

Projeto de Bases de Dados (CC2005)

Grupo nº: G35

Tema: X-Wines Wines Dataset

Elementos do grupo:

Nº mecanográfico	Nome
up202405046	José Paulo Sousa
up202403741	João Paulo Loureiro Borges
up202400836	Alyandro Luciano Jossefa
up202405095	João Miguel Costa

1. Modelação

1.1. Descrição do universo e modelo ER

O caso refere-se à organização e catalogação da produção vinícola.

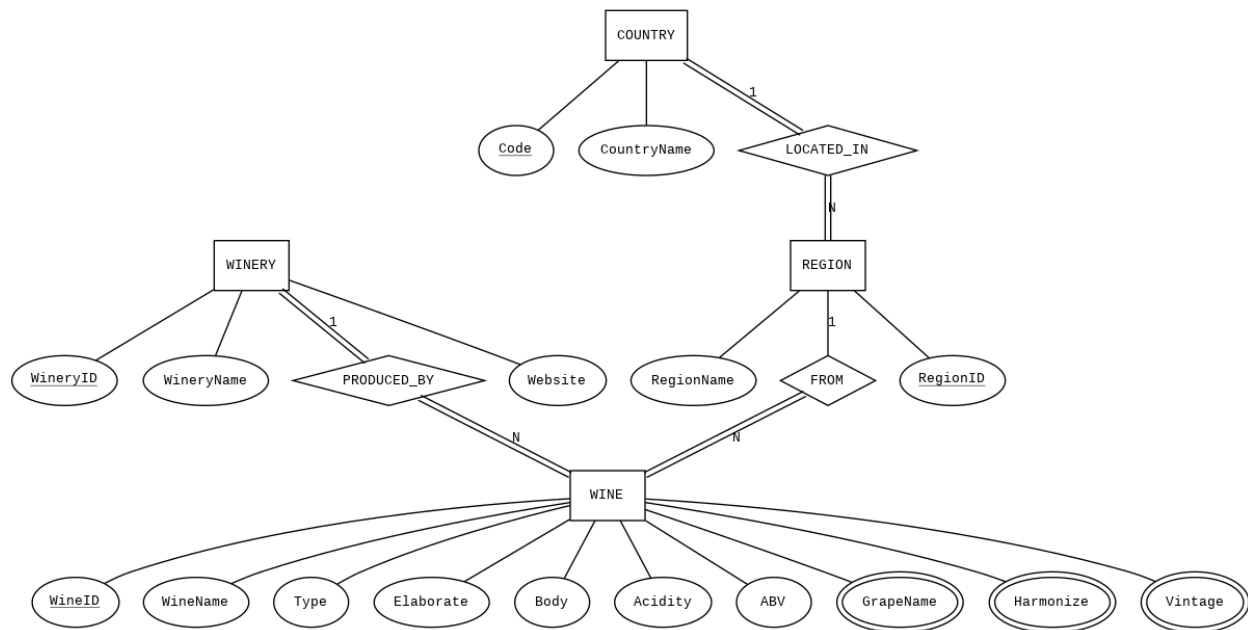
A informação centra-se no vinho enquanto produto final, descrito por diversos atributos, como o id, nome, o tipo do vinho (tinto,branco,rosé,...), o método de elaboração e um conjunto de características sensoriais, entre as quais se destacam o corpo (leve,médio,...) e a acidez (baixa,alta,...). Além disto, para uma caracterização mais completa, cada vinho pode ainda relacionar-se com várias castas de uva , diferentes sugestões de harmonização gastronómica e múltiplos anos de colheita.

Cada vinho é produzido por uma única casa vinícola. Apesar de cada produto ter apenas uma entidade responsável, uma mesma Adega pode ser fabricante de vários vinhos distintos. As casas vinícolas têm igualmente a possibilidade de produzir vinhos oriundos de diferentes regiões, sendo que cada vinho está sempre associado a uma única região. Por sua vez, cada Região encontra-se inserida num só País.

