

L. EIC012 - Bases de Dados

Grupo 901:

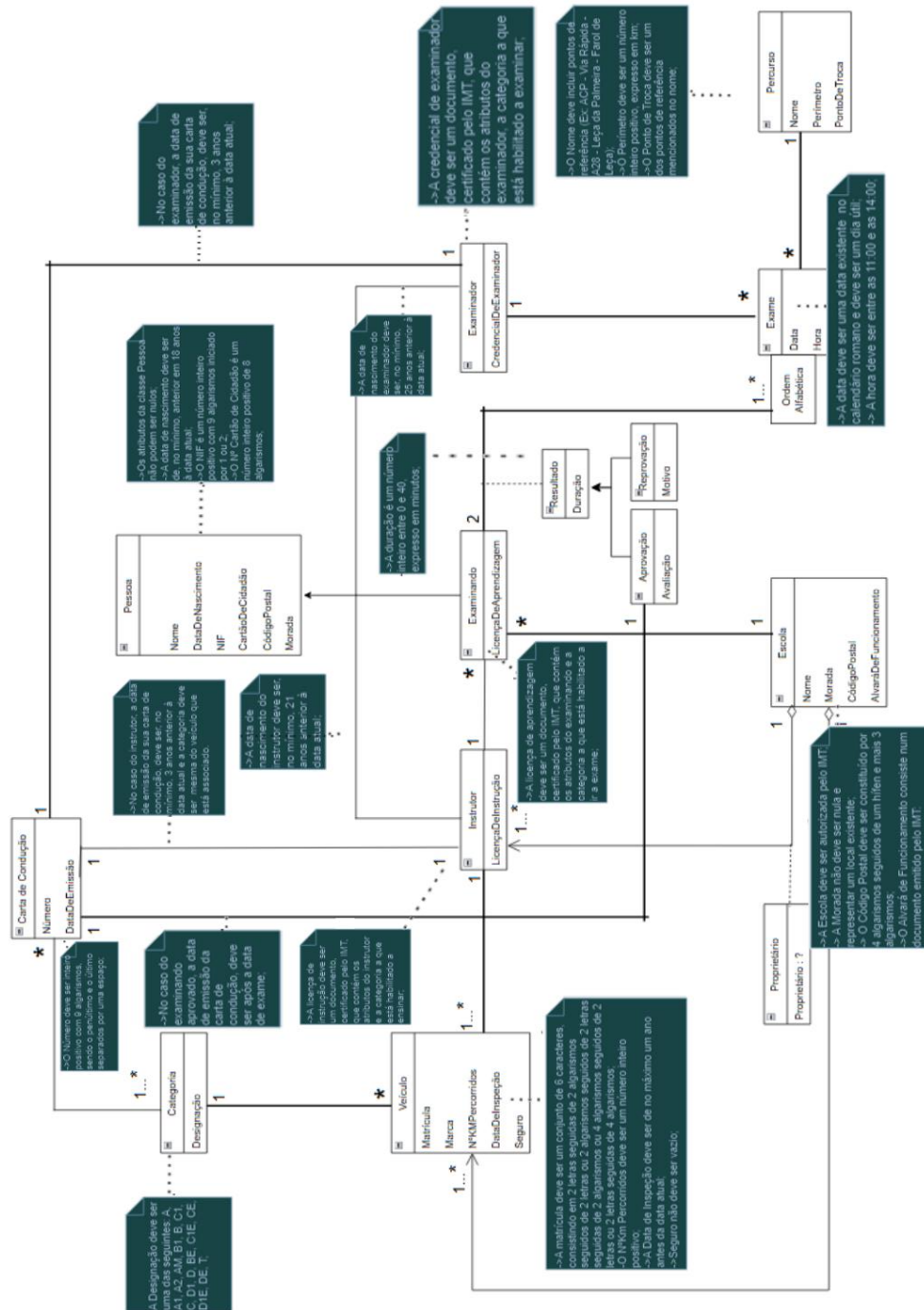
Elton Vaz – up202309925

Henrique Vilarinho – up202307037

Tomás Morais - up202304692

2ª Submissão (Diagrama UML Melhorado, Esquema Relacional,
Implementação em SQLite e Utilização de IA)

Diagrama UML Melhorado



Esquema Relacional

Escola (IDEscola, Nome, Morada, CodigoPostal, AlvaraDeFuncionamento)

Instrutor(IDInstrutor, Nome, DataDeNascimento, NIF, NumeroCartaoCidadao, CodigoPostal, Morada, LicencaDeInstrucao, IDEscola-->Escola, IDCarta-->CartadeConducao)

Proprietario (IDInstrutor-->Instrutor, IDEscola-->Escola, Proprietario ?)

