Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Unidade Curricular: Base de Dados I

Relatório Trabalho Prático de BD1

Tema: Criação de uma base de dados para o Projeto Integrado

Realizado por: Guilherme Bento - 25193

Tiago Portugal - 30816

Lucas Santos - 27440

César Cabral - 6802

Viseu, 2025

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
Departamento de Informática

Relatório relativo ao Trabalho Prático de BD1

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de BD1 Base de dados I

Trabalho Prático de BD1

Ano Letivo 2024/25

Viseu, 2025

ÍNDICE

ÍNDICE DE FIGURAS

Introdução

Neste relatório será documentado o desenvolvimento de uma base de dados relacional trazendo a aplicação prática de seus conceitos. Este trabalho é a junção do desenvolvimento de três partes, as quais são: a aplicabilidade das técnicas de modelagem, implementação de dados e de consulta e por fim o desenvolvimento de uma aplicação cliente-servidor.

Através deste trabalho prático, evidenciamos o domínio das técnicas essenciais de Base de Dados, fazendo a aplicação dos princípios que foram abordado na docência e a implementação de estruturas de dados eficientes, e a construção de meios de trabalho de dados que prova aplicabilidade dos conhecimentos.

Neste relatório será documentado o desenvolvimento de uma base de dados relacional trazendo a aplicação prática de seus conceitos. Este trabalho é a junção do desenvolvimento das três partes do trabalho prático, evidenciando conhecimento e aplicabilidade das técnicas de modelagem, implementação de dados e de consulta e por fim o desenvolvimento de uma aplicação cliente-servidor.

Etapa 1: Modelagem de Dados

Modelo Conceptual

O modelo conceptual foi criado com o uso do Power Designer, simplificando em seu campo as entidades, os atributos, e seus relacionamentos do sistema. Nesta etapa, foram identificadas as principais entidades do domínio da aplicação, por exemplo: os utilizadores e os cursos; e definiu-se a relação entre elas, sem se precisar de preocupar com aspectos de implementação física. Determinaram-se os atributos importantes de cada entidade, chaves primárias para a identificação única dos registos e as cardinalidades dos relacionamentos, assegurando que o modelo conceptual tem como base um domínio factual e aspectos organizacionais reais. Esta visão geral de alto nível proporcionou o fundamento para as fases posteriores do desenvolvimento da base de dados.

