universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

Projeto Final - ZooLife

Bases de Dados

Ano Letivo: 2023/2024
Prof. Carlos Costa
Prof. Joaquim Sousa Pinto

Trabalho realizado por:
Rodrigo Abreu, Nº113626
Ricardo Antunes, Nº115243

Índice

Introdução

Como tema deste projeto, decidimos desenvolver uma base de dados para uma rede de jardim zoológicos, distribuídos por todo o país. Com este sistema, seria possível gerir todos os jardins zoológicos, e os respectivos recintos, funcionários, animais, visitantes, contratos e bilhetes de maneira eficiente.

O sistema foi desenvolvido de forma a permitir o registo de jardins zoológicos, recintos, funcionários, animais, visitantes e contratos, bem como a sua remoção e alteração.

Cada tipo de funcionário terá informações adaptadas à sua especialidade (gerente, tratador, veterinário, segurança, funcionário de bilheteira, funcionário de limpeza e trabalhador de restauração), bem como os respectivos gerentes de cada especialidade (tratador, veterinário, segurança). Demos destaque a certas funções dos funcionários, como a manutenção dos habitats e respectivos habitáculos pelos tratadores, definir o veterinário responsável por cada animal, patrulha dos recintos pelos seguranças, venda dos bilhetes pelos funcionários de bilheteira e limpeza dos recintos pelos funcionários de limpeza.

Cada tipo de recinto, tal como os funcionários terá informações adaptadas à sua especialidade, como Habitats que contém habitáculos que por si, contém animais, bilheteiras que fornecem informações dos bilhetes vendidos. Além destes podem existir outros recintos menos relevantes que não explorámos tanto.

Os animais, terão informações bastante relevantes como o nome, espécie, grupo taxonómico, dieta, habitat e habitáculo que frequentam, respetivo veterinário, entre outros. Além disto, podemos ter informações da relação que um dado animal tem com outros, tal como adicionar novas relações e remover.

Os bilhetes, terão informações relevantes, como o id, preço, data de compra, bilheteira onde foi vendido e o funcionário que vendeu o bilhete. Para complementar, terá algumas informações pessoais sobre o visitante para adicionar o seu registo à lista de visitantes do jardim zoológico. Com isto poderão ser obtidos dados de receitas de bilheteira e registar novos clientes, o que é essencial.

Procurámos também adicionar formas de manter a integridade dos dados. Como não permitir o registo de funcionários menores de 18 anos, não permitir salários menores que o salário mínimo, apenas permitir relacionamentos entre animais da mesma espécie, entre outros.

A nossa interface foi desenvolvida em C# e ficou bastante simples, porém apresenta grande parte das funções essenciais que projetámos implementar.

Estrutura da Pasta do Projeto

Dentro do ficheiro entregue, uma pasta Codigo com uma pasta e uma pasta ZooLifeForm. A pasta SQL contém ficheiros .sql divididos de acordo com o tipo de queries (DDL.sql, DML.sql, INDEX.sql, SP.sql, UDF.sql, VIEWS.sql e a pasta TRIGGERS que contém todos os triggers implementados no projeto). A pasta ZooLifeForm contém o código C# da interface do projeto, encontra-se também neste repositório ZooLifeForm.

Encontra-se também a pasta APFE_113626_115243 com os ficheiros entregues na proposta do projeto, o ficheiro Apresentacao_Projeto_BD_ZooLife com o powerpoint utilizado como recurso na apresentação do projeto, um vídeo de demonstração e este relatório.

Requisitos Funcionais

Acesso a informação sobre:

- cada jardim zoológico.
- cada animal.
- visitantes.
- bilhetes vendidos.
- funcionários da empresa.
- cada recinto.

Gerir:

- registos dos
- registo dos animais no sistema (inserções, remoções, etc)
- registo dos funcionários no sistema (inserções, remoções, etc)
- registo dos recintos no sistema (inserções, remoções, etc)
- registo dos bilhetes vendidos no sistema (inserções, remoções)

Registar novos clientes.

Entidades

Pessoa: Representa uma pessoa que está associada a um Jardim Zoológico, seja como Visitante ou como Funcionário

Funcionário: Representa uma pessoa que trabalha num certo jardim Zoológico. Este funcionário pode ter diversos cargos. A este funcionário está associada a data em que se juntou ao jardim zoológico.

Contrato: Representa o acordo entre o funcionário e o jardim zoológico em termos contratuais. Este é composto por um salário, data de início e de fim, número de contrato e tipo de contrato (part-time, full-time). Esta entidade está dependente do funcionário ao qual está associado.

Veterinário: Funcionário responsável por cuidar de animais. Existe um gerente de veterinários, também ele responsável por cuidar de certos animais. Cada animal tem apenas um veterinário responsável por este.

Gerente: Funcionário que gere o Jardim Zoológico.

Tratador: Funcionário responsável por cuidar da qualidade habitacional dos animais. Podem ser necessários vários tratadores para o mesmo habitat. Existe um gestor de tratadores, também ele um tratador.

Segurança: Funcionário responsável por manter a segurança no Jardim Zoológico. Este é responsável por patrulhar uma certa área. É possível designá-lo a uma diferente área, mas apenas uma de cada vez.

Funcionário de Limpeza: Funcionário responsável por manter a higiene no Jardim Zoológico. Um Funcionário de Limpeza pode estar associado a vários recintos que terá de limpar todos os dias.

Trabalhador de Restauração: Funcionário que trabalha na restauração.

Funcionário de bilheteira: Responsável por vender bilhetes (fisicamente). Cada funcionário trabalha numa bilheteira.

Animal: Entidade que representa todos os animais no Jardim Zoológico. Existem vários dados detalhados sobre cada animal, desde o comprimento ao Grupo Taxonómico. Os animais podem ter relações familiares entre si. Os animais vivem num habitáculo. Vários animais podem viver num habitáculo.

Jardim Zoológico: É aqui que trabalham todos os funcionários - o Jardim Zoológico também comporta todos os recintos, e é gerido pelo gerente. Pode estar aberto ou fechado, para além de ter uma morada, e um nome.

Recinto: Área que contém um ponto relevante do Jardim Zoológico. Este espaço pode estar aberto ou fechado, e tem um identificador, para além de ter um nome. Este recinto pode ser um de várias categorias. Vários recintos podem ser limpos por um funcionário de limpeza

Habitat: Área que representa um ambiente propício à existência de certos animais. Cada habitat contém vários habitáculos. Os habitats podem ser tratados por um tratador.

Habitáculo: Contém um identificador, um tamanho e também o máximo número de animais que comporta. Está dentro de um habitat, estando diretamente relacionado a este. Os habitáculos contém animais.

Restauração: Local onde as pessoas podem ter uma refeição. Tem uma capacidade máxima, sendo um recinto fechado.

Bilheteira: Uma bilheteira vende um certo número de bilhetes, derivado de quantas pessoas os compraram.

Bilhete: Representa a única maneira de entrar no Jardim Zoológico, cada bilhete tem um preço, uma data e um identificador que torna cada bilhete único. Estes bilhetes são vendidos pelos funcionários de bilheteira, nas bilheteiras, e comprados por visitantes

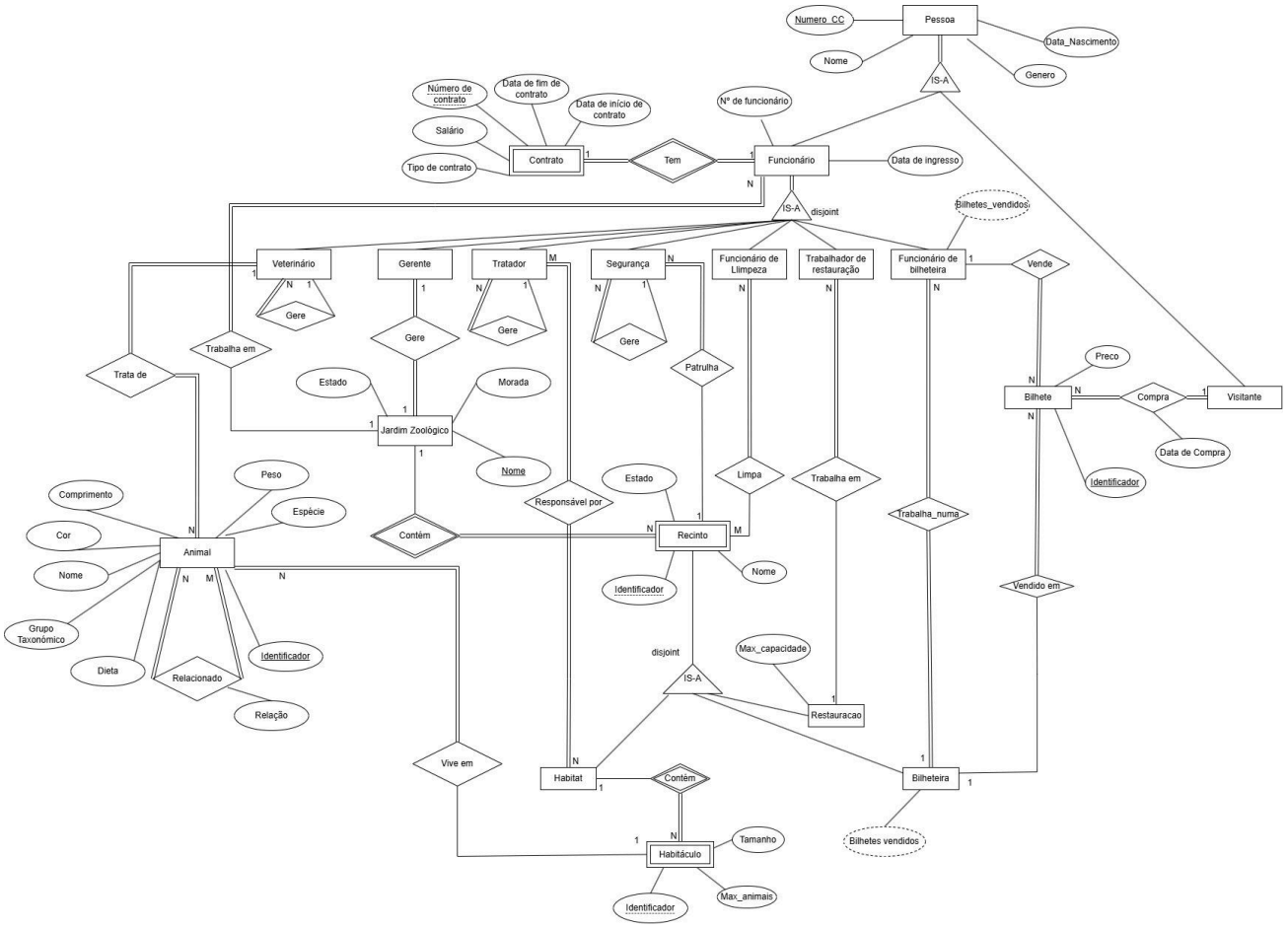
Visitante: Representa uma pessoa que pretende visitar o museu. Esta terá que comprar um bilhete para ter acesso a este.

