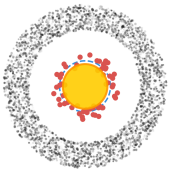
Universidade da Beira Interior
Departamento de Informática
Licenciatura em Engenharia Informática

UC: Bases de Dados

Trabalho Prático

Sistema de Monitorização de Objetos Próximos da Terra



Elaborado por:

Bruno Salvador - 53588
Júlia Santos - 53754
Diogo Nogueira - 53851
Guilherme Sousa - 55010

Orientador:
Professor Doutor João Muranho

3 de janeiro de 2026

Agradecimentos

Gostaríamos de começar por expressar o nosso agradecimento à Universidade da Beira Interior (UBI) e ao Departamento de Informática (DI), pelas infraestruturas e recursos disponibilizados que permitiram a realização deste projeto.

Ao Professor Doutor João Muranho, agradecemos a orientação académica no âmbito da Unidade Curricular de Bases de Dados e a oportunidade de explorar este tema.

Aos nossos colegas de curso, pelo companheirismo e entreajuda demonstrados ao longo do semestre.

Por fim, um agradecimento especial à resiliência e ao empenho incansável investidos na superação dos desafios técnicos deste projeto, sem os quais este resultado final não teria sido possível.

Resumo

A monitorização de Objetos Próximos da Terra (NEO) assume um papel fulcral na **defesa planetária**, dado tratar-se de uma população dinâmica onde, para cada corpo celeste detetado, existem milhares de menor dimensão ainda não catalogados [1].

A incerteza científica atual resulta, em parte, de vieses observacionais e da subestimação do risco de impacto destes objetos. Consequentemente, torna-se imperativo o desenvolvimento de sistemas de informação capazes de armazenar e analisar dados físicos e orbitais complexos.

Este trabalho prático, desenvolvido no âmbito da Unidade Curricular (UC) de **Bases de Dados**, propõe uma solução para este desafio: um modelo relacional implementado em **Microsoft SQL Server**, normalizado até à Terceira Forma Normal (3FN), desenhado para mitigar a dispersão dos dados físicos e suportar a natureza evolutiva dos parâmetros orbitais.

O sistema inclui um ecossistema de aplicações desenvolvidas em **Python**, recorrendo a *Triggers* para a avaliação de risco em tempo real, segundo a Escala de Torino Modificada (ETM) e o critério da Distância Mínima de Interseção Orbital (MOID), bem como para a geração automática de alertas para Asteroide Potencialmente Perigoso (PHA).

Os resultados obtidos privilegiam a consistência e a integridade do armazenamento de grandes volumes de dados científicos, em detrimento da otimização de tempos de execução, alinhando-se com os objetivos pedagógicos da UC. A solução oferece, assim, uma ferramenta funcional e robusta de suporte à decisão face a potenciais ameaças cósmicas.

Palavras-chave: Bases de Dados, Defesa Planetária, Monitorização de Asteroides, Python, SQL Server.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Acrónimos

BD Base de Dados
EA Entidade Associação
UC Unidade Curricular
3FN Terceira Forma Normal
DEA Diagrama Entidade Associação
ETM Escala de Torino Modificada
MOID Distância Mínima de Interseção Orbital
NEO Objetos Próximos da Terra
PHA Asteroide Potencialmente Perigoso
SGBD Sistema de Gestão de Bases de Dados
UBI Universidade da Beira Interior
DI Departamento de Informática
GUI Interface Gráfica do Utilizador

Capítulo

1

Introdução

O Sistema Solar, ao contrário da perceção comum de um vácuo estático, é uma região dinâmica e repleta de matéria. Para além dos oito planetas e das respetivas luas, existe uma infinidade de corpos de dimensões menores, nomeadamente cometas e asteroides. São precisamente estes corpos que representam um risco tangível para a Terra.

Os chamados NEO são asteroides trazidos para as proximidades do nosso planeta devido às suas órbitas excêntricas. Embora a descoberta destes objetos tenha sido consideravelmente mais acelerada nos últimos trinta anos, a monitorização continua a ser um desafio. Estima-se que, para cada corpo detetado, existam milhares de menor dimensão não catalogados. Ainda que a queda de meteoritos recuperáveis seja apenas na ordem das centenas por ano, os eventos catastróficos, mesmo que raros, requerem uma vigilância constante. [2]

Atualmente, a capacidade de armazenar, relacionar e consultar dados orbitais é tão crítica como a observação astronómica. Este relatório descreve o desenvolvimento de um sistema de informação dedicado à gestão de alertas e monitorização destes corpos celestes.

1.1 Objetivos

O objetivo principal deste trabalho prático, realizado na UC de Bases de Dados, é a conceção e implementação de uma Base de Dados (BD) relacional capaz de suportar as necessidades de um observatório astronómico e de um sistema de alertas precoces.

Os objetivos específicos incluem:

- **Importação:** Implementação de mecanismos de extração e tratamento de dados, desenvolvidos em Python, para povoar a base de dados a partir dos *datasets* fornecidos;
- **Modelação:** Criação de um modelo Entidade Associação (EA) e sua posterior conversão para um modelo relacional normalizado até à 3FN, garantindo a integridade e reduzindo a redundância de dados físicos e orbitais;
- **Lógica de Servidor:** Programação de *Triggers* para a automação de alertas e a classificação de risco, baseada na **ETM** e na MOID;
- **Desenvolvimento de Aplicações:** Criação de interfaces gráficas para interagir com a base de dados, permitindo a monitorização estatística e a gestão de alertas de risco em tempo real.

1.2 Metodologia e Ferramentas

A solução foi desenvolvida utilizando o **Microsoft SQL Server** como Sistema de Gestão de Bases de Dados (SGBD). Para a componente cliente e processamento de dados, utilizou-se a linguagem **Python**, com recurso a bibliotecas como:

- `mssql-python`: Para a conectividade e execução de comandos SQL;
- `pandas` e `matplotlib`: Para manipulação de dados e visualização gráfica;
- `tkinter`: Para a construção das interfaces gráficas de utilizador.

O ambiente de desenvolvimento foi configurado para permitir a alternância dinâmica entre ambientes locais, para teste, e remotos.

1.3 Estrutura do Relatório

O presente relatório encontra-se estruturado da seguinte forma:

- **Capítulo 2 - Desenvolvimento da Aplicação Cliente/Servidor:** Descreve a arquitetura Cliente/Servidor, a configuração do ambiente e o desenvolvimento do módulo de importação (ETL), que constitui a base de dados para as restantes aplicações;
- **Capítulo 3 - Modelação:** Apresenta o processo evolutivo de desenho da base de dados, desde o Diagrama Entidade Associação (DEA) até ao Esquema Relacional final e normalizado;
- **Capítulo 4 - Distribuição de Tarefas:** Lista as responsabilidades atribuídas e as tarefas realizadas por cada membro do grupo;
- **Capítulo 5 - Acesso à Base de Dados:** Descreve os mecanismos técnicos utilizados para aceder à base de dados;
- **Capítulo 6 - Funcionalidade:** Apresenta a visão global do sistema e a sua arquitetura, detalhando o fluxo de dados desde a importação até à visualização;
- **Capítulo 7 - Aplicações:** Detalha a implementação física no SQL Server (Tabelas, Triggers, Views) e a integração com as aplicações finais de Gestão de Alertas e Monitorização;
- **Capítulo 8 - Conclusões:** Reflete sobre os resultados alcançados face aos objetivos propostos;
- **Epílogo:** Apresenta uma reflexão crítica acerca da UC;
- **Referências Bibliográficas:** Lista os artigos e fontes citados ao longo do relatório;
- **Apêndices:** Contém os scripts utilizados para a criação e manutenção da BD.