Assim, a estrutura de informação estabelece uma hierarquia que parte do vinho, passa pelo produtor e segue até à região e ao país de origem, permitindo uma visão completa tanto das características do vinho como da sua distribuição geográfica.

Modelo ER (Entidade-Relacionamento)

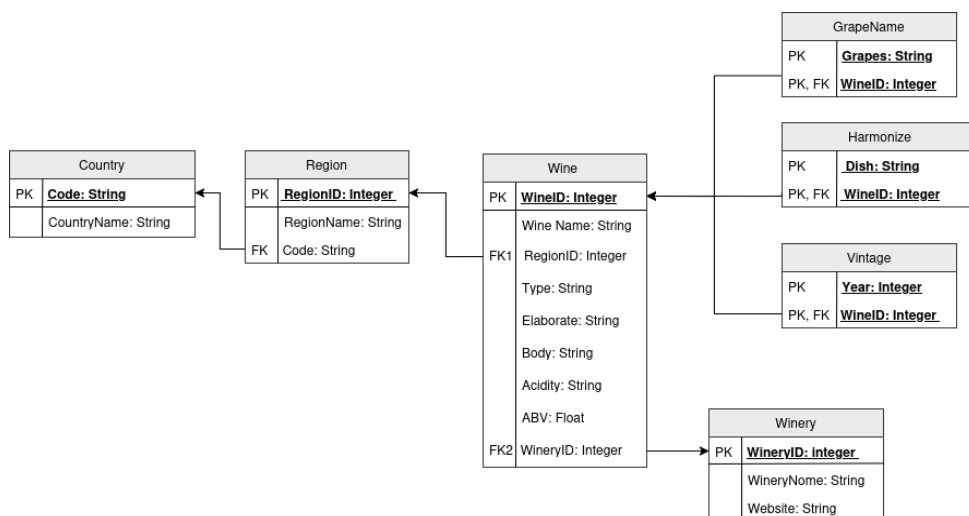
A análise do domínio permitiu identificar as entidades-tipo centrais do sistema: Country, Region, Winery e Wine. Cada uma foi representada no modelo ER com os respectivos atributos descritivos e chaves primárias. Os relacionamentos entre estas entidades estruturam a forma como a informação se organiza: regiões pertencem a países (*LOCATED_IN*), vinhos provêm de regiões (*FROM*) e são produzidos por adegas (*PRODUCED_BY*), refletindo dependências naturais do domínio.



1.2. Modelo relacional

O modelo Entidade-Relacionamento foi transformado num modelo relacional tendo em conta os seguintes aspetos:

1. A entidade *Country* origina a tabela **Country(Code, CountryName)**, onde *Code* é a chave primária.
2. A entidade *Region* origina a tabela **Region(RegionID, RegionName, Code)**. O atributo *Code* é chave externa que implementa o relacionamento **LOCATED_IN**, uma vez que cada região pertence a um único país.
3. A entidade *Winery* gera a tabela **Winery(WineryID, WineryName, Website)**.
4. A entidade *Wine* gera a tabela **Wine(WineID, WineName, Type, Elaborate, Body, Acidity, ABV, WineryID, RegionID)**. Os atributos *WineryID* e *RegionID* são chaves externas que representam os relacionamentos **PRODUCED_BY** (cada vinho é produzido por uma única casa vinícola) e **FROM** (cada vinho pertence a uma única região).
5. Atributos multivalor dão origem a novas tabelas auxiliares com chave externa referenciando a chave primária de Wine:
 - **GrapeName(Grapes, WineID)**: variedades de uva utilizadas em cada vinho.
 - **Harmonize(Dish, WineID)**: sugestões de pratos para harmonizar com o vinho.
 - **Vintage(Year, WineID)**: anos de colheita (vintage) disponíveis para o vinho.