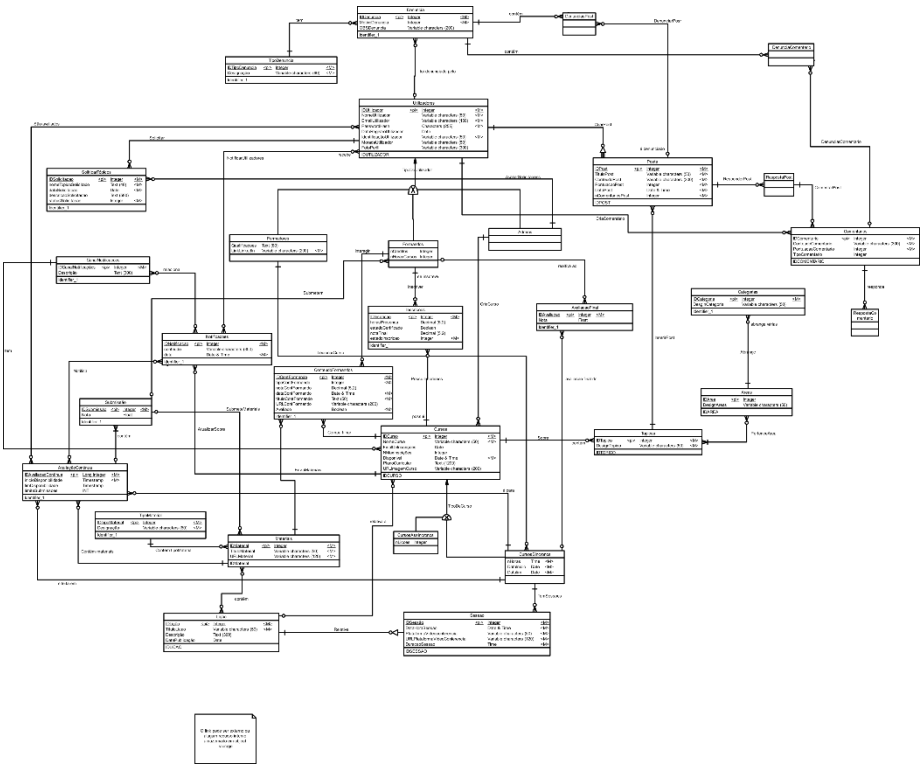


Figura 1 - CDM

Modelo Físico

A partir do modelo conceptual foi gerado o modelo físico e também com o Power Designer, traduzindo as abstrações em estruturas concretas figura de base de dados relacional. Nesta fase, foram estabelecidos os formatos de dados específicos para cada atributo, considerando restrições de tamanho e formato. As relacoções foram transformadas em chaves estrangeiras, estabelecendo a integridade referencial entre as tabelas. Também foram implementadas restrições de integridade com valores únicos e por omissão, otimizando o modelo para o ambiente do SQL Server. O modelo físico é o mais parecido com a estrutura real que será construída no sistema de gestão de base de dados.

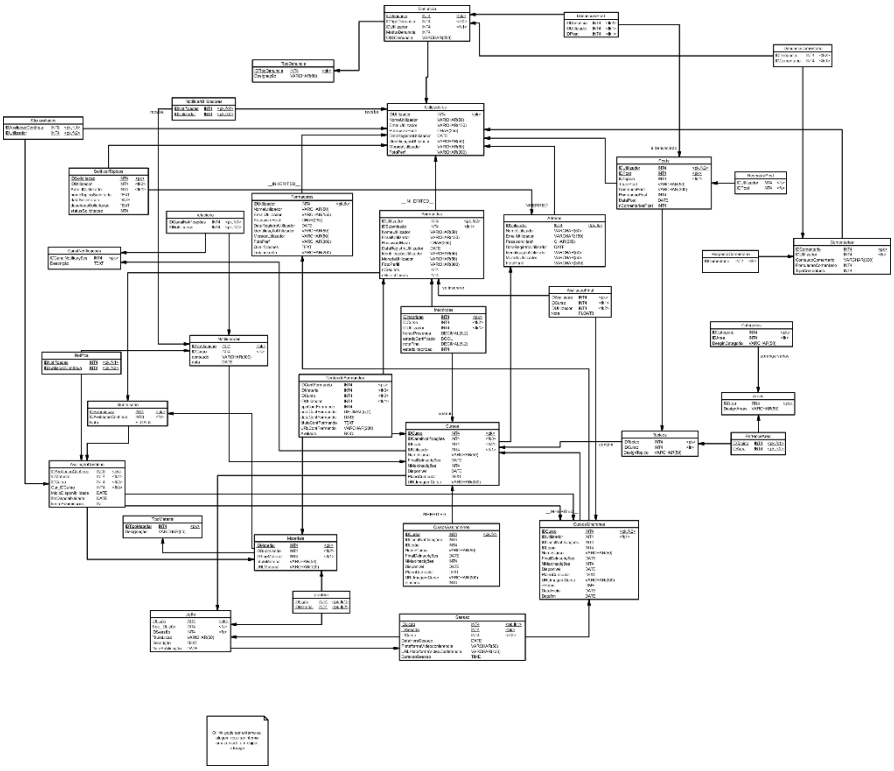


Figura 2 - PDM

Script DDL

Os scripts DDL (Data Definition Language) foram gerados automaticamente pelo Power Designer de acordo com o modelo físico, mantendo consistência, assim como, seu modelo e implantação. Os scripts têm comandos SQL para a criação de todas as tabelas, a configuração das chaves primárias e estrangeiras a implementação de restrições de integridade. Também foram incluídos para a eliminação de tabelas por ordem. Os scripts foram organizados a ponto de garantir que dependências de tabelas sejam respeitadas na hora de serem criadas, evitando erros de referência. Esta abordagem garantiu precisão na transição do modelo físico.

