FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Base de Dados

DCC-FCUP 2024

Billionaires Database

Fábio Elmiro Ramos Semedo	n.º 202300446
Gabriel Adriano Ramalho Almeida Carlos	n.º 202306755
Gonçalo de Carvalho Fernandes Branquinho Mota	n.º 202306501
Rafael Arruda Costa	n.º 202300278

Universo do Dataset:

O universo a ser tratado representa informações variadas sobre os "billionaires" do mundo inteiro, em meados de 2022. Esta informação propraga-se, desde detalhes pessoais, a geográficos, a regionais, a geográficos e económicos nacionais. Consoante analisámos os dados, fomos nos tornando cada vez mais capazes de determinar melhores formas de os organizar, atigindo, conclusivamente, a seguinte estrutura:

Informações pessoais e industriais:

- **a posição de riqueza (inteiro)**
- **a riqueza (inteiro, em milhões)**
- **a industria (sector económico)**
- **o nome de uma ou mais empresas e/ou de um ou mais mercados de interesse (string) (contém sub-informações úteis, se decompostas)**
- **o nome completo (string) (contém sub-informações deriváveis)**
- **a data de nascimento (date) (contém sub-informações deriváveis)**
- **o sexo (string, "M" ou "F")**
- **a nacionalidade (string)**
- **o país de residência (string)**
- **a cidade de residência (string)**

Informações geográficas regionais:

- **a cidade (string)**

Informações geográficas regionais exclusivas aos EUA:

- **o estado (string)**
- **a região do estado (string)**

Informações geográficas nacionais:

- **o país (string)**
- **o continente (string)**
- **a latitude (decimal)**
- **a longitude (decimal)**

Informações económicas nacionais:

- **a população (inteiro)**
- **a esperança de vida (decimal)**
- **índice de preços do consumidor (decimal, representa %)**
- **mudança de índice de preços do consumidor (decimal, representa %)**
- **o produto interno bruto (inteiro)**
- **a taxa de imposto (decimal, representa %)**
- **a receita fiscal (decimal, representa %)**
- **matrícula bruta no ensino primário (decimal, representa %)**
- **matrícula bruta no ensino terciário (decimal, representa %)**

Esta separação e marcação dos dados será essencial para a criação do UML e o Modelo Relacional, e, consequentemente, da Base de Dados e portanto das interrogações SQL.

Agora, antes de avançarmos para o UML, achamos que vale a pena mencionarmos que o dataset não tinha informação totalmente consistente, e às vezes até ausente. Estas adversidades não nos impediram de estruturarmos a informação da maneira que foi mostrada, somente exigiu que tivéssemos estes detalhes em atenção conforme povoámos a base de dados e em alguns casos os modelos.

Para acabar, o universo em questão é referente a uma data entre 2022 e 2023. Esta informação é relevante, e portanto para a derivação da idade de cada “billionaire”, iremos usar a data “2022/06/30” como data atual do data set. Isto faz sentido, uma vez que existe um padrão no data set que confirma que uma data próxima à referida terá sido usada para o cálculo das idades.

UML:

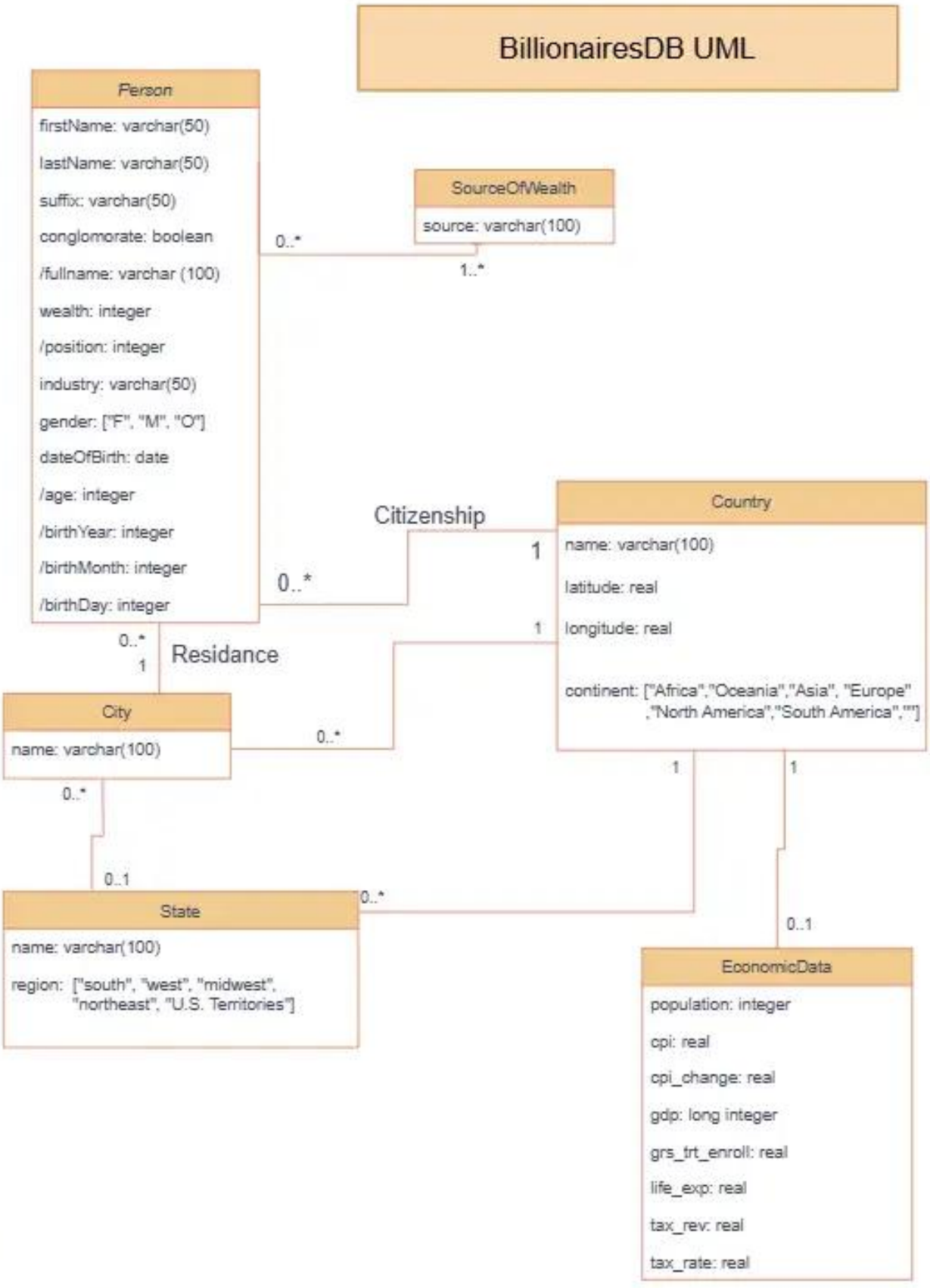
Para a construção do nosso UML, determinámos que podíamos separar cidade e usá-la estabelecer relações com e entre outros vários conjuntos de informações. Como pode ser visto, a tabela "City" liga-se a "Person", "Country" e "State". Nós soubemos que podíamos fazer isto uma vez que havia informação que, no data set, se repetia em função dos países e dos estados, e, da mesma forma, os países e o estados também se repetiam em função das cidades. Para além disso, as cidades são parte de estados e/ou países, ou seja, uma cidade de residência iria estabelecer, com uma dependência transitiva, o estado e/ou o país de residência. Portanto, foi assim que pudemos perceber de que forma é que podíamos separar, agrupar e relacionar os vários atributos do data set!

