER INSERT
AS
BEGIN
    DECLARE @IdProjeto INT;
    DECLARE insert_cursor CURSOR FOR
    SELECT IdProjeto FROM inserted;

    OPEN insert_cursor;
    FETCH NEXT FROM insert_cursor INTO @IdProjeto;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO Projeto_Servico (IdProjeto_Servico, TipoProjeto_Servico)
        VALUES (@IdProjeto, 'Projeto');

        FETCH NEXT FROM insert_cursor INTO @IdProjeto;
    END;

    CLOSE insert_cursor;
    DEALLOCATE insert_cursor;
END;

CREATE TRIGGER InsertProjetoServico_PrestacaoServico
ON PrestacaoServico
AFTER INSERT
AS
BEGIN
    DECLARE @IdPrestacaoServico INT;
    DECLARE insert_cursor CURSOR FOR
    SELECT IdPrestacaoServico FROM inserted;

    OPEN insert_cursor;
    FETCH NEXT FROM insert_cursor INTO @IdPrestacaoServico;
```

```
    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO Projeto_Servico (IdProjeto_Servico, TipoProjeto_Servico)
        VALUES (@IdPrestacaoServico, 'PrestacaoServico');

        FETCH NEXT FROM insert_cursor INTO @IdPrestacaoServico;
    END;

    CLOSE insert_cursor;
    DEALLOCATE insert_cursor;
END;

CREATE TRIGGER DeleteProjetoServico
ON Projeto
AFTER DELETE
AS
BEGIN
    DELETE FROM Projeto_Servico
    WHERE IdProjeto_Servico IN (SELECT IdProjeto FROM deleted)
    AND TipoProjeto_Servico = 'Projeto';
END;
GO

CREATE TRIGGER DeletePrestacaoServico
ON PrestacaoServico
AFTER DELETE
AS
BEGIN
    DELETE FROM Projeto_Servico
    WHERE IdProjeto_Servico IN (SELECT IdPrestacaoServico FROM deleted)
    AND TipoProjeto_Servico = 'PrestacaoServico';
END;
GO
```

Este Procedure trata dos dados a serem inseridos na parte do “Financiamento” do projeto. Torna obrigatório que para a criação de um “Financiamento” seja associado um “Projeto” ou “PrestacaoServico”. Também está programado para lidar com o caso de exclusão:

```
CREATE PROCEDURE InserirFinanciamento
    @Valor DECIMAL(15, 2),
    @TipoFinanciamento VARCHAR(250),
    @OrigemFinanciamento VARCHAR(250),
    @IdFinanciador INT,
    @TipoFinanciador VARCHAR(50),
    @IdProjeto_Servico INT,
    @TipoProjeto_Servico VARCHAR(50)
AS
BEGIN
    DECLARE @IdFinanciamento INT;

    INSERT INTO Financiamento (Valor, TipoFinanciamento, OrigemFinanciamento, IdFinanciador,
TipoFinanciador)
    VALUES (@Valor, @TipoFinanciamento, @OrigemFinanciamento, @IdFinanciador,
@TipoFinanciador);

    SET @IdFinanciamento = SCOPE_IDENTITY();

    INSERT INTO Financiamento_Projeto_PrestacaoServico (IdFinanciamento, IdProjeto_Servico,
TipoProjeto_Servico)
    VALUES (@IdFinanciamento, @IdProjeto_Servico, @TipoProjeto_Servico);
END;
GO

CREATE TRIGGER TR_InsteadOfDeleteFinanciamento
ON Financiamento
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM Financiamento_Projeto_PrestacaoServico
    WHERE IdFinanciamento IN (SELECT IdFinanciamento FROM DELETED);

    DELETE FROM Financiamento
    WHERE IdFinanciamento IN (SELECT IdFinanciamento FROM DELETED);
END;
GO
```

Este trigger está programado para lidar com a divisão do valor total financeiro atribuído pelo “Financiamento”, entre a “Equipa” e o “Projeto” de forma a garantir que não sejam a eles atribuídos mais do que o custo elegível em seu valor próprio ou na soma dos dois:

```
CREATE OR ALTER TRIGGER VerificarCustos
ON CustoElegivelEquipa
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @TotalCustoEquipa DECIMAL(15, 2);
    DECLARE @TotalCustoProjeto DECIMAL(15, 2);
    DECLARE @IdFinanciamento INT;

    SELECT @TotalCustoEquipa = SUM(CustoEquipa)
    FROM CustoElegivelEquipa
    WHERE IdFinanciamento = (SELECT IdFinanciamento FROM inserted);

    SELECT @TotalCustoProjeto = SUM(CustoProjeto)
    FROM CustoElegivelProjeto
    WHERE IdFinanciamento = (SELECT IdFinanciamento FROM inserted);

    SELECT @IdFinanciamento = IdFinanciamento
    FROM inserted;

    DECLARE @ValorFinanciamento DECIMAL(15, 2);

    SELECT @ValorFinanciamento = Valor
    FROM Financiamento
    WHERE IdFinanciamento = @IdFinanciamento;

    IF (@TotalCustoEquipa + @TotalCustoProjeto) > @ValorFinanciamento
    BEGIN
        RAISERROR('A soma dos custos excede o valor total do financiamento.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
```

Este trigger garante que o “Financiamento” atribuído a uma “Equipa” deve ser o mesmo atribuído ao seu respetivo “Projeto”:

```
CREATE TRIGGER AssociarFinanciamentoEquipa
ON CustoElegivelEquipa
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        INNER JOIN Equipa e ON i.IdEquipa = e.IdEquipa
        INNER JOIN Projeto p ON e.IdProjeto = p.IdProjeto
                INNER JOIN Projeto_Servico ps ON p.IdProjeto = ps.IdProjeto_Servico AND
ps.TipoProjeto_Servico = 'Projeto'
            INNER JOIN Financiamento_Projeto_PrestacaoServico fpps ON ps.IdProjeto_Servico =
fpps.IdProjeto_Servico AND ps.TipoProjeto_Servico = fpps.TipoProjeto_Servico
        WHERE i.IdFinanciamento <> fpps.IdFinanciamento
    )
    BEGIN
        RAISERROR('O financiamento associado à equipa não corresponde ao financiamento do
projeto.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
        INSERT INTO CustoElegivelEquipa (IdCustoElegivelEquipa, IdEquipa, CustoEquipa,
IdFinanciamento)
    SELECT IdCustoElegivelEquipa, IdEquipa, CustoEquipa, IdFinanciamento FROM inserted;
END;
```

Este *trigger* garante que ao um “Interno” ser atribuído como líder de um “Projeto”, também é atribuído como líder na tabela “PosicaoInterno”. Também está programado para lidar com a exclusão dos dados:

```
CREATE TRIGGER TRG_AssignProjectLeader
ON Projeto
AFTER INSERT
AS
```

```
BEGIN
    DECLARE @IdProjeto INT, @IdInterno INT
    SELECT @IdProjeto = IdProjeto, @IdInterno = IdInterno FROM inserted
        IF NOT EXISTS (SELECT 1 FROM PosicaoInterno WHERE IdInterno = @IdInterno AND
IdProjeto = @IdProjeto)
    BEGIN
        IF NOT EXISTS (SELECT 1 FROM Posicao WHERE IdPosicao = 1)
        BEGIN
            INSERT INTO Posicao (IdPosicao, NomePosicao)
            VALUES (1, 'Líder')
        END
        INSERT INTO PosicaoInterno (IdPosicao, IdInterno, IdProjeto)
        VALUES (1, @IdInterno, @IdProjeto)
    END
END
CREATE TRIGGER TRG_DeleteProjectIntern
ON Projeto
AFTER DELETE
AS
BEGIN
    DECLARE @IdProjeto INT
    SELECT @IdProjeto = IdProjeto FROM deleted
    DELETE FROM PosicaoInterno WHERE IdProjeto = @IdProjeto
END
```

Este trigger garante que uma “Equipa”, esteja associada apenas a um “Projeto”:

```
CREATE TRIGGER TRG_CheckTeamProjectAssociation
ON Equipa
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @IdEquipa INT, @IdProjeto INT
    IF EXISTS (SELECT 1 FROM inserted WHERE IdProjeto IN (SELECT IdProjeto FROM Projeto))
    BEGIN
        RAISERROR('O projeto já possui uma equipe associada.', 16, 1)
        ROLLBACK TRANSACTION
```

