

**Bases de Dados**

2024/2025

**Modelação e Implementação de uma Base de Dados sobre Contratos Públicos**

Autores:

Bruno Souza Zabot (up202302069@up.pt)

Guilherme Ferreira Klippel (up202300276@up.pt)

Isabela Britto Cartaxo (up202300339@up.pt)

**Índice**

[1. Universo da Base de Dados 1](#_Toc184674801)

[2. UML 2](#_Toc184674802)

[3. Modelo Relacional 5](#_Toc184674803)

[4. Povoamento de tabelas 7](#_Toc184674804)

[5. Interrogações SQL 13](#_Toc184674805)

[6. Aplicação Python 22](#_Toc184674806)

[7. Conclusão 25](#_Toc184674807)

[8. Referências 26](#_Toc184674808)

# Universo da Base de Dados

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.

Abaixo, encontram-se listadas todas as colunas do dataset original:

*Idcontrato, tipoContrato, tipoprocedimento, objectoContrato, adjudicante, adjudicatarios, dataPublicacao, dataCelebracaoContrato, precoContratual, cpv, prazoExecucao, localExecucao, fundamentacao, ProcedimentoCentralizado, DescrAcordoQuadro*

# 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. Por conta do Regulamento Geral sobre a Proteção de Dados (RGPD) algumas entidades não tem o NIF exposto e o valor é assinalado como RGPD.

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 registos 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 registos 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.

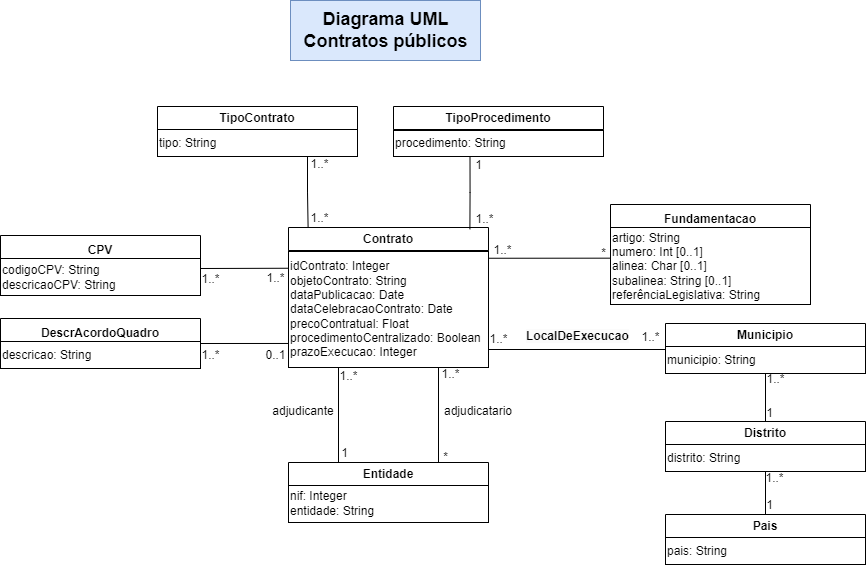
Devido à grande diversidade de formatos, separadores e estruturas, métodos automáticos como o split() (por exemplo, em Python) para separar elementos se mostrariam pouco confiáveis, especialmente em situações como:

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 abstraídas 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.



# 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.

Na descrição do acordo quadro, 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.

