**Projeto de Bases de Dados (CC2005)**

1. Elementos do grupo

**Grupo nº** **G2A**

|  |  |
| --- | --- |
| **Nº mecanográfico** | **Nome** |
| 202300339 | Isabela Britto Cartaxo |
| 202300276 | Guilherme Ferreira Klippel |
| 202302069 | Bruno Souza Zabot |

2. Universo considerado e modelo de classes UML

**Tema:** Contratos públicos 2024

2.1 Univerdo da BD

O universo considerado para a base de dados abrange **Contratos Públicos celebrados em Portugal**, com foco nos contratos cuja **data de celebração** ocorre entre **janeiro e fevereiro de 2024**. Os dados incluem contratos publicados no intervalo entre **dezembro de 2023 e fevereiro de 2024**, refletindo tanto o período de divulgação quanto de efetivação dos contratos em plataformas governamentais.

A base de dados documenta informações detalhadas de cada contrato, como o tipo e o objeto do contrato, as entidades envolvidas (adjudicantes e adjudicatárias), as datas de publicação e celebração, os valores contratados e as justificativas legais ou técnicas que sustentam os processos.

O âmbito geográfico cobre predominantemente o território português, mas também contempla contratos com **locais de execução fora de Portugal**, incluindo múltiplos locais de execução quando aplicável. Exemplos de países presentes nos registros incluem **Espanha, Alemanha, Arábia Saudita, Quénia, Estados Unidos, Cabo Verde**, dentre outros. Para garantir precisão e eliminar redundâncias, as localizações de execução são discriminadas em níveis específicos, como município, distrito e país.

Além disso, o conjunto de dados considera particularidades como contratos vinculados a acordos-quadro e procedimentos centralizados, permitindo uma análise abrangente e detalhada das práticas de contratação pública no período analisado.

2.2 UML

Após a análise do dataset, foi decidido modelar classes separadas para **TipoContrato** e **TipoProcedimento**, considerando que essas categorias aparecem em múltiplos contratos e, portanto, podem ser reutilizadas. Para a classe de **Fundamentação**, optou-se por criar uma estrutura separada devido à presença de múltiplos componentes. Essa decisão garante que essa coluna atenda à Primeira Forma Normal (1FN). Assim, a **Fundamentação** foi decomposta em atributos como **artigo**, **número**, **alínea**, **subalínea** e **referênciaLegislativa**.

De forma similar, o atributo **CPV** foi desmembrado em **CódigoCPV** e **DescriçãoCPV**, enquanto as entidades foram divididas em **NIF** e **Entidade** para manter a integridade dos dados e evitar redundâncias. No caso do **Local de Execução**, decidiu-se modelar essa informação como três classes distintas: **Município**, **Distrito** e **País**. O município está conectado ao contrato, pois é a classe mais específica. Assim, é possível determinar o distrito e o país a partir do município, mas não o contrário.Essa abordagem foi escolhida em vez de tratá-las como atributos, pois permite um melhor relacionamento com outras entidades, promovendo escalabilidade e reutilização, além de facilitar a adição de metadados específicos para cada nível geográfico.

Para o atributo **Descrição Acordo Quadro**, inicialmente foi considerada a possibilidade de separar os elementos em diferentes atributos, com base nos padrões identificados nos dados. Além do texto descritivo, muitos registros apresentam informações adicionais, como a sigla do instrumento (**AQ** - Acordo Quadro, **PRC** - Pedido de Registro de Compras, **CP** - Contrato de Prestação, etc), um número sequencial (indicando a ordem de emissão, como **306**) e o ano de emissão (**2023**). Alguns exemplos desses formatos incluem:

* **AQ 306/2023**
* **CP 2021-25**
* **0029/FISC/2023**
* **PRC\_0006/2023\_RHU**
* **306/2023**
* **2022/56**

No entanto, os formatos dos registros são amplamente variados, como pode ser observado nos exemplos acima. Em alguns casos, é fornecida apenas uma descrição textual, ou surgem informações que não se enquadram claramente em uma categoria definida, ou até mesmo elementos cuja classificação é incerta. Exemplos desses casos incluem:

* **Acordo Quadro para o fornecimento de combustíveis rodoviários**
* **Acordo Quadro 04/2021**
* **ACORDO-QUADRO – DML**
* **PRC\_0507/2021\_GAE - O0760**
* **PRC\_0306/2021\_GAE - 'EB0722**

Essas inconsistências dificultam a criação de uma estrutura uniforme para representar o atributo, especialmente considerando a necessidade de capturar todas as variações possíveis de forma precisa.

Mesmo que os atributos fossem unificados em um único campo, surgiria outro desafio: muitas descrições incluem informações adicionais separadas por diferentes delimitadores, como hífens ("-"), dois-pontos (":"), ou até mesmo o uso de hífens em posições que dificultam a separação correta das informações. Exemplos ilustram essa complexidade:

* **CP 2021/60 - Acordo quadro para fornecimento de radiofármacos, às instituições e serviços do SNS**
* **Acordo Quadro a celebrar com uma única Entidade para a Aquisição de Material de Automação para a Águas do Douro e Paiva, S.A. e SIMDOURO - Saneamento do Grande Porto, S.A.**
  + Nesse caso, a separação com base no hífen dividiria a descrição no local errado, comprometendo a integridade dos dados.
* **CP 2021-25: ACORDO QUADRO PARA FORNECIMENTO DE MATERIAL DE INCONTINÊNCIA, PROTEÇÃO CUTÂNEA E ALÍVIO DE PRESSÃO, PARA AS INSTITUIÇÕES E SERVIÇOS DO SNS**
* **PRC\_0306/2021\_GAE - 'EB0722 - Empreitada geral de substituições / renovações de infraestruturas do Sistema de Águas da Região do Noroeste**
  + Nesse caso, a separação dividiria esse “identificador” no meio, dificultando a extração correta dos dados.

Outro ponto crítico é que, em algumas situações, as informações complementares aparecem no final da descrição, como em:

* **Acordo Quadro para Fornecimento de Gás Natural - AQ/67/2023**

Essas variações estruturais tornam o processo de padronização especialmente desafiador e evidenciam a dificuldade de aplicar uma separação consistente com base nos delimitadores presentes nos dados.

