deti
universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Projeto Final – Estoril Track Manager

Fundamentos de Bases de Dados

Ano Letivo: 2025/2026

Trabalho realizado por:

Tomás Xavier, NMEC: 125493

Guilherme Gomes, NMEC: 125438

Índice

Introdução

Como tema para o nosso projeto, decidimos desenvolver uma base de dados para o autódromo do Estoril. Este sistema foi projetado para centralizar a gestão de competições, permitindo que utilizador comum consiga ver Eventos (sejam estes passados, atuais ou futuros), Recordes, Pilotos e Equipas.

Para além da consulta pública, o sistema fornece também interfaces para três tipos diferentes de utilizadores: Diretores de Corrida, que gerem os eventos e sessões; Diretores de Equipa, que são responsáveis pela inscrição de carros e pilotos da sua equipa em eventos e sessões; e Técnicos de Pista, que registam os tempos de volta dos pilotos em tempo real, assim como as condições meteorológicas da pista.

A nossa interface foi desenvolvida em Python, com o uso do framework Flask, ligada a um servidor SQL Server.

Estrutura da Pasta do Projeto

```
Project_FBD/
|   app.py
|   config.py
|   dados.sql
|
+---routes/
|       auth.py
|       diretor_corrida.py
|       diretor_equipa.py
|       public.py
|       tecnico_pista.py
|
+---static/
|   +---assets/
|   \---css/
|
+---templates/
|       homepage.html
|       events.html
|       records.html
|       ...
|
\---utils/
    |   database.py
```

Requisitos Funcionais

Aceder a informação sobre:

-pilotos;

-equipas;

-eventos;

-recordes.

Gestor de Equipa pode:

-criar uma equipa;

-adicionar/remover pilotos da sua equipa;

-adicionar/remover carros da sua equipa;

-inscrever/desinscrever pilotos e carros de eventos e sessões.

Diretor de Corrida pode:

-criar/apagar um evento;

-adicionar/remover sessões do seu evento;

-ver histórico de eventos.

Técnico de Pista pode:

-definir condições meteorológicas da pista a cada volta;

-definir os tempos de volta de cada piloto.

Entidades

Utilizador: Representa qualquer pessoa com acesso ao sistema do autódromo. É a entidade base que armazena as credenciais de acesso (username, password), o email e o nome.

Diretor de Corrida: Tipo de utilizador responsável pela gestão global do calendário do autódromo. Tem autoridade para criar eventos, definir sessões e gerir o status de conclusão das provas.

Diretor de Equipa: Tipo de utilizador que gere uma equipa específica. É responsável pela frota de carros, pela contratação de pilotos e pela inscrição da equipa em eventos e sessões oficiais.

Técnico de Pista: Tipo de utilizador responsável pelo registo técnico durante as provas. É quem insere os tempos de volta e monitoriza as condições meteorológicas da pista em tempo real.

Equipa: Representa uma organização desportiva associada a um país. Cada equipa é gerida por um Diretor de Equipa e possui um conjunto próprio de pilotos e veículos.

Piloto: Representa o atleta que conduz os veículos. Contém informações como número de licença, data de nascimento e nacionalidade.

Carro: Representa o veículo utilizado nas competições. É identificado pelo VIN (Vehicle Identification Number) e inclui especificações técnicas como marca, modelo, categoria, potência, peso e tipo de motor.

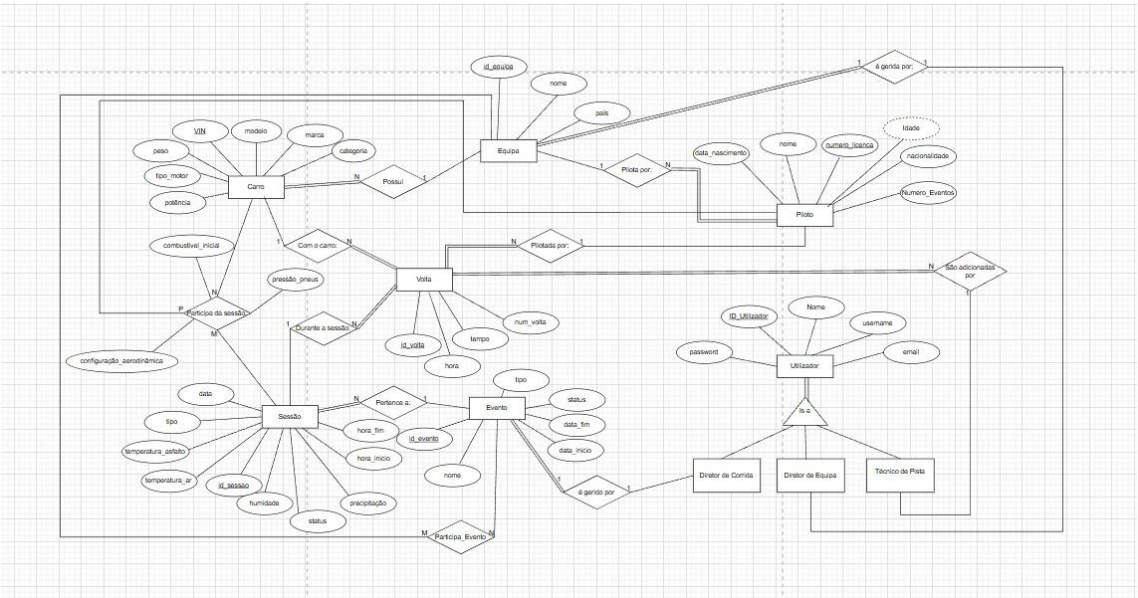
Evento: Representa uma competição no calendário do autódromo. Possui um nome, tipo, datas de início e fim, e um status que indica se está agendado, a decorrer ou concluído.