Limpa: Representa a relação entre um funcionário de limpeza e um certo recinto. Um funcionário pode limpar vários recintos.

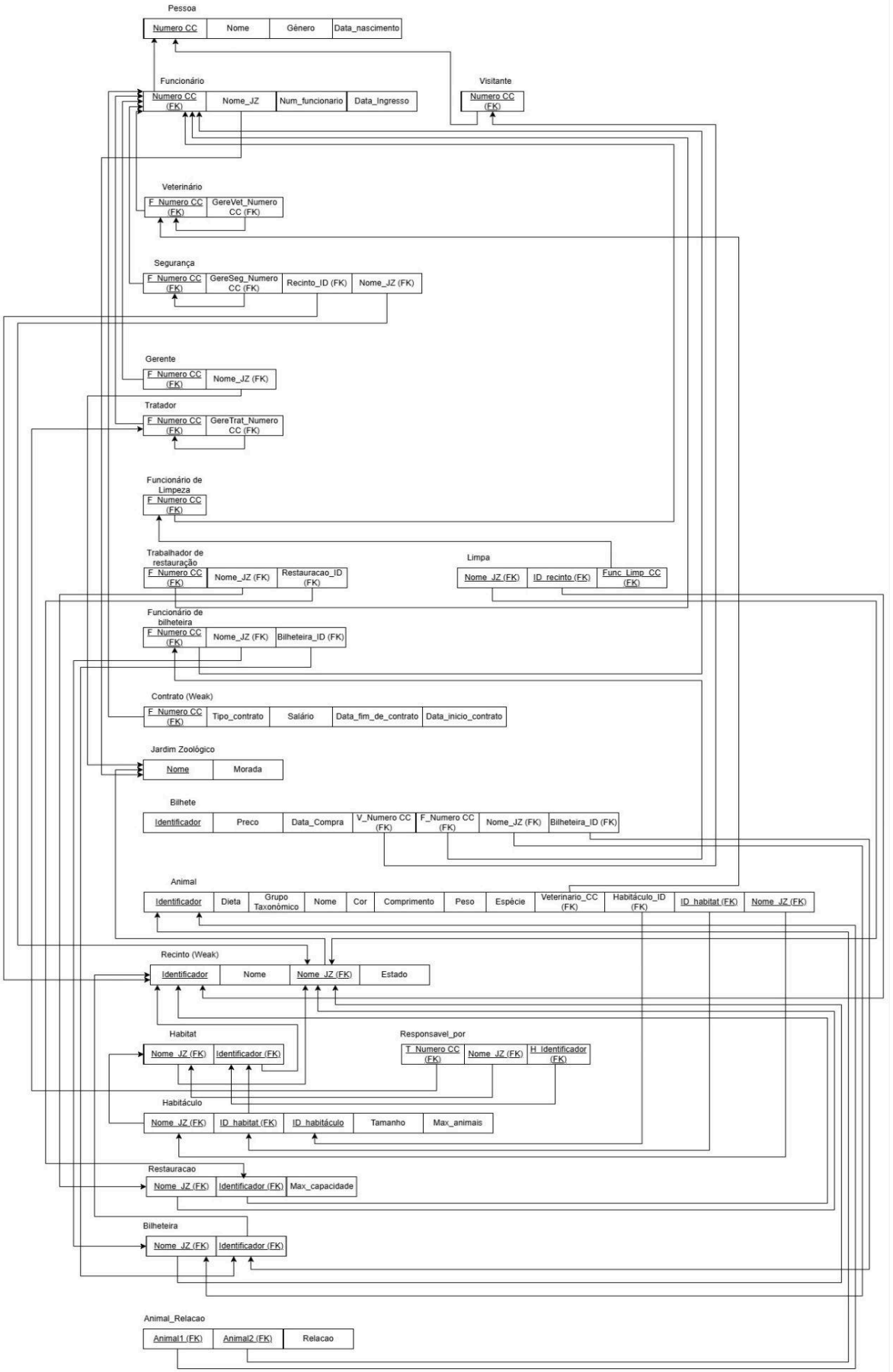
Responsável por: Representa a relação entre um tratador e um habitat. Um tratador pode tratar de vários habitats e um habitat pode ser tratado por vários tratadores. Um tratador, trata de tudo num habitat, incluindo os respectivos habitáculos e animais.

Diagramas

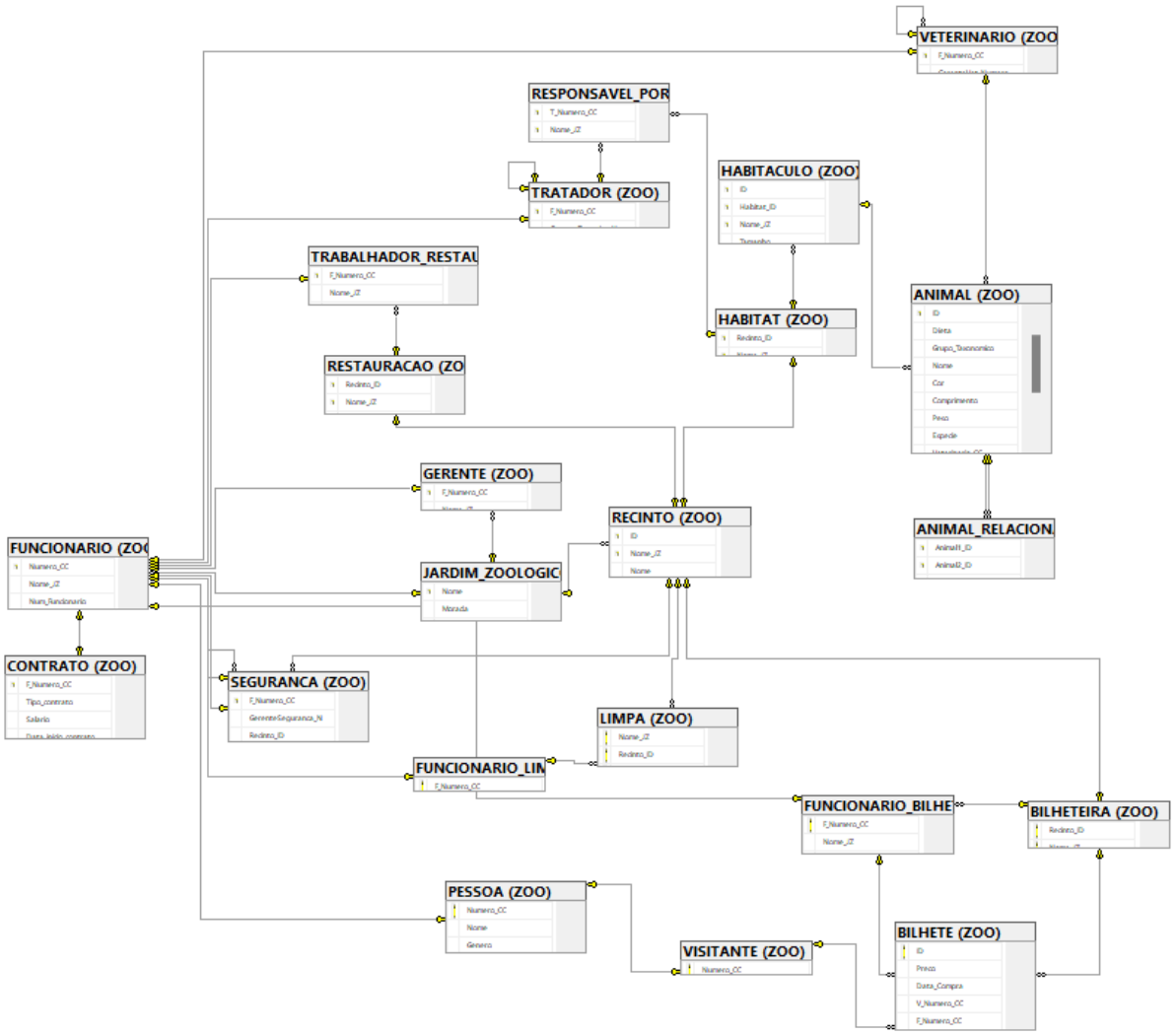
Diagrama Entidade-Relação



Esquema Relacional



Esquema Relacional (SGBD)