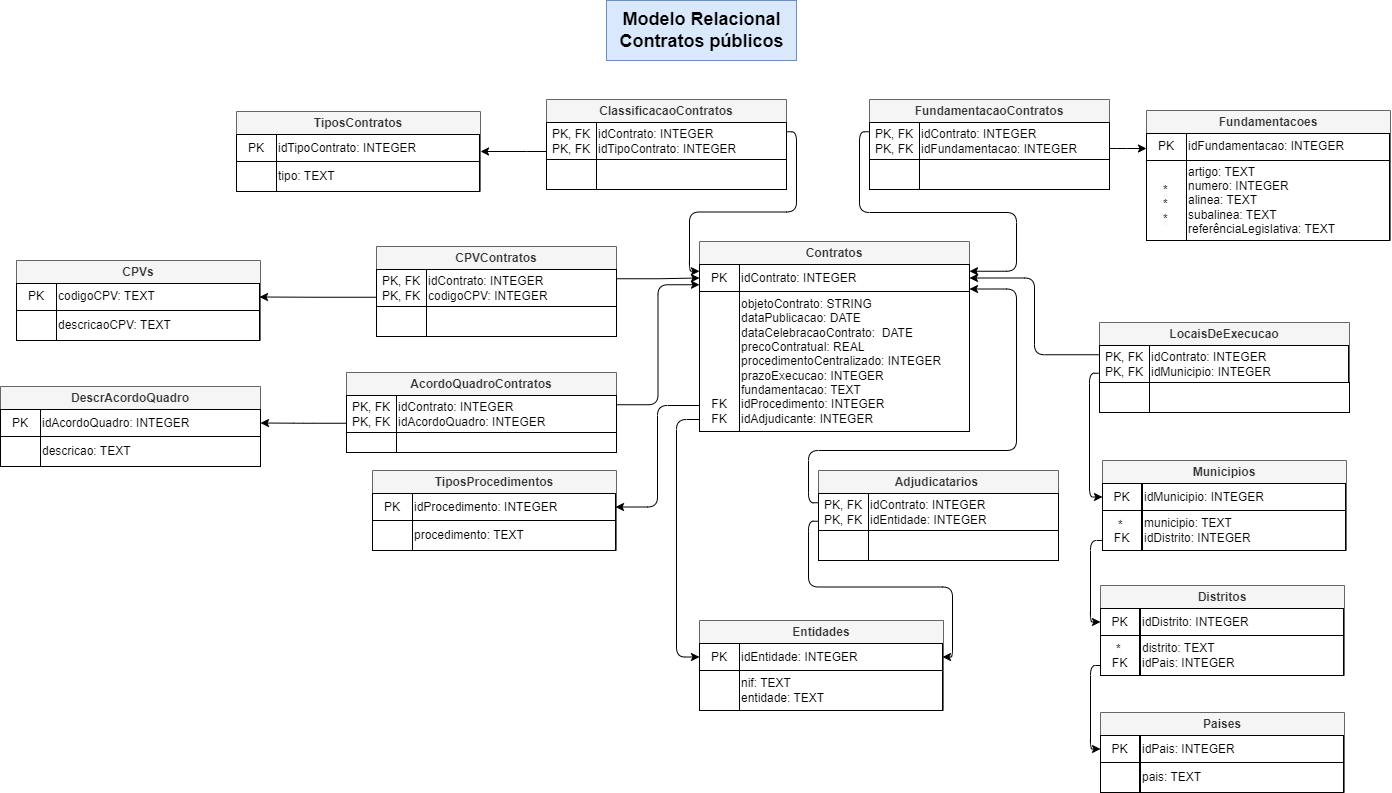
Com isso, também conseguimos atingir as exigências para nos enquadrarmos no modelo 2FN e 3FN.

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:

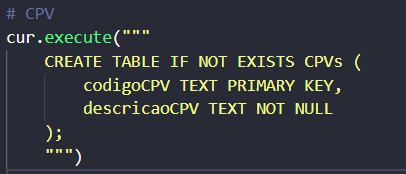
1. Tabelas Associativas: Foram criadas tabelas associativas como CPVsContratos, ClassificacoesContratos, Fundamentacoes, LocaisDeExecucao, e Adjudicatarios para representar as relações muitos-para-muitos entre contratos e outras entidades. Essa abordagem assegura que a tabela principal (por exemplo, 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:

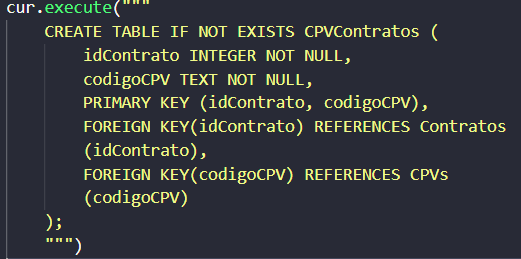
* Descrição e Código CPV: A separação entre o Código CPV e a Descrição CPV garante que não haja dependências transitivas dentro da tabela CPVs. Ambos os atributos dependem diretamente da chave primária, evitando dependências entre atributos não chave.
* Locais de Execução: A modelagem hierárquica entre 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.
* 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.



# 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. A computer screen shot of a code

   Description automatically generated**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.