Também vimos que as nacionalidades eram representadas não pelo adjetivo, mas sim pelo nome referente ao país de origem, exatamente da mesma forma que os países de residência. Assim, decidimos que era boa ideia relacionarmos a nacionalidade ao conjunto de informação dos países ((Person.Citizenship U Country) C= {Countries}), uma vez que desta forma podíamos, também, perceber a relação que existe entre determinado bilionário (da tabela "Person") e o seu país de nacionalidade. Além do mais, para melhor podermos organizar a informação, decidimos que a separação da informação geográfica da informação económica dos países seria útil para que pudéssemos perceber onde certas informações estão.

Continuando, reparámos que as "sources" das pessoas da tabela "Person" eram compostas por sub-"sources", e portanto tomámos a decisão de "atomizar" as sources, facilitando a pesquisa das "sources" que estão relacionadas às pessoas, e, assim, dando origem à tabela "SourceOfWealth".

Concluindo, é importante que seja mencionado que nós tomámos atenção adicional à ideia de que a base de dados poderia ser alterada. Relativamente aos erros na base de dados, nós fizemos um esforço para que houvesse o mínimo de "nulls" o possível, no entanto não conseguimos evitar isso de um modo completo, o que fez com que o dataset não pudesse estar na 1.ª forma normal.

Na seguinte página, estará uma imagem representativa do UML que criámos. Para clarificar, decidimos usar "50" e "100" para o tamanho das strings por ser o standard, nas referências que consultámos.


BillionairesDB UML
Person
firstName: varchar(50)
lastName: varchar(50)
suffix: varchar(50)
conglomorate: boolean
/fullname: varchar (100)
wealth: integer
/position: integer
industry: varchar(50)
gender: ["F", "M", "O"]
dateOfBirth: date
/age: integer
/birthYear: integer
/birthMonth: integer
/birthDay: integer
SourceOfWealth
source: varchar(100)
0..*
1..*
Citizenship
1
0..*
Country
name: varchar(100)
latitude: real
longitude: real
continent: ["Africa","Oceania","Asia", "Europe"
,"North America","South America",""]
0..*
1
Residance
City
name: varchar(100)
0..*
0..1
State
name: varchar(100)
region: ["south", "west", "midwest",
"northeast", "U.S. Territories"]
0..*
1
1
0..1
EconomicData
population: integer
cpi: real
cpi_change: real
gdp: long integer
grs_trt_enroll: real
life_exp: real
tax_rev: real
tax_rate: real