Script DML

Os scripts DML (Data Manipulation Language) foram desenvolvidos para inserir dados de testes em todas as tabelas da base de dados e muito em conformidade com os requisitos do trabalho prático. Foram inserido 5 registos para as tabelas sem chave estrangeira, e 10 para tabelas com chaves estrangeiras, usando dados do contexto real da aplicação. Os comandos de insert foram compilados em um único script, seguindo a ordem de manter a integridade referencial entre as tabelas. Esses dados são fundamentais para tornar testar o funcionamento da base de dados, testar consultas para ver como os objetos lógicos impelmentados, anteriormente, se comportam.

Database Diagram

O Database Diagram foi gerado no DataGrid da Jetbrains no ambiente do SQL Server, para se ter uma visualização da configuração que está sendo implementada na base de dados. Este esquema mostra todas as tabelas, seus campos e relacionamentos entre as mesmas, tudo isso facilita a compreensão da estrutura da base de dados. As chaves primárias e estrangeiras são fortemente destacadas visualmente, permitindo encontrar as dependências entre tabelas de forma rápida. O diagrama é uma documentação gráfica da implementação física e como ferramenta de comunicação, proporcionando uma visão clara e intuitiva da disposição dos dados que suporta a aplicação.

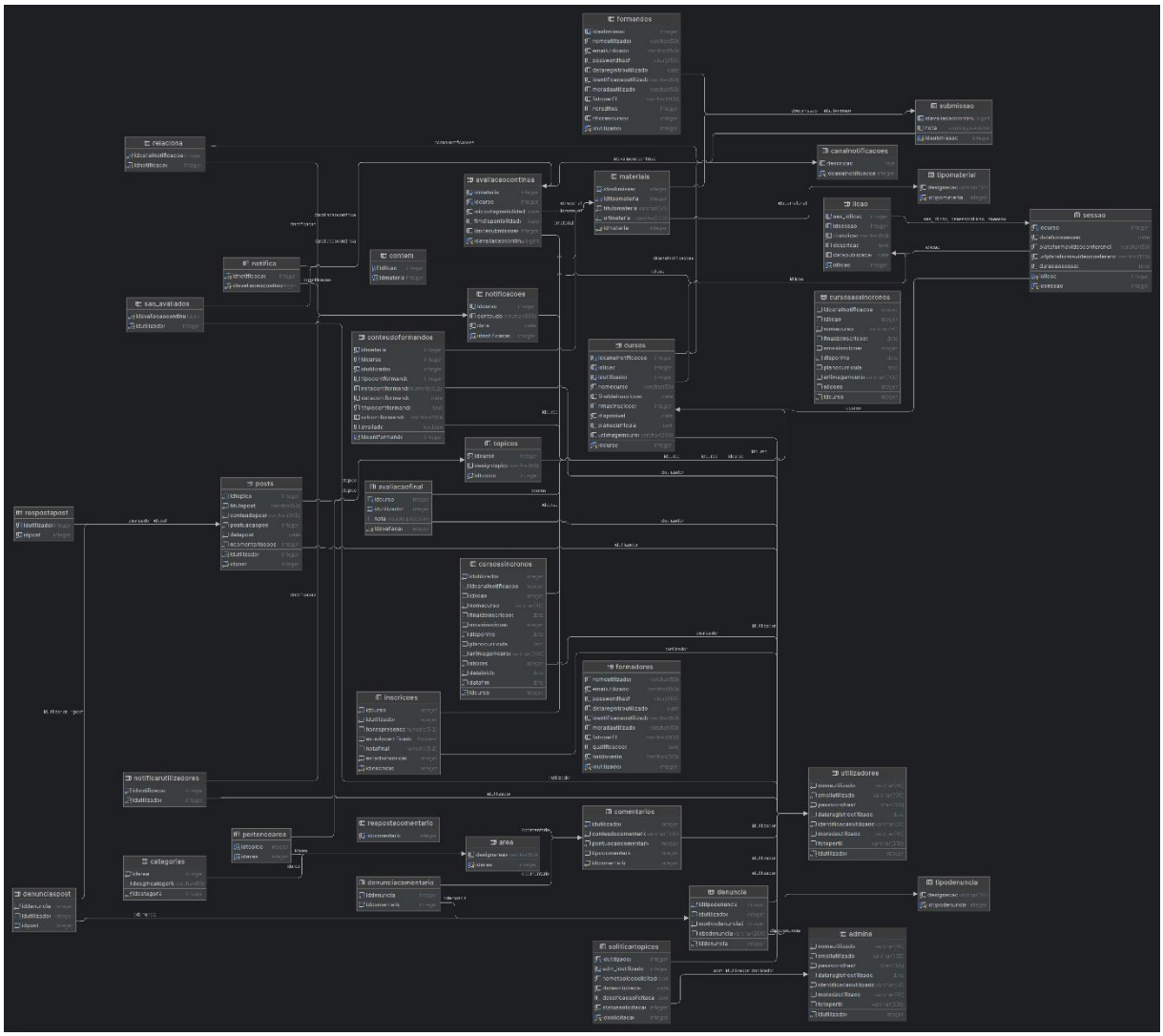


Figura 3 - Database Diagram

Etapa 2: Consultas e Objetos Lógicos

Consultas SQL

Junção externa

Utilizadores

```sql
-- Selecionar Utilizadores que não são administradores (junção externa)

SELECT idutilizador,nome,email
FROM utilizadores
LEFT OUTER JOIN admin ON utilizadores.idutilizador = admin.utilizador
WHERE (admin.idadmin IS NULL OR admin.ativo = false)
  AND utilizadores.ativo = true;
```