2. Implementação

2.1. Povoamento de tabelas

Nome da tabela	Nº de entradas
Countries	31
Grapes	1584
Harmonize	4199
Region	324
Vintages	30251
Wine	1007
Winery	792

Para estruturar e gerir eficientemente os dados contidos no ficheiro X-Wines.csv, desenvolvemos um script em Python que utiliza as bibliotecas **pandas** para análise e manipulação de dados, e **sqlite3** para a gestão da base de dados. O objetivo principal deste script é criar uma Base de Dados Relacional (XWines_Relational.db) com tabelas bem definidas, chaves primárias (PK) e chaves estrangeiras (FK).

1. Preparação e Carregamento de Dados

O processo inicia-se com a configuração dos caminhos dos ficheiros e a preparação do ambiente:

- **Leitura do Ficheiro de Origem:** A biblioteca **pandas** é utilizada para carregar o ficheiro X-Wines.csv para um **DataFrame** (df).

```
print("A carregar ficheiro CSV...")
try:
    df = pd.read_csv(CSV_FILE)
except Exception as e:
    print(f"Erro ao ler CSV: {e}")
    return
```

- **Função de Processamento de Listas:** Uma função (parse_list) é definida para converter listas que estão representadas como *strings* no CSV (como é o caso das colunas 'Grapes', 'Harmonize' e 'Vintages') em listas Python utilizáveis.

```
def parse_list(x):
    if isinstance(x, str) and x.strip().startswith('['):
        try:
            return ast.literal_eval(x)
        except:
            return []
    return []

print("A processar tabelas...")
```

2. Criação de DataFrames Individuais

Nesta etapa dividimos o DataFrame original (df) em múltiplos DataFrames menores, cada um representando uma tabela distinta na base de dados relacional. Para garantir a integridade dos dados, removemos duplicados utilizando a chave primária de cada futura tabela e descartamos valores nulos relevantes.

```
# Países
df_countries = df[['Code', 'Country']].drop_duplicates('Code').dropna()

# Regiões
df_region = df[['RegionID', 'RegionName', 'Code']].drop_duplicates('RegionID').dropna()

# Produtores / Wineries
df_winery = df[['WineryID', 'WineryName', 'Website']].drop_duplicates('WineryID')

# Vinhos
df_wine = df[['WineID', 'WineName', 'Type', 'Elaborate', 'Body',
| | | | 'Acidity', 'ABV', 'WineryID', 'RegionID']].drop_duplicates('WineID')

# Uvas (listas)
df_grapes = df[['WineID', 'Grapes']].copy()
df_grapes['Grapes'] = df_grapes['Grapes'].apply(parse_list)
df_grapes = df_grapes.explode('Grapes').dropna()

# Harmonizações (listas)
df_harm = df[['WineID', 'Harmonize']].copy()
df_harm['Harmonize'] = df_harm['Harmonize'].apply(parse_list)
df_harm = df_harm.explode('Harmonize').dropna()

# Colheitas / Vintages
df_vint = df[['WineID', 'Vintages']].copy()
df_vint['Vintages'] = df_vint['Vintages'].apply(parse_list)
df_vint = df_vint.explode('Vintages').dropna()
```

3. Definição do Esquema e Criação das Tabelas

Após a preparação dos DataFrames, procedemos à criação da base de dados:

- **Conexão SQLite:** É estabelecida a conexão com o ficheiro de base de dados (DB_FILE) utilizando `sqlite3.connect()`.

```
print("A gravar base de dados SQLite...")
conn = sqlite3.connect(DB_FILE)
```

- **Ativação de Foreign Keys:** A instrução SQL `PRAGMA foreign_keys = ON;` é executada para impor as restrições de integridade referencial definidas pelas chaves estrangeiras.

```
conn.execute("PRAGMA foreign_keys = ON;")
cursor = conn.cursor()
```

- **Definição do Esquema SQL:** O esquema (`schema_sql`) é uma *string* de texto que contém todas as instruções `CREATE TABLE`, definindo a estrutura de cada tabela, incluindo a PK e as FKs. Para uma melhor organização daremos o exemplo só da TABLE Wine:

`CREATE TABLE Wine` define as colunas e estabelece as relações de Chave Estrangeira (FOREIGN KEY) com as tabelas Winery e Region, assegurando a integridade dos dados.

```
-- Tabela principal de Vinhos
CREATE TABLE Wine (
    WineID INTEGER PRIMARY KEY,
    WineName TEXT,
    Type TEXT,
    Elaborate TEXT,
    Body TEXT,
    Acidity TEXT,
    ABV REAL,
    WineryID INTEGER,
    RegionID INTEGER,
    FOREIGN KEY (WineryID) REFERENCES Winery(WineryID),
    FOREIGN KEY (RegionID) REFERENCES Region(RegionID)
);
```

- **Execução do Esquema :** O método `cursor.executescript(schema_sql)` é a instrução fundamental que envia e executa todas as instruções SQL contidas no `schema_sql` de uma só vez. Isto cria todas as tabelas vazias com as suas restrições de PK e FK definidas e ativas, estabelecendo a fundação relacional da base de dados.

```
cursor.executescript(schema_sql)
conn.commit()
```

4. População da Base de Dados

Finalmente, os dados processados nos DataFrames pandas são inseridos nas tabelas SQLite correspondentes. O método `.to_sql()` do pandas é utilizado, especificando o nome da tabela, a conexão (conn) e o modo de inserção (if_exists="append"):

```
df_countries.to_sql("Countries", conn, if_exists="append", index=False)
df_region.to_sql("Region", conn, if_exists="append", index=False)
df_winery.to_sql("Winery", conn, if_exists="append", index=False)
df_wine.to_sql("Wine", conn, if_exists="append", index=False)
```

Para as tabelas que resultaram da expansão de listas (df_grapes, df_harm, df_vint), é necessário renomear a coluna expandida para corresponder ao nome definido no esquema SQL (ex: Grapes para Grape) antes de executar o `.to_sql()`:

```
df_grapes.rename(columns={"Grapes": "Grape"}).to_sql("Grapes", conn, if_exists="append", index=False)
df_harm.rename(columns={"Harmonize": "Harmonize"}).to_sql("Harmonize", conn, if_exists="append", index=False)
df_vint.rename(columns={"Vintages": "Vintage"}).to_sql("Vintages", conn, if_exists="append", index=False)
```

Este processo resulta na criação da base de dados **XWines_Relational1.db**, totalmente estruturada com o modelo relacional e povoada com os dados extraídos do ficheiro CSV de origem.

```
conn.close()

print("Base de dados criada com sucesso com PK e FK")
```

2.2. Interrogações SQL

P1) Liste todos os tipos de vinho (Type) distintos que estão presentes na base de dados. Ordene por tipo.

```
SELECT DISTINCT Type
FROM Wine
ORDER BY Type;
```

P2) Qual é o vinho com o teor alcoólico (ABV) mais elevado?

```
SELECT
    WineID,
    WineName,
    ABV
FROM Wine
ORDER BY
    ABV DESC
LIMIT 1
```

P3) Qual é o teor alcoólico médio (ABV) para os vinhos do tipo 'Sparkling' (Espumante), arredondado a duas casas decimais?

```
SELECT ROUND(AVG(ABV), 2) AS AverageSparklingABV
FROM Wine
WHERE Type = 'Sparkling';
```

P4) Qual nome da vinícola (WineryName) e o respetivo website (Website) para todas as vinícolas que produzem vinhos na região 'Douro'.

```
SELECT T2.WineryName, T2.Website
FROM Wine AS T1
  JOIN
    Winery AS T2 ON T1.WineryID = T2.WineryID
  JOIN
    Region AS T3 ON T1.RegionID = T3.RegionID
WHERE T3.RegionName = 'Douro'
GROUP BY T2.WineryName, T2.Website
ORDER BY T2.WineryName;
```

P5)Quais tipos de vinho (Type) harmonizam tanto com "Beef" (Carne de Vaca) quanto com "Poultry" (Aves)?

```
SELECT T1.Type
FROM Wine AS T1
  JOIN Harmonize AS T2 ON T1.WineID = T2.WineID
WHERE T2.Harmonize = 'Beef'
```