```
        RETURN
    END
    INSERT INTO Equipe (IdEquipa, IdProjeto)
    SELECT IdEquipa, IdProjeto FROM inserted
END
```

Este *trigger* garante que uma “Equipa”, esteja associada apenas a um “Projeto”:

```
CREATE TRIGGER TRG_AssociateTeamWithFinancing
ON CustoElegivelEquipa
AFTER INSERT
AS
BEGIN
    DECLARE @IdEquipa INT, @IdFinanciamento INT
    SELECT @IdFinanciamento = IdFinanciamento FROM inserted
      IF NOT EXISTS (SELECT 1 FROM Projeto WHERE IdProjeto IN (SELECT IdProjeto FROM
Equipe WHERE IdEquipa IN (SELECT IdEquipa FROM inserted)) AND IdFinanciamento =
@IdFinanciamento)
    BEGIN
        RAISERROR('A equipe não está associada ao mesmo financiamento do projeto.', 16, 1)
        ROLLBACK TRANSACTION
        RETURN
    END
END
```

Este trigger garante que a “Equipa” não possa se associar a um “Financiamento” diferente do financiamento do “Projeto” ao qual está associada:

```
CREATE TRIGGER TRG_AssociateTeamWithFinancing
ON CustoElegivelEquipa
AFTER INSERT
AS
BEGIN
    DECLARE @IdEquipa INT, @IdFinanciamento INT
```

```
    SELECT @IdFinanciamento = IdFinanciamento FROM inserted
        IF NOT EXISTS (SELECT 1 FROM Projeto WHERE IdProjeto IN (SELECT IdProjeto FROM
Equipe WHERE IdEquipa IN (SELECT IdEquipa FROM inserted)) AND IdFinanciamento =
@IdFinanciamento)
    BEGIN
        RAISERROR('A equipe não está associada ao mesmo financiamento do projeto.', 16, 1)
        ROLLBACK TRANSACTION
        RETURN
    END
END
```

Este *trigger* serve para garantir que uma “Atividade” de um “Projeto”, esteja sendo realizada por um “Membro” que pertence a uma “Equipa” daquele “Projeto”:

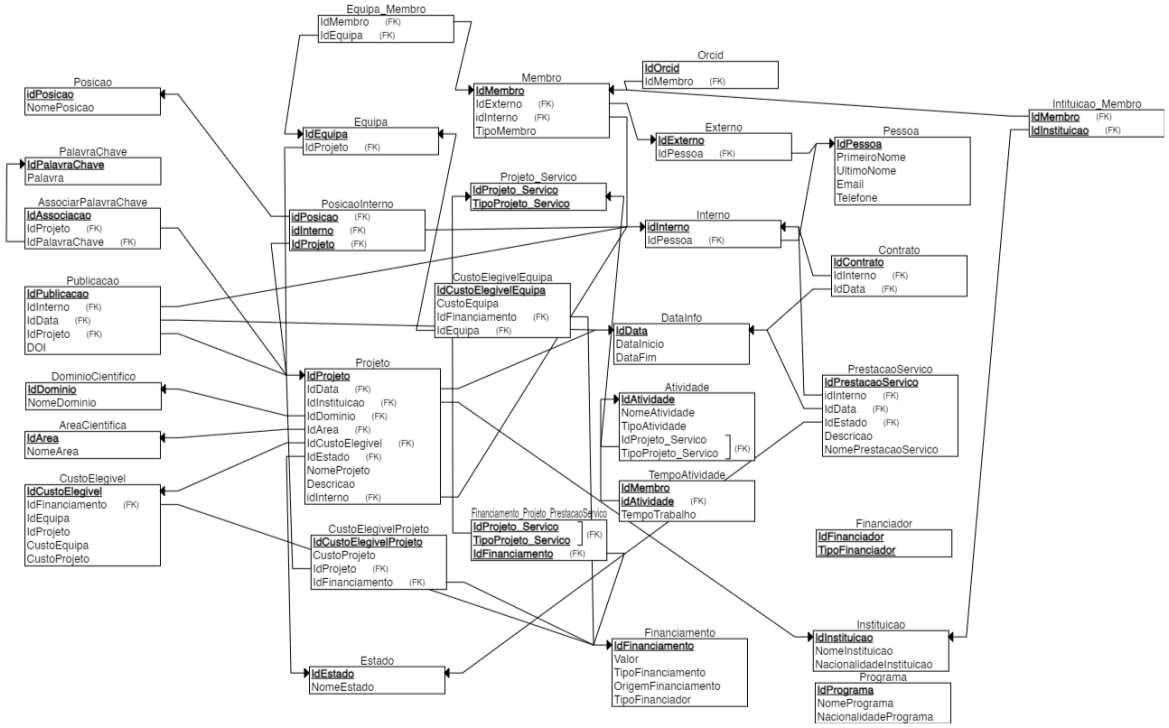
```
CREATE TRIGGER VerificarAssociacaoAtividade
ON TempoAtividade
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        INNER JOIN Atividade a ON i.IdAtividade = a.IdAtividade
            INNER JOIN Projeto_Servico ps ON a.IdProjeto_Servico = ps.IdProjeto_Servico AND
a.TipoProjeto_Servico = ps.TipoProjeto_Servico
        INNER JOIN Equipa_Membro em ON i.IdMembro = em.IdMembro
                INNER JOIN Equipa e ON em.IdEquipa = e.IdEquipa AND e.IdProjeto =
ps.IdProjeto_Servico -- Verifica se o membro está na equipe do projeto associado à atividade
    )
    BEGIN
        INSERT INTO TempoAtividade (IdMembro, TempoTrabalho, IdAtividade)
        SELECT IdMembro, TempoTrabalho, IdAtividade FROM inserted;
    END
    ELSE
    BEGIN
        RAISERROR('Um membro nao pode realizar uma atividade para um projeto se nao estiver
na equipa deste projeto', 16, 1);
```