Capítulo

2

Desenvolvimento da Aplicação Cliente/Servidor

2.1 Introdução

O desenvolvimento de um sistema de informação útil para a monitorização de corpos celestes tem, obrigatoriamente, de ser capaz de suportar o fluxo contínuo de dados entre fontes externas, a base de dados e os utilizadores das aplicações.

O capítulo 2 descreve o desenvolvimento e implementação da aplicação de inserção de dados na BD. Este módulo serve de alicerce para todo o sistema, sendo que é o responsável por carregar e processar ficheiros CSV com dados de NEOs e órbitas, validando e inserindo a informação na BD. Sem ele, não faria sentido a existência das outras duas aplicações desenvolvidas, nomeadamente a de gestão de alertas e a de monitorização, detalhadas no capítulo 7.

2.2 Arquitetura Cliente/Servidor

A solução baseia-se no modelo Cliente/Servidor, essencial para garantir a integridade e a centralização dos dados num ambiente distribuído.

- **Cliente:** O ecossistema cliente é composto por três aplicações desenvolvidas em **Python**. Nesta fase inicial do fluxo de dados, o destaque vai para o *Pipeline*, que atua como um intermediário, sendo que valida, normaliza e insere dados provenientes de ficheiros CSV antes de estes ficarem disponíveis para consulta;
- **Servidor:** O **SQL Server** atua como o repositório central e garante a persistência dos dados, a execução de restrições de integridade relacional e o processamento automático de riscos, através *Triggers* criados.

2.3 SQL Server e Configuração do Acesso

Foi utilizado o Microsoft SQL Server como SGBD, visto ter sido utilizado ao longo de todo o semestre nesta UC.

Para garantir a segurança e a flexibilidade no desenvolvimento, as credenciais de acesso foram abstraídas do código principal, permitindo a alternância entre ambientes através de ficheiros de configuração:

1. **Ambiente de Teste (Local):** Utiliza a autenticação integrada do Windows (`Trusted_Connection=yes`) via `localhost\SQLEXPRESS`;
2. **Ambiente de Produção (Remoto):** Configurado para estabelecer a conexão ao servidor departamental (`192.168.100.14`), através da VPN da UBI, utilizando as credenciais de autenticação fornecidas pelo docente (`User_BD_PL1_01`). Esta configuração assegura o cumprimento dos requisitos de avaliação e simula um cenário de produção seguro com permissões de acesso centralizadas.

2.4 Módulo de Importação

A primeira e mais importante aplicação a ser utilizada é a de **Importação de dados**. O seu objetivo é converter ficheiros CSV em registos estruturados.

2.4.1 Tecnologias e Bibliotecas

- `pandas`**:** Utilizada para a leitura eficiente de ficheiros massivos. Permite o tratamento prévio de valores nulos (conversão de `NaN` para `NULL`) e cálculos de parâmetros orbitais derivados;
- `mssql-python`**:** Responsável pela comunicação com o SQL Server, permitindo a execução eficiente de transações e comandos de inserção;
- `tkinter`**:** Utilizada para fornecer uma **Interface Gráfica do Utilizador (GUI)**, permitindo que o utilizador possa gerir o processo de importação de forma visual, sem necessidade de recorrer à linha de comandos.

2.4.2 Fluxo de Funcionamento

O algoritmo de importação, implementado no script `pipeline.py`, segue o seguinte fluxo para garantir a integridade referencial:

1. **Conexão e Leitura:** O sistema carrega as configurações e lê os *datasets* (`neo.csv`) e (`MPCORB.DAT`);
2. **Limpeza:** Verifica a existência de chaves primárias duplicadas, remove colunas irrelevantes e formata datas, de forma a serem compatíveis com SQL;
3. **Inserção Sequencial:**
   - Insere ou atualiza os registos na entidade `Asteroide`;
   - Insere os dados dependentes na tabela `Orbita`, garantindo que cada órbita fica associada ao ID correto do asteroide.
4. **Feedback:** A interface reporta o progresso em tempo real e notifica o utilizador sobre o sucesso da operação ou eventuais erros de violação de restrições.

2.5 Relação com Aplicações Finais

Embora este capítulo se foque na infraestrutura de dados, é importante notar que o módulo de importação foi desenhado para servir de base às aplicações de exploração visual:

- **Gestor de Alertas:** Consome os dados de risco calculados no momento da inserção.
- **Dashboard de Monitorização:** Visualiza as estatísticas agregadas dos dados importados.

Capítulo

3

Modelação e Base de Dados

3.1 Introdução

Este capítulo descreve a evolução da nossa BD, apresentando as versões iniciais e finais dos modelos concetual e lógico, justificando as alterações introduzidas para corrigir falhas de normalização e melhorar a integridade dos dados.

3.2 Modelo Concetual

A fase inicial focou-se na identificação das entidades principais e das suas relações semânticas, resultando no nosso primeiro DEA.

3.2.1 Evolução do Diagrama

A primeira versão do modelo focava-se essencialmente nas entidades nucleares. Tendo em conta que foi a versão inicial, não estava muito carente de alterações, no entanto acabaram por faltar duas entidades, `Alerta` e `Imagem`. Para as entidades que já tinham sido identificadas, já tinham definida a sua cardinalidade, o que é positivo, mas faltava incluir a participação.

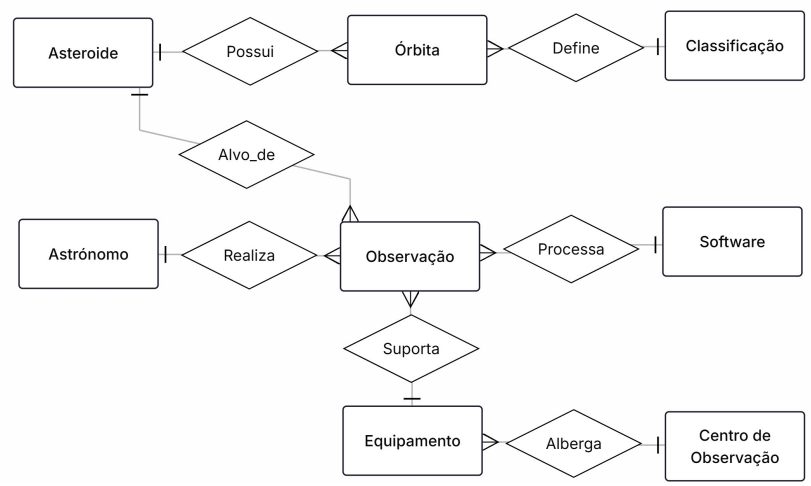


Figura 3.1: Versão inicial do Modelo Concetual.

Após alguma reflexão e análise dos requisitos, o modelo evoluiu para a sua versão final. Foram introduzidas as duas entidades em falta e incluida uma relação entre o `Centro de Observação` e o `Astrónomo`. Além disso, foram esclarecidas as participações de todas as entidades face a todas as suas relações.