Sessão: Representa uma divisão específica de um evento (ex: Treinos Livres, Qualificação, Corrida). Armazena dados meteorológicos como temperatura do ar, do asfalto, humidade e precipitação.

Volta: Representa o registo de performance de um piloto com um carro numa determinada sessão. Contém o tempo (em milissegundos), o número da volta e a pressão dos pneus utilizada no momento.

Diagramas

Diagrama Entidade-Relação



Diferenças da última apresentação:

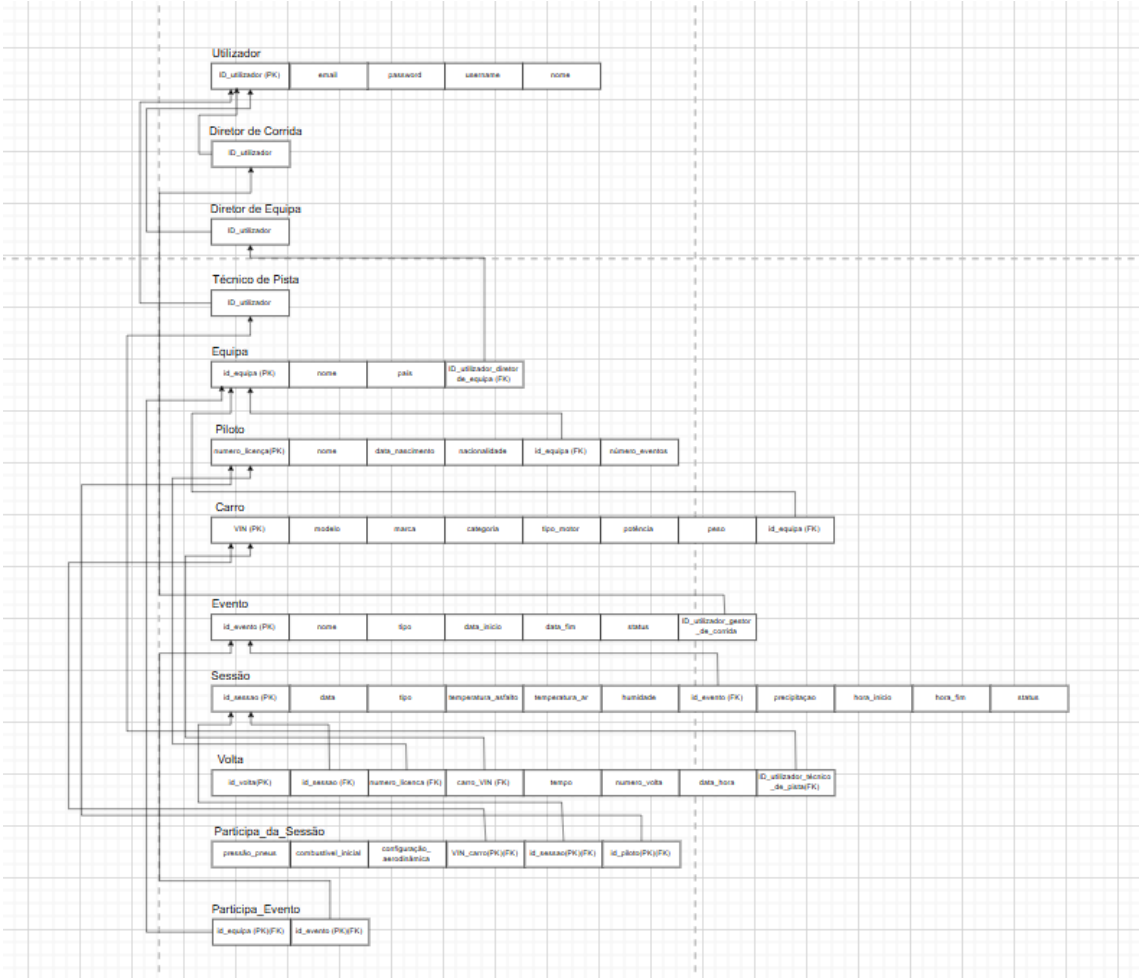
-Remoção da tabela de permissões, visto que acabou por ser uma tabela desnecessária para a nossa implementação.

-Substituímos a relação Está_configurado, pela relação Participa_da_sessao, incluindo também a tabela Piloto na relação de forma a facilitar a implementação.

-Criámos uma nova relação Participa_Evento entre Equipa e Evento, de forma a facilitar a implementação.

-Adicionámos o atributo derivado “Idade” na tabela Piloto, para que o pudéssemos apresentar na aba de detalhes do piloto.

Esquema Relacional



Normalização

Primeira Forma Normal (1FN): todos os atributos das tabelas contêm apenas valores atómicos e não existem grupos repetidos.

-Todas as tabelas do nosso projeto possuem uma Primary_Key única e não utilizam colunas compostas ou multi-valoradas, logo está na 1FN.

Segunda Forma Normal (2FN): cumpre a 1FN e todos os atributos não-chave devem depender da totalidade da chave primária, o que se verifica no nosso projeto.

Terceira Forma Normal (3FN): cumpre a 2FN e não existem dependências transitivas entre atributos não-chave, o que se verifica no nosso projeto.

