## Estrutura do relatório:

1. Delineamento do projeto;
2. Fases do projeto;
3. Descrição de cada fase;
4. Conclusão;
5. Melhoramentos possíveis;

## Delineamento do projeto:

O principal objetivo deste projeto seria criar uma base de dados que apresentasse um largo conjunto de informação relativamente a um determinado número de genes associados a uma query de pesquisa escolhida pelo utilizador.

Partindo do princípio que o objetivo é concluído, a realização desta base de dados poderia de certa forma otimizar diversos aspetos de pesquisa de genes, tais como, pesquisar uma determinada percentagem de um nucleótido, ou genes que estejam associados a artigos onde os autores têm como affiliation a Universidade do Minho, ou, a utilização de informação associada a outra base de dados como a Uniprot em simbiose com a informação descrita no NCBI, etc. Em suma, a nossa base de dados irá permitir retirar informação de uma forma mais facil quando comparada ao NCBI.

## Fases do projeto:

1. Tendo em conta toda a possível informação referente a cada query, realizar o delineamento da Base de dados com aplicação de possíveis entidades, atributos, relacionamentos e respetiva cardinalidade.
2. Análise e construção de código para extração da informação pretendida indo ao encontro da Base de dados criada anteriormente.
3. Análise e construção de código para a povoação das tabelas criadas na fase 1 com a informação obtida na fase 2

## Descrição de cada fase:

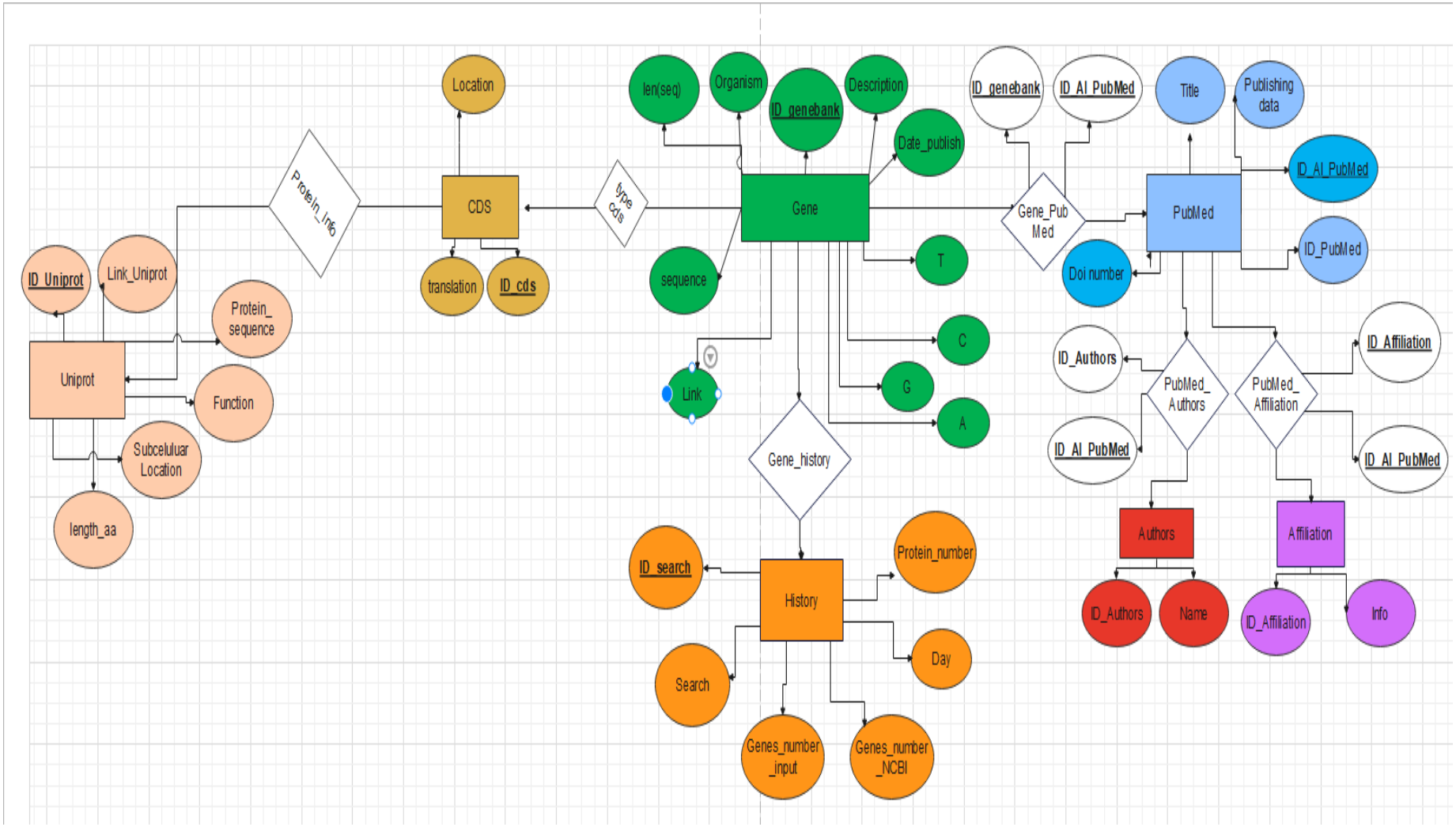
1.