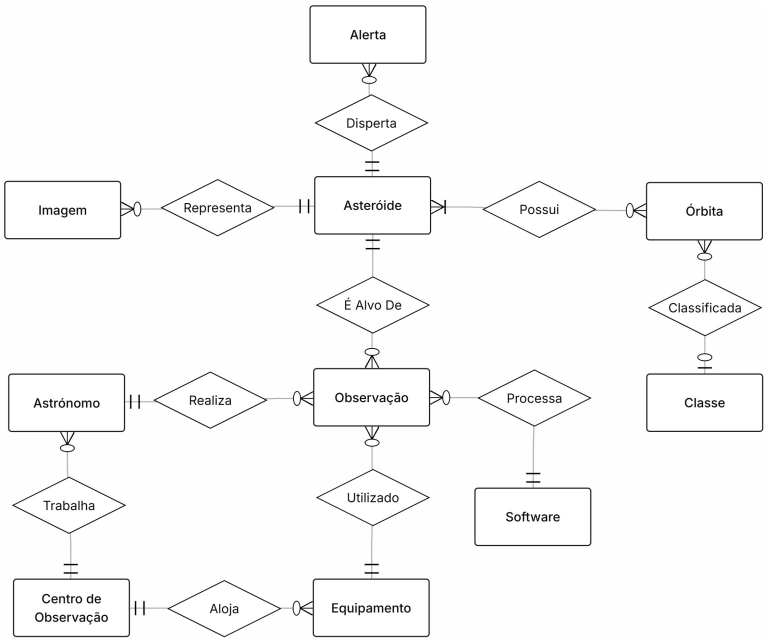


Figura 3.2: Versão final do Modelo Concetual.

O modelo final orbita em torno de duas entidades nucleares:

- **Asteroide:** Representa o corpo celeste em si, isto é, o seus dados estáticos como diâmetro e albedo.
- **Órbita:** Representa a trajetória numa determinada época. A relação `Possui` (1:N) permite manter um histórico evolutivo, fundamental para calcular e recalcular riscos.

3.3 Modelo Lógico

A conversão para o modelo lógico envolveu a tradução das entidades para tabelas e a normalização até à **3FN**.

3.3.1 Evolução do Esquema

Numa primeira fase, a conversão direta resultou em tabelas com algumas redundâncias. O principal problema consistia na repetição de dados textuais especialmente no que toca à tabela `Órbita`, visto que os datasets têm os mesmos dados com nomes diferentes e nesse esquema temos todos eles, o que é redundante. Além disso, faltavam as tabelas `Alerta` e `Imagem`, visto nessa fase ainda não as termos colocado no modelo concetual 3.1.

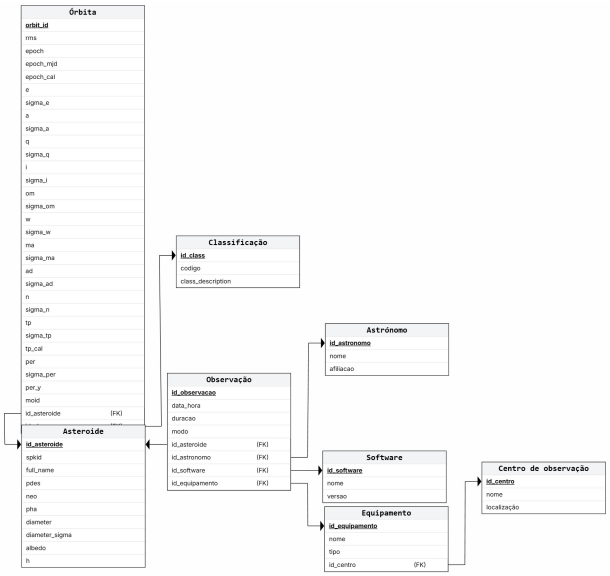


Figura 3.3: Versão inicial do Modelo Lógico.

Na versão final, o esquema foi melhorado, de forma a garantir a integridade referencial, a eficiência de armazenamento e a normalização até à 3FN.

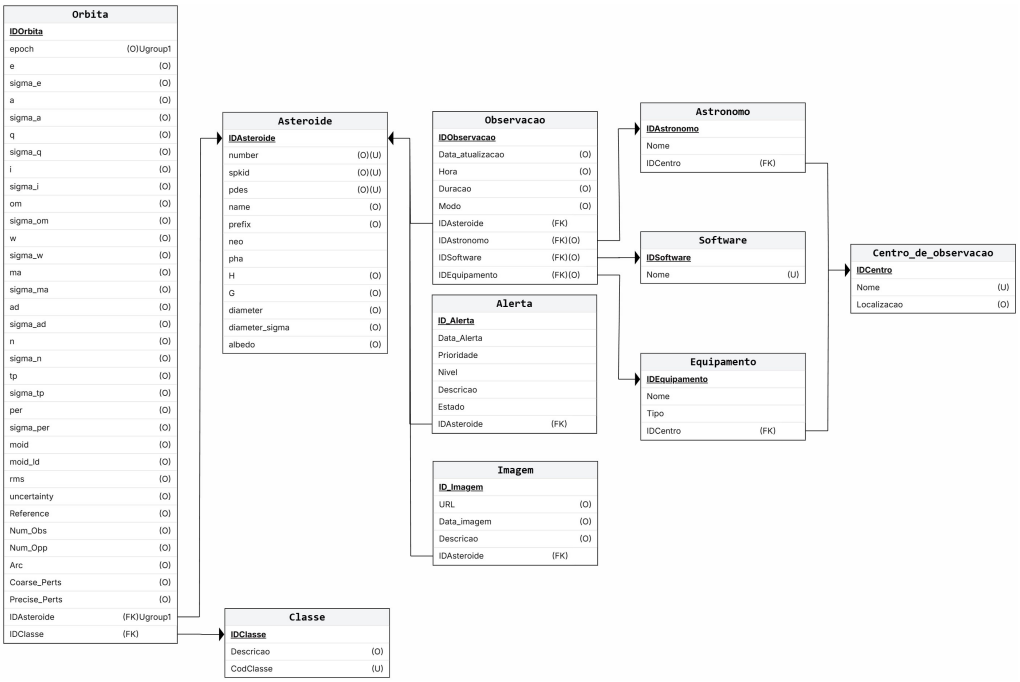


Figura 3.4: Esquema Relacional (Modelo Lógico) final e normalizado.

As principais melhorias na versão final (Figura 3.4) incluem:

- **Tabelas em falta previamente:** Inclusão das tabelas `Imagem` e `Alerta`;
- **Segregação de Dados:** Separação clara entre `Asteroide` e `Orbita`;
- **Tabela `Classe`:** Criação da tabela `Classe`, retirando a redundância da tabela de órbitas.
- **Rastreabilidade:** A tabela `Observacao` foi reestruturada para atuar como tabela de ligação entre as tabelas `Astronomo`, `Equipamento` e `Software` através de chaves estrangeiras.

3.4 Modelo Físico e Implementação SQL

A implementação física no Microsoft SQL Server, detalhada no script `tables.sql` incluido nos Apêndices **??**, introduziu otimizações técnicas cruciais.

3.4.1 Tipos de Dados e Precisão

Dado o contexto astronómico, o tipo de dados `FLOAT` foi substituído pelo uso extensivo de `DECIMAL(30, 10)` para parâmetros orbitais, como, por exemplo, a excentricidade e o semieixo maior, garantindo, assim, a precisão necessária para o cálculo do MOID.

3.4.2 Regras de Integridade

Para além das chaves estrangeiras, foram implementados mecanismos de segurança de dados ao nível do servidor:

- **Verificações:** Restrições `CHECK`, por exemplo, `diameter > 0`, `i <= 180`) impedem a inserção de dados fisicamente impossíveis.
- **Triggers:** O sistema utiliza a função `fn_Torino_Simplificada` e triggers automáticos para inserir alertas na tabela **Alerta** sempre que uma órbita perigosa é detetada.