-NOTA: embora o nosso sistema cumpra a 3FN, mantivémos o campo numero_eventos na tabela Piloto por motivos de performance. Esta desnormalização é gerida automaticamente pelo trigger trg_AtualizarNumeroEventos, que assegura que esta redundância não cause inconsistência.

O nosso sistema **não** cumpre a **BCNF**. Optámos por não normalizar até à BCNF porque esta decomposição levaria à criação de tabelas excessivamente fragmentadas, que prejudicaria a performance das consultas à BD.

-

Queries

DDL – Data Definition Language

```
CREATE TABLE Utilizador (
    ID_utilizador INT IDENTITY(1,1) PRIMARY KEY,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    username VARCHAR(50) NOT NULL UNIQUE,
    nome VARCHAR(100) NOT NULL
);

CREATE TABLE Diretor_de_Corrida (
    ID_utilizador INT PRIMARY KEY,
    FOREIGN KEY (ID_utilizador) REFERENCES Utilizador(ID_utilizador) ON DELETE CASCADE
);

CREATE TABLE Diretor_de_Equipa (
    ID_utilizador INT PRIMARY KEY,
    FOREIGN KEY (ID_utilizador) REFERENCES Utilizador(ID_utilizador) ON DELETE CASCADE
);
```

```
CREATE TABLE Tecnico_de_Pista (
    ID_utilizador INT PRIMARY KEY,
    FOREIGN KEY (ID_utilizador) REFERENCES Utilizador(ID_utilizador) ON DELETE CASCADE
);

CREATE TABLE Equipa (
    id_equipa INT IDENTITY(1,1) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL UNIQUE,
    pais VARCHAR(50) NOT NULL,
    ID_utilizador_diretor_de_equipa INT UNIQUE,
    FOREIGN KEY (ID_utilizador_diretor_de_equipa) REFERENCES Diretor_de_Equipa(ID_utilizador)
);

CREATE TABLE Piloto (
    numero_licenca INT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    data_nascimento DATE NOT NULL,
    nacionalidade VARCHAR(50) NOT NULL,
    id_equipa INT,
    numero_eventos INT DEFAULT 0,
    FOREIGN KEY (id_equipa) REFERENCES Equipa(id_equipa)
);
```

```sql
CREATE TABLE Carro (
    VIN VARCHAR(50) PRIMARY KEY,
    modelo VARCHAR(100) NOT NULL,
    marca VARCHAR(100) NOT NULL,
    categoria VARCHAR(50) NOT NULL,
    tipo_motor VARCHAR(50),
    potencia INT,
    peso INT,
    id_equipa INT,
    FOREIGN KEY (id_equipa) REFERENCES Equipa(id_equipa)
);

CREATE TABLE Evento (
    id_evento INT IDENTITY(1,1) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    data_inicio DATE NOT NULL,
    data_fim DATE NOT NULL,
    status VARCHAR(50) NOT NULL DEFAULT 'Por Iniciar',
    ID_utilizador_gestor_de_corrida INT,
    FOREIGN KEY (ID_utilizador_gestor_de_corrida) REFERENCES Diretor_de_Corrida(ID_utilizador),
    CONSTRAINT CK_Evento_Datas CHECK (data_fim >= data_inicio)
);
```

```sql
CREATE TABLE Sessao (
    id_sessao INT IDENTITY(1,1) PRIMARY KEY,
    data DATE NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    temperatura_asfalto INT,
    temperatura_ar INT,
    humidade INT,
    precipitacao INT,
    id_evento INT NOT NULL,
    hora_inicio TIME,
    hora_fim TIME,
    status VARCHAR(20) DEFAULT 'Por Iniciar',
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento) ON DELETE CASCADE
);
```

```sql
CREATE TABLE Volta (
    id_volta INT IDENTITY(1,1) PRIMARY KEY,
    id_sessao INT NOT NULL,
    numero_licenca INT NOT NULL,
    carro_VIN VARCHAR(50) NOT NULL,
    tempo INT NOT NULL,
    numero_volta INT NOT NULL,
    data_hora DATETIME DEFAULT GETDATE(),
    pressao_pneus INT,
    ID_utilizador_tecnico_de_pista INT,
    FOREIGN KEY (id_sessao) REFERENCES Sessao(id_sessao),
    FOREIGN KEY (numero_licenca) REFERENCES Piloto(numero_licenca),
    FOREIGN KEY (carro_VIN) REFERENCES Carro(VIN),
    FOREIGN KEY (ID_utilizador_tecnico_de_pista) REFERENCES Tecnico_de_Pista(ID_utilizador),
    CONSTRAINT UQ_Volta_Sessao_Carro_Numero UNIQUE (id_sessao, carro_VIN, numero_volta)
);
```

```sql
CREATE TABLE Participa_Sessao (
    id_sessao INT NOT NULL,
    numero_licenca INT NOT NULL,
    VIN_carro VARCHAR(50) NOT NULL,
    combustivel_inicial INT,
    pressao_pneus INT,
    configuracao_aerodinamica INT,
    PRIMARY KEY (id_sessao, numero_licenca, VIN_carro),
    FOREIGN KEY (id_sessao) REFERENCES Sessao(id_sessao) ON DELETE CASCADE,
    FOREIGN KEY (numero_licenca) REFERENCES Piloto(numero_licenca),
    FOREIGN KEY (VIN_carro) REFERENCES Carro(VIN),
    CONSTRAINT UQ_Piloto_Sessao UNIQUE (id_sessao, numero_licenca),
    CONSTRAINT UQ_Carro_Sessao UNIQUE (id_sessao, VIN_carro)
);

CREATE TABLE Participa_Evento (
    id_equipa INT NOT NULL,
    id_evento INT NOT NULL,
    PRIMARY KEY (id_equipa, id_evento),
    FOREIGN KEY (id_equipa) REFERENCES Equipa(id_equipa),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento) ON DELETE CASCADE
);
```