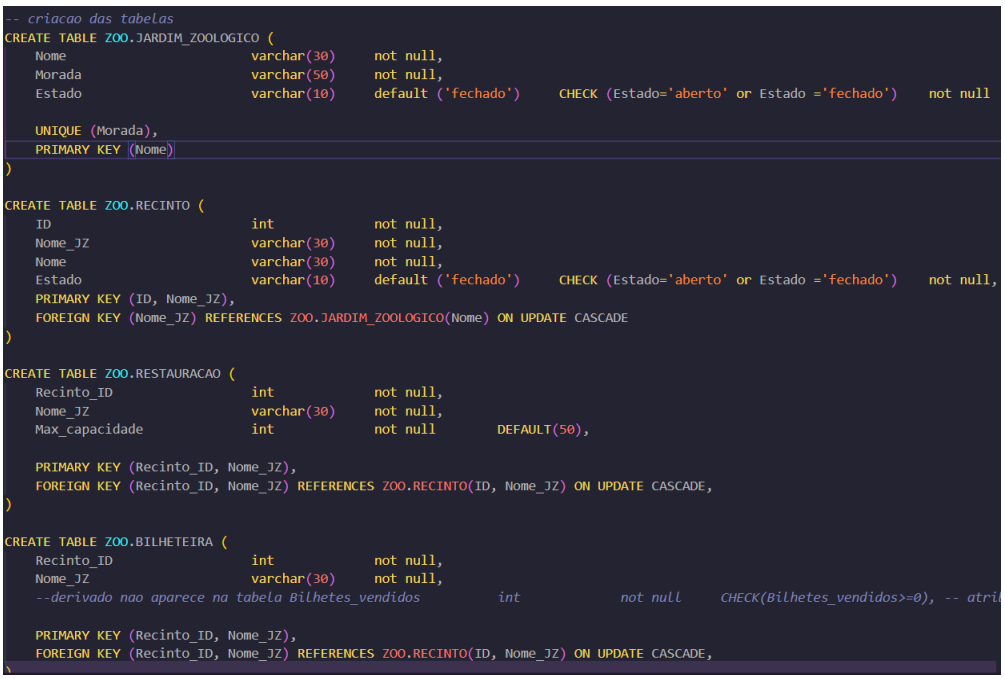


Queries SQL

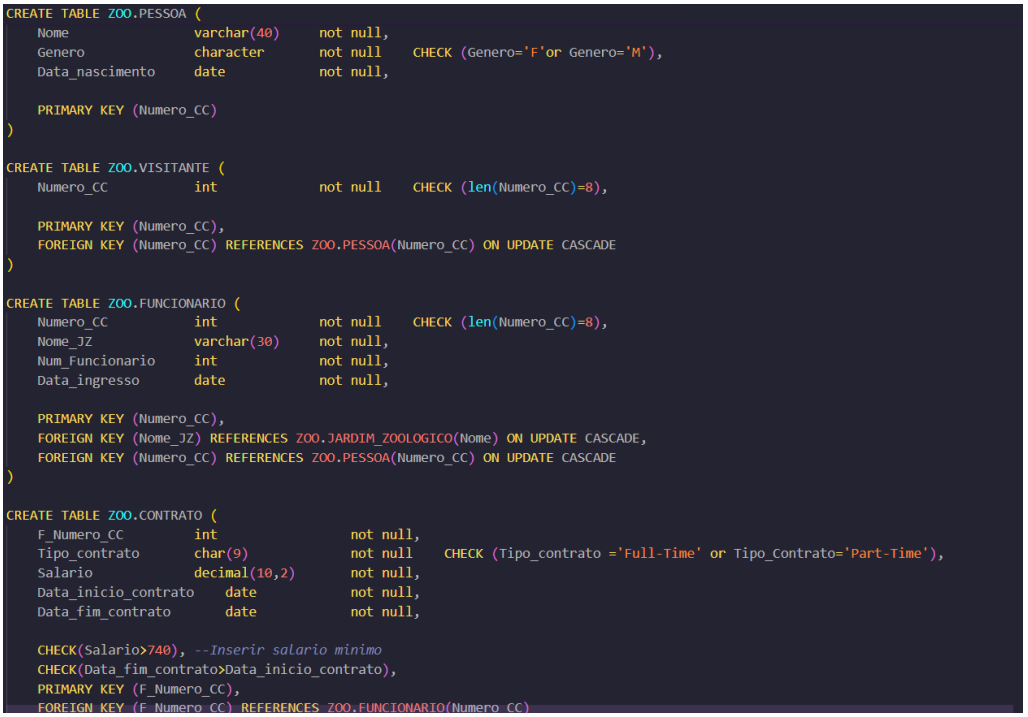
DDL - Data Definition Language

A DDL é uma parte fundamental da linguagem SQL e permite definir a estrutura dos dados dos objetos a ser guardados na base de dados. Através da DDL, definimos a estrutura das tabelas que irão ser guardadas na base de dados. Além disto, definimos algumas verificações para a inserção de dados.

As seguintes imagens apresentam o código desenvolvido:



```
-- criacao das tabelas
CREATE TABLE ZOO.JARDIM_ZOOLOGICO (
    Nome                    varchar(30)     not null,
    Morada                  varchar(50)     not null,
    Estado                  varchar(10)     default ('fechado')     CHECK (Estado='aberto' or Estado ='fechado')    not null

    UNIQUE (Morada),
    PRIMARY KEY (Nome)
)

CREATE TABLE ZOO.RECINTO (
    ID                      int             not null,
    Nome_JZ                 varchar(30)     not null,
    Nome                    varchar(30)     not null,
    Estado                  varchar(10)     default ('fechado')     CHECK (Estado='aberto' or Estado ='fechado')    not null,
    PRIMARY KEY (ID, Nome_JZ),
    FOREIGN KEY (Nome_JZ) REFERENCES ZOO.JARDIM_ZOOLOGICO(Nome) ON UPDATE CASCADE
)

CREATE TABLE ZOO.RESTAURACAO (
    Recinto_ID              int             not null,
    Nome_JZ                 varchar(30)     not null,
    Max_capacidade          int             not null        DEFAULT(50),

    PRIMARY KEY (Recinto_ID, Nome_JZ),
    FOREIGN KEY (Recinto_ID, Nome_JZ) REFERENCES ZOO.RECINTO(ID, Nome_JZ) ON UPDATE CASCADE,
)

CREATE TABLE ZOO.BILHETEIRA (
    Recinto_ID              int             not null,
    Nome_JZ                 varchar(30)     not null,
    --derivado nao aparece na tabela Bilhetes_vendidos          int             not null     CHECK(Bilhetes_vendidos>=0), -- atri

    PRIMARY KEY (Recinto_ID, Nome_JZ),
    FOREIGN KEY (Recinto_ID, Nome_JZ) REFERENCES ZOO.RECINTO(ID, Nome_JZ) ON UPDATE CASCADE,
)
```



```
CREATE TABLE ZOO.PESSOA (
    Nome                varchar(40)     not null,
    Genero              character       not null    CHECK (Genero='F'or Genero='M'),
    Data_nascimento     date            not null,

    PRIMARY KEY (Numero_CC)
)

CREATE TABLE ZOO.VISITANTE (
    Numero_CC           int             not null    CHECK (len(Numero_CC)=8),

    PRIMARY KEY (Numero_CC),
    FOREIGN KEY (Numero_CC) REFERENCES ZOO.PESSOA(Numero_CC) ON UPDATE CASCADE
)

CREATE TABLE ZOO.FUNCIONARIO (
    Numero_CC           int             not null    CHECK (len(Numero_CC)=8),
    Nome_JZ             varchar(30)     not null,
    Num_Funcionario     int             not null,
    Data_ingresso       date            not null,

    PRIMARY KEY (Numero_CC),
    FOREIGN KEY (Nome_JZ) REFERENCES ZOO.JARDIM_ZOOLOGICO(Nome) ON UPDATE CASCADE,
    FOREIGN KEY (Numero_CC) REFERENCES ZOO.PESSOA(Numero_CC) ON UPDATE CASCADE
)

CREATE TABLE ZOO.CONTRATO (
    F_Numero_CC         int                 not null,
    Tipo_contrato       char(9)             not null    CHECK (Tipo_contrato ='Full-Time' or Tipo_Contrato='Part-Time'),
    Salario             decimal(10,2)       not null,
    Data_inicio_contrato    date            not null,
    Data_fim_contrato       date            not null,

    CHECK(Salario>740), --Inserir salario minimo
    CHECK(Data_fim_contrato>Data_inicio_contrato),
    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC)
```

```
CREATE TABLE ZOO.VETERINARIO (
    F_Numero_CC         int             not null,
    GerenteVet_Numero   int                     ,

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (GerenteVet_Numero) REFERENCES ZOO.VETERINARIO(F_Numero_CC)
);

CREATE TABLE ZOO.SEGURANCA (
    F_Numero_CC                 int             not null,
    GerenteSeguranca_Numero     int                     ,
    Recinto_ID                  int             DEFAULT(NULL),
    Nome_JZ                     varchar(30)     DEFAULT(NULL),

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (Recinto_ID, Nome_JZ) REFERENCES ZOO.RECINTO(ID, Nome_JZ) ON DELETE SET NULL,
    FOREIGN KEY (GerenteSeguranca_Numero) REFERENCES ZOO.SEGURANCA(F_Numero_CC),
);


CREATE TABLE ZOO.GERENTE (
    F_Numero_CC                 int             not null,
    Nome_JZ                     varchar(30)     not null,

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC) ON DELETE CASCADE,
    FOREIGN KEY (Nome_JZ) REFERENCES ZOO.JARDIM_ZOOLOGICO(Nome) ON UPDATE CASCADE
);
```

```
CREATE TABLE ZOO.TRATADOR (
    F_Numero_CC                 int             not null,
    GerenteTratador_Numero      int                     ,

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.TRATADOR(F_Numero_CC),
)

CREATE TABLE ZOO.TRABALHADOR_RESTAURACAO (
    F_Numero_CC                 int             not null,
    Nome_JZ                     varchar(30)     DEFAULT(NULL),
    Restauracao_ID              int             DEFAULT(NULL),

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC),
    FOREIGN KEY (Restauracao_ID, Nome_JZ) REFERENCES ZOO.RESTAURACAO(Recinto_ID, Nome_JZ) ON UPDATE SET NULL
)

CREATE TABLE ZOO.FUNCIONARIO_BILHETEIRA (
    F_Numero_CC                 int             not null,
    Nome_JZ                     varchar(30)     DEFAULT(NULL),
    --atributo derivado não vai para a tabela Bilhetes_vendidos        int             not null    CHECK (Bi
    Bilheteira_ID               int             DEFAULT(NULL),

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC),
    FOREIGN KEY (Bilheteira_ID, Nome_JZ) REFERENCES ZOO.Bilheteira(Recinto_ID, Nome_JZ) ON UPDATE SET NULL
)

CREATE TABLE ZOO.FUNCIONARIO_LIMPEZA (
    F_Numero_CC                 int             not null,

    PRIMARY KEY (F_Numero_CC),
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO(Numero_CC) ON UPDATE CASCADE
)
```

```sql
CREATE TABLE ZOO.LIMPA (
    Recinto_ID              int             not null,
    FL_Numero_CC            int             not null,

    PRIMARY KEY(Nome_JZ, Recinto_ID, FL_Numero_CC),
    FOREIGN KEY (FL_Numero_CC) REFERENCES ZOO.FUNCIONARIO_LIMPEZA(F_Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (Recinto_ID, Nome_JZ) REFERENCES ZOO.RECINTO(ID, Nome_JZ) ON UPDATE NO ACTION,
)

CREATE TABLE ZOO.BILHETE (
    ID                      int             not null,
    Preco                   decimal(10,2)   not null    CHECK(Preco >= 0)   DEFAULT(10.00),
    Data_Compra             date            not null,
    V_Numero_CC             int             not null,
    F_Numero_CC             int             not null,
    Nome_JZ                 varchar(30)     default(null),
    Bilheteira_ID           int             default(null),

    PRIMARY KEY (ID),
    FOREIGN KEY (V_Numero_CC) REFERENCES ZOO.VISITANTE(Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (F_Numero_CC) REFERENCES ZOO.FUNCIONARIO_BILHETEIRA(F_Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (Bilheteira_ID, Nome_JZ) REFERENCES ZOO.BILHETEIRA(Recinto_ID, Nome_JZ)
)

CREATE TABLE ZOO.HABITAT (
    Recinto_ID              int             not null,
    Nome_JZ                 varchar(30)     not null,

    PRIMARY KEY (Recinto_ID, Nome_JZ),
    FOREIGN KEY (Recinto_ID, Nome_JZ) REFERENCES ZOO.RECINTO(ID, Nome_JZ) ON UPDATE CASCADE,
)

CREATE TABLE ZOO.HABITACULO (
    ID                      int             not null,
    Habitat_ID              int             not null,
    Nome_JZ                 varchar(30)     not null,
    Tamanho                 varchar(30)     not null,
```

```sql
CREATE TABLE ZOO.RESPONSAVEL_POR (
    Nome_JZ                 varchar(30)     default(null),
    Habitat_ID              int             default(null),

    PRIMARY KEY (T_Numero_CC, Nome_JZ, Habitat_ID),
    FOREIGN KEY (T_Numero_CC) REFERENCES ZOO.TRATADOR(F_Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (Habitat_ID, Nome_JZ) REFERENCES ZOO.Habitat(Recinto_ID, Nome_JZ),
)

CREATE TABLE ZOO.ANIMAL (
    ID                      int             not null,
    Dieta                   varchar(50)     not null,
    Grupo_Taxonomico        varchar(30)     not null,
    Nome                    varchar(30)     not null,
    Cor                     varchar(30)     not null,
    Comprimento             decimal(10,2)   not null, --metros
    Peso                    decimal(10,2)   not null, --kg
    Especie                 varchar(30)     not null,
    Veterinario_CC          int             not null,
    Habitaculo_ID           int             default(null),
    Habitat_ID              int             default(null),
    Nome_JZ                 varchar(30)     default(null),

    PRIMARY KEY (ID),
    FOREIGN KEY (Veterinario_CC) REFERENCES ZOO.VETERINARIO(F_Numero_CC) ON UPDATE CASCADE,
    FOREIGN KEY (Habitaculo_ID, Habitat_ID, Nome_JZ) REFERENCES ZOO.HABITACULO(ID, Habitat_ID, Nome_JZ)
)

CREATE TABLE ZOO.ANIMAL_RELACIONADO (
    Animal1_ID          int             not null,
    Animal2_ID          int             not null,
    Relacao             varchar(50)     not null,

    PRIMARY KEY (Animal1_ID, Animal2_ID),
    FOREIGN KEY (Animal1_ID) REFERENCES ZOO.ANIMAL(ID),
    FOREIGN KEY (Animal2_ID) REFERENCES ZOO.ANIMAL(ID)
)
```