- Seleciona os campos idutilizador, nome e email
- Faz um outer join com o admin para encontrar correspondências entre utilizadores e administradores
- Retorna onde o utilizador não é admin ou onde o admin não está ativo, e onde o utilizador está ativo

Lista todos os usuários ativos que não são administradores ativos (incluindo aqueles que não são administradores)

Junçao interna

```sql
-- Selecionar Utilizadores inscritos no curso 'Desenvolvimento Web' (junção interna + subconsulta)

SELECT idformando, formandos.nome, email FROM inscricao JOIN

( SELECT idformando,nome,email FROM formando
JOIN utilizadores ON utilizador=idutilizador )
AS formandos ON formando=idformando

JOIN curso ON curso=idcurso

WHERE curso.nome='Desenvolvimento Web';
```

Subconsulta

- Seleciona os campos idformando, nome ,email
- Juntas as tabelas formando e utilizadores a partir do campo utilizador

Consulta

- Seleciona idformando,nome e email dos resultados
- Junta a tabela inscricao com a subconsulta formandos
- Junta também com a tabela curso

Lista todos os formandos (com seus IDs, nomes e emails) que estão inscritos no curso de "Desenvolvimento Web"

<u>Função de agregação c/having</u>

Cursos

```sql
-- Selecionar cursos com mais de 3 tópicos (Função de agregação + Having)

SELECT idcurso, nome, COUNT(topico) AS "n_tópicos"
FROM cursotopico JOIN curso ON curso=idcurso
GROUP BY idcurso, nome
HAVING COUNT(topico) > 3;
```

- Seleciona idcurso, nome e conta o número de tópicos (como "n_tópicos")
- Junta as tabelas cursotopico e curso através do campo curso
- Agrupa os resultados por idcurso e nome
- Filtra apenas os grupos que têm mais de 3 tópicos (usando HAVING)

Listar todos os cursos que possuem mais de 3 tópicos associados, mostrando o ID, nome e quantidade de tópicos de cada um