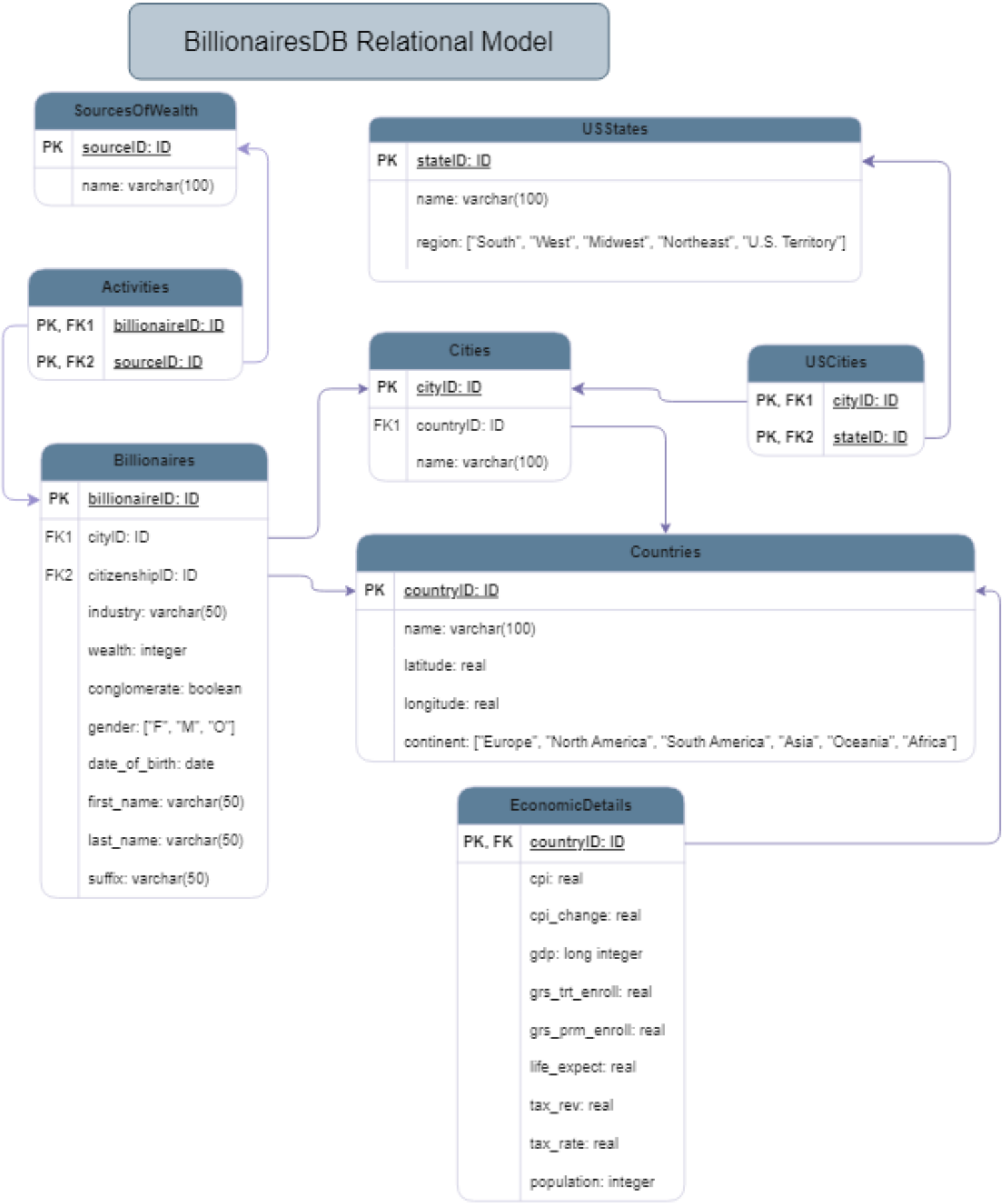
Modelo Relacional:

Para a construção do nosso modelo relacional, o processo de criação usou muito do UML, que já tinha sido feito. Pode ser visto que, semelhantemente ao UML, temos uma tabela "Cities" para as cidades, que aponta para a tabela "Countries" para os países, e que, indiretamente, permite o acesso à tabela "USStates" para os estados e as respetivas regiões, sendo este acesso tornado possível e isento de nulls com o uso da tabela "USCities", que somente contém os ID's de cidades dos Estados Unidos, que vale lembrar, são as únicas cidades que apresentam os atributos dos estados e das regiões sem valores a "null".

A separação da informação geográfica da económica, dos países, foi possível ao fazermos com que cada uma das duas tabelas tenha o ID do respetivo país. Desta forma, torna-se tudo muito mais facilmente acessível com menos ID's e com menos informação desnecessária sempre que se usa algum "join" do SQL. Desta forma, a tabela "Cities" é central para o acesso de uma grande parte da informação que pode ser generalizada, e como podemos ver, é possível extrair informação de qualquer uma das duas tabelas que dizem respeito ao país, ou podemos até extrair informação referente às cidades, que, no caso deste dataset, servem apenas para cidades dos Estados Unidos, sendo a informação referida o estado a que a cidade pertence e a região à qual o estado pertence. Vale mencionar que fizemos com que a nacionalidade, na tabela "billionaires", fosse uma coluna de ID's "citizenshipID" que nos permite extrair informação sobre os países de origem dos vários "billionaires". Esta é apenas mais uma das decisões que tomámos que facilitam a extração de informação para uma maior variedade de ideias e curiosidades.

Finalmente, depois de termos descoberto como "atomizar" as "sources", decidimos criar uma tabela intermediaria "Activities", que fica, relacionalmente, entre a tabela "Billionaires" e a tabela "SourcesOfWealth". Ela serve para podermos ligar um só "billionaire" a zero ou mais "sources". Para isto, a tabela "Activities" tem duas colunas, uma para os ID's dos "billionaires" e outra para os ID's das "sources". Assim, é possível extrairmos informação, por exemplo todas as "sources" de um "billionaire" ou, ao contrário, todos os "billionaires" de uma "source". Termos alcançado esta solução para as "sources" foi um game-changer, uma vez que assim temos muitas mais possibilidades para a extração eficiente de informação relacionada às sources.

Na seguinte página, estará uma imagem do modelo relacional.


BillionairesDB Relational Model
SourcesOfWealth
PK
sourceID: ID
name: varchar(100)
USStates
PK
stateID: ID
name: varchar(100)
region: ["South", "West", "Midwest", "Northeast", "U.S. Territory"]
Activities
PK, FK1
billionaireID: ID
PK, FK2
sourceID: ID
Cities
PK
cityID: ID
FK1
countryID: ID
name: varchar(100)
USCities
PK, FK1
cityID: ID
PK, FK2
stateID: ID
Billionaires
PK
billionaireID: ID
FK1
cityID: ID
FK2
citizenshipID: ID
industry: varchar(50)
wealth: integer
conglomerate: boolean
gender: ["F", "M", "O"]
date_of_birth: date
first_name: varchar(50)
last_name: varchar(50)
suffix: varchar(50)
Countries
PK
countryID: ID
name: varchar(100)
latitude: real
longitude: real
continent: ["Europe", "North America", "South America", "Asia", "Oceania", "Africa"]
EconomicDetails
PK, FK
countryID: ID
cpi: real
cpi_change: real
gdp: long integer
grs_trt_enroll: real
grs_prm_enroll: real
life_expect: real
tax_rev: real
tax_rate: real
population: integer

Povoamento da Base de Dados:

Para o povoamento da base de dados propriamente dito, usámos o que aprendemos nas aulas sobre a criação de tabelas no SQLite e, antes disso, investigámos. Seguidamente, descobrimos que no SQLite Browser é possível converter o .xlsx do data set para um .csv, o que nos deu a oportunidade fazermos a criação da base dados maioritariamente usando o SQLite. Depois disso, as tabelas foram criadas respeitando o modelo relacional. Os ID's foram criados, para melhor prática, usando um campo de ID definido como "Integer Primary Key", ao invés de usando chaves primárias naturais**.** Assim, o SQLite Studio fez a geração automática das chaves primárias para cada inserção que ainda não tivesse um ID. De resto, o mesmo tipo de processo foi aplicado para a criação de todas as tabelas. Era uma questão de usarmos as partições corretas para a função "row_number()".