3.5 Views

Como camada de abstração, foram criadas views como, por exemplo, a `vw_ProximosEventosCriticos`, que lista os asteroides com a maior aproximação à Terra, facilitando a consulta por parte das aplicações finais.

Capítulo

4

Distribuição de Tarefas

4.1 Introdução

O desenvolvimento deste projeto envolveu um esforço conjunto, tendo as tarefas sido distribuídas de forma a maximizar a eficiência e tirar partido das competências individuais de cada membro do grupo.

4.2 Divisão de Responsabilidades

As tarefas foram categorizadas em cinco grandes áreas: Modelação e Base de Dados, Desenvolvimento da aplicação de importação, Desenvolvimento da aplicação de alertas, desenvolvimento da aplicação de monitorização e Documentação.

A distribuição detalhada do esforço realizou-se da seguinte forma:

Tabela 4.1: Responsabilidades individuais e distribuição de tarefas.

Tarefa	**Responsável**	**%**
Base de Dados (SQL Server)		
Modelação Concetual e Lógica (ERDPlus)	Diogo e Júlia	50/50
Implementação Física (DDL, Tabelas e Índices)	Diogo	100%
Programação de Lógica de Servidor (Triggers e Stored Procedures)	Júlia e Guilherme	70/30
Criação de Vistas para as Aplicações	Júlia e Guilherme	70/30
Revisão, correção e melhorias	Bruno	100%
Aplicação de Importação (Python)		
Desenvolvimento do extrator e processador de dados	Bruno	100%
Implementação da conexão `mssql-python` e tratamento de erros	Bruno	100%
Revisão, correção e melhorias	Bruno	100%
Aplicações Cliente (GUI)		
Desenvolvimento da aplicação *Gestor de Alertas* (Tkinter)	Júlia	100%
Desenvolvimento da aplicação *Monitorização*	Júlia	100%
Integração das interfaces com a Base de Dados	Bruno	100%
Revisão, correção e melhorias	Bruno	100%
Relatório		
Redação e Formatação em LaTeX	Júlia	100%
Revisão Final	Bruno	100%

4.3 Calendarização e Metodologia

O trabalho decorreu ao longo do semestre, a partir da data de receção do enunciado. Numa primeira fase, o foco incidiu exclusivamente na modelação da base de dados, visto que foi o tópico trabalhado diretamente nas aulas práticas nessa altura. Depois, a atenção voltou-se para importação correta dos dados dos datasets. Só após a garantia da integridade dos dados é que se procedeu ao desenvolvimento das aplicações de monitorização e alertas.

Capítulo

5

Acesso à Base de Dados

5.1 Introdução

A comunicação entre as aplicações cliente e o servidor de base de dados constitui um ponto crítico da arquitetura do sistema. O acesso aos dados foi desenhado para conciliar os requisitos impostos pela UC, isto é, o alojamento em servidor centralizado com as boas práticas de engenharia de software, segurança e modularidade.

Este capítulo detalha as tecnologias de conectividade utilizadas e a estratégia implementada para gerir as credenciais fornecidas.

5.2 Tecnologia de Conexão: `mssql-python`

Para estabelecer a ponte entre a lógica aplicacional e o SGBD, optou-se pela biblioteca **`mssql-python`**. Esta escolha, em detrimento do driver ODBC tradicional, deve-se à sua arquitetura mais moderna e à capacidade de mapear nativamente os tipos de dados do SQL Server, como, por exemplo, `DATETIME2` e `DECIMAL` para objetos Python, sem perda da sua precisão.

5.3 Segurança e Gestão de Credenciais

O ambiente de produção, remoto, impunha o uso de um utilizador e palavra-passe específicos, definidos pela equipa docente para efeitos de avaliação. Para evitar a exposição desnecessária das credenciais no código-fonte, optou-se por armazená-las em variáveis de ambiente.

Utilizando a biblioteca `python-dotenv`, a aplicação foi configurada para ler as credenciais sensíveis a partir de um ficheiro `.env` local.

```
import mssql_python
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# (...)

def ligar_bd(self):
    try:
        conn_str = os.getenv('SQL_CONNECTION_STRING')
        if not conn_str:
            raise ValueError("A variavel de ambiente
                SQL_CONNECTION_STRING nao esta definida no ficheiro .env
                ")

        self.conn = mssql_python.connect(conn_str)
        self.cursor = self.conn.cursor()
        return True
    except Exception as e:
        messagebox.showerror("Erro Critico", f"Falha SQL:\n{e}")
        self.root.destroy()
        return False
```

Excerto de Código 5.1: Implementação da conexão na aplicação de Alertas

O excerto acima, retirado da aplicação *Gestor de Alertas* (linhas 3-10 e 53-65), demonstra que a string de conexão é recuperada via `os.getenv`, garantindo que as credenciais acabam por não ser escritas diretamente no código-fonte.

5.4 Configuração da String de Conexão

A flexibilidade do sistema permite alternar entre os ambientes de teste e avaliação apenas alterando a variável `SQL_CONNECTION_STRING`:

- **Ambiente de Desenvolvimento e Testes, Local:** Utiliza Autenticação Integrada do Windows, Trusted Connection, para testes rápidos sem gestão de passwords;
- **Ambiente de Produção, Remoto:** Conecta-se ao IP `192.168.100.14` utilizando a Autenticação SQL explícita com as credenciais do grupo (`User_BD_PL1_01`), conforme estipulado no enunciado do trabalho prático.

Capítulo

6

Funcionalidades e Fluxo de Dados

6.1 Introdução

Com a base de dados modelada e o servidor configurado, o sistema foi submetido a testes de carregamento e processamento. Este capítulo descreve o fluxo funcional da solução, validando a integridade dos dados importados e a eficácia dos mecanismos automáticos de deteção de alertas.

As interfaces gráficas desenvolvidas para interagir com este sistema, aplicações cliente, serão detalhadas textual e visualmente no Capítulo 7.

6.2 Importação

A funcionalidade central do sistema é a capacidade de inserção de grandes volumes de dados astronómicos. O módulo de importação foi testado com os datasets fornecidos, `NEO` e `MPC`. A estrutura do ficheiro de dados segue o formato oficial definido pelo **Minor Planet Center** [3], sendo necessário processar colunas de largura fixa e descodificar as flags orbitais.

6.2.1 Resultados da Importação

Tendo sido executado com sucesso, o processo de importação resulta na população das tabelas principais. A validação dos dados na BD confirma a correta normalização de:

- **Asteroides:** Foram criados registos únicos na tabela `Asteroide`, evitando duplicados mesmo quando presentes nos ficheiros de origem;

- **Órbitas:** Cada registo orbital foi associado corretamente ao seu corpo celeste, relação 1:N, mantendo o histórico de observações;
- **Performance:** A utilização do `mssql-python` e de transações em lote permitiu tempos de inserção viáveis para o volume de dados tratado, apesar desse não ser o objetivo principal deste trabalho prático ou da UC.

6.3 Mecanismo de Deteção de Alertas

Outra das funcionalidades críticas do sistema é a classificação automática de riscos, que ocorre ao nível do servidor de BD.