```
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
```

SGBD e Base de Dados

Modelo Relacional

Novo Modelo de Dados com as alterações realizadas após a primeira entrega.



Funcionalidades e Queries

Selecionar o estado pelo nome e id do projeto:

```
queries.get("/estado_projeto/:IdProjeto", async (req, res) => {
    const { IdProjeto } = req.params;
    if (!IdProjeto || isNaN(IdProjeto) || IdProjeto < 0) { res.status(400).send("ID do Projeto Inválido."); return; }
    const query = "SELECT DISTINCT P.IdProjeto, P.NomeProjeto, E.NomeEstado FROM Projeto P, Estado E WHERE P.IdProjeto = @IdProjeto AND P.IdEstado = E.IdEstado";
    const sqlRequest = new sql.Request();
    sqlRequest.input('IdProjeto', sql.Int, IdProjeto);
    const result = await sqlRequest.query(query);
```

```
    res.status(200).send(result.recordset);
});
```

```
queries.get("/financiamento_projetos", async (req, res) => {
    const query = "SELECT DISTINCT P.IdProjeto, P.NomeProjeto, F.Valor FROM Projeto
P, Financiamento F WHERE F.TipoProjeto_Servico = P.IdProjeto AND
F.TipoProjeto_Servico LIKE 'projeto' ORDER BY VALOR DESC";
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.get("/membros_projetos", async (req, res) => {
    const query = "SELECT DISTINCT P.IdProjeto, P.NomeProjeto, COUNT(E.IdMembro)
as QuantidadeMembros FROM Projeto P JOIN Equipa E ON E.IdProjeto = P.IdProjeto
JOIN Membro M ON E.IdMembro = M.IdMembro GROUP BY P.IdProjeto,
P.NomeProjeto ORDER BY QuantidadeMembros DESC";
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.get("/instituicao_projetos_financiamentos", async (req, res) => {
    const query = "SELECT DISTINCT I.NomeInstituicao, COUNT(F.IdFinanciamento) as
QuantidadeFinanciamentos FROM Instituicao I JOIN Financiamento F ON
I.IdFinanciador = I.IdInstituicao JOIN Financiador Fin ON F.IdFinanciadoor =
```