DML - Data Manipulation Language

A DML desempenha um papel importante na inserção de dados na base de dados. Através desta linguagem desenvolvemos o código para preencher as tabelas com dados, de forma a termos material para podermos realizar os procedimentos estipulados.
Seguem-se alguns exemplos dos muitos dados inseridos que se encontram no ficheiro DML.sql :

```
INSERT INTO ZOO.JARDIM_ZOOLOGICO (Nome, Morada, Estado) VALUES
('ZOOLife Norte', 'R. Estação, 4470-184 Maia', 'aberto'),
('ZOOLife Centro', 'Rua Professor Fernando da Fonseca, 1501-806 Lisboa','aberto');

INSERT INTO ZOO.RECINTO (ID, Nome_JZ, Nome, Estado) VALUES
(1, 'ZOOLife Norte', 'Entrada Este', 'aberto'),
(2, 'ZOOLife Norte', 'Entrada Oeste', 'aberto'),
(3, 'ZOOLife Norte', 'Bilheteira Este', 'aberto'),
(4, 'ZOOLife Norte', 'Bilheteira Oeste', 'aberto'),
```

```
INSERT INTO ZOO.RESTAURACAO (Recinto_ID, Nome_JZ, Max_capacidade) VALUES
(5, 'ZOOLife Norte', 50),
(6, 'ZOOLife Norte', 80),
(17, 'ZOOLife Centro', 65),
(18, 'ZOOLife Centro', 100);

INSERT INTO ZOO.Habitat (Recinto_ID, Nome_JZ) VALUES
(7, 'ZOOLife Norte'),
(8, 'ZOOLife Norte'),
```

```
INSERT INTO ZOO.BILHETEIRA (Recinto_ID, Nome_JZ) VALUES
(3, 'ZOOLife Norte'),
(4, 'ZOOLife Norte'),
(15, 'ZOOLife Centro'),
(16, 'ZOOLife Centro');


INSERT INTO ZOO.PESSOA (Numero_CC, Nome, Genero, Data_Nascimento)
VALUES
(12345678, 'João Silva', 'M', '1980-01-01'),
(23456789, 'Maria Oliveira', 'F', '1985-01-01'),
(34567890, 'Pedro Sousa', 'M', '1990-01-01'),
```

```
INSERT INTO ZOO.FUNCIONARIO (Numero_CC, Nome_JZ, Num_Funcionario, Data_Ingresso) VALUES
(12345678, 'ZOOLife Norte', 1, '1998-01-01'),
(23456789, 'ZOOLife Norte', 2, '2003-01-01'),
```

```
INSERT INTO ZOO.CONTRATO (F_Numero_CC, Tipo_Contrato, Salario, Data_inicio_contrato, Data_fim_contrato) VALUES
(12345678, 'Full-Time', 3000.00, '1998-01-01', '2028-01-01'), --GERENTE
(23456789, 'Full-Time', 1250.00, '2003-01-01', '2027-01-01'), --TRATADOR
```

```
INSERT INTO ZOO.Bilhete(ID, Preco, Data_Compra, V_Numero_CC, F_Numero_CC, Nome_JZ, Bilheteira_ID)
VALUES
(1, 10.00, '2023-06-19',  71345678, 25789012,'ZOOLife Norte', 3),
(2, 10.00, '2023-08-17',  72456789, 25789012,'ZOOLife Norte', 3),
(3, 10.00, '2023-09-23',  73567890, 25789012,'ZOOLife Norte', 3),
```

```
INSERT INTO ZOO.HABITACULO (ID, Habitat_ID, Nome_JZ, Tamanho, Max_animais) VALUES
(1, 7, 'ZOOLife Norte', '100m2', 20),
(2, 8, 'ZOOLife Norte', '120m2', 20),
```

```
INSERT INTO ZOO.ANIMAL (ID, Dieta, Grupo_Taxonomico, Nome, Cor, Comprimento, Peso, Especie, Veterinario_CC, Habitaculo_ID, Habitat_ID,

(1, 'Herbívoro', 'Mamífero', 'Elefante', 'Cinza', 3.00, 6000.00, 'Loxodonta africana', 14678901, 12, 10, 'ZOOLife Norte'),
(2, 'Carnívoro', 'Ave', 'Águia', 'Castanho', 1.20, 5.00, 'Aquila chrysaetos', 15789012, 15, 11, 'ZOOLife Norte'),
(3, 'Omnívoro', 'Mamífero', 'Urso', 'Castanho', 2.10, 300.00, 'Ursus arctos', 16890123, 13, 10, 'ZOOLife Norte'),
```

```
INSERT INTO ZOO.ANIMAL (ID, Dieta, Grupo_Taxonomico, Nome, Cor, Comprimento, Peso, Especie, Veterinario_CC, Habitaculo_ID, Habitat_ID,

(1, 'Herbívoro', 'Mamífero', 'Elefante', 'Cinza', 3.00, 6000.00, 'Loxodonta africana', 14678901, 12, 10, 'ZOOLife Norte'),
(2, 'Carnívoro', 'Ave', 'Águia', 'Castanho', 1.20, 5.00, 'Aquila chrysaetos', 15789012, 15, 11, 'ZOOLife Norte'),
(3, 'Omnívoro', 'Mamífero', 'Urso', 'Castanho', 2.10, 300.00, 'Ursus arctos', 16890123, 13, 10, 'ZOOLife Norte'),
```

Views

As views que criámos para este projeto foram-nos bastante úteis particularmente em situações em que tínhamos entidades com relações IS-A com outras entidades das quais queremos retirar informação, sendo assim possível reunir todos os dados necessários numa só tabela. Dando um exemplo em concreto, quando foi necessário resgatar em simultâneo o nome mas também o contrato de um segurança, a criação de uma view permitiu-nos criar uma tabela que continha toda a informação necessária:

```
CREATE VIEW ZOO.SEGURANCA_DETALHADO AS
    SELECT ZOO.RECINTO.ID AS RECINTO_ID, ZOO.RECINTO.Nome AS
RECINTO_NOME, ZOO.RECINTO.Estado AS RECINTO_ESTADO,
ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO.*,
ZOO.SEGURANCA.GerenteSeguranca_Numero FROM ZOO.SEGURANCA
    INNER JOIN ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO ON
ZOO.SEGURANCA.F_Numero_CC =
ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO.Numero_CC
    INNER JOIN ZOO.RECINTO ON ZOO.SEGURANCA.Recinto_ID = ZOO.RECINTO.ID
AND ZOO.SEGURANCA.Nome_JZ = ZOO.RECINTO.Nome_JZ;
```

Como é possivel ver, usámos views que se referenciam a outras views para ir buscar a informação relevante. A cadeia de views que mostramos agora foi particularmente importante para o problema referido acima:

```
CREATE VIEW ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO AS
SELECT FDT.*, C.Tipo_contrato, C.Salario, C.Data_inicio_contrato,
C.Data_fim_contrato
FROM ZOO.FUNCIONARIO_DETALHADO_TOTAL FDT
INNER JOIN ZOO.CONTRATO C ON FDT.Numero_CC = C.F_Numero_CC;
```

```
CREATE VIEW ZOO.FUNCIONARIO_DETALHADO_TOTAL AS
        SELECT FD.*,
            'GERENTE' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.GERENTE G ON FD.Numero_CC = G.F_Numero_CC
        WHERE FD.Numero_CC = G.F_Numero_CC
        UNION ALL
        SELECT FD.*,
            'FUNCIONARIO_BILHETEIRA' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.FUNCIONARIO_BILHETEIRA FB ON FD.Numero_CC =
FB.F_Numero_CC
        WHERE FD.Numero_CC = FB.F_Numero_CC
        UNION ALL
        -- Repeat for other roles
```

```
        SELECT FD.*,
            'FUNCIONARIO_LIMPEZA' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.FUNCIONARIO_LIMPEZA FL ON FD.Numero_CC =
FL.F_Numero_CC
        WHERE FD.Numero_CC = FL.F_Numero_CC
        UNION ALL
        SELECT FD.*,
            'TRABALHADOR_RESTAURACAO' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.TRABALHADOR_RESTAURACAO FM ON FD.Numero_CC =
FM.F_Numero_CC
        WHERE FD.Numero_CC = FM.F_Numero_CC
        UNION ALL
        SELECT FD.*,
            'SEGURANCA' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.SEGURANCA FS ON FD.Numero_CC = FS.F_Numero_CC
        WHERE FD.Numero_CC = FS.F_Numero_CC
        UNION ALL
        SELECT FD.*,
            'VETERINARIO' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.VETERINARIO FV ON FD.Numero_CC = FV.F_Numero_CC
        WHERE FD.Numero_CC = FV.F_Numero_CC
        UNION ALL
        SELECT FD.*,
            'TRATADOR' as Role
        FROM ZOO.FUNCIONARIO_DETALHADO FD
        INNER JOIN ZOO.TRATADOR T ON FD.Numero_CC = T.F_Numero_CC
        WHERE FD.Numero_CC = T.F_Numero_CC;
```

```
CREATE VIEW ZOO.FUNCIONARIO_DETALHADO AS
SELECT ZOO.PESSOA.Numero_CC, ZOO.PESSOA.Nome, ZOO.PESSOA.Genero,
ZOO.PESSOA.Data_nascimento, ZOO.FUNCIONARIO.Nome_JZ,
ZOO.FUNCIONARIO.Num_Funcionario, ZOO.FUNCIONARIO.Data_ingresso FROM
ZOO.FUNCIONARIO
INNER JOIN ZOO.PESSOA ON ZOO.FUNCIONARIO.Numero_CC =
ZOO.PESSOA.Numero_CC;
```

Cada uma destas views liga a informação disponível a uma tabela ligada à entidade Funcionário. A view ZOO.FUNCIONARIO_DETALHADO trata de juntar as entidades Funcionário e Pessoa, a view ZOO.FUNCIONARIO_DETALHADO_TOTAL trata de fazer uma ligação às entidades pertencentes

à relação IS-A com a entidade Funcionário, enquanto que a view ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO junta essa informação aos dados contratuais do funcionário. Esta cadeia de views em particular foi utilizada em diversas views e até mesmo UDF's de forma a facilitar a pesquisa de informação.