6.3.1 Verificação da Lógica de Triggers

Para validar esta funcionalidade, confirmou-se que a inserção de novos dados orbitais aciona corretamente os *Triggers* criados. O fluxo observado foi o seguinte:

1. **Entrada:** Uma nova órbita com `moid_ld < 1` é inserida pelo módulo Python;
2. **Processamento:** O SQL Server aciona o trigger `trg_alerta_alta_prioridade`, que invoca a função `fn_Torino_Simplificada` para calcular o nível de risco e inserir o alerta;
3. **Saída:** Um registo é criado automaticamente na tabela `Alerta`.

Uma simples consulta à tabela `Alerta` é capaz de demonstrar que o sistema foi capaz de identificar e categorizar os eventos de risco com diferentes níveis de prioridade com sucesso.

6.4 Resultados Globais

O sistema final apresenta-se como uma solução integrada capaz de responder aos requisitos propostos:

- **Integridade:** As restrições impedem a entrada de dados fisicamente impossíveis, por exemplo diâmetros negativos;
- **Automatização:** A gestão de riscos é baseada em eventos de inserção;

- **Segurança:** O acesso aos dados respeita as credenciais, garantindo a proteção da informação.

Capítulo

7

Aplicações Cliente e Interface

7.1 Introdução

Enquanto o capítulo anterior validou a lógica de dados, este capítulo foca-se na camada de apresentação. O sistema é composto por três aplicações distintas, desenvolvidas em Python com a biblioteca **Tkinter**.

7.2 Aplicação de Importação de Dados

A primeira interface desenvolvida tem como objetivo simplificar o processo complexo de inserção de dados. Ao abstrair os scripts de linha de comandos numa GUI visualmente mais intuitiva, reduz-se a probabilidade de erro humano na seleção dos ficheiros.

7.2.1 Interface Principal

A janela principal apresenta um layout com botões claros para a seleção dos *datasets* utilizados, `NEO` e `MPC`.

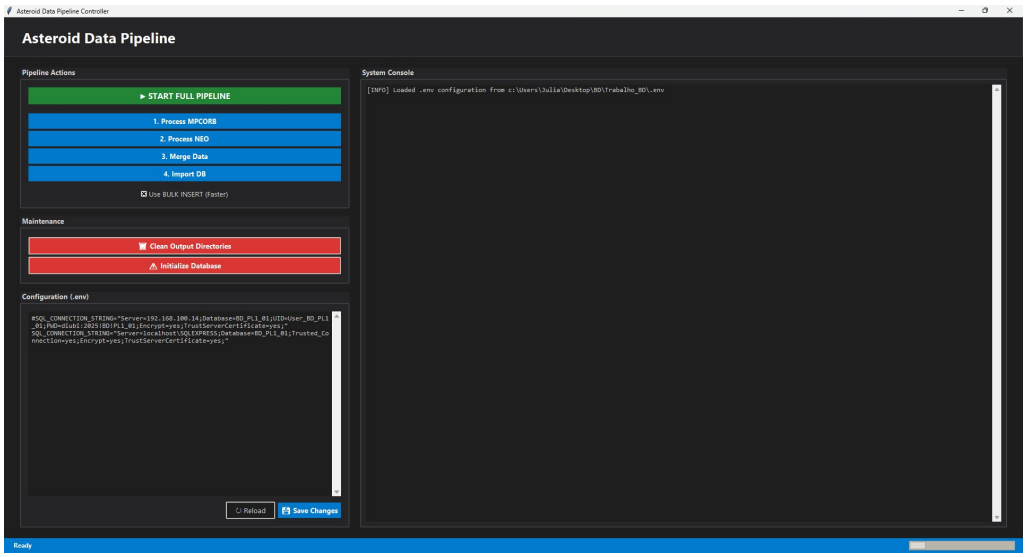


Figura 7.1: Ecrã inicial da aplicação de Importação.

7.2.2 Feedback de Processamento

Uma das principais preocupações relativas a esta aplicação foi a responsividade. Como a inserção de tantos registos, acaba por se tornar uma operação bastante demorada, então o processo corre numa *thread* separada. Isto permite que a interface se mantenha ativa, fornecendo feedback em tempo real através de uma barra de progresso e de uma consola de logs detalhada.

7.3 Gestor de Alertas

A aplicação de gestão de alertas, `alertas.pyw`, é essencial no que toca à segurança. A interface foi desenhada para permitir a identificação imediata de ameaças.

7.3.1 Codificação Visual (Escala de Torino)

Para facilitar a leitura rápida, a tabela de alertas utiliza um esquema de cores baseado na gravidade do evento, mapeando diretamente os níveis da Escala de Torino Modificada, calculados pelo servidor:

- **Vermelho (Nível 4):** Aproximação Iminente ou Perigo Extremo;
- **Laranja (Nível 3):** Eventos de alta probabilidade de impacto;

- **Amarelo (Nível 2):** Eventos que merecem atenção dos astrónomos;
- **Verde (Nível 1):** Eventos de rotina ou de baixo risco.

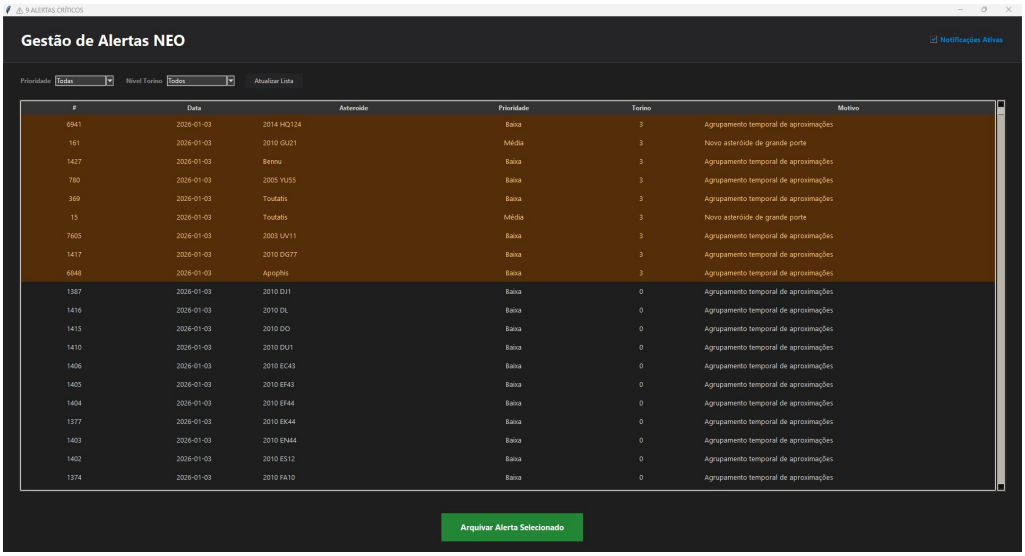


Figura 7.2: Painel de Alertas com codificação de cores por nível de risco.

Como é visível na Figura 7.2, a aplicação permite ainda filtrar os resultados por Prioridade e Nível, consultando dinamicamente a BD para apresentar apenas a informação relevante à nossa consulta.

7.3.2 Ações do Utilizador

Para além da visualização, a interface permite marcar alertas como "Resolvidos". Esta ação remove o alerta da vista ativa, atualizando o campo `Estado` na BD. Isto permite que os utilizadores se foquem apenas nos problemas pendentes, isto é, nos alertas ainda não vistos.

7.4 Monitorização

Por fim, a aplicação de monitorização oferece uma visão macroscópica do catálogo de asteroides. Ao contrário da lista de alertas, que é orientada a eventos, este painel apresenta estatísticas agregadas.