Veiculo (IDVeiculo, Matricula, Marca, NumeroKmPercorridos, DataDeInspecao, Seguro, IDEscola-->Escola, IDInstrutor-->Instrutor, IDCategoria-->Categoria)

Categoria (IDCategoria, Designacao)

CategoriaCarta (IDCarta-->CartaDeConducao, IDCategoria-->Categoria)

CartaDeConducao (IDCarta, Numero, DataDeEmissao)

Percurso (IDPercurso, Nome, Perimetro, PontoDeTroca)

Examinador(IDExaminador, Nome, DataDeNascimento, NIF, CartaoCidadao, CodigoPostal, Morada, CredencialDeExaminador, IDCarta-->CartaDeConducao)

Examinando (IDExaminando, Nome, DataDeNascimento, NIF, NumeroCartaoCidadao, CodigoPostal, Morada, LicencaDeAprendizagem, IDEscola-->Escola, IDInstrutor-->Instrutor)

Exame (IDExame, Data, Hora, IDExaminando-->Examinando, IDExaminador-->Examinador, IDPercurso-->Percurso)

Aprovacao (IDAprovacao, Duracao, Avaliacao, Ordem, IDExaminando-->Examinando, IDExame-->Exame, IDCarta-->CartaDeConducao)

Reprovacao (IDReprovacao, Duracao, Motivo, Ordem, IDExaminando-->Examinando, IDExame-->Exame)

Utilização de AI - Esquema Relacional

IA Utilizada: ChatGPT 4.0

Primeiramente, procuramos verificar a qualidade do nosso esquema relacional, fornecendo-o à IA junto da imagem do Diagrama UML e do seguinte prompt: “Tendo em conta o UML que te envie, analisa qualitativamente este esquema relacional”, tendo sido obtida a seguinte resposta:

Este modelo apresenta um cenário completo para um sistema de gestão de autoescola, incluindo a gestão de veículos, instrutores, examinadores, exames e resultados (aprovações/reprovações). O modelo está consistente com o diagrama UML fornecido, bem estruturado e adequado para implementação em um sistema gerenciador de banco de dados SQLite. Caso precise de ajustes ou normalizações adicionais, posso auxiliar!

De seguida, questionámos acerca de algum aspeto que poderíamos melhorar no nosso esquema, fornecendo o seguinte prompt “Sugeres algo para melhorar o esquema?”. No entanto, o nosso grupo considerou que as modificações sugeridas pela IA eram desnecessárias, uma vez que elas alterariam a estrutura do UML originalmente proposto ou consistiam em normalizações que não consistem no objeto do nosso trabalho, de momento.