INTERSECT

```
SELECT T1.Type
FROM Wine AS T1
  JOIN Harmonize AS T2 ON T1.WineID = T2.WineID
WHERE T2.Harmonize = 'Poultry'
```

```
ORDER BY T1.Type;
```

P6) Top 10 regiões com mais vinhos diferentes produzidos

```
SELECT
    T2.RegionID,
    T2.RegionName,
    COUNT(T1.WineID) AS TotalWines
FROM Wine AS T1
    JOIN Region AS T2
    ON T1.RegionID = T2.RegionID
GROUP BY
    T2.RegionID
ORDER BY
    TotalWines DESC
LIMIT 10
```

P7) Media de alcool por pais

```
SELECT
    T3.Country,
    AVG(T1.ABV)
FROM Wine AS T1
    JOIN Region AS T2
    ON T1.RegionID = T2.RegionID
    JOIN Countries AS T3
    ON T2.Code = T3.Code
WHERE
    T1.ABV IS NOT NULL
GROUP BY
    T3.Country
ORDER BY
    AVG(T1.ABV) DESC;
```

P8) Liste os nomes das vinícolas (WineryName) que apenas produzem vinhos com um nível de acidez (Acidity) 'High' (Alto).

```
SELECT
    T1.WineryName
FROM
    Winery AS T1
    JOIN
        Wine AS T2 ON T1.WineryID = T2.WineryID
GROUP BY
    T1.WineryName
HAVING
    COUNT(T2.WineID) = COUNT(CASE WHEN T2.Acidity = 'High' THEN T2.WineID END)
    AND COUNT(T2.WineID) > 0;
```

P9) Existem adegas que produzem vinhos em regiões de países diferentes? Liste os nomes dessas adegas.

```
SELECT
    T1.WineryID,
    T1.WineryName
FROM Winery AS T1
    JOIN Wine AS T2
        ON T1.WineryID = T2.WineryID
        JOIN Region AS T3
            ON T2.RegionID = T3.RegionID
            JOIN Countries AS T4
                ON T3.Code = T4.Code
GROUP BY
    T1.WineryID
HAVING
    COUNT(DISTINCT T4.Code) > 1
```

P10) Nomes dos vinhos (WineName) que são feitos com mais de uma casta (Grape), e retorne o nome do vinho e o número de castas utilizadas.

```
SELECT
    T1.WineID,
    T1.WineName,
    COUNT(T2.Grape) AS numero_castas
FROM Wine AS T1
JOIN Grapes AS T2
    ON T1.WineID = T2.WineID
GROUP BY
    T1.WineID,
    T1.WineName
HAVING
    COUNT(T2.Grape) > 1
```

P11) Detetar regiões que produzem vinhos acima da média de teor alcoólico nacional

```
WITH AvgABVPerCountry AS (
    SELECT C.Code, AVG(W.ABV) AS CountryAvg
    FROM Wine W
    JOIN Region R ON W.RegionID = R.RegionID
    JOIN Countries C ON C.Code = R.Code
    GROUP BY C.Code
)
SELECT
    R.RegionName,
    C.Country,
    ROUND(AVG(W.ABV), 2) AS RegionAvgABV,
    ROUND(A.CountryAvg, 2) AS CountryAvgABV
FROM Wine W
JOIN Region R ON W.RegionID = R.RegionID
JOIN Countries C ON C.Code = R.Code
JOIN AvgABVPerCountry A ON A.Code = C.Code
GROUP BY R.RegionID
HAVING AVG(W.ABV) > A.CountryAvg
ORDER BY C.Country, RegionAvgABV DESC;
```

2.3. Aplicação Python

“Endpoint”	Funcionalidade
/	Página de entrada
/<table_name>/	Lista todos os registos de cada uma das tabelas da base de dados
/<table_name>/<pk_val>/	Exibe com detalhe todos os atributos de um registo e permite o acesso a dados relacionados (e.g. num registo da tabela Wine é possível aceder a toda a informação do vinho em todas as tabelas da base de dados).
/queries/	Apresenta as interrogações relevantes para o dataset em questão
/queries/<query_id>	Permite a visualização do sql que resolve a querie e ao mesmo tempo o resultado que é clicável e permite o redirecionamento para a página do registo em questão
/ai-sommelier	Apresenta a página de pesquisa com o "Sommelier IA" e sugestões de perguntas. Esta permite que o usuário faça perguntas que são respondidas através de uma API do google gemini.