DML – Data Manipulation Language e Formulários na Interface

Para realizarmos a inserção de dados recorremos a ferramentas de inteligência artificial para que nos gerassem os dados a serem inseridos. Para além dos dados apresentados abaixo, temos também, na base de dados, dados que foram introduzidas de forma dinâmica através do website.

```
-- UTILIZADORES E ROLES
SET IDENTITY_INSERT Utilizador ON;
INSERT INTO Utilizador (ID_utilizador, username, email, password, nome) VALUES
(100, 'admin_corrida', 'corrida@estoril.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'Carlos Silva'),
(101, 'tecnico1', 'tecnico1@estoril.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'António Santos'),
(102, 'tecnico2', 'tecnico2@estoril.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'João Ferreira'),
(103, 'equipa_ferrari', 'ferrari@racing.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'Marco Rossi'),
(104, 'equipa_mercedes', 'mercedes@racing.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'Hans Müller'),
(105, 'equipa_redbull', 'redbull@racing.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'Max Verstappen Sr'),
(106, 'equipa_mclaren', 'mclaren@racing.pt', 'a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3', 'Zak Brown');
SET IDENTITY_INSERT Utilizador OFF;
-- Diretor de Corrida
INSERT INTO Diretor_de_Corrida (ID_utilizador) VALUES (100);
-- Técnicos de Pista
INSERT INTO Tecnico_de_Pista (ID_utilizador) VALUES (101), (102);
-- Diretores de Equipa
INSERT INTO Diretor_de_Equipa (ID_utilizador) VALUES (103), (104), (105), (106);
-- EQUIPAS
SET IDENTITY_INSERT Equipa ON;
INSERT INTO Equipa (id_equipa, nome, pais, ID_utilizador_diretor_de_equipa) VALUES
(100, 'Scuderia Ferrari', 'Itália', 103),
(101, 'Mercedes-AMG Petronas', 'Alemanha', 104),
(102, 'Red Bull Racing', 'Áustria', 105),
(103, 'McLaren Racing', 'Reino Unido', 106);
```

```
SET IDENTITY_INSERT Equipa OFF;
-- PILOTOS
INSERT INTO Piloto (numero_licenca, nome, data_nascimento, nacionalidade, id_equipa, numero_eventos) VALUES
-- Ferrari
(116, 'Charles Leclerc', '1997-10-16', 'Mónaco', 100, 5),
(155, 'Carlos Sainz', '1994-09-01', 'Espanha', 100, 5),
-- Mercedes
(144, 'Lewis Hamilton', '1985-01-07', 'Reino Unido', 101, 10),
(163, 'George Russell', '1998-02-15', 'Reino Unido', 101, 3),
-- Red Bull
(101, 'Max Verstappen', '1997-09-30', 'Países Baixos', 102, 8),
(111, 'Sergio Pérez', '1990-01-26', 'México', 102, 6),
-- McLaren
(104, 'Lando Norris', '1999-11-13', 'Reino Unido', 103, 4),
(181, 'Oscar Piastri', '2001-04-06', 'Austrália', 103, 2),
-- Pilotos sem equipa (disponíveis)
(114, 'Fernando Alonso', '1981-07-29', 'Espanha', NULL, 15),
(177, 'Valtteri Bottas', '1989-08-28', 'Finlândia', NULL, 7);
```

```sql
-- CARROS
INSERT INTO Carro (VIN, modelo, marca, categoria, tipo_motor, potencia, peso) VALUES
-- Ferrari
('FERR2024001', 'SF-24', 'Ferrari', 'GT3', 'V8 Turbo', 680, 1300),
('FERR2024002', 'SF-24 Evo', 'Ferrari', 'GT3', 'V8 Turbo', 700, 1280),
-- Mercedes
('MERC2024001', 'AMG GT3', 'Mercedes', 'GT3', 'V8 Biturbo', 650, 1320),
('MERC2024002', 'AMG GT3 Evo', 'Mercedes', 'GT3', 'V8 Biturbo', 670, 1300),
-- Red Bull
('REDB2024001', 'RB20 GT', 'Red Bull', 'GT3', 'V6 Híbrido', 720, 1250),
('REDB2024002', 'RB20 GT Pro', 'Red Bull', 'GT3', 'V6 Híbrido', 740, 1240),
-- McLaren
('MCLA2024001', '720S GT3', 'McLaren', 'GT3', 'V8 Biturbo', 690, 1290),
('MCLA2024002', '720S GT3 Evo', 'McLaren', 'GT3', 'V8 Biturbo', 710, 1270),
-- Carros sem equipa