Infelizmente, havia uma questão que teriamos mais cedo ou mais tarde de enfrentar – os problemas que as inconsistências e os erros do dataset nos iriam causar. Para começar, encontrámos erros de codificação com a ajuda de interrogações SQL "glob", que nos deixaram encontrar os caracteres especiais dos atributos. Foi a este ponto quando reparámos o quão problemáticas as inconsistências no dataset realmente eram. Por exemplo, quando a tabela "USStates" era criada havia erros ou entradas com atributos a vazios, graças a estados que, ao longo do dataset, apareciam relacionados a várias regiões diferentes, algo que não devia ser possível, não só por desrespeitar as relações que tinhamos estabelecido, mas também e especialmente por não fazer sentido no universo do dataset. Para nos desgastar ainda mais, o dataset mostrou que havia cidades com nomes diferentes, idênticos, que se referiam à mesma cidade, e outras vezes a cidades diferentes, portanto, de modo a reduzirmos semelhanças distintas, mudamos todas as letras dos nomes de todas as cidades para maiúsculas. Outro aspeto do dataset que nos incomodou foi o facto de haver "sources" semelhantes, por exemplo haver tanto a forma da palavra referente à "source" no singular e no plural ao mesmo tempo, ou gralhas nos nomes. Apesar disto, e de termos encontrado uma forma de, com interrogações SQL, agruparmos essas semelhancas, decidimos manter estas mudanças subtis, especialmente por alguns casos destas pequenas diferenças serem válidos. Mais pontualmente, fomos capazes de encontrar erros de numeração através da comparação entre contagens das interrogações SQL "distinct" e "all", e ,também, com a ajuda de scripts de Python que combinavam atributos e verificavam se estes eram únicos.

Agora, após termos localizado as irregularidades do dataset, começámos a pensar em formas de tratar o data set. Para isso, implementámos a substituição de caracteres especiais pelos seus equivalentes mais próximos em ASCII, utilizando a biblioteca Unidecode e também implementámos a adição de aspas de modo a evitarmos erros durante a execução de interrogações SQL. Para além disso, nós fomos capazes de encontrar erros de numeração através da comparação entre contagens das interrogações SQL "distinct" e "all", e ,também, com a ajuda de scripts de Python que combinavam atributos e verificavam se estes eram únicos. Relativamente à atomização das "sources", nós criámos um .csv a partir das "sources", atribuimos "personID's" temporários à tabela "SourcesOfWealth" após a atomizarmos com Python e seguidamente reinserimos a tabela no SQLite Studio.

Para acabar, foram realizados ajustes pontuais, específicos tais como a adição de ID's extraordinários para entradas que tinham atributos irregulares ou atributos semelhantes que representavam a mesma coisa ou até casos de cidades com nomes iguais que se localizam em estados americanos diferentes, e casos de nacionalidades cujos países não têm informação, fazendo com que tivessemos de criar linhas adicionais para a tabela "Countries" de forma a que nacionalidade não fosse um ID a null, que teve como consequência a existência de países sem informação económica.

Segue-se a tabela que inclui os nomes das tabelas da base de dados, cada um seguido do número de entradas da tabela a que se referem:

Nome da tabela:	Número de entradas:
Billionaires	2591
Activities	2965
Cities	759
Countries	88
EconomicDetails	78
SourcesOfWealth	758
USCities	275
USStates	45

Interrogações SQL:

1.- Riqueza total, ou seja, somatório de riqueza dos "billionaires", e quantidade de "billionaires" cuja riqueza não é parte de um "conglomerate", ou seja, o "conglomerate" tem de ser igual a 0, organizada em função dos níveis de educação nacional:

```
SELECT
    quantity,
    printf("%.2f", ((total_wealth / 1.0) /
    (
    SELECT
        SUM(wealth_millions)
    FROM
        (
        SELECT
            personID, conglomorate, wealth_millions
        FROM
            Billionaires
        )
    WHERE
      conglomorate = 0
    GROUP BY personID, conglomorate
    ))) AS total_wealth_perc, grs_prm_enroll, grs_trt_enroll
FROM
    (
    SELECT
        COUNT(wealth_millions) AS quantity, SUM(wealth_millions) AS total_wealth,
        e.grs_prm_enroll, ABS(e.grs_prm_enroll - 100), e.grs_trt_enroll, ABS(e.grs_trt_enroll)
    FROM
        Billionaires b
    JOIN
        Cities c ON b.cityID = c.cityID
    JOIN
        EconomicDetails e ON c.countryID = e.countryID
    WHERE b.conglomorate = 0
    GROUP BY ABS(grs_prm_enroll - 100), ABS(grs_trt_enroll - 100)
    ORDER BY ABS(grs_prm_enroll - 100), ABS(grs_trt_enroll - 100)
    )
```

QUANTITY	TOTAL_WEALTH_PERCENTAGE	GROSS PRIMARY ENROLL	GROSS TERTIARY ENROLL
2	0.06	100.0	82.0
4	0.09	100.0	67.8
7	0.08	100.2	88.2
385	7.82	100.2	50.6
28	0.56	99.8	49.3