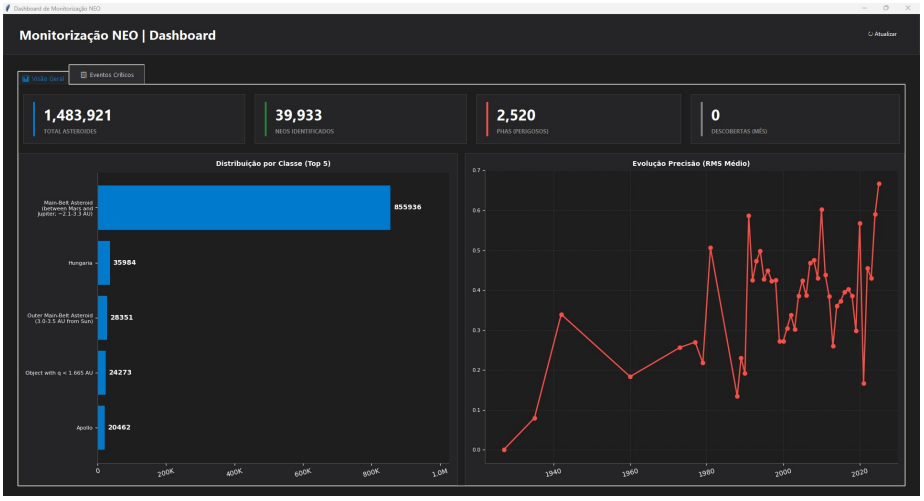


Figura 7.3: Dashboard com estatísticas globais do sistema.

Esta interface utiliza as views criadas no SQL Server, como, por exemplo, a `vw_EstatisticasDescoberta`, garantindo que os dados apresentados estão sempre sincronizados com a realidade da base de dados sem sobrecarregar o sistema com cálculos complexos.

7.5 Experiência de Utilizador

Transversalmente a todas as aplicações, optou-se por um tema escuro, utilizando uma paleta de cores, fundo `#1e1e1e` e texto `#d4d4d4`, que minimiza o cansaço visual. Esta decisão de design é particularmente relevante num contexto de astronomia, onde os operadores trabalham frequentemente em ambientes com pouca luz.

Capítulo

8

Conclusões

8.1 Síntese do Trabalho

Este trabalho prático tinha como objetivo o desenvolvimento de um sistema de informação robusto para a monitorização de NEOs. Finalizada a etapa de implementação e testes, é plausível afirmar que todos os requisitos propostos foram cumpridos com sucesso.

A solução desenvolvida demonstrou a eficácia da arquitetura Cliente/Servidor na gestão de dados. A BD, implementada em Microsoft SQL Server, não serve apenas como um repositório passivo, mas atua como um motor ativo de regras de negócio, garantindo a integridade dos dados orbitais e a deteção automática de riscos através dos *Triggers* criados.

Paralelamente, o ecossistema de aplicações em Python é a ponte entre os dados brutos e o utilizador final. A aplicação de importação resolveu o desafio de normalizar e carregar grandes volumes de informação, enquanto as interfaces gráficas permitiram uma visualização clara e intuitiva dos alertas de segurança.

8.2 Análise Crítica e Principais Desafios

O desenvolvimento deste sistema permitiu consolidar diversos conceitos teóricos e práticos, destacando-se os seguintes pontos:

- **Precisão e Tipos de Dados:** A natureza científica do projeto impôs desafios específicos. A transição inicial de `FLOAT` para `DECIMAL(30,10)` acabou por ser crucial para evitar erros de arredondamento no cálculo do parâmetro MOID, garantindo a fiabilidade dos alertas gerados.

- **Normalização:** O processo de normalização até à 3FN obrigou a uma reestruturação dos dados originais nos ficheiros CSV, resultando num esquema lógico mais eficiente e sem redundâncias.

8.3 Considerações Finais

Em suma, este projeto permitiu aplicar de forma integrada competências de Modelação de Dados, SQL Avançado e Programação. O resultado final é um sistema funcional, resiliente a erros e capaz de auxiliar na tarefa crítica de monitorizar corpos celestes potencialmente perigosos.

Capítulo

9

Epílogo

A realização deste trabalho permitiu consolidar, de forma prática, os conhecimentos adquiridos ao longo da UC. A transição da teoria, modelação, para a prática, implementação física e desenvolvimento de aplicações, proporcionou uma perspetiva realista dos desafios enfrentados na engenharia de software.

Globalmente, o balanço da disciplina é positivo. A estrutura do projeto obrigou-nos a lidar com problemas reais, como, por exemplo, a integridade de dados, a normalização e a conectividade entre linguagens diferentes, que, apesar de discutidos em aulas teóricas e práticas, se aprendem muito melhor trabalhando.

No entanto, na opinião do grupo, podiam ter-se explorado temas como *Triggers* nas aulas práticas, visto que, ao desenvolver este trabalho prático, não tínhamos bem noção de como os fazer. Talvez com a sua inclusão numa aula prática fosse de certa forma mais fácil avançar nesse campo. Por exemplo, tivemos uma aula prática relativa à modelação, o que ajudou bastante na parte relativa à modelação no trabalho.

Ainda assim, este obstáculo acabou por se revelar formativo, pois obrigou o grupo a desenvolver autonomia, o que também é essencial para o futuro profissional (acreditamos que esta tenha sido a ideia do docente).

Bibliografia

[1] Maria Elizabeth Zucolotto, Ariadne C Fonseca, and Loiva L Antonello. O alvo terra. *Revista Carbono*, 1(05):2014, 2013.

[2] Amy Mainzer, T Grav, J Bauer, J Masiero, RS McMillan, RM Cutri, R Walker, E Wright, P Eisenhardt, DJ Tholen, et al. Neowise observations of near-earth objects: Preliminary results. *The Astrophysical Journal*, 743(2):156, 2011.

[3] J.L. Galache. *Guide to the Extended Versions of MPC Data Files Based on the MPCORB Format*. Minor Planet Center (MPC), May 2024. Last updated: 2024/05/08.