Análise de Dependências Funcionais e Formas Normais

Relação	Pessoa	Escola	Veículo
Chaves	IDPessoa (Chave Primária), CartaoDeCidadao, NIF	IDEscola (Chave primária), Nome, AlvaraFuncionamento	IDVeiculo (Chave primária), Matrícula
Funcional Dependencies	IDPessoa →Nome, DataDeNascimento, NIF, CartaoDeCidadao, Morada, CartaoDeCidadao →Nome, DataDeNascimento, NIF, CodigoPostal, Morada, NIF →Nome, DataDeNascimento, CartaoDeCidadao, CodigoPostal, Morada.	IDEscola → Nome, Morada, CodigoPostal, AlvaraFuncionamento. Nome → Morada, CodigoPostal, AlvaraFuncionamento. AlvaraFuncionamento → Nome.	IDVeiculo → Matrícula, Marca, Seguro, KmPercorridos, DataInspeção. Matrícula → Marca, Seguro, DataInspeção
Justificação e Decomposição	Não há necessidade de haver decomposição da classe Pessoa pois ID, CartaoDeCidadao e NIF são superkeys desta classe.	Não há necessidade de haver decomposição da classe Escola pois Nome e AlvaraFuncionamento são superkeys desta classe. Cada escola de condução so pode ter um alvara de funcionamento, logo é uma superkey da classe Escola.	Não há necessidade de haver decomposição da classe Veiculo pois ID e Matrícula são superkeys desta classe.
Integração da AI (ChatGPT-4.0)	A Inteligencia Artificial adicionou CodigoPostal → Morada . Como esta <i>Functional Dependency</i> tem como termo da esquerda um atributo que não é uma <i>superkey</i> , a IA disse que esta violava tanto a <i>Boyce Codd Normal Form</i> quando a <i>Third Normal Form</i> , separando a relação em <i>Pessoa</i> (IDPessoa, Nome, DataDeNascimento, NIF, NumeroCartaoCidadao, CodigoPostal) e <i>Endereco</i> (CodigoPostal, Morada).	A Inteligencia Artificial adicionou CodigoPostal → Morada e alterou AlvaraFuncionamento → Nome , para AlvaraDeFuncionamento → Nome , Morada , CodigoPostal . Como a primeira <i>Functional Dependency</i> tem como termo da esquerda um atributo que não é uma <i>superkey</i> , a IA disse que esta violava tanto a <i>Boyce Codd Normal Form</i> quando a <i>Third Normal Form</i> , separando a relação em <i>Escola</i> (IDEscola, Nome, CodigoPostal, AlvaraDeFuncionamento) e <i>Endereco</i> (CodigoPostal, Morada).	A Inteligencia Artificial adicionou Seguro → Matrícula . Como esta <i>Functional Dependency</i> tem como termo da esquerda um atributo que não é uma <i>superkey</i> a IA disse que esta violava tanto a <i>Boyce Codd Normal Form</i> quando a <i>Third Normal Form</i> , separando a relação em <i>Veiculo</i> (IDVeiculo, Matrícula, Marca, NumeroKmPercorridos) e <i>Seguro</i> (Seguro, Matrícula).
Final	Apesar das sugestões da IA, nós chegamos a conclusão de que, como o código postal não permite saber todas as informações sobre uma determinada morada, nem o inveros, não há violação da BCNF e da 3NF. Logo a versão final é igual a inicial.	Apesar das sugestões da IA, nós chegamos a conclusão de que, como o código postal não diz todas as informações sobre uma determinada morada, ne o invernos, não há violação da BCNF e da 3NF. Porém levamos em consideração as alterações feitas a AlvaraFuncionamento → Nome e trocamos esta para AlvaraDeFuncionamento → Nome , Morada , CodigoPostal	Apesar das sugestões da IA, ao analisar a <i>Functional Dependency</i> que esta adicionou chegamos a conclusão que um seguro pode ter mais de um veiculo e por isso este não é capaz de dizer com certeza uma determinada matrícula. Logo a versão inicial é igual a final

Relação	CartaDeConducao	Reprovacao
Chaves	IDCarta (Chave primaria), Numero	IDReprovacao (Chave primaria)
Functional Dependencies	IDCarta → Numero, DataDeEmissao	IDReprovacao → Duracao, Motivo
Justificação e Decomposição	Só há uma Funcional Dependency com o elemento da esquerda sendo uma superkey. Logo, não há necessidade de haver decomposição da classe	Só há uma Funcional Dependency com o elemento da esquerda sendo uma superkey. Logo, não há necessidade de haver decomposição da classe
Integração da AI (ChatGPT-4.0)	A Inteligencia Artificial não detetou nenhuma violação da BCNF ou da 3NF.	A Inteligencia Artificial não detetou nenhuma violação da BCNF ou da 3NF.
Final	Como não há violação da BCNF ou da 3NF, a versão final é igual a inicial.	Como não há violação da BCNF ou da 3NF, a versão final é igual a inicial.

Análise de Dependências Funcionais e Formas Normais - Utilização de IA

De modo, a conhecermos a avaliação da IA relativamente à existência de dependências funcionais e violações da Terceira Forma Normal ou da Forma Normal de Boyce-Codd, fornecemos à IA, o esquema relacional, na sua integridade, junto do seguinte prompt: “Considerando a relação acima, analisa se existem dependências funcionais e violações da BCNF ou da 3NF e, se existirem, resolve-as”.

As repostas obtidas e as alterações realizadas encontram-se registadas nas tabelas acima.