```
Fin.IdFinanciador AND Fin.TipoFinanciador LIKE 'Projeto' GROUP BY I.NomeInstituicao
ORDER BY QuantidadeFinanciamentos DESC";
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.get("/publicacoes_projetos", async (req, res) => {
    const query = "SELECT DISTINCT P.IdProjeto, P.NomeProjeto,
COUNT(Pub.IdPublicacao) as QuantidadePublicacoes FROM Projeto P JOIN
Publicacao Pub ON P.IdProjeto = Pub.IdProjeto GROUP BY P.IdProjeto, P.NomeProjeto
ORDER BY QuantidadePublicacoes DESC";
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.get("/instituições financiamento", async (req, res) => {
    const query = "SELECT DISTINCT I.NomeInstituicao, COUNT(F.IdFinanciamento) as
QuantidadeFinanciamentos FROM Instituicao I JOIN Financiamento F ON
I.IdFinanciador = I.IdInstituicao JOIN Financiador Fin ON F.IdFinanciadoor =
Fin.IdFinanciador AND Fin.TipoFinanciador LIKE 'Instituição' GROUP BY
I.NomeInstituicao ORDER BY QuantidadeFinanciamentos DESC";
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.patch("/atualizar_estado_projeto/:IdProjeto", async (req, res) => {
    const { IdProjeto } = req.params;
    if (!IdProjeto || isNaN(IdProjeto) || IdProjeto < 0) { res.status(400).send("ID do Projeto Inválido."); return; }
    const NovoEstado = (req.body && req.body.NovoEstado ? req.body.NovoEstado : null)
    if (!NovoEstado || NovoEstado == null) { res.status(400).send("Novo Estado do Projeto Inválido."); return; }
    const query = "SELECT DISTINCT IdEstado FROM Projeto WHERE IdProjeto = @IdProjeto";
    var sqlRequest = new sql.Request();
    sqlRequest.input('IdProjeto', sql.Int, IdProjeto);
    const selectResult = await sqlRequest.query(query);
    if (selectResult.IdEstado == null || selectResult.IdEstado == "[]") {
res.status(400).send("Esse ID de Estado é Inválido."); return; }
    const updateQuery = "UPDATE `Estado` WHERE IdEstado = @IdEstado SET NomeEstado = @NomeEstado";
    sqlRequest = new sql.Request();
    sqlRequest.input('IdEstado', sql.Int, selectResult.IdEstado);
    sqlRequest.input('NomeEstado', sql.Int, NovoEstado);
    const updateResult = await sqlRequest.query(updateQuery);
    res.status(200).send(updateResult.recordset);
});
```

```
queries.patch("/atualizar_tempo_membro/:IdAtividade", async (req, res) => {
    const { IdAtividade } = req.params;
    if (!IdAtividade || isNaN(IdAtividade) || IdAtividade < 0) { res.status(400).send("ID da Atividade Inválido."); return; }
    const NovoTempo = (req.body && req.body.NovoTempo ? req.body.NovoTempo : null);
    if (!NovoTempo || NovoTempo == null) { res.status(400).send("Novo Tempo a Adicionar Inválido."); return; }
    const IdMembro = (req.body && req.body.IdMembro ? req.body.IdMembro : null);
```

```
    if (!IdMembro || IdMembro == null) { res.status(400).send("ID do Membro Inválido.");
return; }
    const selectQuery = "SELECT DISTINCT IdMembro FROM Membro WHERE
IdMembro = @IdMembro";
    var sqlRequest = new sql.Request();
    sqlRequest.input('IdMembro', sql.Int, IdMembro);
    const selectResult = await sqlRequest.query(selectQuery);
    if (selectResult.IdMembro == null || selectResult.IdMembro == "[]") {
res.status(400).send("Esse ID de Membro é Inválido."); return; }
    const selectTempoQuery = "SELECT DISTINCT TempoTrabalho FROM
TempoAtividade WHERE IdMembro = @IdMembro AND IdAtividade = @IdAtividade";
    var sqlRequest = new sql.Request();
    sqlRequest.input('IdMembro', sql.Int, IdMembro);
    sqlRequest.input('IdAtividade', sql.Int, IdAtividade);
    const selectTempoResult = await sqlRequest.query(selectTempoQuery);
    if (selectTempoResult.TempoTrabalho == null || selectResult.TempoTrabalho == "[]") {
res.status(400).send("Tempo de Atividade Para Esse Membro e Atividade Inválido.");
return; }
    const tempoFinal = (parseInt(selectTempoResult.TempoTrabalho) +
parseInt(NovoTempo))
    const updateQuery = "UPDATE `TempoAtividade` WHERE IdMembro = @IdMembro
AND IdAtividade = @IdAtividade SET TempoTrabalho = @TempoTrabalho";
    sqlRequest = new sql.Request();
    sqlRequest.input('TempoTrabalho', sql.Decimal, tempoFinal);
    const updateResult = await sqlRequest.query(updateQuery);
    res.status(200).send(updateResult.recordset);
});
```