Também criámos outras views, relativas a atributos abstratos e outras informações que ajudaria estarem relacionadas. A utilização de views simplificou bastante a pesquisa de informação em que tal pesquisa não estava dependente de um funcionário (i.e. esta view ajudou com pesquisa em todos os funcionário e não um funcionário em específico) para além de permitir um ponto de entrada para aceder a todos os dados de um funcionário sem entrar em todas as tabelas em específico (ou seja, os dados do funcionário estavam disponíveis mas não as suas responsabilidades ou supervisor)

UDFs - User Defined Functions

UDFs permitem criar funções personalizadas da linguagem SQL, permitindo assim a reutilização de código e realização de pesquisas complexas de maneira mais eficiente.

Elas foram de extrema importância no nosso projeto, tornando as consultas de dados SQL bastante mais eficientes e o código mais legível.

Seguem-se alguns exemplos das muitas UDFs usadas no nosso projeto e que se encontram no ficheiro UDF.sql :

```sql
CREATE FUNCTION ZOO.GET_TIPO_RECINTO
(
    @recintoID INT
)
RETURNS VARCHAR(30)
AS
BEGIN
    DECLARE @tipo VARCHAR(30) = NULL;

    -- Check specific tables first
    IF EXISTS (SELECT 1 FROM ZOO.RESTAURACAO WHERE Recinto_ID = @recintoID)
    BEGIN
        SELECT @tipo = 'RESTAURACAO';
    END
    ELSE IF EXISTS (SELECT 1 FROM ZOO.BILHETEIRA WHERE Recinto_ID = @recintoID)
    BEGIN
        SELECT @tipo = 'BILHETEIRA';
    END
    ELSE IF EXISTS (SELECT 1 FROM ZOO.HABITAT WHERE Recinto_ID = @recintoID)
    BEGIN
        SELECT @tipo = 'HABITAT';
    END
    -- Check the main RECINTO table last
    ELSE IF EXISTS (SELECT 1 FROM ZOO.RECINTO WHERE ID = @recintoID)
    BEGIN
        SELECT @tipo = 'RECINTO';
    END

    RETURN @tipo;
END
GO
```

```sql
CREATE FUNCTION ZOO.GET_ALL_BILHETE_INFO
(
    @ID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        b.ID,
        b.Preco,
        b.Data_Compra,
        b.V_Numero_CC,
        b.F_Numero_CC,
        r.Nome AS Recinto_Nome,
        r.Nome_JZ,
        p.Nome AS Pessoa_Nome,
        p.Genero
    FROM ZOO.BILHETE b
    JOIN ZOO.BILHETEIRA bil ON b.Bilheteira_ID = bil.Recinto_ID
    JOIN ZOO.RECINTO r ON bil.Recinto_ID = r.ID AND bil.Nome_JZ = r.Nome_JZ
    JOIN ZOO.PESSOA p ON b.V_Numero_CC = p.Numero_CC
    WHERE b.ID = @ID
)
```

```sql
DROP FUNCTION IF EXISTS ZOO.FUNCIONARIO_BILHETEIRAS_DISPONIVEIS
GO
CREATE FUNCTION ZOO.FUNCIONARIO_BILHETEIRAS_DISPONIVEIS(
    @NUMERO_CC INT,
    @Nome_JZ NVARCHAR(50)
) RETURNS TABLE
AS
RETURN
(
    SELECT ZOO.RECINTO.*
    FROM ZOO.RECINTO
    INNER JOIN ZOO.BILHETEIRA ON ZOO.RECINTO.ID = ZOO.BILHETEIRA.Recinto_ID
        AND ZOO.BILHETEIRA.Nome_JZ = ZOO.BILHETEIRA.Nome_JZ
    LEFT JOIN (SELECT Bilheteira_ID FROM ZOO.FUNCIONARIO_BILHETEIRA WHERE F_Numero_CC = @NUMERO_CC) AS FB
        ON ZOO.RECINTO.ID = FB.Bilheteira_ID
    WHERE ZOO.RECINTO.Nome_JZ = @Nome_JZ
        AND FB.Bilheteira_ID IS NULL
)
GO
```

```sql
DROP FUNCTION IF EXISTS ZOO.PESQUISA_RELACOES_ANIMAL;
GO
CREATE FUNCTION ZOO.PESQUISA_RELACOES_ANIMAL (
    @AnimalId INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT CASE WHEN RELACAO.Animal1_ID = @AnimalId THEN ANIMAL2.Nome ELSE ANIMAL1.Nome END AS OtherAnimalName,
           CASE WHEN RELACAO.Animal1_ID = @AnimalId THEN RELACAO.Animal2_ID ELSE RELACAO.Animal1_ID END AS OtherAnimalId,
           RELACAO.Relacao
    FROM ZOO.ANIMAL_RELACIONADO AS RELACAO
    INNER JOIN ZOO.ANIMAL AS ANIMAL1 ON RELACAO.Animal1_ID = ANIMAL1.ID
    INNER JOIN ZOO.ANIMAL AS ANIMAL2 ON RELACAO.Animal2_ID = ANIMAL2.ID
    WHERE RELACAO.Animal1_ID = @AnimalId OR RELACAO.Animal2_ID = @AnimalId
)
GO
```

```sql
DROP FUNCTION IF EXISTS ZOO.BILHETES_VENDIDOS
GO
CREATE FUNCTION ZOO.BILHETES_VENDIDOS
(
    @recintoID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @bilhetes INT = NULL;
    IF EXISTS (SELECT * FROM ZOO.BILHETEIRA WHERE Recinto_ID = @recintoID)
    BEGIN
        SELECT @bilhetes = COUNT(*)
        FROM ZOO.BILHETE
        WHERE Bilheteira_ID = @recintoID;
    END
    RETURN @bilhetes;
END
GO
```

Stored Procedures

Os Stored Procedures são conjuntos de comandos SQL pré-compilados que são armazenados no servidor de base de dados e podem ser executados quando necessário no código. Eles permitem que os utilizadores executem operações complexas de base de dados de maneira eficiente e consistente.

Isto foi dos pontos mais importantes do nosso trabalho, permitindo-nos a reutilização de código, torná-lo mais eficiente e sobretudo, mais seguro.

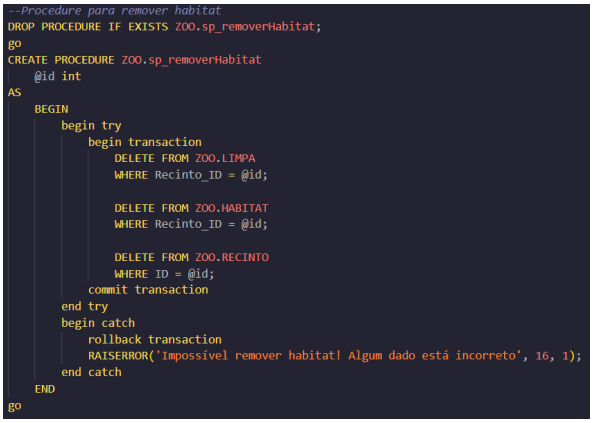
Segue aqui o nome e a utilidade de cada stored procedure usado, tal como algumas imagens do código, que se encontram no ficheiro SP.sql :

- ZOO.sp_adicionarAnimal - Adiciona um animal.
- ZOO.sp_alterarAnimal - Altera os dados de um animal.
- ZOO.sp_DeleteAnimal - Permite remover um animal.
- ZOO.sp_transferirAnimal - Permite transferir um animal para outro habitáculo.
- ZOO.sp_adicionarAnimalRelacionado - Permite adicionar uma relação a um animal.
- ZOO.sp_removerAnimalRelacionado - Permite remover relação de um animal.
- ZOO.sp_adicionarVeterinario - Permite adicionar um veterinário.
- ZOO.sp_removerVeterinario - Permite remover um veterinário.
- ZOO.sp_adicionarGerente - Permite adicionar um gerente.
- ZOO.sp_removerGerente - Permite remover um gerente.
- ZOO.sp_adicionarSeguranca - Permite adicionar segurança.
- ZOO.sp_removerSeguranca - Permite remover segurança.
- ZOO.sp_adicionarTratador - Permite adicionar um tratador.
- ZOO.sp_removerTratador - Permite remover um tratador.
- ZOO.sp_adicionarTrabalhadorRestauracao - Permite adicionar um trabalhador de restauração.
- ZOO.sp_removerTrabalhadorRestauracao - Permite remover um trabalhador de restauração.
- ZOO.sp_adicionarFuncionarioBilheteira - Permite adicionar um funcionário de bilheteira.
- ZOO.sp_removerFuncionarioBilheteira - Permite remover um funcionário de bilheteira.
- ZOO.sp_adicionarFuncionarioLimpeza - Permite adicionar um funcionário de limpeza.
- ZOO.sp_removerFuncionarioLimpeza - Permite remover um funcionário de limpeza.
- ZOO.sp_adicionarLimpa - Permite adicionar uma relação entre o funcionário de limpeza e o recinto que ele limpa.
- ZOO.sp_removerLimpa - Permite remover uma relação entre o funcionário de limpeza e o recinto que ele limpa.
- ZOO.sp_definirVeterinarioComoGerente - Permite definir um veterinário como novo gerente.
- ZOO.sp_definirSegurancaComoGerente - Permite definir um segurança como novo gerente.
- ZOO.sp_definirTratadorComoGerente - Permite definir um tratador como novo gerente.
- ZOO.sp_adicionarRecinto - Permite adicionar um novo recinto.
- ZOO.sp_removerRecinto - Permite remover um recinto.
- ZOO.sp_adicionarHabitat - Adiciona um habitat novo.
- ZOO.sp_removerHabitat - Remove um habitat.
- ZOO.sp_adicionarHabitaculo - Adiciona um habitáculo a um habitat.
- ZOO.sp_removerHabitaculo - Remove um habitáculo de um habitat.
- ZOO.sp_adicionarBilheteira - Adiciona uma bilheteira nova.
- ZOO.sp_removerBilheteira - Remover bilheteira.