<u>Função de agregação</u>

```sql
-- Obter horas totais lecionadas em cada curso sincrono (função de agregação + subquery + junção interna)

SELECT idcurso,nome,horaslecionadas  FROM cursosincrono JOIN

(
    SELECT cursosincrono, SUM(duracaohoras) AS horaslecionadas
    FROM sessao
    WHERE CURRENT_TIMESTAMP > datahora + (duracaohoras * INTERVAL '1 hour')
    GROUP BY cursosincrono
) AS t
ON idcursosincrono = cursosincrono
JOIN curso ON cursosincrono.curso = idcurso;
```

Subconsulta

- Seleciona cada cursosincrono e soma as duracaohoras das sessões (como "horaslecionadas")
- Filtra apenas sessões já concluídas (onde a hora atual é maior que datahora + duração)
- Agrupa os resultados por cursosincrono

Consulta

- Seleciona idcurso, nome e horaslecionadas
- Junta a tabela cursosincrono com a subconsulta ***t***
- Junta também a tabela curso para obter o nome do curso

Calcula o total de horas já lecionadas em cada curso síncrono, considerando apenas sessões já concluídas, e mostrar o ID, nome do curso e horas lecionadas

Objetos Lógicos

Procedures

setStatus

```
-- Adiconar roles de admin, formando ou formador para um utilizador

CREATE OR REPLACE PROCEDURE setStatus(
    p_admin BOOLEAN,
    p_formando BOOLEAN,
    p_formador BOOLEAN,
    p_id BIGINT
)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
    IF p_admin THEN
        INSERT INTO admin(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM admin WHERE utilizador = p_id;
    END IF;

    IF p_formando THEN
        INSERT INTO formando(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM formando WHERE utilizador = p_id;
    END IF;

    IF p_formador THEN
        INSERT INTO formador(utilizador) VALUES (p_id);
    ELSE
        DELETE FROM formador WHERE utilizador = p_id;
    END IF;
END;
$$;
```

Recebe 4 parâmetros:

- p_admin: booleano que indica se o usuário deve ser admin
- p_formando: booleano que indica se o usuário deve ser formando
- p_formador: booleano que indica se o usuário deve ser formador
- p_id: ID do utilizador a ser modificado

- Para cada tipo de role (admin, formando, formador):
- Se o parâmetro for true, insere o utilizador na tabela correspondente
- Se o parâmetro for false, remove o utilizador da tabela correspondente

Atribuir ou remover roles (admin, formando, formador) de um utilizador de forma controlada através de uma única chamada

List_utilizadores

```
-- Listar utilizadores com um cursor (só porque sim)

CREATE OR REPLACE PROCEDURE list_utilizadores()
LANGUAGE plpgsql
AS $$
DECLARE
    cur CURSOR FOR
        SELECT idUtilizador, nome, email, telefone, ativo
        FROM Utilizadores;

    v_id BIGINT;
    v_nome VARCHAR(60);
    v_email VARCHAR(60);
    v_telefone CHAR(9);
    v_ativo BOOLEAN;
BEGIN
```

```
    OPEN cur;

    LOOP
        FETCH cur INTO v_id, v_nome, v_email, v_telefone, v_ativo;

        EXIT WHEN NOT FOUND;

        RAISE NOTICE 'ID: %, Nome: %, Email: %, Telefone: %, Ativo: %',
            v_id, v_nome, v_email, v_telefone, v_ativo;
    END LOOP;

    CLOSE cur;
END;
$$;
```

- Declara um cursor que seleciona todos os utilizadores com seus dados básicos
- Declara variáveis para armazenar os valores de cada coluna
- Abre o cursor e entra em um loop:
- Busca o próximo registo do cursor
- Sai do loop quando não encontrar mais registos
- Para cada registo, exibe uma mensagem (NOTICE) com os dados do utilizador
- Fecha o cursor ao final

Demonstrar o uso de cursor para percorrer e exibir todos os registos da tabela Utilizadores. A mensagem é exibida no log do PostgreSQL (não retorna resultados diretamente ao cliente)