```
queries.put("/inserir_projeto", async (req, res) => {
    const { TipoProjeto_Servico, Nome, Descricao, IdData, IdInterno, IdInstituicao,
IdEstado, IdArea, IdDominio, IdMembro } = req.body;
    var query = "";
    const sqlRequest = new sql.Request();
```

```
    if (TipoProjeto_Servico == "Projeto") {
        query = "INSERT INTO `Projeto` (NomeProjeto, Descricao, IdData, IdInstituicao,
IdEstado, IdArea, IdDominio, IdMembro) VALUES (@NomeProjeto, @Descricao,
@IdData, @IdInstituicao, @IdEstado, @IdArea, @IdDominio, @IdMembro)";
        sqlRequest.input('NomeProjeto', sql.VarChar, Nome);
        sqlRequest.input('Descricao', sql.VarChar, Descricao);
        sqlRequest.input('IdData', sql.Int, IdData);
        sqlRequest.input('IdInstituicao', sql.Int, IdInstituicao);
        sqlRequest.input('IdEstado', sql.Int, IdEstado);
        sqlRequest.input('IdArea', sql.Int, IdArea);
        sqlRequest.input('IdDominio', sql.Int, IdDominio);
        sqlRequest.input('IdMembro', sql.Int, IdMembro);
    } else if (TipoProjeto_Servico == "Serviço") {
        query = "INSERT INTO `PrestacaoServico` (NomePrestacaoServico, Descricao,
IdInterno, IdData, IdEstado, IdFinanciamento) VALUES (@NomePrestacaoServico,
@Descricao, @IdInterno, @IdData, @IdEstado, @IdFinanciamento)";
        sqlRequest.input('NomePrestacaoServico', sql.VarChar, Nome);
        sqlRequest.input('Descricao', sql.VarChar, Descricao);
        sqlRequest.input('IdInterno', sql.Int, IdInterno);
        sqlRequest.input('IdData', sql.Int, IdData);
        sqlRequest.input('IdEstado', sql.Int, IdEstado);
        sqlRequest.input('IdFinanciamento', sql.Int, IdFinanciamento);
    }
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

```
queries.get("/estado_membro", async (req, res) => {
    const { IdMembro } = req.body;
    if (!IdMembro || isNaN(IdMembro) || IdMembro < 0) { res.status(400).send("ID do
Membro Inválido."); return; }
    const query = "SELECT DISTINCT TA.IdMembro, TA.TempoTrabalho, TA.IdAtividade
FROM TempoAtividade";
```

```
    const sqlRequest = new sql.Request();
    const result = await sqlRequest.query(query);
    res.status(200).send(result.recordset);
});
```

Aplicação

Neste trabalho foram seguidas as seguintes funcionalidades:

- Inserir projetos/serviços incluindo todos os elementos necessários (equipa de investigação e investigador responsável e entidades externas participantes).
- Inserir entidades financiadoras e programas.
- Alterar o estado de um projeto (por ex.: adicionar novo membro de equipa, e alterar o tempo de dedicação do IR).
- Mostrar a situação atual de um projeto/serviço.
- Mostrar a situação atual de um docente/investigador.
- Mostrar as estatísticas identificadas.
- Garantir a integridade e consistência dos dados.
- Utilizar vistas para queries específicas e gatilhos por exemplo na validação de tempo máximo associado do investigador responsável.