('PORS2024001', '911 GT3 R', 'Porsche', 'GT3', 'Flat-6', 565, 1280),
('ASTO2024001', 'Vantage GT3', 'Aston Martin', 'GT3', 'V8 Turbo', 600, 1340);
-- Vincular carros às equipas
UPDATE Carro SET id_equipa = 100 WHERE VIN IN ('FERR2024001', 'FERR2024002');
UPDATE Carro SET id_equipa = 101 WHERE VIN IN ('MERC2024001', 'MERC2024002');
UPDATE Carro SET id_equipa = 102 WHERE VIN IN ('REDB2024001', 'REDB2024002');
UPDATE Carro SET id_equipa = 103 WHERE VIN IN ('MCLA2024001', 'MCLA2024002');
-- EVENTOS
SET IDENTITY_INSERT Evento ON;
INSERT INTO Evento (id_evento, nome, tipo, data_inicio, data_fim, status, ID_utilizador_gestor_de_corrida) VALUES
(100, 'GP Estoril 2025', 'Corrida', '2025-01-15', '2025-01-17', 'Por Iniciar', 100),
(101, 'Treinos de Inverno', 'Treino', '2025-01-20', '2025-01-22', 'Por Iniciar', 100),
(102, '24 Horas de Estoril', 'Resistência', '2025-02-10', '2025-02-11', 'Por Iniciar', 100),
(103, 'Test Day Primavera', 'Treino', '2024-12-01', '2024-12-01', 'Concluído', 100),
(104, 'GT Challenge Round 1', 'Corrida', '2024-12-15', '2024-12-16', 'Concluído', 100);
SET IDENTITY_INSERT Evento OFF;
-- SESSÕES
SET IDENTITY_INSERT Sessao ON;
INSERT INTO Sessao (id_sessao, data, tipo, hora_inicio, hora_fim, id_evento, status, temperatura_asfalto, temperatura_ar, humidade, precipitacao) VALUES
-- GP Estoril 2025
(100, '2025-01-15', 'Treino Livre 1', '09:00', '10:30', 100, 'Por Iniciar', NULL, NULL, NULL, NULL),
(101, '2025-01-15', 'Treino Livre 2', '14:00', '15:30', 100, 'Por Iniciar', NULL, NULL, NULL, NULL),
(102, '2025-01-16', 'Qualificação', '10:00', '11:00', 100, 'Por Iniciar', NULL, NULL, NULL, NULL),
(103, '2025-01-17', 'Corrida', '15:00', '17:00', 100, 'Por Iniciar', NULL, NULL, NULL, NULL),
-- Treinos de Inverno
(104, '2025-01-20', 'Sessão Manhã', '09:00', '12:00', 101, 'Por Iniciar', NULL, NULL, NULL, NULL),
(105, '2025-01-20', 'Sessão Tarde', '14:00', '18:00', 101, 'Por Iniciar', NULL, NULL, NULL, NULL),
-- Evento Concluído (Test Day)
(106, '2024-12-01', 'Treino', '09:00', '17:00', 103, 'Concluída', 28, 18, 45, 0),
-- GT Challenge (Concluído)
(107, '2024-12-15', 'Qualificação', '10:00', '11:00', 104, 'Concluída', 25, 15, 50, 0),
(108, '2024-12-16', 'Corrida', '14:00', '16:00', 104, 'Concluída', 30, 20, 40, 0);
SET IDENTITY_INSERT Sessao OFF;
-- PARTICIPAÇÃO EM EVENTOS
INSERT INTO Participa_Evento (id_equipa, id_evento) VALUES
(100, 100), (101, 100), (102, 100), (103, 100),  -- Todas no GP Estoril
(100, 101), (101, 101),                          -- Ferrari e Mercedes nos Treinos
(100, 103), (101, 103), (102, 103),              -- Test Day concluído
(100, 104), (101, 104), (102, 104), (103, 104);  -- GT Challenge concluído
-- PARTICIPAÇÃO EM SESSÕES (Eventos concluídos)
INSERT INTO Participa_Sessao (id_sessao, numero_licenca, VIN_carro, combustivel_inicial, pressao_pneus, configuracao_aerodinamica) VALUES
-- Test Day (sessão 106)
(106, 116, 'FERR2024001', 80, 28, 3),
(106, 144, 'MERC2024001', 75, 27, 4),
(106, 101, 'REDB2024001', 85, 29, 2),
-- GT Challenge Qualificação (sessão 107)
(107, 116, 'FERR2024001', 50, 28, 5),
(107, 155, 'FERR2024002', 50, 28, 5),
(107, 144, 'MERC2024001', 55, 27, 4),
(107, 101, 'REDB2024001', 60, 29, 3),
-- GT Challenge Corrida (sessão 108)
(108, 116, 'FERR2024001', 100, 28, 3),
(108, 155, 'FERR2024002', 100, 28, 3),
(108, 144, 'MERC2024001', 95, 27, 4),
(108, 163, 'MERC2024002', 95, 27, 4),
(108, 101, 'REDB2024001', 110, 29, 2),
(108, 104, 'MCLA2024001', 90, 28, 3);
```