Além disso, como dito no item 2, por conta do RGPD, tivemos que lidar com os casos em que o NIF das entidades não está exposto. Por conta disso, para não gerar duplicidade, a comparação de duplicidade foi feita utilizando o NIF e o nome da empresa, para que, nos casos RGPD não sejam reconhecidas empresas diferentes como a mesma.

A computer screen shot of a program

Description automatically generatedA computer screen shot of text

Description automatically generatedPor exemplo, para verificar a existência de um tipo de contrato antes de inseri-lo na tabela *TiposContratos*, utilizamos as seguintes funções:

1. **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**.

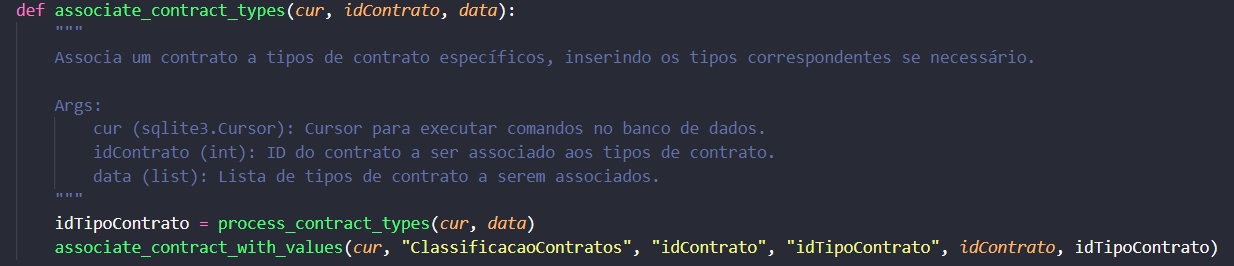
Além disso, existem casos muito específicos de processamento, como no caso dos **locais de execução**, **entidades** ou **fundamentação**, que exigem um tratamento mais detalhado e customizado. Nestes casos, a criação de funções específicas para cada um desses casos de processamento foi fundamental para 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.

A screen shot of a computer screen

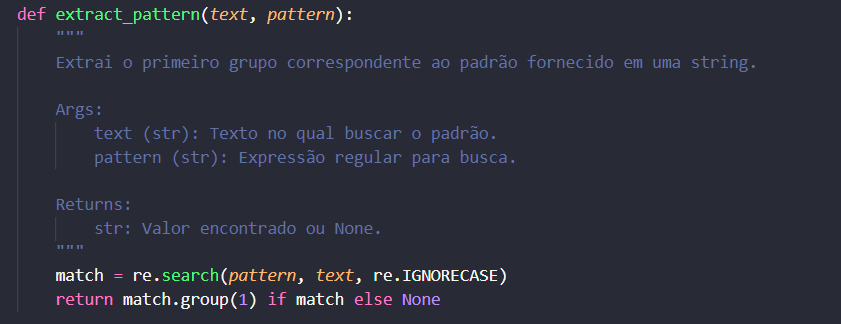
Description automatically generatedPor exemplo, a função para processar os **tipos de contratos** pode ser estruturada da seguinte forma:

1. **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.

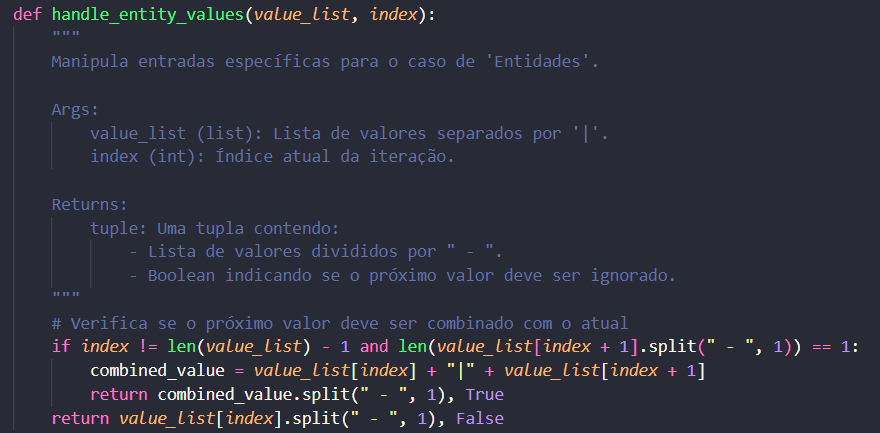
Um exemplo de uma função específica que associa o **id** do contrato com um atributo pode ser a função associate\_contract\_types. Esta função é responsável por associar o **id** de um contrato com o tipo de contrato correspondente. Ela chama uma função específica para processar os tipos de contrato (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.