2.- Percentagem de “billionaires” relativa à população, com ordem alfabetica decrescente por país, em notação científica:

```
SELECT
    o.name, printf("%.2E", (count(b.personID) / (e.population / 1.0))) AS ratio
FROM
    Billionaires b
JOIN
    Cities c ON b.cityID = c.cityID
JOIN
    EconomicDetails e ON c.countryID = e.countryID
JOIN
    Countries o ON e.countryID = o.countryID
GROUP BY population
ORDER BY o.name DESC
```

COUNTRY	RATIO
Vietnam	6.22E-08
Uzbekistan	2.98E-08
Uruguay	2.89E-07
United States	2.31E-06
United Kingdom	1.21E-06

3.- “Billionaires” com idade maior do que a esperança de vida cuja indústria envolve trabalhos perigosos:

```
SELECT
    b.industry, COUNT(b.personID) AS quantity
FROM
    Billionaires b
JOIN
    Cities c ON b.cityID = c.cityID
JOIN
    EconomicDetails e ON c.countryID = e.countryID
WHERE
    (CAST((julianday('2022-06-30') - julianday(b.birth_date)) AS integer) / 365.25) > e.life_expect
    AND
    (b.industry = "Metals & Mining" OR b.industry = "Construction & Engineering")
GROUP BY b.industry
ORDER BY b.industry DESC
```

INDUSTRY	QUANTITY
Metals & Mining	5
Construction & Engineering	3

4.- A quantidade e respetiva percentagem de “billionaires” que não são tão ricos quanto a “billionaire”, mulher, em questão, sendo que a percentagem é representada pela quantidade a dividir pelo número total de membros do sexo oposto:

```
SELECT
    rank, personId, first_name, last_name, wealth_millions,
    (2591 - rank - (gender_row - 1)) AS quantity,
    printf("%.4f", ((100*(2591 - rank - (gender_row - 1)) / 1.0) / (
        (
        SELECT
            count(gender)
        FROM
            Billionaires
        WHERE
            gender = "M"
        GROUP BY gender
        )))) AS percentage
FROM
    (SELECT
         *, Rank() OVER (PARTITION BY gender ORDER BY rank DESC) AS gender_row
     FROM
         (Billionaires b
             JOIN
          Ranks r ON b.personId = r.personId))
WHERE gender = "F"
ORDER BY quantity DESC, wealth_millions DESC
```

RANK	NAME	WEALTH	QUANTITY	PERCENTAGE
11	Francoise Bettencourt Meyers	$80500M	2258	99.5591
17	Julia Koch	$59000M	2253	99.3386
21	Alice Walton	$56700M	2250	99.2063
31	Jacqueline Mars	$38300M	2241	98.8095
35	Miriam Adelson	$35000M	2238	98.6772

4.- A quantidade e respetiva percentagem de “billionaires” que não são tão ricos quanto o “billionaire”, homem, em questão, sendo que a percentagem é representada pela quantidade a dividir pelo número total de membros do sexo oposto:

```
SELECT
    rank, personId, first_name, last_name, wealth_millions,
    (2591 - rank - (gender_row - 1)) AS quantity,
    printf("%.4f", ((100*(2591 - rank - (gender_row - 1)) / 1.0) / (
        (
        SELECT
            count(gender)
        FROM
            Billionaires
        WHERE
            gender = "F"
        GROUP BY gender
        )))) AS percentage
FROM
    (SELECT
         *, Rank() OVER (PARTITION BY gender ORDER BY rank DESC) AS gender_row
     FROM
         (Billionaires b
              JOIN
          Ranks r ON b.personId = r.personId))
WHERE gender = "M"
ORDER BY quantity DESC, wealth_millions DESC
```

RANK	NAME	WEALTH	AMOUNT	PERCENTAGE
1	Bernard Arnault	$211000M	323	100.0000
2	Elon Musk	$180000M	323	100.0000
3	Jeff Bezos	$114000M	323	100.0000
4	Larry Ellison	$107000M	323	100.0000
5	Warren Buffett	$106000M	323	100.0000

5.- Total de dinheiro (somatório) dos “billionaires” para cada indústria:

```
SELECT
    industry, SUM(wealth_millions) AS total
FROM
    Billionaires
GROUP BY industry
ORDER BY total DESC, industry
```

INDUSTRY	TOTAL WEALTH
Technology	$1870700M
Fashion & Retail	$1675200M
Finance & Investments	$1593600M
Manufacturing	$1006400M
Food & Beverage	$946000M

6.- Percentagem de “billionaires” localizados em cada continente:

```sql
SELECT
    o.continent, printf("%.1f", ((100 * COUNT(b.personID)) / 2591.0)) AS percentage
FROM
    Billionaires b
JOIN
    Cities c ON b.cityID = c.cityID
JOIN
    Countries o ON c.countryID = o.countryID
GROUP BY o.continent
ORDER BY o.continent
```

CONTINENT	PERCENTAGE
Africa	0.7
Asia	41.0
Europe	22.7
North America	31.7
Oceania	1.7

7.- Indústrias mais populares e a riqueza total das mesmas, considerando apenas “billionaires” dos Estados Unidos, para cada região dos Estados Unidos:

```
SELECT
    s.region, b.industry, COUNT(b.personID) AS quantity,SUM(b.wealth_millions) AS total_wealth
FROM
    Billionaires b
JOIN
    Cities c ON b.cityID = c.cityID
JOIN
    USCities uc ON c.cityID = uc.cityID
JOIN
    USStates s ON uc.stateID = s.stateID
GROUP BY s.region, b.industry
ORDER BY s.region, quantity DESC, b.industry
```

REGION	INDUSTRY	QUANTITY	TOTAL WEALTH
Midwest	Finance & Investments	16	$165000M
Midwest	Fashion & Retail	10	$71800M
Midwest	Food & Beverage	10	$36600M
Midwest	Technology	10	$32900M
Midwest	Manufacturing	8	$29500M