Segundo a informação pretendida, realizou-se a organização desta em entidades, representadas por diversos atributos com diferentes tipos de relacionamentos entre si., como é demonstrado nas tabelas abaixo. Tabelas que serviram de orientação para a realização das restantes fases do trabalho.

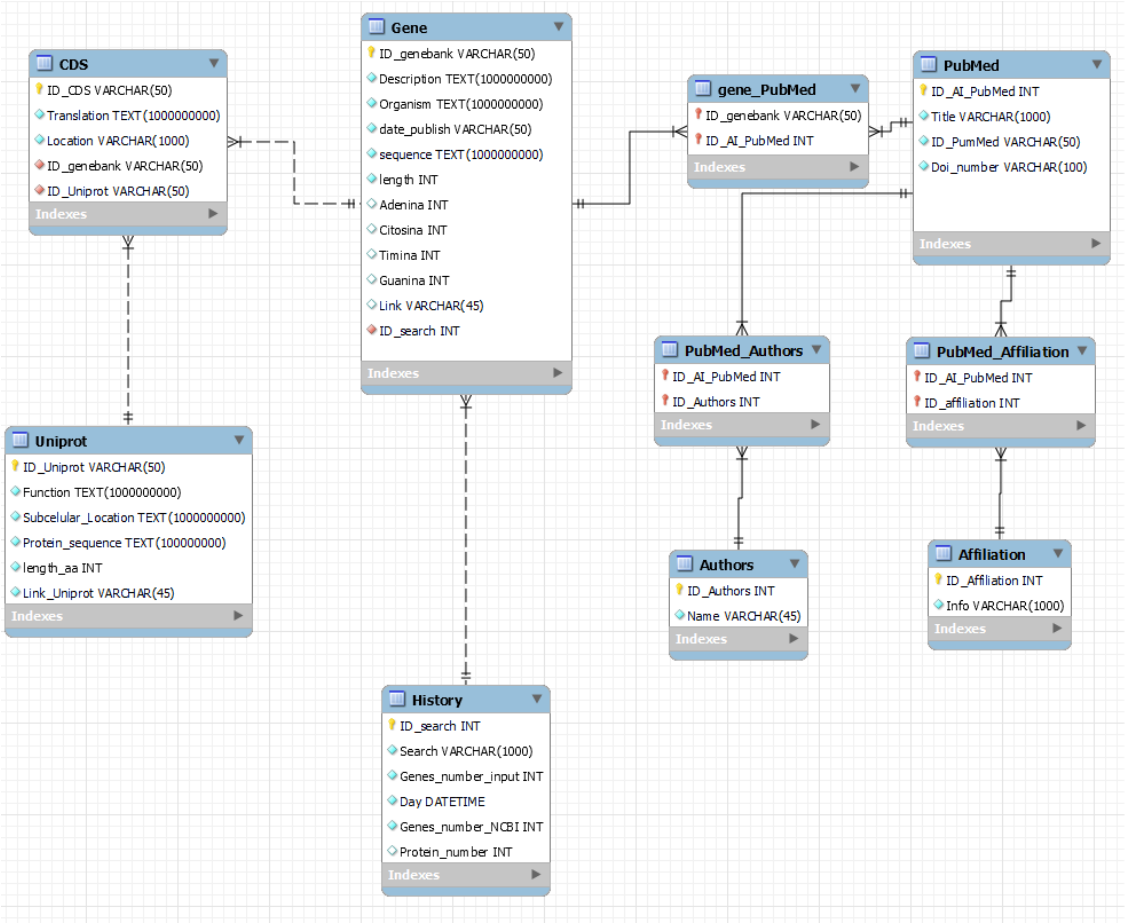
| Identity | Name | Tipo e dominio | Multivalue | Zero | Keys |
|---|---|---|---|---|---|
| Gene | **ID_genbank** | VARCHAR | N | N | S- |
| | Description | TEXT | N | N | N |
| | Organism | TEXT | N | N | N |
| | Date_publish | VARCHAR | N | N | N |
| | sequence | Text | N | N | N |
| | length | INT | N | N | N |
| | Adenina | INT | N | S | N |
| | Citosina | INT | N | S | N |
| | Guanina | INT | N | S | N |
| | Timina | INT | N | S | N |
| | Link | VARCHAR | N | S | N |
| PubMed | **ID_AI_PubMed** | INT | N | N | S |
| | Title | VARCHAR | N | N | N |
| | ID_PubMed | VARCHAR | N | N | N |
| | Doi_number | VARCHAR | N | N | N |
| Authors | **ID_Authors** | INT | N | N | S |
| | Name | VARCHAR | N | N | N |
| Affiliation | **ID_Affiliation** | INT | N | N | S |
| | Info | VARCHAR | N | N | N |
| History | **ID_search** | INT | N | N | S |
| | Search | VARCHAR | N | N | N |
| | Genes_number_input | INT | N | N | N |
| | Day | DATETIME | N | N | N |
| | Genes_number_NCBI | INT | N | N | N |
| | Protein_number | INT | N | N | N |
| CDS | **ID_CDS** | VARCHAR | N | N | S |
| | Translation | TEXT | N | N | N |
| | Location | INT | N | N | N |
| Uniprot | **ID_Uniprot** | VARCHAR | N | N | S |
| | Function | TEXT | N | N | N |
| | Subcellular_Location | TEXT | N | N | N |
| | Protein sequence | TEXT | N | N | N |
| | Length (a.a) | INT | N | N | N |
| | Link_Uniprot | VARCHAR | N | N | N |