- ZOO.sp_adicionaRestauracao - Adiciona um novo recinto de restauração (bares, restaurantes, etc).
- ZOO.sp_removerRestauracao - Remove um recinto de restauração.
- ZOO.sp_adicionarBilhete - Permite adicionar um bilhete vendido ao registo.
- ZOO.sp_transferirSeguranca - Permite transferir um segurança para outro recinto.
- ZOO.sp_transferirVeterinario - Permite transferir um veterinário para outro jardim zoológico.
- ZOO.sp_transferirTratador - Permite transferir um tratador para outro habitat.
- ZOO.sp_transferirFuncionarioBilheteira - Permite transferir um funcionário de bilheteira para outra bilheteira.
- ZOO.sp_transferirTrabalhadorRestauracao - Permite transferir um trabalhador de restauração para outro recinto de restauração.
- ZOO.sp_transferirFuncionarioLimpeza - Permite transferir um funcionário de limpeza para outro zoo.
- ZOO.sp_removerBilhete - Permite remover um bilhete vendido do registo.
- ZOO.sp_editarRecinto - Permite editar um recinto.
- ZOO.sp_eliminarFuncionario - Permite remover qualquer tipo de funcionário.
- ZOO.sp_editarFuncionario - Permite editar um funcionário.

```
--Procedure para adicionar funcionario de bilheteira
DROP PROCEDURE IF EXISTS ZOO.sp_adicionarFuncionarioBilheteira;
go
CREATE PROCEDURE ZOO.sp_adicionarFuncionarioBilheteira
    @cc int,
    @nome varchar(40),
    @genero character,
    @data_nascimento date,
    @jardim_zoologico varchar(30),
    --num_funcionario e calculado
    @data_ingresso date,
    @tipo_contrato char(9),
    @salario decimal(10,2),
    @data_inicio_contrato date,
    @data_fim_contrato date,
    @bilheteira_id int
AS
    BEGIN
        begin try
            begin transaction
                DECLARE @num_funcionario int;

                SELECT @num_funcionario = MAX(Num_Funcionario) + 1 FROM ZOO.FUNCIONARIO;

                INSERT INTO ZOO.PESSOA VALUES (@cc, @nome, @genero, @data_nascimento);
                INSERT INTO ZOO.FUNCIONARIO VALUES (@cc, @jardim_zoologico, @num_funcionario, @data_ingresso);
                INSERT INTO ZOO.CONTRATO VALUES (@cc, @tipo_contrato, @salario, @data_inicio_contrato, @data_fim_contrato);
                INSERT INTO ZOO.FUNCIONARIO_BILHETEIRA VALUES (@cc, @jardim_zoologico, @bilheteira_id);
            commit transaction
        end try
        begin catch
            rollback transaction
            RAISERROR('Impossível adicionar funcionário de bilheteira! Algum dado está incorreto', 16, 1);
        end catch
    END
```

```
--Procedure para criar um habitat
DROP PROCEDURE IF EXISTS ZOO.sp_adicionarHabitat;
go
CREATE PROCEDURE ZOO.sp_adicionarHabitat
    @nome_jz varchar(30),
    @nome varchar(30),
    @estado varchar(10)
AS
    BEGIN
        begin try
            begin transaction
                DECLARE @id int;

                SELECT @id = MAX(ID) + 1 FROM ZOO.RECINTO;

                INSERT INTO ZOO.RECINTO VALUES (@id, @nome_jz, @nome, @estado);
                INSERT INTO ZOO.HABITAT VALUES (@id, @nome_jz);
            commit transaction
        end try
        begin catch
            rollback transaction
            RAISERROR('Impossível adicionar habitat! Algum dado está incorreto', 16, 1);
        end catch
    END
go
```



```
--Procedure para remover habitat
DROP PROCEDURE IF EXISTS ZOO.sp_removerHabitat;
go
CREATE PROCEDURE ZOO.sp_removerHabitat
    @id int
AS
    BEGIN
        begin try
            begin transaction
                DELETE FROM ZOO.LIMPA
                WHERE Recinto_ID = @id;

                DELETE FROM ZOO.HABITAT
                WHERE Recinto_ID = @id;

                DELETE FROM ZOO.RECINTO
                WHERE ID = @id;
            commit transaction
        end try
        begin catch
            rollback transaction
            RAISERROR('Impossível remover habitat! Algum dado está incorreto', 16, 1);
        end catch
    END
go
```

```
CREATE PROCEDURE ZOO.sp_adicionarBilhete
    @v_genero character,
    @v_data_nascimento date,
    @preco decimal(10,2),
    @data_compra date,
    @f_numero_cc int,
    @nome_jz varchar(30),
    @bilheteira_id int

AS
    BEGIN
        begin try
            begin transaction
                DECLARE @id int;

                SELECT @id = MAX(ID) + 1 FROM ZOO.BILHETE;
                if NOT EXISTS(SELECT * FROM ZOO.PESSOA WHERE Numero_CC = @v_numero_cc)
                BEGIN
                    INSERT INTO ZOO.PESSOA VALUES (@v_numero_cc, @v_nome, @v_genero, @v_data_nascimento);
                    INSERT INTO ZOO.VISITANTE VALUES (@v_numero_cc);
                END
                ELSE
                BEGIN
                    IF NOT EXISTS (SELECT * FROM ZOO.VISITANTE WHERE Numero_CC = @v_numero_cc)
                    BEGIN
                        INSERT INTO ZOO.VISITANTE VALUES (@v_numero_cc);
                    END
                END

                INSERT INTO ZOO.BILHETE VALUES (@id, @preco, @data_compra, @v_numero_cc, @f_numero_cc, @nome_jz, @bilheteira_id);
            commit transaction
        end try
        begin catch
            rollback transaction
            RAISERROR('Impossível adicionar bilhete! Algum dado está incorreto', 16, 1);
        end catch
    END
```

Triggers

Triggers são objetos de base de dados que são ativados automaticamente em resposta a eventos como inserção, atualização ou exclusão de dados em uma tabela. Eles ajudam a controlar, automatizar e manter a integridade dos dados.

No nosso projeto, necessitámos de criar triggers para as seguintes situações:

- Não deixar remover recintos caso estes tenham funcionários associados.
- Não permitir a criação e alteração para que recintos tenham nomes iguais.
- Não deixar adicionar animais num habitáculo caso este já tenha atingido a capacidade máxima.
- Não permitir que contratos tenham uma data inicial posterior à data final do contrato.

Segue aqui alguns exemplos, dos triggers que se encontram na pasta TRIGGERS:

```sql
CREATE TRIGGER trg_check_recinto_sem_funcionarios
ON ZOO.RECINTO
INSTEAD OF DELETE
AS
BEGIN
    DECLARE @ID int, @Nome_JZ varchar(30);
    -- Seleciona os valores de ID e Nome_JZ da tentativa de exclusão
    SELECT @ID = DELETED.ID, @Nome_JZ = DELETED.Nome_JZ
    FROM DELETED;
    -- Verifica se há funcionários associados ao recinto que está sendo excluído
    IF EXISTS
    (SELECT 1 FROM ZOO.HABITAT WHERE Recinto_ID = @ID )
    OR EXISTS
    (SELECT 1 FROM ZOO.BILHETEIRA WHERE Recinto_ID = @ID)
    OR EXISTS
    ( SELECT 1 FROM ZOO.RESTAURACAO WHERE Recinto_ID = @ID)
    BEGIN
        RAISERROR('Não é possível remover o recinto. Existe um recinto especifico associado.', 16, 1);
    END
    ELSE
    BEGIN
        -- Se não houver funcionários associados, permite a exclusão
        DELETE FROM ZOO.RECINTO
        WHERE ID = @ID AND Nome_JZ = @Nome_JZ;
    END
END;
```

```sql
CREATE TRIGGER trg_validar_datas_contrato
ON ZOO.CONTRATO
INSTEAD OF INSERT
AS BEGIN
    DECLARE @Data_Inicio date, @Data_Fim date;
    SELECT @Data_Inicio = INSERTED.Data_inicio_contrato, @Data_Fim = INSERTED.Data_fim_contrato
    FROM INSERTED;

    IF @Data_Inicio > @Data_Fim
    BEGIN
        RAISERROR('Data de início do contrato não pode ser posterior à data de fim.', 16, 1);
    END
    ELSE
    BEGIN
        INSERT INTO ZOO.CONTRATO (F_Numero_CC, Tipo_contrato, Salario, Data_inicio_contrato, Data_fim_contrato)
        SELECT F_Numero_CC, Tipo_contrato, Salario, Data_inicio_contrato, Data_fim_contrato
        FROM INSERTED;
    END
END;
```

```sql
CREATE TRIGGER trg_check_max_animais
ON ZOO.ANIMAL
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @Habitaculo_ID int, @Habitat_ID int, @Nome_JZ varchar(30);

    SELECT @Habitaculo_ID = INSERTED.Habitaculo_ID,
           @Habitat_ID = INSERTED.Habitat_ID,
           @Nome_JZ = INSERTED.Nome_JZ
    FROM INSERTED;

    IF @Habitaculo_ID IS NOT NULL AND @Habitat_ID IS NOT NULL AND @Nome_JZ IS NOT NULL
    BEGIN
        DECLARE @CurrentCount int, @MaxAnimals int;

        SELECT @CurrentCount = COUNT(*)
        FROM ZOO.ANIMAL
        WHERE Habitaculo_ID = @Habitaculo_ID AND Habitat_ID = @Habitat_ID AND Nome_JZ = @Nome_JZ;

        SELECT @MaxAnimals = Max_animais
        FROM ZOO.HABITACULO
        WHERE ID = @Habitaculo_ID AND Habitat_ID = @Habitat_ID AND Nome_JZ = @Nome_JZ;

        IF @CurrentCount > @MaxAnimals
        BEGIN
            RAISERROR ('Número máximo de animais no habitáculo excedido.', 16, 1);
            ROLLBACK TRANSACTION;
        END
    END
END;
```

```sql
CREATE TRIGGER trg_check_recinto_nome_unico
ON ZOO.RECINTO
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @Nome_JZ varchar(30), @Nome varchar(30);

    -- Seleciona os valores de Nome_JZ e Nome da tentativa de inserção
    SELECT @Nome_JZ = INSERTED.Nome_JZ, @Nome = INSERTED.Nome
    FROM INSERTED;

    -- Verifica se já existe um recinto com o mesmo nome no mesmo jardim zoológico
    IF EXISTS (
        SELECT 1
        FROM ZOO.RECINTO
        WHERE Nome_JZ = @Nome_JZ AND Nome = @Nome
    )
    BEGIN
        RAISERROR('Já existe um recinto com este nome neste jardim zoológico.', 16, 1);
    END
    ELSE
    BEGIN
        -- Se não houver conflito, permite a inserção
        INSERT INTO ZOO.RECINTO (ID, Nome_JZ, Nome, Estado)
        SELECT ID, Nome_JZ, Nome, Estado
        FROM INSERTED;
    END
END;
```