1. **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.
2. **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.





1. **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 **" | "**.



1. **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://euc-word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=pt-PT&rs=pt-PT&wopisrc=https%3A%2F%2Fuporto-my.sharepoint.com%2Fpersonal%2Fup202302069_up_pt%2F_vti_bin%2Fwopi.ashx%2Ffiles%2F75aaa5a828c947c48a490560c98ee680&wdprevioussession=03bc18c7-dc3d-a219-1a3a-6e712f73ff10&wdenableroaming=1&wdfr=1&mscc=1&wdodb=1&hid=67EE6BA1-4062-9000-CCDC-48ADB3F32D17.0&uih=sharepointcom&wdlcid=pt-PT&jsapi=1&jsapiver=v2&corrid=8fc68717-0fb6-70e8-7b09-57bd11b452cc&usid=8fc68717-0fb6-70e8-7b09-57bd11b452cc&newsession=1&sftc=1&uihit=docaspx&muv=1&ats=PairwiseBroker&cac=1&sams=1&mtf=1&sfp=1&sdp=1&hch=1&hwfh=1&dchat=1&sc=%7B%22pmo%22%3A%22https%3A%2F%2Fuporto-my.sharepoint.com%22%2C%22pmshare%22%3Atrue%7D&ctp=LeastProtected&rct=Normal&wdorigin=BrowserReload&csc=1&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush#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.

Para obter as tabelas basta executar o script */povoamento/schema.py* e, em sequência, o script */povoamento/seed.py*. O arquivo *contratos\_publicos.db* será criado.

|  |  |
| --- | --- |
| 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 | 79 |
| FundamentacaoContrato | 21793 |

# Interrogações SQL

As interrogações em SQL foram divididas em três principais categorias: as fáceis, médias e difíceis. Todas as respostas também podem ser verificadas na interface (verificar item 6).

1. **Fáceis**
   1. **Quais são os tipos de contrato em ordem alfabética?**

SELECT tipo

FROM TiposContratos

ORDER BY tipo;

A list of black text

Description automatically generated

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

SELECT pais, tipo, SUM(precoContratual) AS total

FROM Contratos

NATURAL JOIN LocaisDeExecucao

NATURAL JOIN Municipios

NATURAL JOIN Distritos

NATURAL JOIN Paises

NATURAL JOIN ClassificacaoContratos

NATURAL JOIN TiposContratos

GROUP BY pais, tipo;

A screenshot of a white sheet with black text

Description automatically generated

* 1. **Qual é o adjudicante com o maior valor de contrato?**

SELECT entidade AS adjudicante, MAX(precoContratual) AS valor\_maximo

FROM Contratos

JOIN Entidades ON idAdjudicante = idEntidade;

A black text on a white background

Description automatically generated

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

SELECT nif, entidade

FROM Contratos

NATURAL JOIN LocaisDeExecucao

NATURAL JOIN Municipios

NATURAL JOIN Distritos

NATURAL JOIN Paises

NATURAL JOIN Adjudicatarios

NATURAL JOIN Entidades

WHERE prazoExecucao < 365

GROUP BY idEntidade; -- pois, um adjudicatario pode estar em mais de um contrato

A screenshot of a computer

Description automatically generated

* 1. **Quais são os adjudicatários que começam com a letra “M”?**

SELECT entidade

FROM Entidades

NATURAL JOIN Adjudicatarios

WHERE entidade LIKE "M%";

A screenshot of a computer

Description automatically generated

1. **Médias**
   1. **Qual é o número de contratos em cada tipo diferente de procedimento?**

SELECT COUNT(idcontrato) as quantidade, procedimento as tipo\_procedimento

FROM contratos

NATURAL JOIN tiposProcedimentos

GROUP BY idProcedimento

A white text box with black text

Description automatically generated

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

SELECT entidade, COUNT(idContrato) AS num\_contratos

FROM Contratos

NATURAL JOIN Adjudicatarios

NATURAL JOIN Entidades

GROUP BY idEntidade

ORDER BY num\_contratos DESC

LIMIT 1;