Associámos os seguintes formulários a Stored Procedures em vez de comandos DML diretos no python, de forma a termos uma maior segurança contra ataques de SQL Injection e para que não existam dados inconsistentes caso ocorra um erro a meio dum processo.

Formulário	Ação do utilizador	Operação DML	Descrição
Página de Registo	Criar conta nova	*sp_RegistarUtilizador*	Executa um **INSERT** na tabela *Utilizador* e na tabela de perfil correspondente.
Gestão de Equipas	Criar Equipa	*sp_CriarEquipa*	Realiza um **INSERT** na tabela *Equipa* associando-a ao ID do Diretor logado.
Painel do Diretor	Adicionar Piloto	*sp_AdicionarPiloto*	Executa um **INSERT** na tabela *Piloto* após validar a unicidade da licença.
Painel do Técnico	Registar Volta	*sp_RegistarVolta*	Converte o tempo e faz um **INSERT** na tabela *Volta*.
Página de Eventos	Cancelar Evento	*sp_CancelarEventoCompleto*	Executa um **DELETE** em cascata para remover o evento e todos os dados dependentes.
Definições de Perfil	Alterar Nome/Senha	*UPDATE Utilizador*	Comando DML direto (**UPDATE**) para modificar atributos do perfil do utilizador.
Gestão de Frota	Editar Carro	*sp_AtualizarCarro*	Executa um **UPDATE** nos atributos técnicos da tabela *Carro*.
Inscrições	Inscrever em Sessão	*sp_InscreverSessao*	Realiza um **INSERT** na tabela de associação *Participa_Sessao*.

Views

Recorremos à criação de views para otimizar o acesso aos dados e simplificar a lógica no lado do servidor. O uso destas permitiu-nos encapsular joins complexos e criar uma camada de abstração que simplificou a comunicação entre a base de dados e o framework Flask. Implementámos duas views fundamentais no nosso projeto:

vw_RankingPilotos: Esta view centraliza a informação necessária para a listagem e filtragem de pilotos. Ao consolidar os dados da tabela Piloto com os nomes das respetivas equipas e integrar a função escalar *fn_IdadePiloto*, eliminámos a necessidade de realizar múltiplos *left joins* e cálculos de data em tempo de execução na aplicação.

```
CREATE VIEW vw_RankingPilotos AS
SELECT
    P.numero_licenca,
    P.nome,
    P.data_nascimento,
    P.nacionalidade,
    E.nome AS nome_equipa,
    P.numero_eventos,
    P.id_equipa,
    dbo.fn_IdadePiloto(P.numero_licenca) as idade
FROM Piloto P
LEFT JOIN Equipa E ON P.id_equipa = E.id_equipa;
GO
```

vw_Recordes: Esta view foi desenvolvida especificamente para a página de tempos, ela agrega cinco tabelas distintas para fornecer um histórico de performance. A sua grande vantagem reside na integração das funções de formatação de tempo e no cálculo dinâmico do *gap* para o melhor tempo feito, permitindo que a tabela de recordes seja alimentada por uma única consulta simples, o que melhora drasticamente a performance da página.

```
CREATE VIEW vw_Recordes AS
SELECT
    dbo.fn_FormatarTempoMS(V.tempo) as tempo_formatado,
    P.nome as piloto,
    C.marca + ' ' + C.modelo as carro,
    Ev.nome as evento,
    V.tempo,
    S.data,
    V.id_volta,
    dbo.fn_FormatarTempoMS(dbo.fn_GapParaMelhor(V.id_volta)) as gap
FROM Volta V
INNER JOIN Piloto P ON V.numero_licenca = P.numero_licenca
INNER JOIN Carro C ON V.carro_VIN = C.VIN
INNER JOIN Sessao S ON V.id_sessao = S.id_sessao
INNER JOIN Evento Ev ON S.id_evento = Ev.id_evento;
GO
```

UDF'S – User Defined Functions

Implementámos quatro UDF's no nosso projeto, que nos permitiram tornar o código mais legiivel, assim como as consultas de dados SQL mais eficientes.

Tratamento da Cronometragem(**TempoParaMs** e **fn_FormatarTempoMS**): Estas funções formam o sistema de tempos do nosso projeto. Enquanto que a primeira converte entradas de texto em milissegundos para ser possível realizar cálculos no SQL, a segunda faz o oposto, formatando os dados para uma leitura humana "mm:ss:ms". Isto assegura um armazenamento eficiente e uma exibição precisa dos dados.

```
CREATE FUNCTION dbo.TempoParaMs (@tempo VARCHAR(12))
RETURNS INT
AS
BEGIN
    DECLARE @minutos INT, @segundos INT, @ms INT;
    SET @minutos = CAST(SUBSTRING(@tempo, 1, 2) AS INT);
    SET @segundos = CAST(SUBSTRING(@tempo, 4, 2) AS INT);
    SET @ms = CAST(SUBSTRING(@tempo, 7, 2) AS INT) * 10;
    RETURN (@minutos * 60000) + (@segundos * 1000) + @ms;
END;
GO
```

```
CREATE FUNCTION dbo.fn_FormatarTempoMS (@ms INT)
RETURNS VARCHAR(20)
AS
BEGIN
    DECLARE @minutos INT, @segundos INT, @milissegundos INT;
    DECLARE @resultado VARCHAR(20);

    SET @minutos = @ms / 60000;
    SET @segundos = (@ms % 60000) / 1000;
    SET @milissegundos = @ms % 1000;

    SET @resultado = RIGHT('0' + CAST(@minutos AS VARCHAR), 2) + ':' +
                     RIGHT('0' + CAST(@segundos AS VARCHAR), 2) + '.' +
                     RIGHT('00' + CAST(@milissegundos AS VARCHAR), 3);
    RETURN @resultado;
END;
GO
```