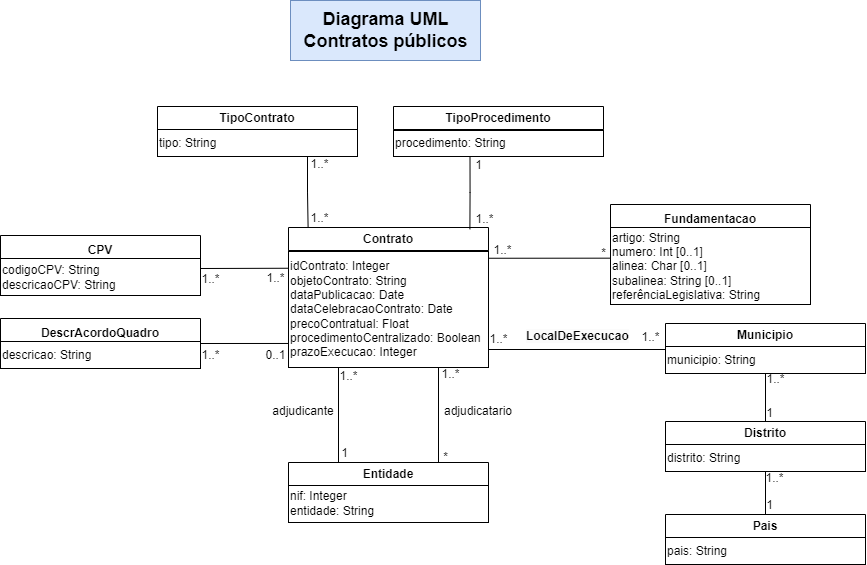
Devido à grande diversidade de formatos, separadores e estruturas, métodos automáticos como o split() (por exemplo, em Python) para separar elementos se mostrariam pouco confiáveis, especialmente em situações como:

1. Descrições longas e complexas, onde a separação no hífen pode ocorrer em locais indevidos.
2. Alternância de separadores, como o uso de ":" em vez de "-".
3. Casos onde a descrição principal aparece antes ou depois dos dados formais.

Para os demais atributos, decidimos mantê-los dentro da classe **Contrato**, pois não se repetem o suficiente para justificar a criação de uma classe separada ou não há necessidade dessa separação. Por exemplo, no caso da **data de celebração**, apesar de existirem datas que se repetem em vários contratos, não faz sentido movê-la para uma classe separada, pois isso aumentaria a complexidade do modelo sem trazer benefícios claros. Ao manter a data dentro da classe **Contrato**, evitamos a necessidade de associar um novo objeto ou classe para cada ocorrência de uma data, o que tornaria o modelo mais complexo e dificultaria o gerenciamento e a leitura dos dados, além de potencialmente resultar em maior custo de processamento e manutenção.

As escolhas de modelação foram fundamentadas nos princípios de normalização de banco de dados, buscando atender à 1FN e minimizar redundâncias. Sempre que possível, valores reutilizáveis e categorias foram abstraídos para classes separadas, enquanto atributos complexos, mas estáveis, foram mantidos como texto para reduzir a complexidade operacional.

Segue abaixo o diagrama de classes UML, ilustrando as relações entre as classes e destacando os pontos abordados.



3. Modelo relacional

Para as classes que poderiam ter mais de uma associação com o **Contrato**, optamos por criar novas tabelas para representar essas associações. Exemplos disso são as tabelas **CPVsContratos**, **ClassificacoesContratos**, **Fundamentacoes**, **LocaisDeExecucao** e **Adjudicatarios**. Nesses casos, associamos o **id** do **Contrato** ao **id** correspondente da tabela relacionada.

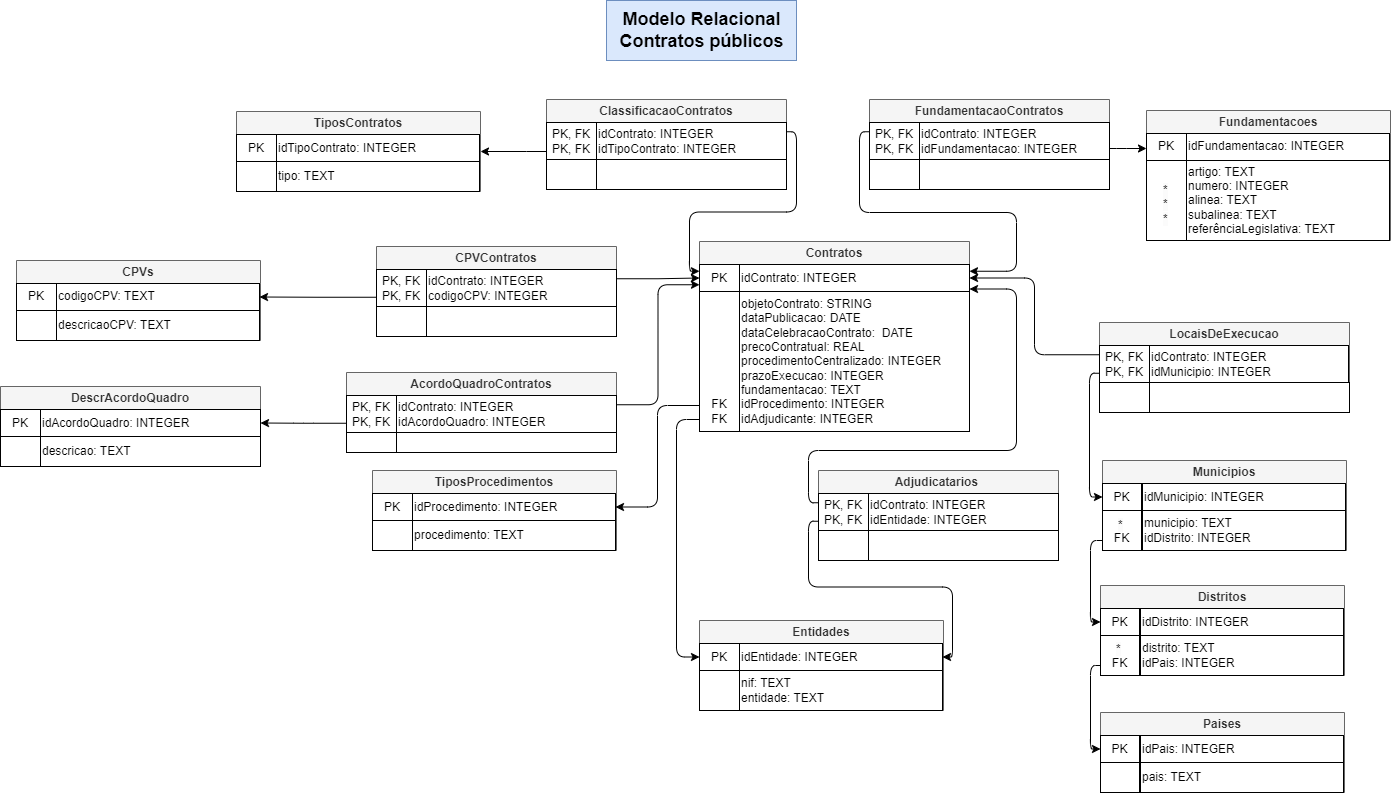
Nos casos em que existe apenas uma associação por contrato, como entre as tabelas **Contratos** e **TiposProcedimentos**, ou entre **Contratos** e **Adjudicantes**, optamos por adicionar uma chave estrangeira (FK) diretamente na tabela **Contratos**, pois não há necessidade de criar uma tabela separada. Isso ocorre porque não há múltiplos atributos por contrato para serem associados, e nem muitos valores nulos que justifiquem essa separação.