Para isto, foi criada uma interface em linguagem HTML onde criou-se a seguinte navegação:

- Home: Página inicial onde colocamos os Objetivos do Trabalho para que o utilizador soubesse o que consegue aceder e até mesmo fazer.
- Ver dados: Página onde o utilizador poderá consultar todos os dados tanto dos projetos como das prestações de serviço.
- Criar dados: Página onde poderá ser criado tanto projetos como prestações de serviço.
- Mudar dados: Página onde o utilizador terá a possibilidade de mudar dados de um projeto ou de uma prestação de serviço.
- Eliminar dados: Página onde será possível eliminar ou um projeto ou uma prestação de serviço.
- Créditos: Página criada para que se possa identificar os autores do trabalho, os docentes e a Unidade Curricular.

Em todas as páginas, ao clicar no botão ou “Projeto” ou “Prestação de Serviço”, poderá ver/criar/mudar/eliminar os dados que deseja dessa determinada atividade.

Ver Dados

Aqui, poderá ver todos os dados de todos(as) os(as) Projetos/Prestações de Serviço registados na Base de Dados.

Criar Dados

Selecionando esta página, terá a opção de preencher o formulário e depois, ao submeter, criará assim um(a) novo(a) Projeto/Prestação de Serviço.

Mudar Dados

Aqui, poderá alterar dados específicos como por exemplo o estado, pois este irá com o tempo evoluir.

Eliminar Dados

Por fim, nesta página poderá eliminar um(a) Projeto/Prestação de Serviço. Eliminando assim também todos os dados.