Utilização de IA - Criação da Base de Dados em SQLite

IA Utilizada: ChatGPT 4.0

Primeiramente, questionámos acerca da existência de algum problema no ficheiro create1.sql que poderia comprometer a sua executabilidade e de avaliar a qualidade do mesmo, através do seguinte prompt: *“O seguinte script, em SQLite, é executável”*, seguido do conteúdo integral do ficheiro, ao que respondeu:

Sim, este script deve ser executado sem problemas em um banco de dados SQLite, desde que:

1. O PRAGMA `foreign_keys` esteja ativado antes da execução do script (como está no seu caso).
2. Não existam dependências cíclicas entre as tabelas que possam causar conflitos na criação das tabelas ou na deleção de dados.
3. As tabelas mencionadas como `FOREIGN KEY REFERENCES` existam antes de quaisquer operações que dependam delas.

Observações sobre o script:

1. NULL em chaves estrangeiras:

- Algumas `FOREIGN KEY` permitem que o valor seja `NULL`, como em `IDEscola` e `IDInstructor`. Isso é funcional, mas se o valor de uma chave estrangeira precisar ser obrigatório, você deve marcar o campo como `NOT NULL`.

2. CHECK com GLOB:

- O uso de `GLOB` para validação de formatos é válido, mas tenha em mente que as regras de `GLOB` são diferentes de `LIKE` ou `REGEX` e podem ser menos flexíveis.

3. Restrição de exclusão (`ON DELETE SET NULL`):

- Você escolheu definir os campos dependentes como `NULL` ao excluir registros pai. Isso faz sentido em muitos casos, mas lembre-se de que isso deixa registros órfãos se o valor for opcional.

4. Validação de datas:

- O uso de condições como `DATE('now', '-1 years')` depende da configuração do SQLite. Certifique-se de que o formato de data utilizado seja consistente ao inserir dados.

Se tudo isso está alinhado com seus requisitos, o script está bem estruturado e deve funcionar conforme esperado. Caso encontre algum erro, compartilhe a mensagem para que possamos ajustar o script.

De maneira a compararmos uma versão de create1.sql criada por IA com a desenvolvida por nós, fornecemos o esquema relacional da base de dados, seguido do seguinte prompt: *“Imagina que com base no esquema relacional que te envie, terias que criar uma implementação da base de dados correspondente em SQLite, chamada “create1.sql”. Mostra como o fizias e o resultado final.”*, sendo enviado o seguinte código:

```
-- Criação do banco de dados
CREATE DATABASE EscolaDB;
USE EscolaDB;

-- Tabela Escola
CREATE TABLE Escola (
    IDEscola INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    Morada VARCHAR(255) NOT NULL,
    CodigoPostal VARCHAR(10) NOT NULL,
    AlvaraDeFuncionamento VARCHAR(50) NOT NULL
);

-- Tabela Categoria
CREATE TABLE Categoria (
    IDCategoria INT PRIMARY KEY,
    Designacao VARCHAR(100) NOT NULL
);

-- Tabela CartaDeConducao
CREATE TABLE CartaDeConducao (
    IDCarta INT PRIMARY KEY,
    Numero VARCHAR(20) NOT NULL,
    DataDeEmissao DATE NOT NULL
);

-- Tabela Instrutor
CREATE TABLE Instrutor (
    IDInstrutor INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    DataDeNascimento DATE NOT NULL,
    NIF VARCHAR(15) NOT NULL,
    NumeroCartaoCidadao VARCHAR(20) NOT NULL,
    CodigoPostal VARCHAR(10) NOT NULL,
    Morada VARCHAR(255) NOT NULL,
    LicencaDeInstrucao VARCHAR(50) NOT NULL,
    IDEscola INT,
    IDCarta INT,
    FOREIGN KEY (IDEscola) REFERENCES Escola(IDEscola),
```

```

        FOREIGN KEY (IDCarta) REFERENCES CartaDeConducao(IDCarta)
    );
-- Tabela Proprietario
CREATE TABLE Proprietario (
    IDInstrutor INT,
    IDEscola INT,
    PRIMARY KEY (IDInstrutor, IDEscola),
    FOREIGN KEY (IDInstrutor) REFERENCES Instrutor(IDInstrutor),
    FOREIGN KEY (IDEscola) REFERENCES Escola(IDEscola)
);
-- Tabela Veiculo
CREATE TABLE Veiculo (
    IDVeiculo INT PRIMARY KEY,
    Matricula VARCHAR(20) NOT NULL,
    Marca VARCHAR(100) NOT NULL,
    NumeroKmPercorridos INT NOT NULL,
    DataDeInspecao DATE NOT NULL,
    Seguro VARCHAR(50) NOT NULL,
    IDEscola INT,
    IDInstrutor INT,
    IDCategoria INT,
    FOREIGN KEY (IDEscola) REFERENCES Escola(IDEscola),
    FOREIGN KEY (IDInstrutor) REFERENCES Instrutor(IDInstrutor),
    FOREIGN KEY (IDCategoria) REFERENCES Categoria(IDCategoria)
);
-- Tabela CategoriaCarta
CREATE TABLE CategoriaCarta (
    IDCarta INT,
    IDCategoria INT,
    PRIMARY KEY (IDCarta, IDCategoria),
    FOREIGN KEY (IDCarta) REFERENCES CartaDeConducao(IDCarta),
    FOREIGN KEY (IDCategoria) REFERENCES Categoria(IDCategoria)
);
-- Tabela
CREATE TABLE Percurso (
    IDPercurso INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    Perimetro INT NOT NULL,
    PontoDeTroca VARCHAR(255) NOT NULL
);
-- Tabela Examinador
CREATE TABLE Examinador (
    IDExaminador INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,