Em relação à **descrição do acordo quadro**, a situação é um pouco diferente. Como apenas 30% dos contratos possuem essa descrição, decidimos manter essa informação fora da tabela **Contratos** para evitar muitos valores **NULL**. Assim, mesmo havendo apenas uma descrição de acordo quadro por contrato, essa abordagem evita que a tabela **Contratos** contenha muitas entradas nulas.

No caso dos **Locais de Execução**, os dados podem ser fornecidos de três formas: **país, distrito e município**, **país e distrito**, ou apenas **país**. Por isso, optamos por permitir que, quando o **distrito** ou **município** não forem informados, esses valores sejam atribuídos como **NULL**. Por exemplo, se o país for **Espanha** e o distrito e município não forem fornecidos, eles serão **NULL**. Como existe um outro atributo de vinculação, podemos ter mais de um atributo **NULL** nos campos **distrito** e **município**. Para garantir a unicidade, a combinação da tupla **idPais, distrito** ou **idDistrito, município** é usada, o que assegura que não haja repetições, pois o **(idDistrito, NULL)** ou **(idPais, NULL)** será único, mesmo com o valor **NULL**.

Além disso, optamos por diferenciar os nomes dos atributos em cada tabela para que possamos utilizar o natural join de forma mais intuitiva nas queries, melhorando a legibilidade e facilitando a identificação da tabela a que cada atributo pertence.

Por fim, para facilitar a leitura do modelo relacional, é importante observar que cada chave estrangeira (FK) possui uma seta na linha que indica a direção para a chave primária (PK) correspondente da outra tabela.



Com isso, também conseguimos atingir as exigências para nos enquadrarmos no modelo 2FN e 3FN.

A 2FN exige que não existam dependências parciais, ou seja, todos os atributos não chave devem depender completamente da chave primária. No modelo relacional apresentado:

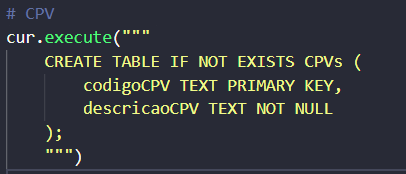
1. Tabelas Associativas: Foram criadas tabelas associativas como CPVsContratos, ClassificacoesContratos, Fundamentacoes, LocaisDeExecucao, e Adjudicatarios para representar as relações muitos-para-muitos entre contratos e outras entidades. Essa abordagem assegura que a tabela principal (por exemplo, Contratos) não armazene dados redundantes que dependam apenas de parte da chave primária, eliminando dependências parciais.
2. Locais de Execução: A modelagem de locais de execução em três tabelas separadas (País, Distrito e Município) impede que atributos como Município dependam apenas de parte da chave (por exemplo, a chave primária em Contratos). Cada nível de localização é representado por identificadores únicos, garantindo que as dependências sejam completas e não parciais.
3. Descrição do Acordo Quadro: O fato de manter a Descrição do Acordo Quadro fora da tabela Contratos, devido à baixa ocorrência, ajuda a evitar muitos valores nulos na tabela Contratos. Essa decisão, por sua vez, mantém a integridade da 2FN ao evitar a dependência de atributos não chave que poderiam ter um impacto em uma chave composta.

A 3FN elimina dependências transitivas, assegurando que atributos não chave dependam diretamente apenas da chave primária:

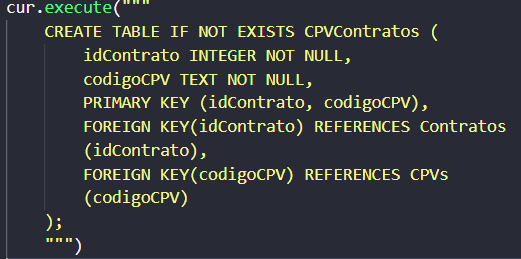
1. Descrição e Código CPV: A separação entre o Código CPV e a Descrição CPV garante que não haja dependências transitivas dentro da tabela CPVs. Ambos os atributos dependem diretamente da chave primária, evitando dependências entre atributos não chave.
2. Locais de Execução: A modelagem hierárquica entre País, Distrito, e Município evita dependências transitivas. A tabela Município está conectada a Distrito e País por chaves estrangeiras, garantindo que as dependências sejam diretas e que as tabelas não contenham atributos não chave que dependam de outros atributos não chave.
3. Nomes Diferenciados nos Atributos: A escolha de nomes distintos para atributos em diferentes tabelas, como idPais e idContrato, ajuda a evitar ambiguidades e facilita a identificação de qual tabela um atributo pertence. Essa prática promove consultas mais precisas e evita dependências transitivas, pois cada tabela tem seus próprios atributos e chaves que são dependentes apenas da chave primária da tabela.