8.- Para cada valor de imposto, selecionar os “billionaires” mais ricos, até um máximo de 5, ordenando pelo nível do imposto:

```
SELECT
     k.first_name, k.last_name, k.wealth_millions, k.name, k.tax_rate
FROM
    (SELECT
         b.first_name, b.last_name, b.wealth_millions, o.name, e.tax_rate,
         (ROW_NUMBER() OVER (PARTITION BY e.tax_rate ORDER BY b.wealth_millions DESC)) AS ROW
     FROM
         (
             Billionaires b
         JOIN
             Cities c ON b.cityID = c.cityID
         JOIN
             EconomicDetails e ON c.countryID = e.countryID
         JOIN
             Countries o ON e.countryID = o.countryID
         )
     ORDER BY e.tax_rate DESC) k
 WHERE row <= 5
```

NAME	WEALTH	COUNTRY	TAX RATE
Marcos Galperin	4900	Argentina	106.3
Gregorio Perez Companc	2900	Argentina	106.3
Eduardo Eurnekian	1900	Argentina	106.3
Eduardo Costantini	1300	Argentina	106.3
Luis Carlos Sarmiento	6400	Colombia	71.2

9.- Dinheiro que iria para cada membro da população de cada país se a riqueza de todos os "billionaires" de cada país fosse distribuida pelo povo do país do próprio ("billionaire"):

```
SELECT k.name, (k.sum / k.population) AS wealth
FROM
    (
    SELECT
        o.name AS name,
        sum(b.wealth_millions)*1000000 AS sum,
        e.population AS population
    FROM
        (
            Billionaires b
        JOIN
            Cities c ON b.cityID = c.cityID
        JOIN
            Countries o ON c.countryID = o.countryID
        JOIN
            EconomicDetails e ON o.countryID = e.countryID
        )
    GROUP BY o.name
    ) k
```

NAME	MONEY PER PERSON
Algeria	$106
Andorra	$19444
Argentina	$244
Armenia	$405
Australia	$6733

10.- “Sources” mais populares agrupadas por país e para cada grupo de país, ordenadas pelo dinheiro total que é feito através das mesmas:

```
SELECT
    name, source, MAX(soma) AS best_total
FROM
   (
   SELECT
        o.name AS name, s.source AS source, SUM(b.wealth_millions) AS soma
   FROM
       Billionaires b
   JOIN
       Activities a ON b.personID = a.personID
   JOIN
       SourcesOfWealth s ON a.sourceID = s.sourceID
   JOIN
       Cities c ON b.cityID = c.cityID
   JOIN
       Countries o ON c.countryID = o.countryID
   GROUP BY o.name
   )
GROUP BY name
ORDER BY best_total DESC, name, source
```

COUNTRY	SOURCE	TOTAL MONEY
United States	Tesla	$5344500M
China	Beverages	$1983800M
India	Diversified	$714900M
Russia	Metals	$546600M
France	LVMH	$529100M

11.- Número de “billionaires” por par país de residência-nacionalidade:

```sql
SELECT
   c1.name AS Origin_Country,
   c2.name AS Destination_Country,
   COUNT(b.personId) AS Total_Billionaires
FROM
   Billionaires b
JOIN
   Countries c1 ON b.citizenshipID = c1.countryid
JOIN
   Cities ci ON b.cityID = ci.cityid
JOIN
   Countries c2 ON ci.countryid = c2.countryid
WHERE
   b.citizenshipID != ci.countryid
GROUP BY c1.name, c2.name
ORDER BY Total_Billionaires DESC;
```

ORIGIN COUNTRY	DESTINATION COUNTRY	TOTAL BILLIONAIRES
Hong Kong	China	18
Canada	United States	8
United States	China	8
China	Hong Kong	7
Russia	Switzerland	7

12.- “Billionaires” jovens, com idade inferior a 45 anos, em indústrias pouco populares, tal que o número de “billionaires” nessas indústrias é inferior ou igual a 50:

```sql
WITH IndustryCounts AS (
    SELECT
        b.industry,
        COUNT(*) AS Total_Billionaires
    FROM
        Billionaires b
    GROUP BY
        b.industry
    HAVING
        COUNT(*) <= 50
)
SELECT
    b.first_name AS First_name,
    b.last_name AS Last_name,
    b.industry AS Industry,
    b.wealth_millions AS Wealth_in_Millions,
    Total_billionaires,
    (CAST((JULIANDAY('2022-06-30') - JULIANDAY(b.birth_date)) AS INTEGER) / 365.25) AS Age
FROM
    Billionaires b
JOIN
    IndustryCounts ic ON b.industry = ic.industry
WHERE
    (CAST((JULIANDAY('2022-06-30') - JULIANDAY(b.birth_date)) AS INTEGER) / 365.25) < 45
ORDER BY Age ASC, Wealth_in_Millions DESC;
```

NAME	INDUSTRY	NUMBER OF BILLIONAIRES	WEALTH	AGE
LeBron James	Sports	39	1000	37.497604380561256
Maria Angelicoussis	Logistics	38	5600	40.36139630390144
Fahed Hariri	Construction & Engineering	42	1200	41.50034223134839
Ayman Hariri	Construction & Engineering	42	1400	44.12320328542094