```

```

        DataDeNascimento DATE NOT NULL,
        NIF VARCHAR(15) NOT NULL,
        NumeroCartaoCidadao VARCHAR(20) NOT NULL,
        CodigoPostal VARCHAR(10) NOT NULL,
        Morada VARCHAR(255) NOT NULL,
        CredencialDeExaminador VARCHAR(50) NOT NULL,
        IDCarta INT,
        FOREIGN KEY (IDCarta) REFERENCES CartaDeConducao(IDCarta)
    );
-- Tabela Examinando
CREATE TABLE Examinando (
    IDExaminando INT PRIMARY KEY,
    Nome VARCHAR(255) NOT NULL,
    DataDeNascimento DATE NOT NULL,
    NIF VARCHAR(15) NOT NULL,
    NumeroCartaoCidadao VARCHAR(20) NOT NULL,
    CodigoPostal VARCHAR(10) NOT NULL,
    Morada VARCHAR(255) NOT NULL,
    LicencaDeAprendizagem VARCHAR(50) NOT NULL,
    IDEscola INT,
    IDInstrutor INT,
    FOREIGN KEY (IDEscola) REFERENCES Escola(IDEscola),
    FOREIGN KEY (IDInstrutor) REFERENCES Instrutor(IDInstrutor)
);
-- Tabela Exame
CREATE TABLE Exame (
    IDExame INT PRIMARY KEY,
    Data DATE NOT NULL,
    Hora TIME NOT NULL,
    IDExaminando INT,
    IDExaminador INT,
    IDPercurso INT,
    FOREIGN KEY (IDExaminando) REFERENCES Examinando(IDExaminando),
    FOREIGN KEY (IDExaminador) REFERENCES Examinador(IDExaminador),
    FOREIGN KEY (IDPercurso) REFERENCES Percurso(IDPercurso)
);
-- Tabela Aprovacao
CREATE TABLE Aprovacao (
    IDAprovacao INT PRIMARY KEY,
    Duracao INT NOT NULL,
    Avaliacao VARCHAR(50) NOT NULL,
    OrdemAlfabetica VARCHAR(255) NOT NULL,
    IDExaminando INT,
    IDExame INT,

```

```

        IDCarta INT,
        FOREIGN KEY (IDExaminando) REFERENCES Examinando(IDExaminando),
        FOREIGN KEY (IDExame) REFERENCES Exame(IDExame),
        FOREIGN KEY (IDCarta) REFERENCES CartaDeConducao(IDCarta)
    );
-- Tabela Reprovacao
CREATE TABLE Reprovacao (
    IDReprovacao INT PRIMARY KEY,
    Duracao INT NOT NULL,
    Motivo VARCHAR(255) NOT NULL,
    OrdemAlfabetica VARCHAR(255) NOT NULL,
    IDExaminando INT,
    IDExame INT,
    FOREIGN KEY (IDExaminando) REFERENCES Examinando(IDExaminando),
    FOREIGN KEY (IDExame) REFERENCES Exame(IDExame)
);

```

Explicação do Script:

1. Criação das Tabelas:

- Cada tabela é criada com os campos definidos no esquema relacional. As chaves primárias são especificadas com **PRIMARY KEY**.
- As colunas de relacionamento com outras tabelas, como **IDEscola**, **IDCarta**, **IDInstructor**, etc., são definidas como chaves estrangeiras (**FOREIGN KEY**).