Triggers

Roles(exemplo de adicionar + remover)

```
CREATE TRIGGER adicionar_admin
BEFORE INSERT ON admin
FOR EACH ROW
EXECUTE FUNCTION adicionar_role();
```

```
CREATE TRIGGER remover_admin
BEFORE DELETE ON admin
FOR EACH ROW
EXECUTE FUNCTION remover_role();
```

- Cria triggers chamado adicionar_admin e remover_admin

- É acionado antes (BEFORE) de qualquer operação de inserção (INSERT) na tabela admin
- Executa para cada linha afetada (FOR EACH ROW)
- Chama a função adicionar_role()e remover_role() respetivamente quando disparado

Tópico

```
CREATE TRIGGER remover_topico_trigger
BEFORE DELETE ON topico
FOR EACH ROW
EXECUTE FUNCTION remover_topico();
```

- Cria trigger chamado remover_topico_trigger

- É acionado antes (BEFORE) de qualquer operação de inserção (INSERT) na tabela topico
- Executa para cada linha afetada (FOR EACH ROW)
- Chama a função remover_topico() quando disparado

Utilizadores

```
CREATE TRIGGER desativar_utilizador_trigger
BEFORE DELETE ON utilizadores
FOR EACH ROW
EXECUTE FUNCTION desativar_utilizador();

CREATE OR REPLACE TRIGGER inserir_utilizador_trigger
BEFORE INSERT ON utilizadores
FOR EACH ROW
EXECUTE FUNCTION inserir_utilizador();
```

- Cria trigger chamado desativar_utilizador_trigger e inserir_utilizador_trigger

- É acionado antes (BEFORE) de qualquer operação de inserção (INSERT) na tabela utilizadores
- Executa para cada linha afetada (FOR EACH ROW)
- Chama a função destivar_utilizador() e inserir_utilizador() respetivamente quando disparado

UDF(funções de trigger)

Exemplo de UDF

```
CREATE OR REPLACE FUNCTION adicionar_role()
RETURNS TRIGGER AS $$
DECLARE
    userActive BOOLEAN;
    inRole BOOLEAN;
BEGIN
    EXECUTE 'SELECT ativo FROM utilizadores WHERE idutilizador = $1'
    INTO userActive
    USING NEW.utilizador;

    IF NOT userActive THEN
        RAISE EXCEPTION 'Utilizador não está ativo';
        RETURN NULL;
    END IF;

    EXECUTE format(
        'SELECT EXISTS (SELECT 1 FROM %I WHERE utilizador = $1)',
        TG_TABLE_NAME
    )
    INTO inRole
    USING NEW.utilizador;

    IF inRole THEN
        EXECUTE format(
            'UPDATE %I SET ativo = true WHERE utilizador = $1',
            TG_TABLE_NAME
        )
        USING NEW.utilizador;

        RETURN NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

1.Verifica se o utilizador está ativo:

```
EXECUTE 'SELECT ativo FROM utilizadores WHERE idutilizador = $1'
INTO userActive
USING NEW.utilizador;

IF NOT userActive THEN
    RAISE EXCEPTION 'Utilizador não está ativo';
    RETURN NULL;
END IF;
```

- Consulta o status do utilizador na tabela utilizadores
- Se o utilizador não estiver ativo, aborta a operação com uma exceção

2.Verifica se o utilizador já tem a role:

```
EXECUTE format(
    'SELECT EXISTS (SELECT 1 FROM %I WHERE utilizador = $1)',
    TG_TABLE_NAME
)
INTO inRole
USING NEW.utilizador;
```

- Usa TG_TABLE_NAME para identificar dinamicamente a tabela que disparou o trigger
- Verifica se o utilizador já existe na tabela de roles

3.Se já tiver a role:

```
IF inRole THEN
    EXECUTE format(
        'UPDATE %I SET ativo = true WHERE utilizador = $1',
        TG_TABLE_NAME
    )
    USING NEW.utilizador;