Testes

Para a inserção dos dados de teste usamos os seguintes inserts:

```
EXEC InserirPessoa 'João', 'Silva', 'joao.silva@example.com',
'123456789', 'Interno';
EXEC InserirPessoa 'Maria', 'Oliveira',
'maria.oliveira@example.com', '987654321', 'Interno';

INSERT INTO Interno (IdInterno, IdPessoa) VALUES
(1, 1),
(2, 2);

INSERT INTO Externo (IdExterno, IdPessoa) VALUES
(1, 3);

INSERT INTO Membro (IdMembro, IdPessoa, TipoMembro) VALUES
(1, 1, 'Interno'),
(2, 2, 'Interno'),
(3, 3, 'Externo');

INSERT INTO Projeto_Servico (IdProjeto_Servico,
TipoProjeto_Servico) VALUES
(1, 'Servico'),
(2, 'Projeto');

INSERT INTO DataInfo (IdData, DataInicio, DataFim) VALUES
(1, '2023-01-01', '2023-12-31'),
(2, '2024-01-01', '2024-12-31');

INSERT INTO Estado (IdEstado, NomeEstado) VALUES
(1, 'Aprovado'),
(2, 'Cancelado'),
(3, 'Concluído'),
(4, 'Em curso'),
(5, 'Encerrado'),
(6, 'Renovado'),
(7, 'Em submissão');

INSERT INTO Programa (IdPrograma, NacionalidadePrograma,
```

```
NomePrograma) VALUES
(1,'Portugal', 'Programa A'),
(2,'Espanha', 'Programa B');

INSERT INTO Instituicao (IdInstituicao, NomeInstituicao,
NacionalidadeInstituicao) VALUES
(1, 'Universidade da Beira Interior', 'Portugal'),
(2, 'Universidade de Aveiro', 'Portugal'),
(3, 'Fundação Gulbenkian', 'Portugal'),
(4, 'Universidade Complutense de Madrid', 'Espanha');

INSERT INTO DominioCientifico (IdDominio, NomeDominio) VALUES
(1, 'Ciências Naturais'),
(2, 'Engenharia');

INSERT INTO AreaCientifica (IdArea, NomeArea) VALUES
(1, 'Biologia'),
(2, 'Computação');

INSERT INTO PalavraChave (IdPalavraChave, PalavraChave) VALUES
(1, 'Inteligência Artificial'),
(2, 'Sustentabilidade');

INSERT INTO Posicao (IdPosicao, NomePosicao) VALUES
(1, 'Líder'),
(2, 'Promotor'),
(3, 'Copromotor'),
(4, 'Participante');

INSERT INTO Financiador (IdFinanciador, TipoFinanciador) VALUES
(1, 'Instituição'),
(2, 'Programa');

INSERT INTO Projeto (IdProjeto, NomeProjeto, Descricao, IdData,
IdInstituicao, IdEstado, IdArea, IdDominio, IdInterno) VALUES
(1, 'Projeto Alpha', 'Pesquisa sobre sustentabilidade', 1, 1, 1,
1, 1, 1),
(2, 'Projeto Beta', 'Desenvolvimento de IA', 2, 2, 6, 2, 2, 2);
```

```
EXEC InserirFinanciamento
    @Valor = 5000.00,
    @TipoFinanciamento = 'Competitivo',
    @OrigemFinanciamento = 'Externo',
    @IdFinanciador = 1,
    @TipoFinanciador = 'Instituicao',
    @IdProjeto_Servico = 1,
    @TipoProjeto_Servico = 'Projeto';

INSERT INTO Equipa (IdEquipa, IdProjeto) VALUES
(1, 1),

INSERT INTO Equipa_Membro (IdEquipa, IdMembro) VALUES
(1, 1)

INSERT INTO CustoElegivelEquipa (IdCustoElegivelEquipa, IdEquipa,
CustoEquipa, IdFinanciamento) VALUES
(1, 1, 600.00, 2),
(2, 2, 1000.00, 2);

INSERT INTO CustoElegivelProjeto (IdCustoElegivelProjeto,
IdProjeto, CustoProjeto, IdFinanciamento) VALUES
(1, 1, 400.00, 1),
(2, 2, 10000.00, 2);

INSERT INTO AssociarPalavraChave (IdAssociacao, IdProjeto,
IdPalavraChave) VALUES
(1, 1, 2),
(2, 2, 1);

INSERT INTO PrestacaoServico (IdPrestacaoServico,
NomePrestacaoServico, Descricao, IdInterno, IdData, IdEstado)
VALUES
(1, 'Consultoria Ambiental', 'Serviço de consultoria em práticas
sustentáveis', 1, 1, 1),
(2, 'Desenvolvimento de Software', 'Serviço de desenvolvimento de
aplicações de IA', 2, 2, 2);

INSERT INTO Atividade (IdAtividade, NomeAtividade, TipoAtividade,
IdProjeto_Servico, TipoProjeto_Servico) VALUES
```

```
(1, 'Pesquisa de Campo', 'Pesquisa', 1, 'Servico'),
(2, 'Desenvolvimento de Algoritmo', 'Desenvolvimento', 2,
'Projeto');

INSERT INTO TempoAtividade (IdMembro, TempoTrabalho, IdAtividade)
VALUES (10, DATEADD(HOUR, 500, '00:00:00'), 10);

INSERT INTO PosicaoInterno (IdPosicao, IdInterno, IdProjeto)
VALUES
(1, 1, 1),
(2, 2, 2);

INSERT INTO Publicacao (IdPublicacao, DOI, IdProjeto, IdInterno,
IdData) VALUES
(1, '10.1234/abcd.2023.001', 1, 1, 1),
(2, '10.1234/abcd.2024.002', 2, 2, 2);
```

Conclusão

Neste trabalho, foi possível consolidar o conhecimento sobre modelação de bases de dados utilizando diagramas de entidade-relacionamento (ER) e derivar o modelo relacional correspondente.

Implementamos uma base de dados no SQL Server, adicionamos os dados de teste, e desenvolvemos uma aplicação cliente que interage eficientemente com a base de dados.

Conseguimos implementar funcionalidades essenciais, como a inserção e atualização de projetos e entidades financiadoras, além de garantir a integridade dos dados através de triggers e views.

Os testes realizados confirmaram a funcionalidade esperada da aplicação. Embora tenhamos enfrentado desafios, como garantir a consistência dos dados e otimizar a aplicação para diferentes consultas, conseguimos superá-los e alcançar os objetivos propostos.

O projeto não só reforçou nosso entendimento teórico como também aprimorar as nossas habilidades práticas em SQL e desenvolvimento de aplicações.