| Identity | # | Relationship | Identity | # |
|---|---|---|---|---|
| Gene | 1…N | Gene_PubMed | PubMed | 1…N |
| | 1…1 | Type_CDS | CDS | 1…N |
| | 1…N | Gene_History | History | 1…1 |
| PubMed | 1…N | PubMed_Authors | Authors | 1…N |
| | 1…N | PubMed_Affiliation | Affiliation | 1…N |
| CDS | 1…N | Protein_info | Uniprot | 1…1 |

As tabelas apresentadas serviram como guia para a realização do modelo conceptual no software *Wondershare EdrawMax*.


Location
len(seq)
Organism
ID_genebank
Description
Date_publish
ID_genebank
ID_Al PubMed
Title
Publishing data
Protein_Info
CDS
type cds
Gene
Gene_Pub Med
PubMed
ID_Al_PubMed
ID_Uniprot
Link_Uniprot
translation
ID_cds
sequence
T
Doi number
ID_PubMed
Protein_ sequence
C
ID_Authors
ID_Affiliation
Uniprot
Function
Link
G
PubMed_ Authors
PubMed_ Affiliation
A
ID_Al_PubMed
Subceluluar Location
Gene_history
ID_Al_PubMed
length_aa
Authors
Affiliation
ID_search
Protein_number
History
ID_Authors
Name
ID_Affiliation
Info
Search
Day
Genes_number _input
Genes_number _NCBI

Após o desenho do modelo conceptual e, tendo em particular atenção a cardinalidade entre cada entidade/relacionamento, realizou-se o modelo lógico utilizando o software *MySQL Workbench.*




Após a realização do modelo lógico, seguiu-se para o modelo físico usando o mesmo software referido anteriormente.