fn_GapParaMelhor: Esta função permitiu-nos calcular a diferença de tempo entre uma volta específica e a melhor volta da mesma sessão. Isto permite que a tabela de recordes mostre instantaneamente a distância de tempo entre pilotos, sem ser necessário a criação de queries complexas.

```sql
CREATE FUNCTION dbo.fn_GapParaMelhor(@id_volta INT)
RETURNS INT AS BEGIN
    DECLARE @tempo INT, @id_sessao INT;
    SELECT @tempo = tempo, @id_sessao = id_sessao FROM Volta WHERE id_volta = @id_volta;
    RETURN @tempo - (SELECT MIN(tempo) FROM Volta WHERE id_sessao = @id_sessao);
END;
GO
```

fn_IdadePiloto: Utilizamos esta função para tornar os dados dos pilotos mais dinâmicos, calculando automaticamente a idade deste com base na sua data de nasimento. Como temos esta lógica integrada diretamente no SQL, garantimos que a informação de idade esteja sempre atualizada.

```sql
CREATE FUNCTION dbo.fn_IdadePiloto(@numero_licenca INT)
RETURNS INT AS BEGIN
    RETURN (SELECT DATEDIFF(YEAR, data_nascimento, GETDATE()) FROM Piloto WHERE numero_licenca = @numero_licenca);
END;
GO
```

Index's

O uso de índices é essencial para tornar mais eficaz a pesquisa de dados numa base de dados. À medida que o volume de dados em tabelas como Volta cresce, se não usássemos índices, o SQL teria de realizar leituras longas e complexas às tabelas, o que demoraria mais tempo e degradaria a experiência do utilizador. Implementámos três Non-Clustered Indexes para garantir uma pesquisa otimizada e rápida, independentemente do crescimento da quantidade de dados na base de dados.

IX_Utilizador_Login: Criámos este índice para facilitar o fluxo do login. Ao usarmos este índice, o SQL Server consegue validar as crendencias de um utilizador e recuperar o perfil deste diretamente do índice, sem precisar de consultar a tabela principal.

```sql
CREATE NONCLUSTERED INDEX IX_Utilizador_Login
ON Utilizador(username, password)
INCLUDE (id_utilizador, email);
GO
```

IX_Volta_Tempo: Uma vez que a página de recordes é uma das mais consultada, criámos este índice para armazenar os tempos de volta já ordenados de forma ascendente. Assim, quando o Flask solicita a lista de recordes, estes já vêem previamente ordenados do SQL, poupando o esforço necessário para uma ordenação em tempo real para diversos registos.

```
CREATE NONCLUSTERED INDEX IX_Volta_Tempo
ON Volta(tempo ASC)
INCLUDE (id_sessao, numero_licenca, carro_VIN);
GO
```

IX_Piloto_Nome: Criámos este índice para facilitar a pesquisa na página de pilotos, otimizando as consultas baseadas no nome (o atributo mais comum de ser pesquisado).

```
CREATE NONCLUSTERED INDEX IX_Piloto_Nome
ON Piloto(nome)
INCLUDE (numero_licenca, data_nascimento, nacionalidade, id_equipa, numero_eventos);
GO
```

Triggers

A implementação de triggers foi necessária no nosso projeto para garantir que o sistema funciona de forma consistente, independente das ações feitas na interface. Desenvolvemos três triggers, estes sendo:

trg_ValidaçãoCondicoesVolta: Este trigger INSTEAD OF INSERT atua como uma barreira de segurança/verificação técnica. Antes de ser permitido o registo de qualquer volta na tabela *Volta* , o sistema verifica se os dados das condições meteorológicas já foram inseridos para a respetiva sessão. Caso esse não seja o caso, a operação é interrompida com a mensagem de erro “Não é possível registar voltas. As condições climáticas da sessão ainda não foram registadas.”.

```
CREATE TRIGGER trg_ValidarCondicoesVolta
ON Volta
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;


    IF EXISTS (
        SELECT 1 FROM inserted i
        INNER JOIN Sessao S ON i.id_sessao = S.id_sessao
        WHERE S.temperatura_asfalto IS NULL
           OR S.temperatura_ar IS NULL
           OR S.humidade IS NULL
    )
    BEGIN
        RAISERROR('Não é possível registar voltas. As condições climáticas da sessão ainda não foram registadas!', 16, 1);
        RETURN;
    END


    INSERT INTO Volta (id_sessao, numero_licenca, carro_VIN, tempo, numero_volta, data_hora, pressao_pneus, ID_utilizador_tecnico_de_pista)
    SELECT id_sessao, numero_licenca, carro_VIN, tempo, numero_volta, GETDATE(), pressao_pneus, ID_utilizador_tecnico_de_pista
    FROM inserted;
END;
GO
```

trg_AtualizarStatusEvento: Este trigger automatiza o ciclo de vida das competições. Sempre que uma sessão é atualizada, a base de dados verifica se todas as sessões associadas a esse evento já estão concluídas, ou se a data de fim do evento já passou. Se for este o caso, o status do *Evento* é alterado para "Concluído", automatizando o sistema e garantindo que o calendário oficial esteja sempre atualizado.

```
CREATE TRIGGER trg_AtualizarNumeroEventos
ON Volta
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE P
    SET P.numero_eventos = ISNULL(P.numero_eventos, 0) + 1
    FROM Piloto P
    INNER JOIN inserted i ON P.numero_licenca = i.numero_licenca
    INNER JOIN Sessao S ON i.id_sessao = S.id_sessao
    WHERE NOT EXISTS (
        SELECT 1
        FROM Volta v2
        INNER JOIN Sessao s2 ON v2.id_sessao = s2.id_sessao
        WHERE v2.numero_licenca = i.numero_licenca
          AND s2.id_evento = S.id_evento
          AND v2.id_volta != i.id_volta
    );
END;
GO
```