table_name: Countries, Region, Winery, Wine, Grapes, Harmonize, Vintages

3. “Sommelier IA” - Extra

Como funcionalidade complementar ao projeto, implementou-se um motor de busca inteligente (“Sommelier IA”) capaz de interpretar linguagem natural e convertê-la em instruções de base de dados. Para tal, recorreu-se à API do Google Gemini (modelo gemini-2.5-flash), utilizando as funções nativas da biblioteca google.generativeai.

Apesar da integração técnica via Python ser direta, o foco principal deste desenvolvimento centrou-se no desenvolvimento do *prompt*. Para garantir que o modelo gera resultados úteis e executáveis sem erros na nossa base de dados SQLite, o *prompt* foi desenhado tendo em conta três requisitos fundamentais:

1. **Interpretação Semântica:** O modelo foi instruído para interpretar pedidos tanto em Português como em Inglês, realizando o mapeamento conceptual para os valores da base de dados (e.g., associar “Peixe” à tabela **Harmonize**).
2. **Formato de Saída (Output):** Foi imposta uma restrição estrita para que a resposta seja exclusivamente código SQL sintaticamente correto, sem formatação Markdown ou explicações textuais, permitindo a sua execução direta pelo motor da base de dados.
3. **Seleção de Colunas e Navegabilidade:** Para manter a consistência com a interface da aplicação, instruiu-se o modelo a selecionar obrigatoriamente as Chaves Primárias (e.g., **WineID**) como primeira coluna. Isto é crucial para que o sistema consiga gerar automaticamente as hiperligações para as páginas de detalhes dos resultados.

Adicionalmente, para mitigar “alucinações”, injetamos no contexto do *prompt* o esquema da base de dados (DDL) e uma amostragem dos valores existentes nas colunas principais. Isto garante que a IA “conhece” os dados antes de escrever a query. Dado o custo computacional desta operação, este contexto é gerado apenas no arranque da aplicação e armazenado em **cache** (memória), minimizando o impacto na performance.

O *prompt* final desenvolvido para cumprir estes objetivos é apresentado abaixo:

Role: Expert SQL Data Analyst. Dialect: SQLite.

Database Schema & Data Samples:

{schema context} <-- (DDL e Amostras de valores injetados via Cache)

Goal: Generate a SQL query to answer: "{user_question}"

Strict Rules:

1. Output ONLY raw SQL. No markdown.
2. MANDATORY: Select the Primary Key FIRST (e.g. WineID), THEN select meaningful columns.
3. Use LIKE for text searches (case-insensitive).
4. If searching for food pairings, join 'Wine' with 'Harmonize'.
5. If searching for grapes, join 'Wine' with 'Grapes' (column 'Grape').
6. Do NOT use LIMIT unless the user specifically asks.

4. Conclusão

Este projeto permitiu a implementação bem-sucedida de uma base de dados relacional robusta para o dataset "X-Wines". Através de um processo de modelação rigoroso e do desenvolvimento de um script em Python para a migração dos dados, garantimos a integridade e normalização da informação, solucionando eficazmente o desafio dos atributos multivalor.

A funcionalidade do sistema foi validada através de diversas interrogações SQL complexas e enriquecida pelo desenvolvimento do "Sommelier IA". Esta componente extra demonstrou a mais-valia da integração de Inteligência Artificial com bases de dados, permitindo consultas intuitivas em linguagem natural.

Em suma, este projeto permitiu-nos enriquecer o nosso conhecimento, principalmente sobre o processo de transição entre um dataset e a sua transformação numa base de dados funcional que pode ser utilizada em situações reais.

Referências

<https://ai.google.dev/gemini-api/docs?hl=pt-br>