4. Povoamento de tabelas

A criação das tabelas na base de dados foi realizada utilizando **Python** e a biblioteca **sqlite3**, conforme demonstrado nas aulas. Este processo envolveu a definição da estrutura das tabelas, incluindo as chaves primárias (PK) e as chaves estrangeiras (FK), para garantir a integridade referencial dos dados. Abaixo segue um exemplo de código para a criação de uma tabela:



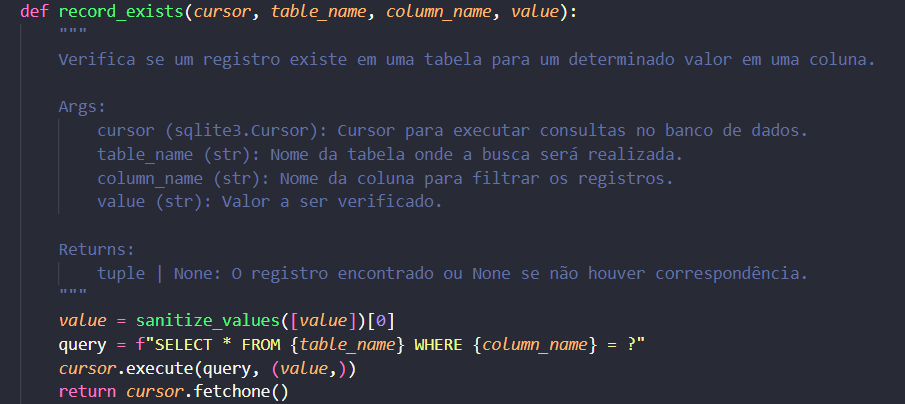
Abaixo segue um exemplo de código para a criação de uma tabela, com vinculação de uma chave estrangeira:



O **povoamento** das tabelas foi feito utilizando **Python**, juntamente com as bibliotecas **sqlite3**, **openpyxl** e **re**. A fonte de dados utilizada para povoar a base de dados foi o arquivo **ContratosPublicos2024.xlsx**, disponibilizado no **Moodle** da disciplina. Para ler e manipular os dados deste arquivo, utilizamos a biblioteca **openpyxl**, que facilita a extração de dados de planilhas do Excel.

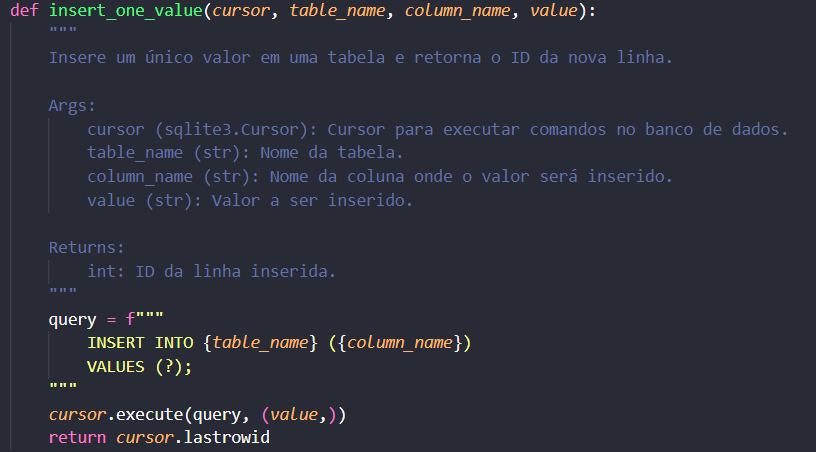
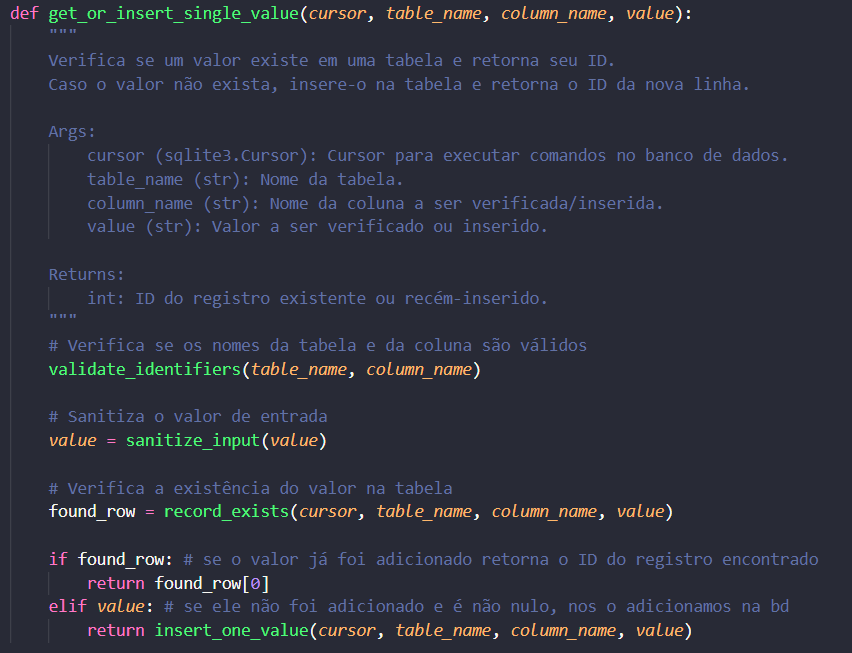
Para garantir a integridade e qualidade dos dados, foram tomadas algumas precauções importantes durante o povoamento:

**1. Limpeza de dados**: Todos os valores extraídos foram limpos, removendo espaços desnecessário, para garantir que apenas dados consistentes fossem inseridos na base de dados.



**2. Verificação de duplicidade:** Antes de inserir qualquer dado, é realizada uma verificação para garantir que não exista registros duplicados. Para isso, foi criado um processo de checagem que verifica a existência de um determinado valor na tabela antes de adicionar o novo valor. Caso o dado já exista na tabela, o código pega o id correspondente a esse valor. Caso contrário, o valor é inserido na tabela e o id gerado é recuperado. Esse processo é essencial para garantir a integridade dos dados e permitir a associação entre as tabelas utilizando o id obtido.

Por exemplo, formos verificar a existência de um tipo de contrato antes de inseri-lo na tabela TiposContratos, utilizamos as seguintes funções:

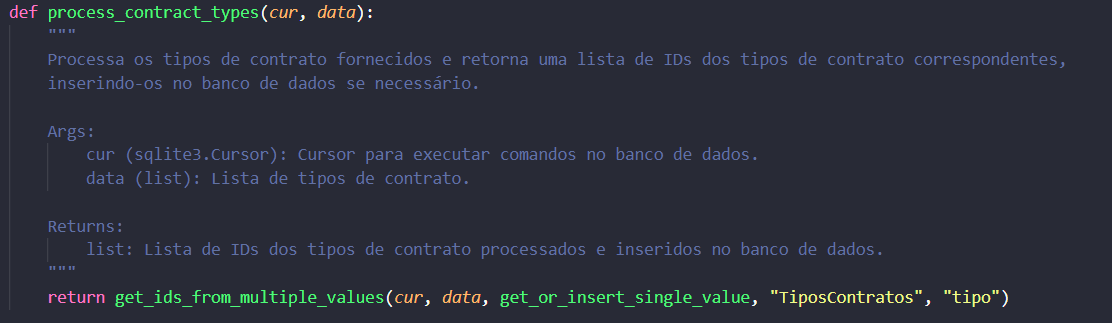


**3. Processamento de dados**: Para facilitar a manutenção e melhorar a legibilidade do código, optamos por criar funções específicas para o processamento de cada coluna das tabelas. Isso permitiu isolar as tarefas de processamento, tornando o código mais modular, organizado e de fácil entendimento. Por exemplo, para processar os tipos de contratos, criamos uma função específica dedicada a essa tarefa. Dentro dessa função, chamamos uma função mais geral que executa operações comuns, como a verificação de duplicidade e a obtenção do **id**.