A black text on a white background

Description automatically generated

* 1. **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?**

SELECT distrito, COUNT(idContrato) AS total

FROM Contratos

NATURAL JOIN AcordoQuadroContratos -- pega só os que tem descrição acordo quadro

NATURAL JOIN LocaisDeExecucao

NATURAL JOIN Municipios

NATURAL JOIN Distritos

WHERE dataPublicacao > '2024-01-10'

AND dataPublicacao < '2024-01-14'

AND distrito IS NOT NULL -- retira os distritos que não foram cadastrados

GROUP BY distrito;

A table with black text

Description automatically generated

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

WITH municipiosCentra AS ( -- municipios com contratos com procedimentos centralizados

SELECT idMunicipio

FROM Contratos

NATURAL JOIN LocaisDeExecucao

NATURAL JOIN Municipios

WHERE procedimentoCentralizado = "Sim"

GROUP BY idMunicipio

)

SELECT municipio

FROM Municipios

WHERE idMunicipio NOT IN municipiosCentra

AND municipio IS NOT NULL

ORDER BY municipio;

A screenshot of a phone

Description automatically generated

* 1. **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.**

WITH Adjudicantes5Cont AS ( -- retorna o id dos adjjudicantes com pelo menos 5 contratos

SELECT idAdjudicante

FROM Contratos

JOIN Entidades ON idAdjudicante = idEntidade

GROUP BY idAdjudicante

HAVING COUNT(idContrato) <= 5

)

SELECT nif, entidade AS designacao, objetoContrato

FROM Contratos

JOIN Entidades ON idAdjudicante = idEntidade

WHERE idAdjudicante IN Adjudicantes5Cont

ORDER BY entidade;

A screenshot of a computer

Description automatically generated

1. **Difíceis**
   1. **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.**

WITH MaiorValorPorPais AS (

SELECT

idContrato,

precoContratual,

idPais,

pais AS nomePais

FROM Contratos

NATURAL JOIN LocaisDeExecucao

NATURAL JOIN Municipios

NATURAL JOIN Distritos

NATURAL JOIN Paises

WHERE precoContratual IS NOT NULL

GROUP BY idPais

HAVING precoContratual = MAX(precoContratual)

)SELECT

MVPC.idContrato,

CP.codigoCPV,

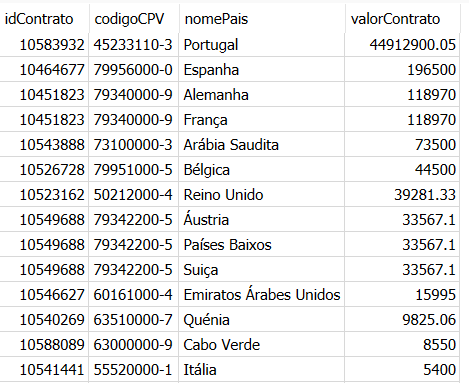
MVPC.nomePais,

MVPC.precoContratual AS valorContrato

FROM MaiorValorPorPais MVPC

NATURAL JOIN CPVContratos CP

ORDER BY valorContrato desc, CP.codigoCPV ASC;



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

SELECT

TC.tipo AS tipoContrato,

M.municipio,

COUNT(C.idContrato) AS quantidadeContratos,

DATE(C.dataCelebracaoContrato) AS dia

FROM Contratos C

JOIN LocaisDeExecucao L ON C.idContrato = L.idContrato

JOIN Municipios M ON L.idMunicipio = M.idMunicipio

JOIN TiposContratos TC ON C.idProcedimento = TC.idTipoContrato

WHERE DATE(C.dataCelebracaoContrato) = '2024-01-15'

GROUP BY TC.tipo, M.municipio, dia

ORDER BY TC.tipo, M.municipio;

Tabela

Descrição gerada automaticamente

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

WITH MediaPorTipoProcedimentoDistrito AS (

SELECT

D.distrito,

TP.procedimento AS tipoProcedimento,

ROUND(AVG(C.precoContratual), 2) AS mediaValor

FROM Contratos C

JOIN LocaisDeExecucao L ON C.idContrato = L.idContrato

JOIN Municipios M ON L.idMunicipio = M.idMunicipio

JOIN Distritos D ON M.idDistrito = D.idDistrito

JOIN TiposProcedimentos TP ON C.idProcedimento = TP.idProcedimento

WHERE C.precoContratual IS NOT NULL

AND D.distrito IS NOT NULL

GROUP BY D.distrito, TP.procedimento

)