trg_AtualizarNumeroEventos: Este trigger foi criado com o objetivo de manter a consistência dos dados históricos à medida que são incrementadas novas voltas. Quando um piloto tem a sua primeira volta registada por um técnico de pista, num evento onde ainda não tinha participado, o sistema incrementa o contador *numero_eventos* na tabela *Piloto*. Assim evutamos contagens duplas, e garantimos que o perfil do atleta reflete a realidade.

```
CREATE TRIGGER trg_AtualizarStatusEvento
ON Sessao
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;


    UPDATE Evento
    SET status = 'Concluído'
    FROM Evento E
    WHERE E.id_evento IN (SELECT DISTINCT id_evento FROM inserted)
    AND (
        NOT EXISTS (
            SELECT 1 FROM Sessao S
            WHERE S.id_evento = E.id_evento
            AND S.status != 'Concluída'
        )
        OR E.data_fim < CAST(GETDATE() AS DATE)
    );
END;
GO
```

Stored Procedures

O uso de Stored Procedures no nosso projeto apresentou diversas vantagens, como a abstração de lógica complexa, tornando o código da aplicação python muito mais limpo, reduzindo as queries extensas que seriam necessárias de executar; a melhoria da performance, uma vez que estas são pré-compiladas no SQL Server; e o controlo de transações, para garantir que se ocorrer algum erro a meio da criação duma conta por exemplo, não fique nada gravado.

sp_RegistarVolta: Converte uma string de tempo (usando **TempoParaMs**) para milissegundos e insere um novo registo de volta na base de dados.

sp_AdicionarPiloto: Insere um novo piloto no sistema, verificando previamente se o número de licença já existe de forma a evitar duplicados.

sp_VincularPiloto: Associa um piloto já existente a uma equipa, desde que este ainda não esteja vinculado a nenhuma outra.

sp_AdicionarCarro: Regista um novo veículo na base de dados com todas as suas especificações técnicas (VIN, potência, peso, etc.).

sp_VincularCarro: Atribui um carro disponível a uma equipa específica.

sp_CancelarEventoCompleto: Remove um evento e todos os dados dependentes (voltas, sessões, participações e inscrições), garantindo a integridade dos dados.

sp_ListarEventosComTotais: Gera um relatório de eventos que inclui a contagem agregada de quantas equipas e pilotos participaram em cada um.

sp_ManutencaoStatusEventos: Atualiza automaticamente o estado dos eventos ("A Decorrer" ou "Concluído") comparando as datas de início e fim com a data atual.

sp_RegistarUtilizador: Gere o registo de novos utilizadores e atribui-os automaticamente às tabelas de perfil específicas (Técnico de Pista, Diretor de Equipa ou Diretor de Corrida).

sp_AlterarStatusEvento: Permite a alteração manual do estado de um evento e, caso seja concluído, encerra automaticamente todas as sessões associadas.

sp_AlterarStatusSessao: Modifica o estado de uma sessão específica, impedindo alterações caso a sessão já tenha sido concluída.

sp_InscreverEquipaEvento: Regista a participação de uma equipa num evento específico, validando se o evento ainda permite inscrições.

sp_CancelarInscricaoEvento: Remove a inscrição de uma equipa de um evento, garantindo que não existem pilotos dessa equipa já inscritos em sessões desse mesmo evento.

sp_InscreverSessao: Inscreve um conjunto (piloto e carro) numa sessão, validando a disponibilidade de ambos para evitar conflitos de horários ou duplicidade.

sp_CancelarInscricaoSessao: Remove a participação de um piloto/carro de uma sessão, desde que esta ainda não tenha começado ou terminado.

sp_CriarEquipa: Cria uma nova equipa e associa-a a um Diretor de Equipa, garantindo que cada diretor só possui uma organização.

sp_CriarSessao: Adiciona uma nova sessão a um evento, validando se a hora de fim é posterior à hora de início.

sp_AtualizarEvento: Permite editar os detalhes de um evento existente e as suas datas.

sp_AtualizarSessao: Permite a edição dos dados de uma sessão, desde que esta ainda não tenha sido iniciada.

sp_RemoverSessao: Remove uma sessão do sistema se esta ainda estiver com o status "Por Iniciar".

sp_AtualizarCondicoesPista: Atualiza os dados meteorológicos e de estado do asfalto para uma sessão específica.

sp_AtualizarPiloto: Permite editar as informações biográficas e de licença de um piloto.

sp_AtualizarCarro: Permite editar as especificações técnicas de um veículo.

sp_DesvincularPiloto: Remove o vínculo entre um piloto e a sua equipa, impedindo a ação se a equipa estiver em eventos ativos.

sp_DesvincularCarro: Remove um veículo da frota de uma equipa, também validando a inatividade da equipa em eventos.

Conclusão

O desenvolvimento do **Estoril Track Manager** foi um desafio interessante para pôr em prática a ligação entre uma base de dados relacional e uma interface real. O maior foco do nosso projeto foi garantir que o sistema não fosse apenas um sítio para guardar nomes, mas sim uma ferramenta capaz de manipular os dados.

Em conclusão, o projeto superou as expectativas iniciais. Conseguimos um sistema funcional onde a base de dados é o "cérebro" da aplicação, garantindo que os dados de telemetria e as participações dos pilotos estão sempre consistentes e fáceis de consultar.