13.- Média do produto interno bruto dos países que têm “billionaires”, por continente:

```
SELECT
    t1.continent, ROUND(t1.RelativeGdp,2) AS Relative_Gdp,
    ROUND(t1.RelativeGdp/(SUM(t2.RelativeGdp))*100, 2) AS percentage
FROM
    (
    SELECT
        continent, AVG(AdjGdp) as RelativeGdp
    FROM
        (
        SELECT
            cn.name, cn.continent, e1.gdp/e1.population AS AdjGdp, e1.*
        FROM EconomicDetails e1 INNER JOIN Countries cn ON e1.countryid = cn.countryid
        )
    GROUP BY continent
    ORDER BY AVG(AdjGdp) Desc
    ) t1,
    (
    SELECT
        continent, AVG(AdjGdp) AS RelativeGdp
    FROM
        (
        SELECT
            cn.name, cn.continent, e1.gdp/e1.population AS AdjGdp, e1.*
        FROM
            EconomicDetails e1 INNER JOIN Countries cn ON e1.countryid = cn.countryid
        )
    GROUP BY continent
    ORDER BY AVG(AdjGdp) DESC
    ) t2
GROUP BY t1.continent
ORDER BY percentage DESC
```

CONTINENT	RELATIVE GDP	PERCENTAGE
North America	51235.63	28.05
Oceania	48352.0	26.47
Europe	47696.75	26.11
Asia	21471.78	11.76
South America	10526.0	5.76

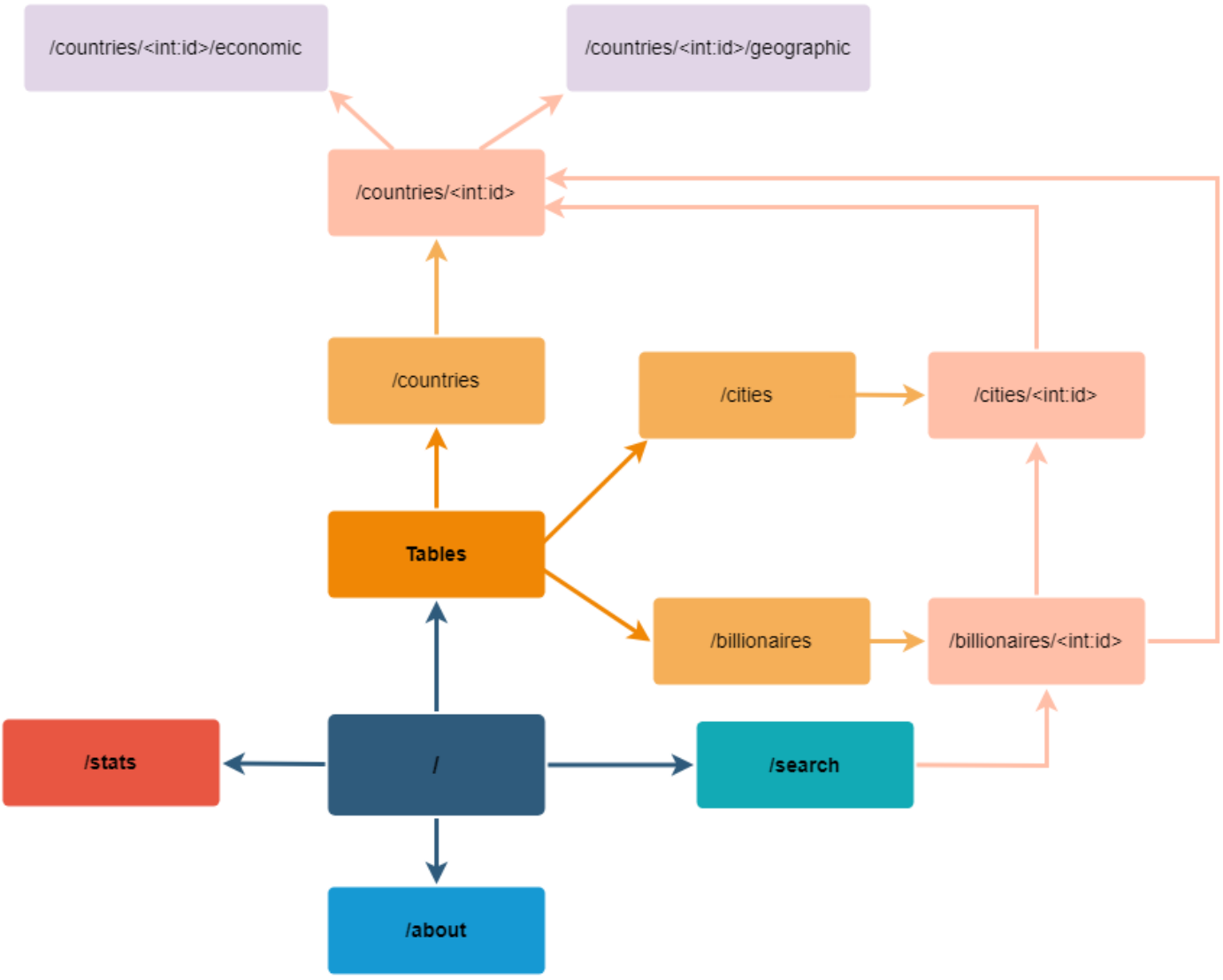
Aplicação de Python:

"Endpoint"	Funcionalidade
/	Mostra os top 50 "billionaires" em função da riqueza e saúda o utilizador da aplicação.
/search	Sistema de pesquisa avançado que permite pesquisar por "billionaires" que respeitem os filtros selecionados, permitindo então filtragem e ordenação.
/billionaires	Permite pesquisar por "billionaires" de forma rápida e simplificada.
/billionaires/<int:id>	O "id" é o personID, primário da tabela "Billionaires", e é do tipo integer. Este endpoint permite a visualização de informação referente ao "billionaire" associado ao "id".
/countries	Permite pesquisar por países, com base nas suas informações economicas, geográficas e pelo nome dos mesmos, de forma rápida e simplificada.
/countries/<int:id>	O "id" é o countryID, primário da tabela "Countries", e é do tipo integer. Permite a escolha entre informações geográficas ou económicas, cada uma do país associado ao "id".
/countries/<ind:id>/geographic	O "id" é o countryID, primário da tabela "Countries", e é do tipo integer. Permite a visualização da informação geográfica do país associado ao "id".
/countries/<int:id>/economic	O "id" é o countryID, primário da tabela "Countries", e é do tipo integer. Permite a visualização da informação económica do país associado ao "id".
/cities	Permite pesquisar por cidades, através do nome ou localização das mesmas, de forma rápida e simplificada,
/cities/<int:id>	O "id" é o cityID, primário da tabela "Cities", e é do tipo integer. Permite a visualização da localização da cidade, particularmente o país em que se situa, e, se a mesma for uma cidade dos Estados Unidos, em que estado se situa e a que região esse mesmo estado pertence.
/stats	Permite a visualização das 13 interrogações SQL.
/about	Mostra informação que serve para dar contexto à aplicação e à base de dados.