Além disso, existem casos muito específicos de processamento, como no caso dos **locais de execução**, **entidades** ou **fundamentação**, que exigem um tratamento mais detalhado e customizado. Nestes casos, a criação de funções específicas para cada um desses casos de processamento foi fundamental para garantir a consistência e a clareza do código. A separação das lógicas de processamento em funções distintas ajuda a evitar que o código se torne confuso e difícil de manter, especialmente quando há diferentes formas de tratar dados dependendo das tabelas ou dos atributos envolvidos.

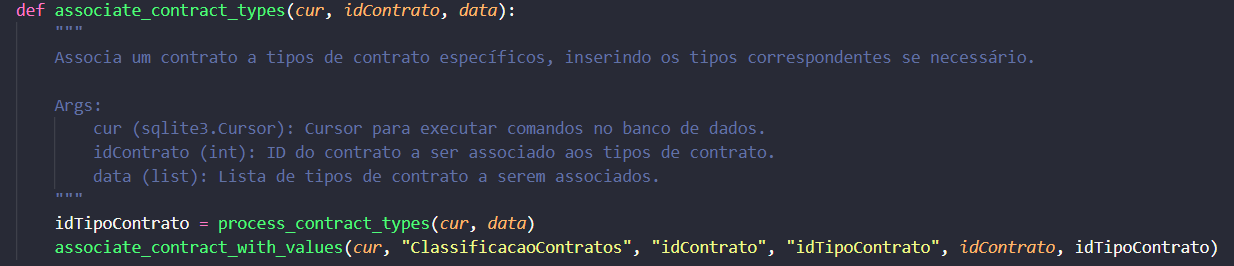
Dessa forma, cada caso de processamento específico, como os locais de execução, entidades ou fundamentação, fica isolado em sua própria função, o que facilita a manutenção e o entendimento do código à medida que o sistema evolui. Essa separação também torna mais fácil identificar e corrigir possíveis erros, já que cada função trata de uma parte específica do processo.

Por exemplo, a função para processar os **tipos de contratos** pode ser estruturada da seguinte forma:



**4. Associação com o contrato**: A associação entre as tabelas foi feita por meio de **chaves estrangeiras (FK)**. No caso de tabelas que precisavam se relacionar diretamente com a tabela **Contratos**. Para isso criamos funções especificas para cada situação, com a mesma justificação do tópico anterior, facilidar a legibilidade e manutenção do código.

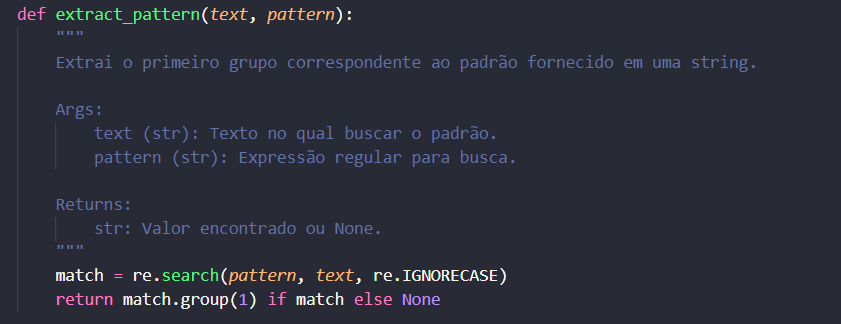
Um exemplo de uma função específica que associa o **id** do contrato com um atributo pode ser a função associate\_contract\_types. Esta função é responsável por associar o **id** de um contrato com o tipo de contrato correspondente. Ela chama uma função específica para processar os tipos de contrato (como a process\_contract\_types mostrada anteriormente) para garantir que o tipo seja adicionado à tabela **TiposContratos** caso ainda não exista, e para recuperar o **id** do tipo de contrato. Em seguida, ela chama uma função genérica para realizar a associação entre o **id** do contrato e o **id** do tipo de contrato na tabela de associação.



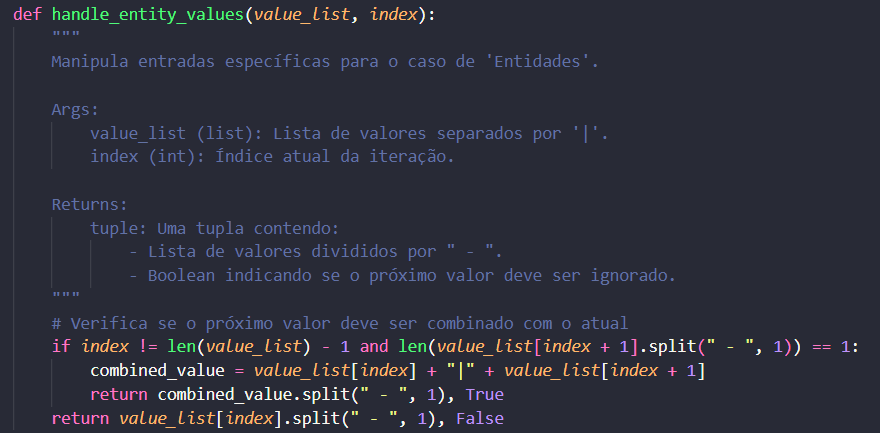
**5. Povoamento de Locais de Execução**: Um dos casos mais complexo de povoamento foi o da tabela **LocaisDeExecucao**, pois os dados de localização eram fornecidos de diferentes maneiras: país, distrito e município, ou apenas país e distrito. Por isso, criamos um processo específico para separar corretamente essas informações e garantir que, quando um campo não fosse fornecido (como o distrito ou **município**), esses valores fossem deixados como **NULL**. Além disso, o relacionamento entre as localidades foi tratado de maneira a permitir mais de um local associado a um contrato.