    RETURN NULL;
END IF;
```

- Atualiza o registo existente, definindo ativo = true
- Retorna NULL para cancelar a inserção (já que estamos atualizando)

4.Se não tiver a role:

```
RETURN NEW;
```

- Permite a inserção prosseguir normalmente

Index

Exemplo de index(indexNome)

```
CREATE INDEX IF NOT EXISTS "indexNome"
    ON public.curso USING btree
    (nome COLLATE pg_catalog."default" ASC NULLS LAST)
    INCLUDE(idcurso, nome, disponivel)
    WITH (deduplicate_items=True)
    TABLESPACE pg_default;
```

- Cria um index do tipo arvore-B(btree)

Coluna Indexada:

```
(nome COLLATE pg_catalog."default" ASC NULLS LAST)
```

- Indexa a coluna nome dos cursos
- Usa collation padrão para ordenação de strings
- Ordenação ascendente (ASC)
- Valores NULL aparecem no final (NULLS LAST)

Colunas Incluídas (INCLUDE):

```
INCLUDE(idcurso, nome, disponivel)
```

- Adiciona estas colunas ao índice como dados adicionais (não usados para busca)
- Permite "consultas cobertas" (quando todos os dados necessários estão no índice)
- Evita acessos desnecessários à tabela principal

Otimizações:

```
WITH (deduplicate_items=True)
TABLESPACE pg_default;
```

- *WITH (deduplicate_items=True):* Compacta valores duplicados para economizar espaço
- *TABLESPACE pg_default:* Armazena no tablespace padrão do PostgreSQL

Etapa 3: Aplicação Cliente-Servidor

Tabelas

Para a aplicação que nesta etapa foi nos pedido desenvolver decidimos criar um pequeno conjunto de páginas a partir do qual é possível fazer os seguintes :

- Listar e pesquisar utilizadores
- Desativar/ativar administradores
- Inserir utilizador como administrador

As tabelas presentes nestas operações são a `utilizadores` e `admin`.

Conexão com a base de dados

Como no projeto utilizamos Postgresql como motor da base de dados, em vez do MSsql que utilizado ao longo do semestre foi necessário instalar o pacote Npgsql para estabelecer uma conexão á nossa base de dados remota.
Nas fichas anteriores fizemos uso da classe SQLDataSource e do SQLDataClient para proceder á manipulação de dados. Esta biblioteca fornece classes que implementam estas mesmas interfaces, pelo que foi possível incorporar os dados em widgets pré-existentes, utilizados em passadas tarefas orientadas.

```
string connectionString =
  "Host=thesoftskills.duckdns.org; \
   Username=grupo;Password=**************; \
   Database=thesoftskills";

var dataSource = NpgsqlDataSource.Create(connectionString);

using (var command = dataSource.CreateCommand("Comando SQL"))
using (var reader = await command.ExecuteReaderAsync())
{
   var dt = new DataTable();
       dt.Load(reader);
       AdminsTable.DataSource = dt;
       AdminsTable.DataBind();
}
```

Páginas da aplicação

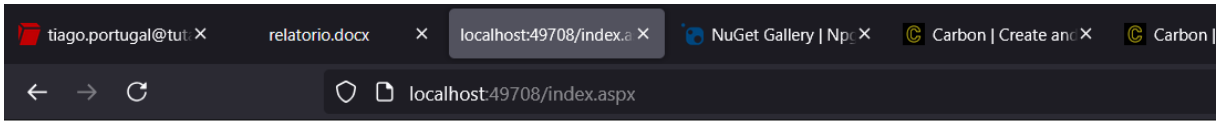
A aplicação é composta por 3 páginas, um índex onde é possível navegar para as operações disponíveis, duas para interagir com as tabelas na base de dados utilizadores e administradores respetivamente.

Os caminhos são os seguintes:

- /index.aspx : index
- /listarUtilizadores.aspx : utilizadores
- /gerirAdmins.aspx : administradores

Index

Esta página tem o aspeto apresentado abaixo:



Bem vindo á pagina backoffice protótipo do Thesoftskills

Não temos muitas funcionalidades, ainda estamos numa fase de desenvolvimento as que temos são as seguintes :

Listar Utilizadores
Gerir Admins

A partir daqui conseguimos navegar para a página dos utilizadores e dos administradores.

Utilizadores

Nesta página é possível pesquisar por um utilizador em específico ou ver todos os utilizadores.

A pesquisa é feita com a seguinte query fazendo uso do operador LIKE