2. Relacionamentos:

- As relações entre as tabelas são feitas através de chaves estrangeiras, que referenciam as chaves primárias de outras tabelas. Por exemplo, a tabela **Instructor** tem uma chave estrangeira **IDEscola**, que referencia a chave primária **IDEscola** da tabela **Escola**.

3. Chaves Estrangeiras:

- Em várias tabelas, como **Instructor**, **Veiculo**, **Examinador**, entre outras, são criadas chaves estrangeiras para estabelecer os vínculos entre as diferentes entidades.

Considerações:

- **Índices:** Embora não tenha adicionado índices adicionais, em um banco de dados real, seria interessante adicionar índices para melhorar o desempenho das consultas, especialmente em colunas de busca, como **NIF**, **Matricula**, etc.
- **Validação de Dados:** Dependendo das necessidades, poderias incluir validações como **NOT NULL** ou restrições de valores para certos campos (ex: **DataDeNascimento** deve ser uma data válida).

Este script cria as tabelas de acordo com o esquema relacional fornecido e prepara o banco de dados para ser utilizado em um sistema de gestão escolar.

Observando os dois códigos, concluímos que, fundamentalmente, são semelhantes, reforçando a nossa confiança na correta implementação da base de dados.

Finalmente, com o intuito de saber se existia algum aspeto a melhorar na nossa implementação, forenecemos o seguinte prompt: “Tendo em conta o teu script, o meu e o esquema relacional, sugeres alguma alteração a fazer ao meu script de modo a torná-lo mais robusto?”, tendo sido respondido com a seguinte resposta:

Algumas alterações e sugestões específicas para o teu script:

1. Definir os Tipos de Dados Corretos e Consistentes:

- Campos como `CodigoPostal`, `NIF`, `NumeroCartaoCidadao`: Se eles seguem um padrão fixo (como `NIF` e `NumeroCartaoCidadao` que têm tamanhos específicos), podemos definir tamanhos mais exatos.
- Datas: Garante que os campos de data e hora usem `DATE` ou `DATETIME` de acordo com o necessário.

2. Definir as Ações de `ON DELETE` e `ON UPDATE` nas Chaves Estrangeiras:

- Para garantir integridade referencial, é importante definir como o banco vai tratar a exclusão ou atualização de registos relacionados.

3. Adicionar Índices e Restrições `UNIQUE` em Campos Críticos:

- Índices ajudam nas buscas e são necessários em campos como `NIF`, `Matricula`, `NumeroCartaoCidadao`, entre outros, para garantir eficiência nas consultas.
- Campos que devem ser únicos como `Matricula` e `Numero` devem ter restrições `UNIQUE`.

Das alterações sugeridas, achamos pertinentes a primeira e terceira, tendo alterado o código em conformidade, no ficheiro `create2.sql`, de modo a tornar o código mais conciso e de mais fácil leitura, no caso da primeira e, no caso da terceira, de *“melhorar a integridade, performance e manutenção do banco de dados a longo prazo”* (ChatGPT,2024).

Utilização de IA - Introdução de Dados

IA Utilizada: ChatGPT 4.0

Primeiramente, procurando saber acerca da opinião da IA acerca do nosso ficheiro populate1.sql, fornecemos o seguinte prompt: “O código que se segue corresponde aos inserts relativos à implementação que te mostrei anteriormente. O que achas?”, tendo recebido a seguinte resposta:

Os `INSERTs` apresentados estão bem estruturados e, em princípio, devem ser executados corretamente, desde que:

1. As tabelas tenham sido criadas conforme o modelo que enviaste anteriormente.
2. Não haja restrições violadas, como chaves primárias duplicadas, violação de integridade referencial (FKs) ou tamanhos de campos incompatíveis com os valores inseridos.
3. Os valores estejam no formato compatível com o SGBD, como o formato de datas (`'YYYY-MM-DD'`) e os tamanhos definidos para colunas do tipo `VARCHAR` .