6. **Fundamentação**: Para tratar os dados da coluna **Fundamentação**, utilizamos a biblioteca **re** (expressões regulares). Essa abordagem foi escolhida para separar e processar informações complexas contidas em um único campo, permitindo extrair e organizar os dados de maneira mais eficiente.



**7. Entidades:** No caso da tabela **entidades**, criamos uma função específica para a separação dos valores, pois, quando há mais de uma entidade, elas são separadas pelo caractere **" | "**. No entanto, algumas designações de entidades contêm esse caractere, o que exigiria um tratamento especial. Por essa razão, foi necessário criar uma função separada para lidar com essa situação, garantindo que a separação das entidades fosse feita corretamente sem interferir nas designações que já utilizam o caractere **" | "**.



**8. Representação do atributo ProcedimentoCentralizado**: A única diferença em relação à representação dos dados fornecidos foi no atributo **ProcedimentoCentralizado**. Embora os valores para este atributo sejam **Sim** ou **Não**, optamos por não foi possivel representa-lo como tipo **bool**, uma vez que o SQLite não oferece suporte nativo a esse tipo de dado [[1]](https://www.sqlite.org/datatype3.html#boolean_datatype). Em vez disso, decidimos representá-lo como um **integer**, utilizando o valor **0** para **Não** e o valor **1** para **Sim**, conforme indicado no diagrama UML e no modelo relacional. Essa abordagem garante que os dados sejam armazenados de forma eficiente, ao mesmo tempo que preserva a clareza na interpretação dos valores, facilitando a consulta e manipulação no banco de dados.

Em resumo, o processo de povoamento da base de dados foi cuidadosamente planejado para garantir que todos os dados fossem inseridos de maneira eficiente, limpa e sem duplicações. As fontes de dados utilizadas foram essencialmente o arquivo ContratosPublicos2024.xlsx, e o processo foi automatizado e otimizado utilizando Python com bibliotecas adequadas para manipulação de arquivos Excel e processamento de dados. O código foi estruturado de forma modular, com funções específicas para cada tipo de dado, facilitando a manutenção e a legibilidade do código.

**Número de linhas por tabela:**

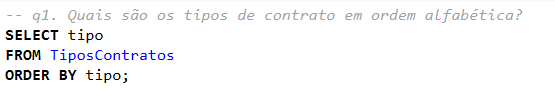
|  |  |
| --- | --- |
| **Nome da tabela** | **Nº de entradas** |
| Contratos | 21748 |
| Entidades | 11375 |
| Adjudicatarios | 22246 |
| CPVs | 2211 |
| CPVContratos | 22067 |
| DescrAcordoQuadro | 256 |
| AcordoQuadroContratos | 3545 |
| TiposContratos | 8 |
| ClassificacaoContratos | 21911 |
| Paises | 18 |
| Distritos | 40 |
| Municipios | 348 |
| LocaisDeExecucao | 23317 |
| TiposProcedimentos | 11 |
| Fundamentacoes |  |
| FundamentacaoContratos |  |

4. Interrogações SQL

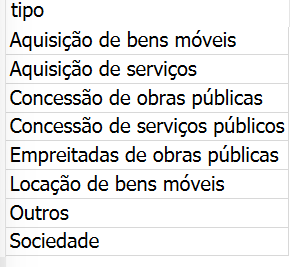
**Perguntas fáceis**

Q1. Quais são os tipos de contrato em ordem alfabética?

Código SQL:

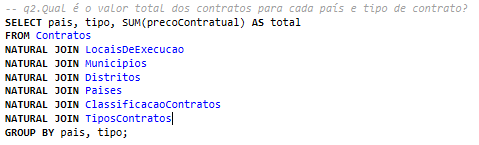


Extrato do resultado:



Q2. Qual é o valor total dos contratos para cada país e tipo de contrato?

Código SQL:

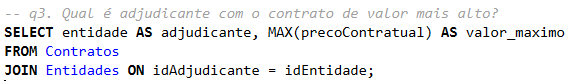


Extrato do resultado:



Q3. Qual é adjudicante com o contrato de valor mais alto?

Código SQL:

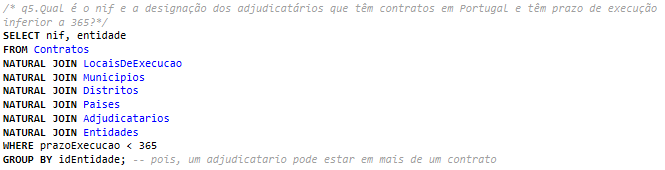


Extrato do resultado:



Q4.Qual é o NIF e a designação dos adjudicatários que têm contratos em Portugal e têm prazo de execução inferior a 365 dias?

Código SQL:

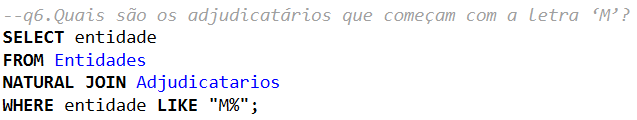


Extrato do resultado:



Q5. Quais são os adjudicatários que começam com a letra ‘M’?

Código SQL:



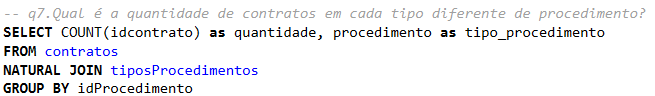
Extrato do resultado:



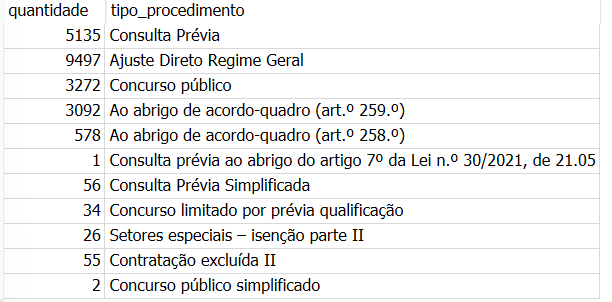
**Perguntas Médias**

Q6. Qual é o número de contratos em cada tipo diferente de procedimento?

Código SQL:

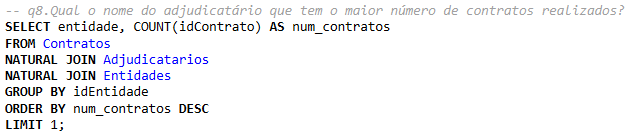


Extrato do resultado:



Q7. Qual o nome do adjudicatário que tem o maior número de contratos realizados?