Apêndice

A

Script de Criação da Base de Dados

Apresenta-se abaixo o *script* SQL desenvolvido para a criação da estrutura física da BD, incluindo a definição de tabelas, chaves primárias e restrições de integridade.

```
CREATE TABLE Classe (
  IDClasse INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Descricao VARCHAR(255),
  CodClasse VARCHAR(50) NOT NULL UNIQUE
);
GO

CREATE TABLE Software (
  IDSoftware INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Nome VARCHAR(100) NOT NULL UNIQUE
);
GO

CREATE TABLE Centro_de_observacao (
  IDCentro INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Nome VARCHAR(100) NOT NULL UNIQUE,
  Localizacao VARCHAR(100)
);
GO

CREATE TABLE Asteroide (
  IDAsteroide INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  number INT,
  spkid VARCHAR(20),
  pdes VARCHAR(20),
  name VARCHAR(100),
  prefix VARCHAR(10),
  neo BIT NOT NULL DEFAULT 0,
  pha BIT NOT NULL DEFAULT 0,
  H DECIMAL(10, 5),
  G DECIMAL(10, 5),
```

```
  diameter DECIMAL(10, 5),
  diameter_sigma DECIMAL(10, 5),
  albedo DECIMAL(10, 5),
  CONSTRAINT CK_Asteroide_Diameter CHECK (diameter > 0),
  CONSTRAINT CK_Asteroide_Albedo CHECK (albedo >= 0 AND albedo <=
      1)
);
GO

CREATE UNIQUE NONCLUSTERED INDEX IX_Asteroide_spkid ON Asteroide(
    spkid) WHERE spkid IS NOT NULL;
CREATE UNIQUE NONCLUSTERED INDEX IX_Asteroide_pdes ON Asteroide(
    pdes) WHERE pdes IS NOT NULL;
CREATE UNIQUE NONCLUSTERED INDEX IX_Asteroide_number ON Asteroide(
    number) WHERE number IS NOT NULL;
CREATE NONCLUSTERED INDEX IX_Asteroide_name ON Asteroide(name)
    WHERE name IS NOT NULL;
CREATE INDEX IX_Asteroide_Flags ON Asteroide(pha, neo) INCLUDE (
    diameter, H);
GO

CREATE TABLE Astronomo (
  IDAstronomo INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Nome VARCHAR(100) NOT NULL,
  IDCentro INT NOT NULL,
  FOREIGN KEY (IDCentro) REFERENCES Centro_de_observacao(IDCentro)
);
GO

CREATE TABLE Equipamento (
  IDEquipamento INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  Nome VARCHAR(100) NOT NULL,
  Tipo VARCHAR(50) NOT NULL,
  IDCentro INT NOT NULL,
  FOREIGN KEY (IDCentro) REFERENCES Centro_de_observacao(IDCentro)
);
GO

CREATE TABLE Orbita (
  IDOrbita INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  IDAsteroide INT NOT NULL,
  epoch DATE,
  e DECIMAL(20, 10),
  sigma_e DECIMAL(30, 10),
  a DECIMAL(30, 10),
  sigma_a DECIMAL(30, 10),
  q DECIMAL(30, 10),
  sigma_q DECIMAL(30, 10),
  i DECIMAL(20, 10),
  sigma_i DECIMAL(30, 10),
  om DECIMAL(20, 10),
  sigma_om DECIMAL(30, 10),
  w DECIMAL(20, 10),
  sigma_w DECIMAL(30, 10),
  ma DECIMAL(20, 10),
  sigma_ma DECIMAL(30, 10),
  ad DECIMAL(30, 10),
  sigma_ad DECIMAL(30, 10),
  n DECIMAL(20, 10),
```

```
  sigma_n DECIMAL(30, 10),
  tp DATETIME2(7),
  sigma_tp DECIMAL(30, 10),
  per DECIMAL(30, 10),
  sigma_per DECIMAL(30, 10),
  moid DECIMAL(20, 10),
  moid_ld DECIMAL(20, 10),
  rms DECIMAL(10, 5),
  uncertainty VARCHAR(10),
  Reference VARCHAR(50),
  Num_Obs INT,
  Num_Opp INT,
  Arc VARCHAR(20),
  Coarse_Perts VARCHAR(20),
  Precise_Perts VARCHAR(20),
  IDClasse INT,
  FOREIGN KEY (IDAsteroide) REFERENCES Asteroide(IDAsteroide),
  FOREIGN KEY (IDClasse) REFERENCES Classe(IDClasse),
  CONSTRAINT CK_Orbita_Eccentricity CHECK (e >= 0),
  CONSTRAINT CK_Orbita_Inclination CHECK (i >= 0 AND i <= 180),
  CONSTRAINT CK_Orbita_Perihelion CHECK (q > 0)
);
GO

CREATE UNIQUE INDEX UQ_Orbita_Asteroide_Epoch ON Orbita(IDAsteroide
    , epoch) WITH (IGNORE_DUP_KEY = ON);
GO

CREATE TABLE Observacao (
  IDObservacao INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  IDAsteroide INT NOT NULL,
  IDAstronomo INT,
  IDSoftware INT,
  IDEquipamento INT,
  Data_atualizacao DATE,
  Hora TIME,
  Duracao DECIMAL(10, 2),
  Modo VARCHAR(50),
  FOREIGN KEY (IDAsteroide) REFERENCES Asteroide(IDAsteroide),
  FOREIGN KEY (IDAstronomo) REFERENCES Astronomo(IDAstronomo),
  FOREIGN KEY (IDEquipamento) REFERENCES Equipamento(IDEquipamento)
      ,
  FOREIGN KEY (IDSoftware) REFERENCES Software(IDSoftware)
);
GO

CREATE TABLE Imagem (
  ID_Imagem INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  IDAsteroide INT NOT NULL,
  URL VARCHAR(255),
  Data_imagem DATE,
  Descricao VARCHAR(255),
  FOREIGN KEY (IDAsteroide) REFERENCES Asteroide(IDAsteroide)
);
GO

CREATE TABLE Alerta (
  ID_Alerta INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
  IDAsteroide INT NOT NULL,
```

```
  Data_Alerta DATE NOT NULL,
  Prioridade VARCHAR(20) NOT NULL,
  Nivel INT NOT NULL,
  Descricao VARCHAR(255) NOT NULL,
  Estado VARCHAR(20) NOT NULL DEFAULT 'Ativo',
  FOREIGN KEY (IDAsteroide) REFERENCES Asteroide(IDAsteroide),
  CONSTRAINT CK_Alerta_Prioridade CHECK (Prioridade IN ('Baixa', '
      M dia', 'Alta')),
  CONSTRAINT CK_Alerta_Estado CHECK (Estado IN ('Ativo', 'Resolvido
      ', 'Ignorado')),
  CONSTRAINT CK_Alerta_Nivel CHECK (Nivel BETWEEN 0 AND 4)
);
GO
```

Excerto de Código A.1: Estrutura de Tabelas (tables.sql)

Apêndice

B

Programação de Base de Dados

Este apêndice agrupa as *Views* e os *Triggers* da BD.