A aplicação foi feita com Python, Jinja, Flask, HTML e CSS. As ideias dos mecanismos do website foram evoluindo conforme pensávamos no que podíamos usar para navegar pela base de dados. Também usámos, como base para a aplicação, no começo, o exemplo da aplicação sobre os filmes, que nos foi mostrada pelos professores. Por último, para motivos de segurança, usámos interrogações SQL parameterizadas, de modo a prevenirmos injeções de SQL. Também usamos Flask Route com conversor de int:id de tal forma que as funções da aplicação não recebam informação errada, ou seja, no contexto desta aplicação, valores que não sejam inteiros, e, se alguma vez receberem, acontece um “404 Not Found”.

Segue-se uma representação gráfica do endpoints.

Representação dos endpoints em grafo:



Conclusão:

O trabalho deixou-nos a perceber a utilidade dos procedimentos todos que aprendemos nesta cadeira. Desde os UML's aos Modelos Relacionais, à base de dados em si. Aprendermos a fazer as queries em SQLite também nos deu formas de exprimirmos e materializarmos interrogações que poderíamos querer fazer a uma base de dados. Na "bigger picture", aprendemos a pesar num conjunto de informação e a tornar possível a extração de dados com relacionados de modo elaborado.

Também queremos salientar que a qualidade da estrutura dos dados teve um impacto na dificuldade do trabalho, uma vez que erros na recolha de informação para o conjunto de dados ou na estruturação do mesmo tornavam os modelos que criámos para as bases de dados inconsistentes. Graças a isto, tivemos de fazer alterações tanto aos modelos da base de dados como à informação que fomos inserindo na base de dados a partir do conjunto de dados.

Finalmente, sentimos bastante satisfação sempre que conseguimos ultrapassar os desafios apresentados pelo dataset, tal que acabámos por formar uma aplicação que consideramos útil e de uso agradável! Desenvolvemos várias competências transversais à área da cadeira, às bases de dados, e portanto consideramos que este projeto nos foi muito útil para o desenvolvimento pessoal, interpessoal, académico e profissional.

Referências:

Front-End App Development:

- GeeksforGeeks. (n.d.). - . Retrieved from https://www.geeksforgeeks.org
- Reddit. (n.d.). - . Retrieved from https://www.reddit.com
- Stack Overflow. (n.d.). -. Retrieved from https://stackoverflow.com
- OpenAI. (2024). ChatGPT.

SQL and Documentation:

- Datacamp. (n.d.). *How to use ROW_NUMBER in SQL*. Retrieved from https://www.datacamp.com
- Class slides. (n.d.). *SQL and database documentation*. Internal material from UC: Bases de Dados | moodle2425

SQL and Data validation:

- OpenAI. (2024). ChatGPT .
- SQLite. (n.d.). SQLite documentation. Retrieved from https://www.sqlite.org
- Ferreira, P. G. D. (n.d.). *Data science notebook*. Internal material.

SQL, UML, Relational Models, Data Entry, and testing:

- SQLite Tutorial. (n.d.). *Learn SQLite step by step*. Retrieved from https://www.sqlitetutorial.net
- Class slides. (n.d.). *UML Diagrama de Classes, Tradução UML-Relacional*. Internal material from UC: Bases de Dados | moodle2425
- UML Diagrams. (n.d.). - . Retrieved from https://www.uml-diagrams.org/data-type.html

SQLite Research:

- SQLite Tutorial. (n.d.). *Learn SQLite step by step*. Retrieved from https://www.sqlitetutorial.net
- SQLite. (n.d.). *Datatypes documentation*. Retrieved from https://sqlite.org/datatype3.html
- Stack Overflow. (n.d.). *Using ROW_NUMBER in SQLite*. Retrieved from https://stackoverflow.com/questions/16847574/how-to-use-row-number-in-sqlite

Additional research for data interpretation:

- National Geographic. (n.d.). *Antarctica*. Retrieved from https://education.nationalgeographic.org/resource/antarctica
- SQLite. (n.d.). *Maximum string length*. Retrieved from https://sqlite.org/limits.html#max_length
- Atlas Obscura. (n.d.). *The longest place name*. Retrieved from https://www.atlasobscura.com
- Largest.org. (n.d.). *Longest city names*. Retrieved from https://largest.org/geography/city-names/
- Wikipedia. (n.d.). *List of long place names*. Retrieved from https://en.wikipedia.org/wiki/List_of_long_place_names
- WorldAtlas. (n.d.). *Longest country names*. Retrieved from https://www.worldatlas.com
- World Population Review. (n.d.). *Longest country names*. Retrieved from https://worldpopulationreview.com
- Inshorts. (n.d.). *Countries without a capital*. Retrieved from https://inshorts.com
- World Population Review. (n.d.). *Countries with states*. Retrieved from https://worldpopulationreview.com
- WorldAtlas. (n.d.). *How many countries are there in the world?*. Retrieved from https://www.worldatlas.com
- Worldometers. (n.d.). *How many countries are there in the world?*. Retrieved from https://www.worldometers.info
- Wikipedia. (n.d.). *List of sovereign states*. Retrieved from https://en.wikipedia.org/wiki/List_of_sovereign_states