Código SQL:

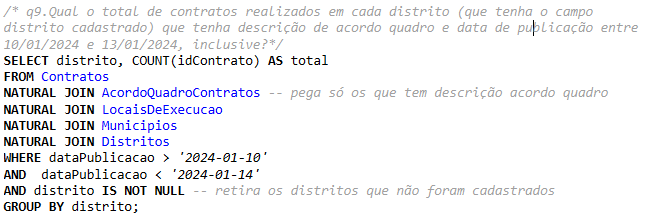


Extrato do resultado:

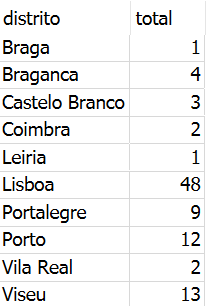


Q8. Qual o total de contratos realizados em cada distrito (que tenha o campo distrito cadastrado) que tenha descrição de acordo quadro e data de publicação entre 10/01/2024 e 13/01/2024, inclusive?

Código SQL:

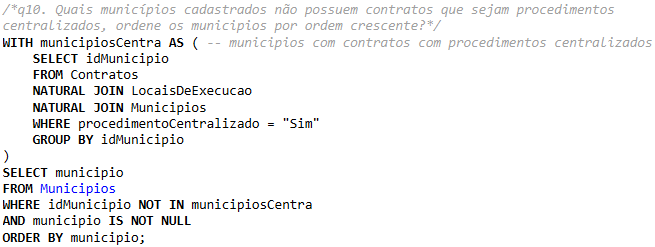


Extrato do resultado:

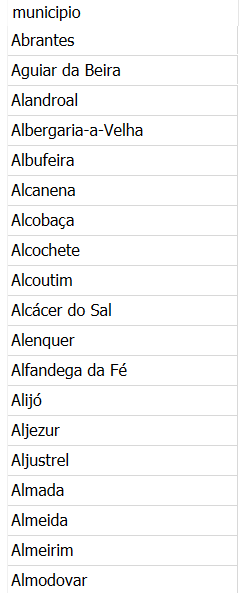


Q9. Quais municípios cadastrados não possuem contratos que sejam procedimentos centralizados, ordene os municípios por ordem crescente?

Código SQL:

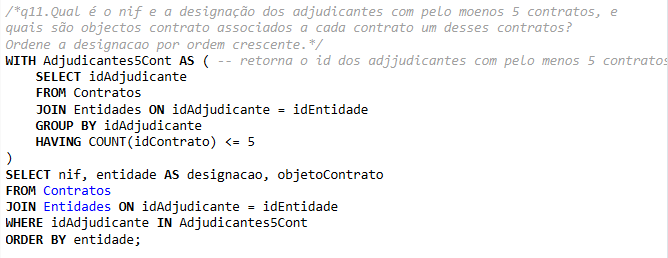


Extrato do resultado:

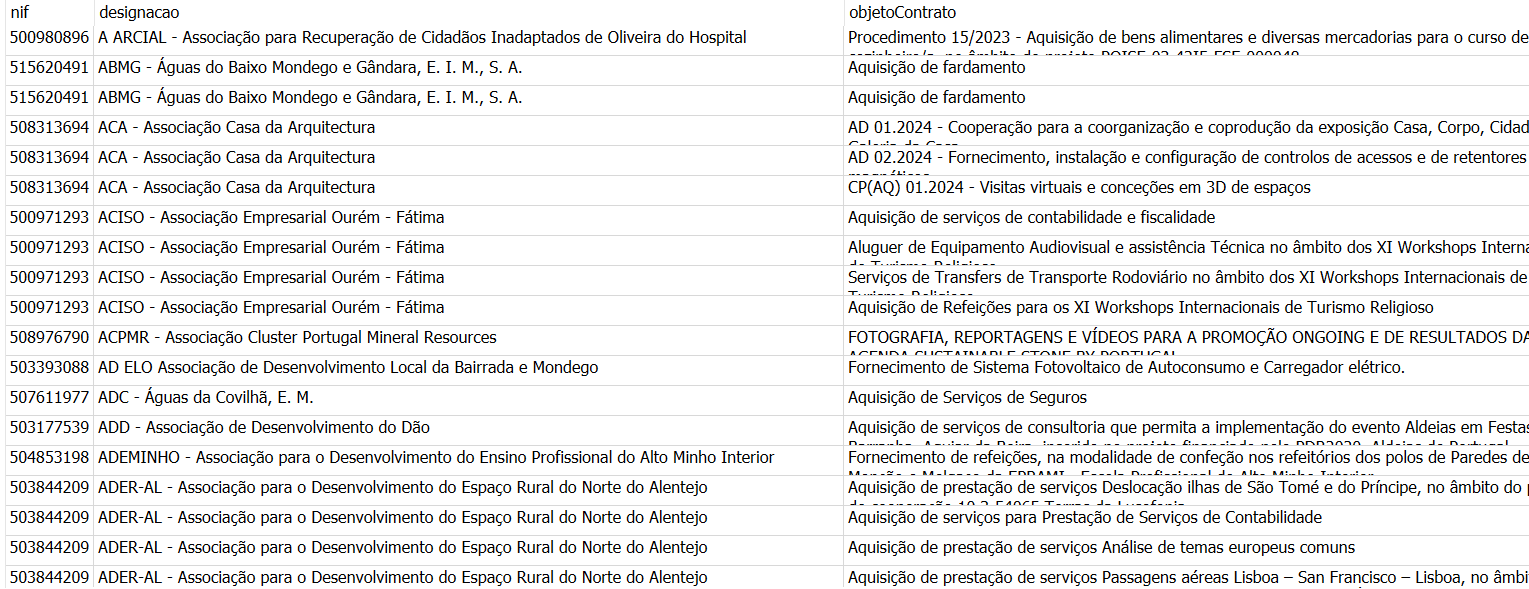


Q10. Qual é o NIF e a designação dos adjudicantes com pelo menos 5 contratos, e quais são objectos contrato associados a cada contrato um desses contratos? Ordene a designação por ordem crescente.