B.1 Views

As views foram criadas para abstrair consultas complexas e fornecer dados diretamente ao *Dashboard* de monitorização da aplicação cliente.

```
CREATE OR ALTER VIEW vw_EstatisticasAlerta AS
SELECT
  (SELECT COUNT(*) FROM Alerta WHERE Nivel = 4 AND Estado = 'Ativo'
     ) AS Alertas_Vermelhos,
  (SELECT COUNT(*) FROM Alerta WHERE Nivel = 3 AND Estado = 'Ativo'
     ) AS Alertas_Laranja,
  (SELECT COUNT(*) FROM Asteroide WHERE pha = 1 AND diameter > 0.1)
      AS Total_PHAs_Monitorizados_GT_100m;
GO

CREATE OR ALTER VIEW vw_ProximosEventosCriticos AS
SELECT TOP 5
  o.tp AS Data_Proxima_Aproximacao,
  a.IDAsteroide AS IDAsteroide,
  a.name AS Nome,
  a.pdes as Designacao_Provisoria,
  o.moid_ld AS Distancia_Lunar,
  a.diameter AS Diametro
FROM Orbita o
JOIN Asteroide a ON o.IDAsteroide = a.IDAsteroide
WHERE o.moid_ld IS NOT NULL
  AND o.moid_ld < 5
  AND o.tp IS NOT NULL
  AND CAST(o.tp AS DATE) >= CAST(GETDATE() AS DATE)
ORDER BY o.tp ASC;
GO

CREATE OR ALTER VIEW vw_EstatisticasDescoberta AS
SELECT
```

```
  COUNT(a.IDAsteroide) AS Novos_NEOs_ultimo_mes
FROM Asteroide a
JOIN Orbita o ON a.IDAsteroide = o.IDAsteroide
WHERE a.neo = 1
  AND CAST(o.epoch as DATE) BETWEEN DATEADD(MONTH, -1, GETDATE())
      AND GETDATE();
GO

CREATE OR ALTER VIEW vw_EvolucaoPrecisao AS
SELECT TOP 1000
  YEAR(epoch) AS Ano,
  AVG(rms) AS rms_medio
FROM Orbita
WHERE epoch IS NOT NULL AND rms IS NOT NULL
GROUP BY YEAR(epoch)
ORDER BY Ano ASC;
GO

CREATE OR ALTER VIEW vw_Last5Detected AS
SELECT TOP 5
  a.IDAsteroide AS IDAsteroide,
  a.pdes AS Designacao_Provisoria,
  a.name AS Nome,
  o.epoch AS Data_Deteccao
FROM Asteroide a
JOIN Orbita o ON a.IDAsteroide = o.IDAsteroide
ORDER BY o.epoch DESC;
GO

CREATE OR ALTER VIEW vw_PHA_NEO AS
SELECT
  a.IDAsteroide AS IDAsteroide,
  a.pdes AS Designacao_Provisoria,
  a.name AS Nome,
  a.neo AS NEO,
  a.pha AS PHA
FROM Asteroide a
WHERE a.neo = 1 AND a.pha = 1;
GO

CREATE OR ALTER VIEW vw_TopCenters AS
SELECT TOP 10
  c.IDCentro AS IDCentro,
  c.Nome AS Nome,
  c.Localizacao AS Localizacao,
  COUNT(DISTINCT o.IDObservacao) AS total_observacoes
FROM Centro_de_observacao c
JOIN Equipamento e ON c.IDCentro = e.IDCentro
JOIN Observacao o ON o.IDEquipamento = e.IDEquipamento
GROUP BY c.IDCentro, c.Nome, c.Localizacao
ORDER BY total_observacoes DESC;
GO

CREATE OR ALTER VIEW vw_LargestPHAs AS
SELECT TOP 20
  a.IDAsteroide AS IDAsteroide,
  a.pdes AS Designacao_Provisoria,
  a.name AS Nome,
  a.diameter AS Diametro
```

```
FROM Asteroide a
WHERE a.pha = 1
ORDER BY a.diameter DESC;
GO
```

Excerto de Código B.1: Vistas de Estatística

B.2 Stored Procedures

Desenvolveu-se um procedimento armazenado para encapsular a lógica de manipulação de dados, garantindo a consistência das operações.

```
CREATE PROCEDURE dbo.sp_AlterarEstadoAlerta
    @ID_Alerta INT,
    @NovoEstado VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;
    IF @NovoEstado IN ('Ativo', 'Resolvido', 'Ignorado')
        UPDATE Alerta SET Estado = @NovoEstado WHERE ID_Alerta =
            @ID_Alerta;
    ELSE
        RAISERROR('Estado inv lido!', 16, 1);
END;
GO
```

Excerto de Código B.2: Procedimento Armazenado

B.3 Triggers e Funções

Implementação da função de cálculo de risco, Escala de Torino Modificada, e automatização dos alertas.

```
CREATE OR ALTER FUNCTION dbo.fn_Torino_Simplificada (
  @diameter DECIMAL(10,5),
  @moid_ld DECIMAL(20,10),
  @rms DECIMAL(10,5),
  @tp DATETIME2(7),
  @pha BIT
)
RETURNS INT
AS
BEGIN
  DECLARE @hoje DATE = CAST(GETDATE() AS DATE);
  DECLARE @data_evento DATE = CAST(@tp AS DATE);

  IF @diameter > 0.03
     AND @moid_ld < 1
     AND @data_evento BETWEEN @hoje AND DATEADD(DAY,30,@hoje)
    RETURN 4;

  IF @diameter > 0.05
```

```
      AND @moid_ld < 5
      AND @rms < 0.3
    RETURN 3;

  IF @diameter > 0.1
      AND @moid_ld BETWEEN 5 AND 20
      AND @data_evento BETWEEN @hoje and DATEADD(DAY, 180, @hoje)
    RETURN 2;

  IF @pha = 1
      AND @diameter BETWEEN 0.05 AND 0.5
      AND @moid_ld BETWEEN 20 AND 100
    RETURN 1;

  RETURN 0;
END;
GO

CREATE TRIGGER trg_alerta_alta_prioridade
ON Orbita
AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @hoje DATE = CAST(GETDATE() AS DATE);

  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide,
    @hoje,
    'Alta',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'Aproxima  o iminente',
    'Ativo'
  FROM inserted i
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE i.moid_ld < 1
    AND a.diameter > 0.01
    AND CAST(i.tp AS DATE) BETWEEN @hoje AND DATEADD(DAY, 7, @hoje)
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'Aproxima  o iminente'
        AND al.Estado = 'Ativo');

  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide, @hoje,
    'Alta',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'PHA com trajet ria incerta',
    'Ativo'
  FROM inserted i
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE a.pha = 1
    AND a.diameter > 0.1
    AND i.rms > 0.8
```

```
    AND i.moid_ld < 20
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'PHA com trajet ria
        incerta' AND al.Estado = 'Ativo');
END;
GO

CREATE TRIGGER trg_alerta_media_prioridade
ON Orbita
AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @hoje DATE = CAST(GETDATE() AS DATE);

  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide, @hoje,
    'M dia',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'Novo aster ide de grande porte',
    'Ativo'
  FROM inserted i
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE a.diameter > 0.5
    AND i.moid_ld < 50
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'Novo aster ide de grande
        porte' AND al.Estado = 'Ativo');

  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide, @hoje,
    'M dia',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'Mudan a orbital significativa',
    'Ativo'
  FROM inserted i
  JOIN deleted d ON i.IDOrbita = d.IDOrbita
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE (ABS(i.e - d.e) > 0.05 OR ABS(i.i - d.i) > 2)
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'Mudan a orbital
        significativa' AND al.Estado = 'Ativo');
END;
GO

CREATE TRIGGER trg_alerta_baixa_prioridade
ON Orbita
AFTER INSERT, UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @hoje DATE = CAST(GETDATE() AS DATE);
```

```
  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide, @hoje,
    'Baixa',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'Caracter sticas an malas',
    'Ativo'
  FROM inserted i
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE a.albedo > 0.3
    AND i.e > 0.8
    AND i.i > 70
    AND a.diameter > 0.2
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'Caracter sticas an malas
        ' AND al.Estado = 'Ativo');

  INSERT INTO Alerta (IDAsteroide, Data_Alerta, Prioridade, Nivel,
      Descricao, Estado)
  SELECT
    i.IDAsteroide, @hoje,
    'Baixa',
    dbo.fn_Torino_Simplificada(a.diameter, i.moid_ld, i.rms, i.tp,
        a.pha),
    'Agrupamento temporal de aproxima  es',
    'Ativo'
  FROM inserted i
  JOIN Asteroide a ON a.IDAsteroide = i.IDAsteroide
  WHERE i.moid_ld < 10
    AND (
      SELECT COUNT(*)
      FROM Orbita o
      WHERE o.moid_ld < 10
      AND MONTH(o.tp) = MONTH(i.tp)
      AND YEAR(o.tp) = YEAR(i.tp)
    ) > 5
    AND NOT EXISTS (SELECT 1 FROM Alerta al WHERE al.IDAsteroide =
        i.IDAsteroide AND al.Descricao = 'Agrupamento temporal de
        aproxima oes' AND al.Estado = 'Ativo');
END;
GO
```

Excerto de Código B.3: Função de Risco e Triggers