```
SELECT idutilizador, nome, email FROM utilizadores WHERE nome
LIKE '{pesquisaInput.Text}%' ORDER BY idutilizador"
```

E a listagem é uma simples seleção de todos os utilizadores ativos

```
SELECT idutilizador, nome, email FROM utilizadores WHERE ativo=true ORDER BY idutilizador
```

Tendo a aparecia abaixo

Administradores

Na página de administradores podemos ver o seu nome e mudar o seu estado, é também possível inserir um utilizador nesta tabela a partir do seu id.

Ao inserir um administrador na aplicação é executada a instrução abaixo

```
INSERT INTO admin(utilizador) VALUES ({id})
```

E o utilizador é alertado se a inserção foi feita com sucesso, neste exemplo foi inserido na tabela o utilizador com id 40

E é nos confirmada a sua inserção.

Como podemos ver o utilizador está agora na tabela, clicando no botão mudar estado

Todos os Administradores

IdAdmin	Utilizador	Nome	Estado	
3	8	Sofia Rocha	ativo	Mudar estado
4	3	Carlos Santos	ativo	Mudar estado
28	37	João Matos	ativo	Mudar estado
29	38	Pedro Santos	ativo	Mudar estado
33	42	Bernardo Melo	ativo	Mudar estado
34	10	Inês Mendes	ativo	Mudar estado
36	1	João Silva	ativo	Mudar estado
37	2	Maria Oliveira	ativo	Mudar estado
39	41	Daniel Costa	ativo	Mudar estado
41	40	Rui Sebastião	ativo	Mudar estado

Inserir Admin

Id Utilizador : 40 Inserir

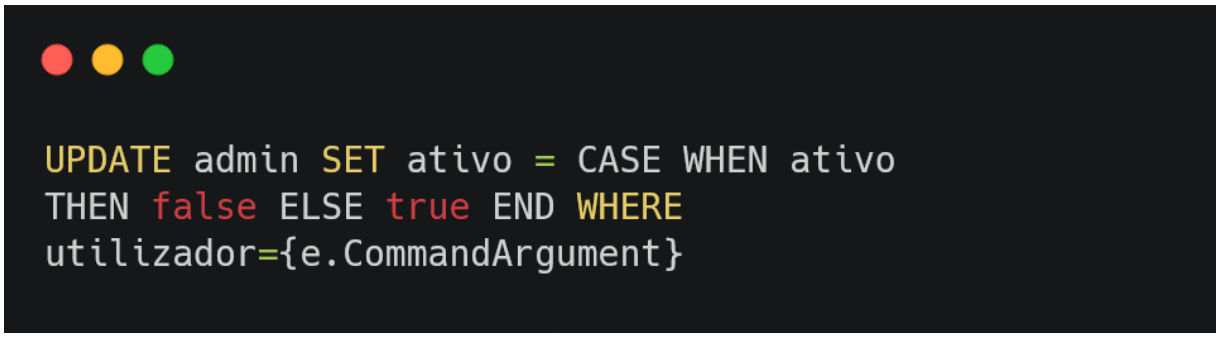
Navegação

Home
Listar Utilizadores

Conseguimos alterar o seu estado de ativo para desativo

39	41	Daniel Costa	ativo	Mudar estado
41	40	Rui Sebastião	desativo	Mudar estado

Este ação é feita através desta query



Onde o *`e.CommandArgument`* é o id do utilizador.