Indexes

Os indexes são ideais para a eficiência da pesquisa de dados numa base de dados, logo foram bastante úteis para otimização da pesquisa de certos dados na nossa base de dados, nas operações de filtragem. Segue aqui os indexes usados:

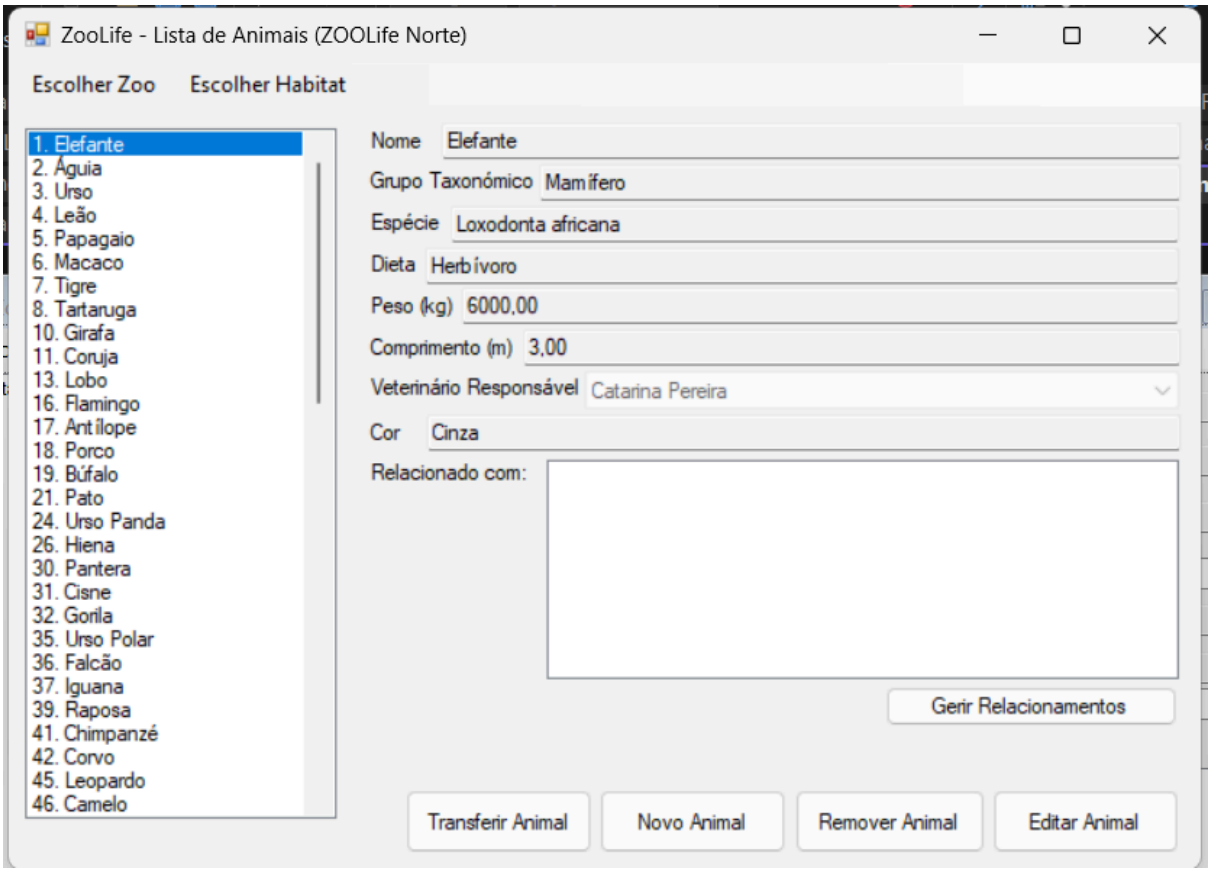
```
CREATE INDEX IDXRecintoID ON ZOO.RECINTO(ID);
CREATE INDEX IDXRecintoNome ON ZOO.RECINTO(Nome);
CREATE INDEX IDXFuncionarioNome ON ZOO.PESSOA(Nome);
CREATE INDEX IDXFuncionarioDataNascimento ON
ZOO.PESSOA(Data_nascimento);
CREATE INDEX IDXFuncionarioZoo ON ZOO.FUNCIONARIO(Nome_JZ);
CREATE INDEX IDXFuncionarioSalario ON ZOO.CONTRATO(Salario);
CREATE INDEX IDXFuncionarioDataIngresso ON
ZOO.FUNCIONARIO(Data_ingresso);
```

A escolha da criação destes indexes baseou-se na utilização frequente dos atributos nestes criados aquando pesquisa de informação. Existiu um maior foco neste aspeto na página dos funcionários, que contém filtros e ordenação, pelo que a maior parte dos indexes são direcionados à volta destes funcionários

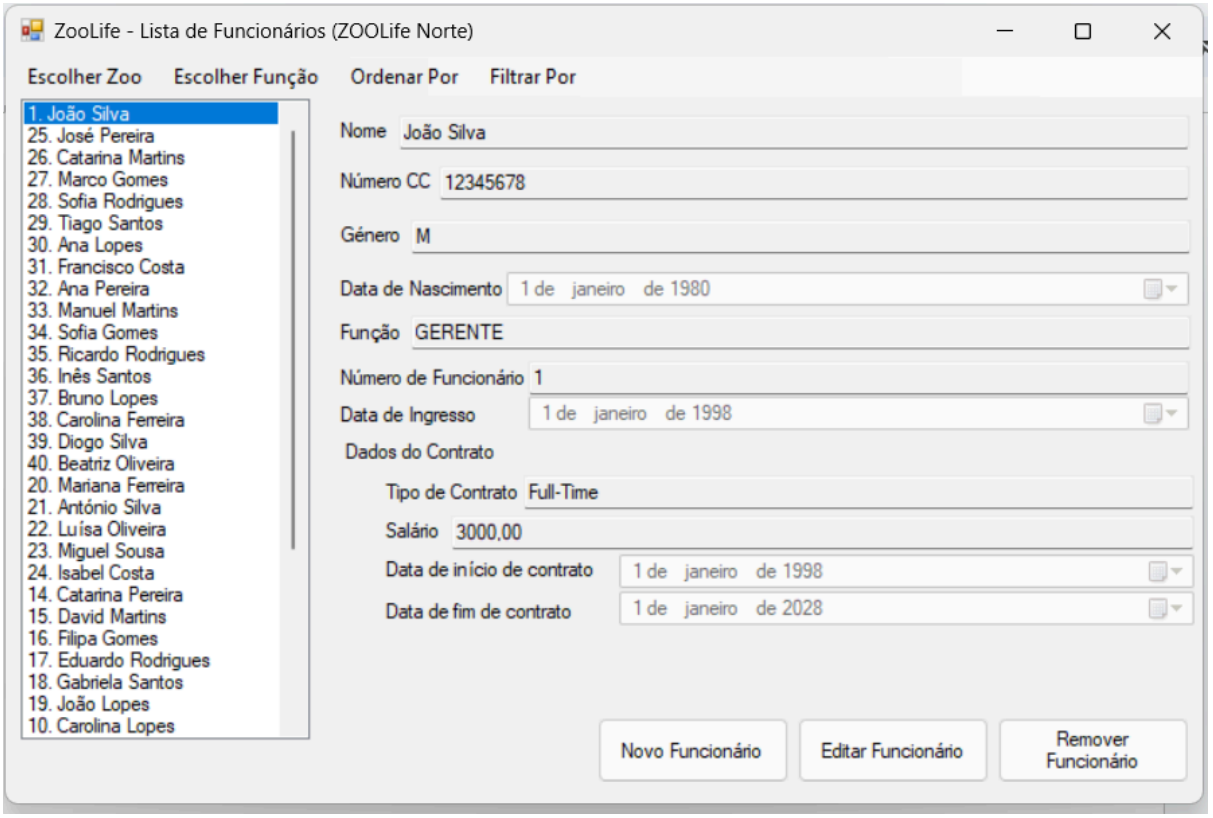
Interface

Tentámos incluir na nossa interface, todas as funções importantes do nosso sistema, mas por falta de tempo acabámos por deixar algumas funcionalidades de fora, embora nos ficheiros sql tenhamos o código para executar essas funções. Exemplos disto são, a impossibilidade de, na interface, criar ou remover jardins zoológicos, transferir funcionários e obter informações mais detalhadas sobre clientes e estatísticas de bilhetes vendidos.

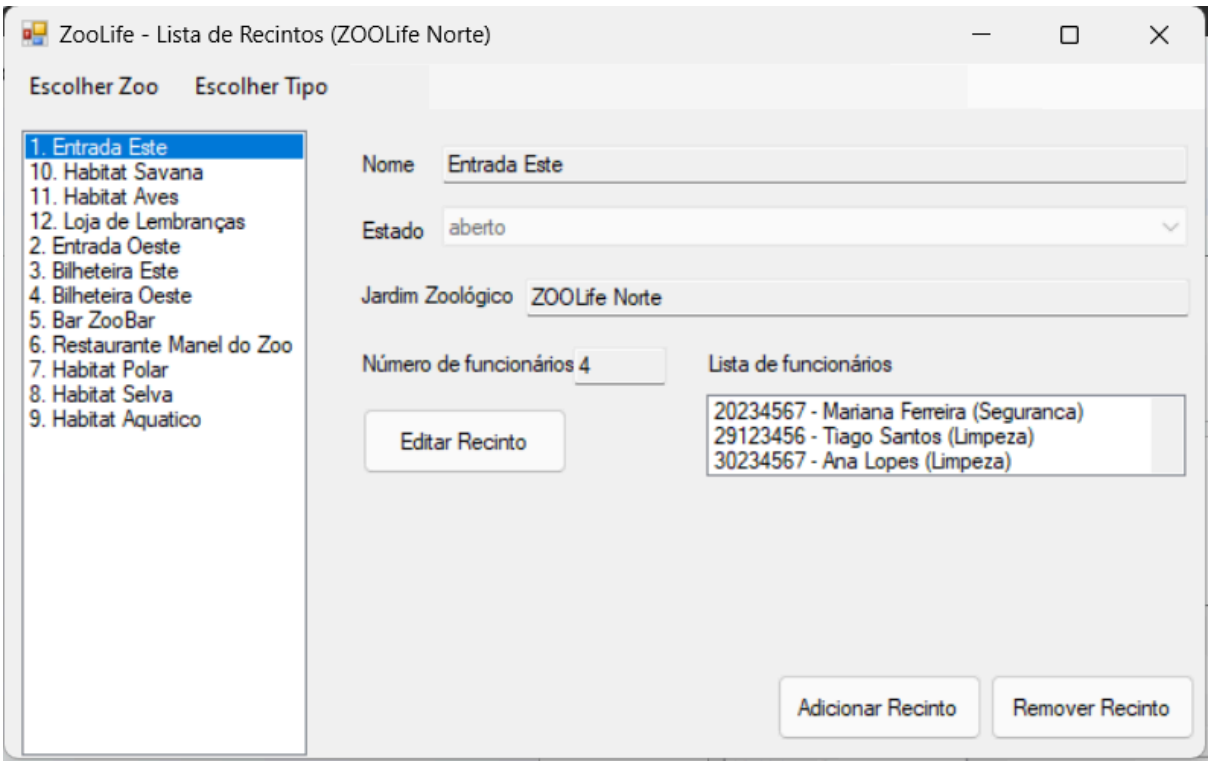
Seguem-se algumas imagens a exemplificar as funcionalidades do sistema:



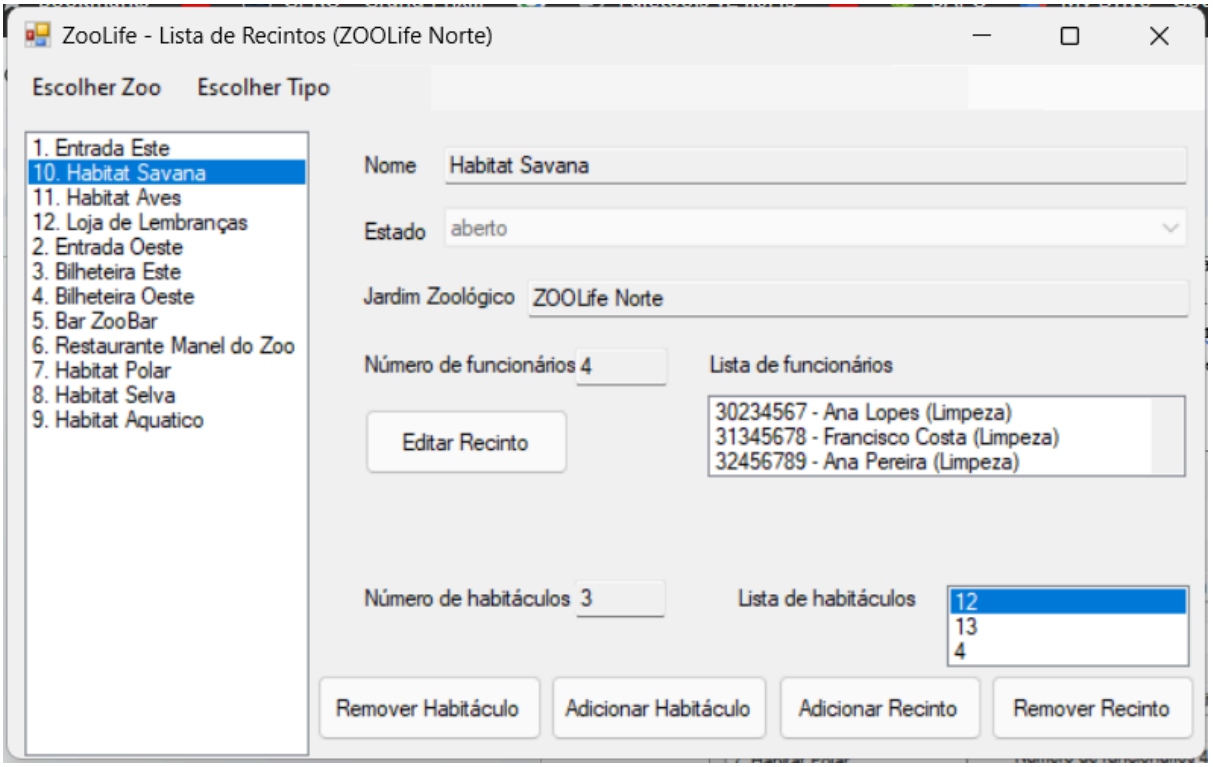
Esta página é onde o utilizador pode pesquisar todos os animais pertencentes a um Jardim Zoológico. O utilizador também pode colocar um animal num Jardim Zoológico / Habitat / Habitáculo diferente, para além de conseguir criar um novo animal, remover o animal selecionado e editar um animal existente (caso existam gralhas na inserção de dados, por exemplo)



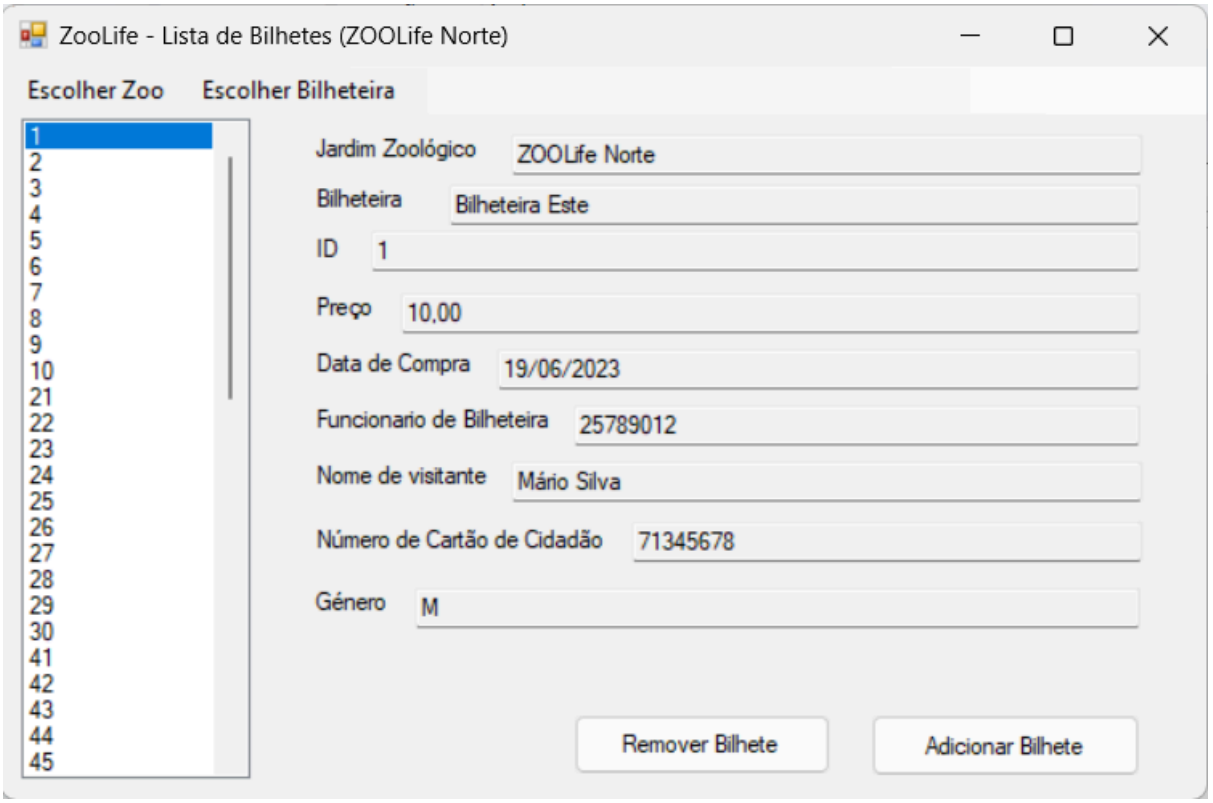
Esta página é onde o utilizador pode observar todos os dados relevantes sobre os funcionários. Esta tabela recolhe dados da view ZOO.FUNCIONARIO_DETALHADO_TOTAL_CONTRATO que já foi explicada anteriormente. Aqui o utilizador pode, para além de filtrar por Zoo e função, filtrar também por Nome e Género, para além de ordenar a lista por Salário, Data de Nascimento e Data de Ingresso no Jardim Zoológico (tanto por ordem crescente ou decrescente). O utilizador pode também adicionar um novo funcionário, remover o funcionário atualmente selecionado e editar alguns dados do funcionário atual.



Esta página é onde o utilizador consegue ver os dados relevantes a todos os recintos de cada Jardim Zoológico. Aqui é possível filtrar por Jardim Zoológico e tipo de recinto (relativamente à relação IS-A demonstrada no diagrama inicial). O utilizador consegue mais uma vez, gerir os recintos ao permitir a edição, remoção e adição de recintos. Adicionalmente, poderão aparecer mais opções dependendo do tipo de recinto que temos:



Quando o utilizador seleciona um habitáculo relativo a um habitat, aparecem as opções de remover e adicionar um habitáculo.



Esta página é onde o utilizador consegue analisar todos os bilhetes vendidos e os dados referentes a estes. O utilizador pode também adicionar e remover bilhetes, se assim o pretender. O utilizador pode filtrar por Jardim Zoológico e Bilheteira.

Conclusão

Este projeto envolveu várias atividades relativas à criação de uma base de dados e a sua implementação:

- Numa primeira fase, tivemos que delinear o que queríamos retirar do projeto e o que era relevante para uma base de dados de um conjunto de Jardins Zoológicos. Feito esse exercício mental, tivemos que criar as entidades relevantes à construção de um Diagrama Entidade-Relação, ao qual aplicamos as técnicas aprendidas nas aulas de Bases de Dados para criar um Modelo Relacional.
- Feitas as verificações que estávamos perante um Modelo Relacional na 3FN, e certificado que estávamos satisfeitos com os requisitos e objetivos do projeto, traduzimos o Modelo Relacional para SQL, criando assim um script DDL que usámos para criar o Esquema Relacional apresentado no início do relatório.
- Com a base de dados construída, construímos a interface de interação do utilizador com a base de dados, refletindo sobre que Views, UDF's, Stored Procedures, Triggers, etc. eram necessários ao longo do desenvolvimento do projeto.

Apesar de apresentarmos aqui um resultado satisfatório, este projeto tem algumas limitações que, devido à sobrecarga de projetos com que nos encaramos, não foi possível resolver, nomeadamente:

- Não é possível adicionar, editar ou remover Jardins Zoológicos.
- Não é possível transferir funcionários entre Jardins Zoológicos.
- Não é possível transferir recintos entre Jardins Zoológicos.
- Não existe diferenciação entre um administrador e um potencial visitante, que obviamente não deveria poder editar dados, e portanto não existe função de autenticação
- Aquando a criação de um funcionário / visitante, não temos uma funcionalidade de providenciar ao utilizador os dados existentes de uma pessoa com um certo Número CC (mas se tal inserção for tentada, os triggers existentes irão prevenir essa adição)
- Os filtros existentes na página dos funcionários são o que provavelmente deveria estar presente em todas as páginas, evidentemente com outros atributos a serem filtrados / ordenados

Estes problemas foram deixados de parte, dado que considerámos que o restante trabalho que realizamos evidenciou que conseguimos aplicar os conhecimentos aprendidos na cadeira de Base de Dados, em que aplicámos vários conceitos e estratégias para construir a base de dados, e fazer a ligação desta com uma interface a ser utilizada por outrem.