Código SQL:



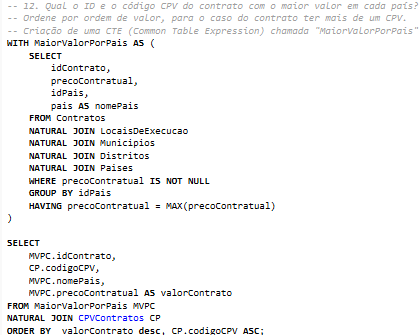
Extrato do resultado:



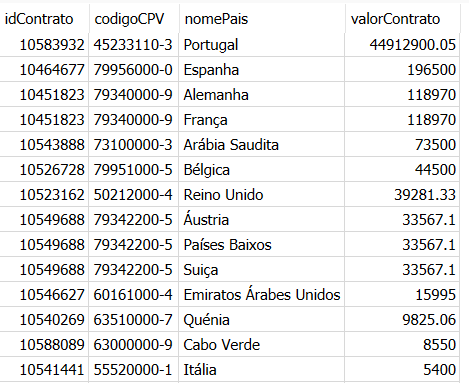
**Perguntas Difíceis**

Q11. Qual o ID e o código CPV do contrato com o maior valor em cada país? Ordene por ordem crescente de id de contrato, para o caso de o contrato ter mais de um CPV.

Código SQL:

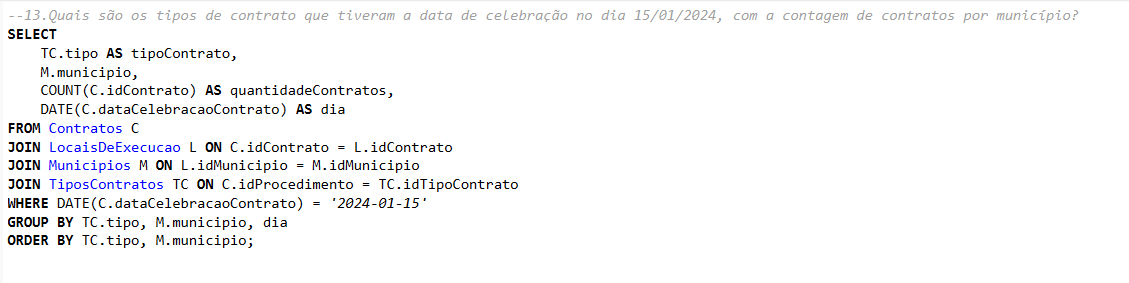


Extrato do resultado:



Q12. Quais são os tipos de contrato que tiveram a data de celebração no dia 15/01/2024, com a contagem de contratos por município?

Código SQL:

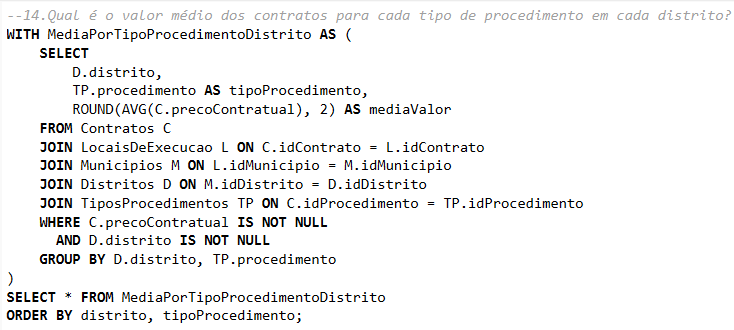


Extrato do resultado:

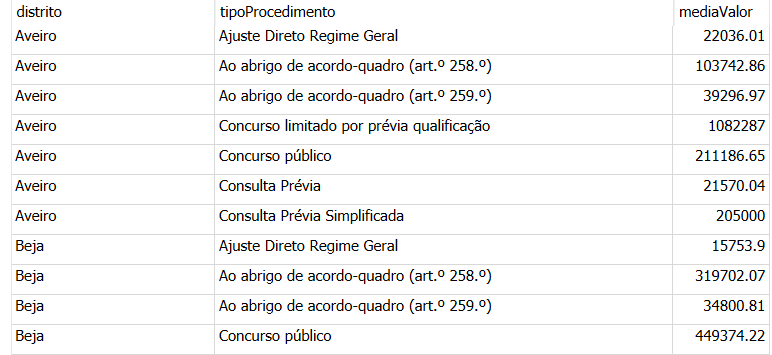


Q.13 Qual é o valor médio dos contratos para cada tipo de procedimento em cada distrito?

Código SQL:



Extrato do resultado:



Q.14 Qual é a fundamentação que mais aparece em cada distrito?

Código SQL:

Texto

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

Extrato da resposta:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

Q.15 Quais municípios não possuem contratos cujo valor contratual seja superior 1.000.000 euros? E para cada um desses municípios mostre o contrato de maior valor.

Código SQL:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Extrato da resposta:

Tabela

Descrição gerada automaticamente

5. Aplicação Python

**Indique os “endpoints” implementados e um sumário da respetiva funcionalidade na tabela abaixo.**

|  |  |
| --- | --- |
| **“Endpoint”** | **Funcionalidade** |
| **/** | Página de entrada |
|  |  |
|  |  |

6.Conclusão

Nesse trabalho, nosso grupo se dedicou a criar um modelo de base de dados focado nos contratos públicos celebrados em 2024. Começamos entendendo o universo dos dados, analisando as informações relevantes e, com base nisso, montamos o modelo UML para organizar as classes e os relacionamentos.

Ao longo do desenvolvimento, fomos ajustando o modelo relacional, sempre buscando seguir as formas normais (1FN, 2FN e 3FN). Fizemos isso para garantir que os dados ficassem organizados, sem redundâncias e fáceis de consultar. Além disso, tomamos cuidado com detalhes, como evitar muitos valores nulos, para que o banco fosse o mais eficiente possível.

Na hora de povoar as tabelas, usamos Python com as bibliotecas **sqlite3**, **openpyxl** e **re**, o que facilitou muito o trabalho com os dados reais. Criamos funções específicas para tratar cada coluna, deixando o código modular e mais fácil de entender. Um dos maiores desafios foi lidar com as descrições variadas dos contratos, mas conseguimos criar uma lógica que tratasse essas diferenças de forma consistente.

Por fim, fizemos várias consultas SQL para explorar os dados, respondendo a perguntas que iam desde coisas simples até questões mais complexas com o intuito de colocar em prática tudo aquilo que aprendemos nas aulas de base de dados e demonstrar que o modelo que construímos está preparado para análises detalhadas.

7. Referências

[[1]](https://www.sqlite.org/datatype3.html#boolean_datatype) SQLite Documentation. Boolean Datatype. Disponível em: <https://www.sqlite.org/datatype3.html#boolean_datatype>. Acesso em: 7 dez. 2024.