SELECT \*

FROM MediaPorTipoProcedimentoDistrito

ORDER BY distrito, tipoProcedimento;

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

* 1. **Qual é a fundamentação que mais aparece em cada distrito?**

WITH FundamentacaoPorDistrito AS (

SELECT

d.distrito,

f.artigo,

f.numero,

f.alinea,

f.referenciaLegislativa,

(COALESCE(f.artigo, '') || ' ' ||

COALESCE(f.numero, '') || ' ' ||

COALESCE(f.alinea, '') || ' - ' ||

COALESCE(f.referenciaLegislativa, '')) AS fundamentacao,

COUNT(\*) as total

FROM LocaisDeExecucao le

NATURAL JOIN Municipios m

NATURAL JOIN Distritos d

NATURAL JOIN FundamentacaoContratos fc

NATURAL JOIN Fundamentacoes f

GROUP BY d.distrito, f.artigo, f.numero, f.alinea, f.referenciaLegislativa, fundamentacao

),

MaxFundamentacao AS (

SELECT

distrito,

MAX(total) as max\_total

FROM FundamentacaoPorDistrito

GROUP BY distrito

)

SELECT

fpd.distrito,

fpd.artigo,

fpd.numero,

fpd.alinea,

fpd.referenciaLegislativa,

fpd.fundamentacao,

fpd.total

FROM FundamentacaoPorDistrito fpd

NATURAL JOIN MaxFundamentacao mf

WHERE fpd.total = mf.max\_total

ORDER BY fpd.distrito;

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

* 1. **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.**

WITH MunicipiosSemContratosAltos AS (

SELECT DISTINCT m.municipio,

MAX(c.precoContratual) AS valorMaisProximo

FROM Municipios m

NATURAL JOIN LocaisDeExecucao le

NATURAL JOIN Contratos c

WHERE m.idMunicipio NOT IN (

SELECT DISTINCT m2.idMunicipio

FROM Municipios m2

NATURAL JOIN LocaisDeExecucao le2

NATURAL JOIN Contratos c2

WHERE c2.precoContratual > 1000000

)

AND m.municipio IS NOT NULL

GROUP BY m.municipio

HAVING MAX(c.precoContratual) IS NOT NULL

)

SELECT

municipio,

valorMaisProximo

FROM MunicipiosSemContratosAltos

ORDER BY municipio;

Tabela

Descrição gerada automaticamente

Uma versão com uma melhor formatação e amostragem encontra-se na interface do item abaixo.

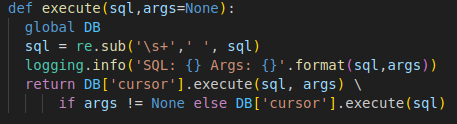
# Aplicação Python

A aplicação em Python foi feita com base em Flask e implementa diversos endpoints que tem o objetivo de esclarecer e mostrar a maior quantidade de dados da base de dados dos contratos públicos. A interface funciona com base em três scripts: db.py, server.py e app.py.

Para executar e conferir o funcionamento da interface basta executar o script *interface/server.py*, com o arquivo .db da base de dados criado e disponível no diretório correspondente ao indicado no script *server.py*.

1. db.py

Esse script contém as funções que implementam as consultas em SQL, via SQLite3. A principal função é a execute:

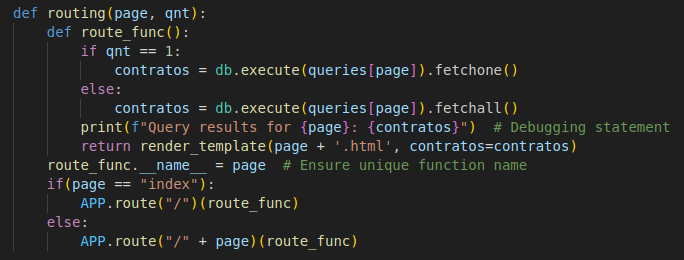
Essa função recebe um código em SQL e, utilizando a base de dados, retorna o resultado que pode ser consultado.

1. server.py

É responsável por inicializar a interface na porta 9000 e também se conecta a base de dados, permitindo a realização de consultas.

1. app.py

Script contém todas as queries responsáveis pela busca e criação de endpoint para mostrar as informações. No dicionário “queries” encontra-se todo o código SQL das queries.