Se quiseres, posso testar estes comandos localmente (ou ajudar-te a testar). Deixa-me saber!

De seguida, com o intuito de saber de alguma alteração que poderíamos fazer de modo a melhorar a qualidade do ficheiro, fornecemos o seguinte prompt: “Tendo em conta, o meu script create1.sql e o meu populate1.sql, tens alguma sugestão para melhorar o populate1.sql?”.

1. Verificação de Existência de Registos (INSERT Condicional)

- Uma boa prática é garantir que os dados sejam inseridos apenas se não existirem previamente na tabela, para evitar duplicação. Pode ser interessante usar a instrução `INSERT ... SELECT` ou `INSERT IGNORE` para evitar falhas de chave primária ou duplicação.

2. Uso de Transações

- Para garantir que todos os `INSERT` sejam realizados com sucesso e de forma atômica (ou seja, todos ou nenhum dos registos sejam inseridos), você pode envolver seus comandos `INSERT` dentro de uma transação. Caso algum erro ocorra, todas as alterações podem ser revertidas.

3. Inserções em Cascata (Com Relações de Chave Estrangeira)

- Ao inserir registos nas tabelas que têm chaves estrangeiras, a ordem de inserção deve seguir o fluxo correto para evitar erros de chave estrangeira. No seu caso, por exemplo, `Instrutor` depende de `Escola` e `CartaDeConducao`, então a ordem dos `INSERT` deve respeitar essas dependências.

4. Uso de Funções para Gerar Dados Dinâmicos

- Caso você precise gerar valores dinâmicos (como datas ou valores aleatórios), pode usar funções do banco de dados para isso. Por exemplo, para gerar uma data de nascimento aleatória ou um número de matrícula.

5. Verificação de Tipos de Dados

- Verifique se os dados inseridos no `populate1.sql` estão de acordo com os tipos de dados das colunas das tabelas. Se houver incompatibilidade, pode ocorrer erro de inserção. No seu caso, como exemplo, o campo `DataDeNascimento` deve ter o formato `YYYY-MM-DD` e o campo `CodigoPostal` parece ter o formato `XXXXX-XXX`.

6. Inserção em Massa (Batch Inserts)

- Ao inserir um grande número de registros, é mais eficiente agrupar múltiplos `INSERT` em uma única consulta, o que reduz o número de transações. Isso melhora a performance de execução.

7. Uso de Valores Padrão

- Para evitar a necessidade de especificar valores para todas as colunas durante a inserção, pode ser útil definir valores padrão nas colunas que permitem nulos ou têm um valor fixo. Isso ajuda a simplificar os `INSERT`.

8. Manter o Controle de Chaves Primárias e Estrangeiras

- Se as chaves primárias são geradas automaticamente (`AUTO_INCREMENT`), certifique-se de não incluí-las no `INSERT`, ou utilize o `DEFAULT` ou `NULL` para que o banco de dados gere as chaves.

Das sugestões apresentadas a única que considerámos pertinente para aplicar, foi a sexta, tendo sido procedida à sua alteração em conformidade no `populate2.sql`, de modo a melhorar não só a performance da execução, como refere a IA, mas também a simplicidade e a fácil leitura do código.

Contributos dos membros do grupo

Ao contrário do que ocorreu no âmbito da primeira submissão do projeto, devido ao facto desta segunda submissão consistir, fundamentalmente, em três tarefas, decidiu-se, entre os três elementos do grupo, que cada um ficava encarregue do desenvolvimento de uma delas.

Elton Vaz - Conversão do Modelo Conceptual em Esquema Relacional e respetiva Integração de IA;

Henrique Vilarinho - Identificação de Dependências Funcionais e violações das Formas Normais e respetiva Integração de IA;

Tomás Moraes - Implementação da Base de Dados em SQLite, Introdução de Dados e respetiva Integração de IA.

Porém, esta etapa do projeto foi, maioritariamente desenvolvida, com os elementos do grupo a trabalhar de forma conjunta, ou seja, cada um a executar a tarefa que lhe foi atribuída, porém dando apoio a outro colega sempre quando necessário.

Deste modo, todos os membros contribuíram, de forma igual e cooperativa, para o desenvolvimento desta segunda etapa do projeto, tendo sido também evidente que os três estávamos igualmente interessados e empenhados de modo a atingir o melhor resultado possível.