A função “routing” é responsável pela criação e envio de informações ao HTML com base em dois parâmetros. O “page” que identifica a respectiva página e qual query irá utilizar do dicionário. Além disso recebe o argumento “qnt” que é responsável por verificar a quantidade de resultados da query, ou seja, 1 para que seja mostrado somente o primeiro resultado e 2 ou mais para que sejam mostrados mais resultados.

Além dos três scripts, a interface em Flask define um HTML base (“base.html”), em que consta o header com o retorno a página inicial e uma função de procura pelo ID do contrato. Para todas as outras páginas se encontram seus respectivos HTMLs representando um endpoint da aplicação. Os principais endpoints estão listados abaixo:

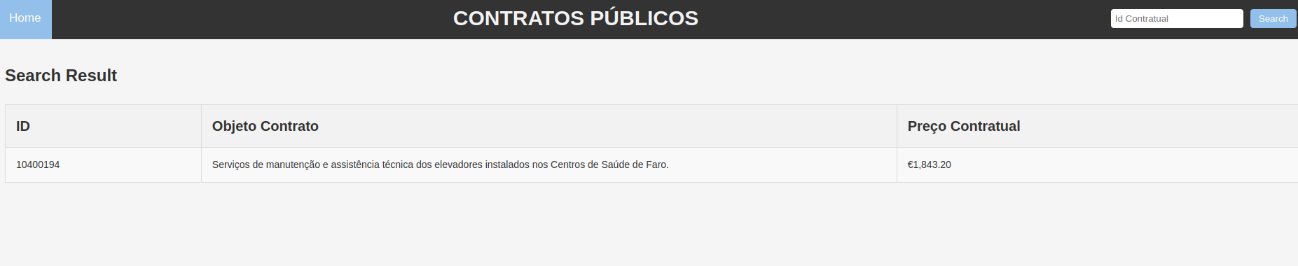
1. / (index.html)

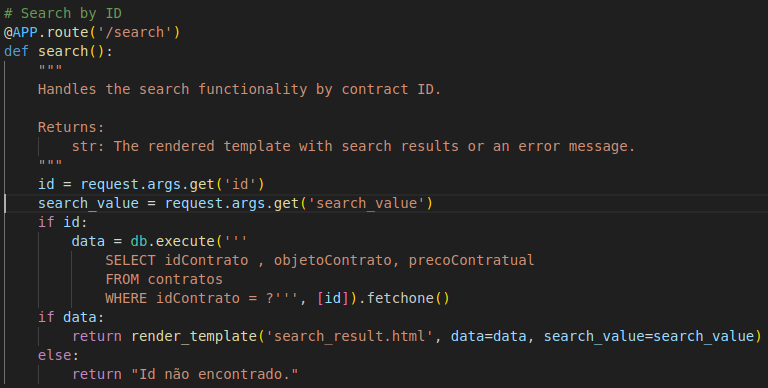
A página inicial da aplicação, com o header, um resumo inicial dos contratos, com o número de contratos e o valor total dos contratos (somados), e três divisões indicando todas as questões divididas por dificuldade.

1. /p1 ... /p15

 As páginas das perguntas, que correspondem ao header, um título com a pergunta e uma tabela com a resposta da query.

1. /search

 Esse endpoint é o mecanismo de pesquisa, do canto superior direito, em que é retornado o id, objeto do contrato e preço contratual.



A função de busca utiliza o recurso de parametrização do Flask, o que evita ataques maliciosos de SQL Injection.

|  |  |
| --- | --- |
| Endpoint | Descrição |
| / | Página inicial com acesso a todas as perguntas e mecanismo de busca |
| /p1 ... /p15 | Tabela com a resposta para a pergunta selecionada |
| /search?id=(...) | Página com os dados do contrato buscado |

# 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.

Além disso, 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.

Por fim, exploramos como nosso modelo poderia se relacionar em uma interface web, buscando realizar as queries via Flask em um navegador e utilizar input de um usuário (busca pelo ID do contrato) para se comunicar com nossa Base de Dados. Uma versão completa do trabalho pode ser conferida no seguinte repositório do GitHub <https://github.com/bzabot-CS/databases>.

# 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.

SQLite Documentation. Disponível em: <https://www.sqlite.org/docs.html>. Acesso em 8 dez. 2024.

Flask Documentation. Disponível em: <https://flask.palletsprojects.com/en/stable/>. Acesso em 8 dez. 2024.