## 2.

Em primeiro lugar, criou-se um código com o prepósito de através de uma função (*url get*), obter os links associados ao NCBI a partir de uma determinada query e, destes links com auxílio de expressões regulares extrair os respetivos IDS genebank. Referir, que criamos a possibilidade ao utilizador de escolher um determinado número de IDs tendo sempre como limite máximo o valor 20, o que está associado à própria paginação do NCBI. Como os IDs poderiam aparecer repetidos a nossa estratégia passou por criar uma lista(n_genes) onde os ids apenas apareciam uma única vez, sem repetições.

```
#Extrair ids genebank:
data_e_hora_atuais = datetime.now()

query= input('escolha o que quer pesquisar: ')


try:
    def url_get(i):
        url_list_id=[]
        url = f"https://www.ncbi.nlm.nih.gov/gene/?term={i}"
        url_list_id.append(url)
        return url_list_id

    url_get(query)

    content = []
    for url in url_get(query):
        r = requests.get(url)
        content.append(r.content)

    for c in content:
        soup = BeautifulSoup(c, 'html.parser')
        a= soup.get_text()

    existe = re.findall(r"ID:\s+\d*(?=\D)", a, re.DOTALL)

    c= ', '.join(existe)
    h= c.replace('ID: ','')
    IDS= h.split(', ')

    n_gene= IDS[0:1+(int(input('escolha o nº de genes que quer obter (máximo 20): ')))]

    n_genes = []
    seen = set()
    for item in n_gene:
        if item not in seen:
            seen.add(item)
            n_genes.append(item)

    numero_genes= len(n_genes)

    print(n_genes)
    print (url_get(query))

    if n_genes == ['']:
        print()
        print('0 resultados para a sua pesquisa, pesquise de novo.')

except:
    ('0 resultados para a sua pesquisa, pesquise de novo.')
```

.

Dos IDs obtidos por *web scraping*, alguns já foram removidos do genebank e, como tal, a sua associação dava erro. Desta forma, a solução encontrada foi o recurso ao biopython, nomeadamente o modulo Entrez, para extrair apenas os que não foram removidos.

```
#Extrair ids ncbi:
Ids=[]
database = 'nucleotide'
email= 'rodrigoce9@gmail.com'
idlist= n_genes
handle = Entrez.efetch(db=database, id=idlist, rettype="gb")
records = list(SeqIO.parse(handle,"gb"))
handle.close()
for info in records:
    Ids.append(info.id)
```

Tendo em atenção que um dos atributos pretendidos para a entidade Gene é o Link_Gene fizemos uma associação entre este atributo (variavel no código) e os links obtidos com a utilização de uma query.

A partir da estratégia referida anteriormente e com principal recurso a Biopython foi possível obter descrições, organismos, sequência, tamanho de sequência, percentagem dos nucleótidos, data de publicação do gene. Posto isto, a informação necessária para preencher os atributos da entidade “Gene” foi extraída.

```
#extrair links ncbi
def url_get_id(i):
    url_list= [ ]
    for id in i:
        url = "https://www.ncbi.nlm.nih.gov/nuccore/{}".format( id )
        url_list.append(url)
    return url_list
link_genebank= url_get_id(Ids)
print(link_genebank)

#Extrair description NCBI:
description=[]
database = 'nucleotide'
email= 'rodrigoce9@gmail.com'
idlist= n_genes
handle = Entrez.efetch(db=database, id=idlist, rettype="gb")
records = list(SeqIO.parse(handle,"gb"))
handle.close()
for info in records:
    description.append(info.description)

#organismos
Organismos=[]
database = 'nucleotide'
email= 'rodrigoce9@gmail.com'
idlist= n_genes
handle = Entrez.efetch(db=database, id=idlist, rettype="gb")
records = list(SeqIO.parse(handle,"gb"))
handle.close()
for info in records:
    Organismos.append(info.annotations['organism'])
```

O próximo aspeto focou-se na obtenção da informação relativa à entidade PubMed, onde através dos links dos IDs já "processados" referentes ao NCBI extrair o seu conteúdo utilizando o módulo *Beautifulsoup* do *bs4*. Todavia, para ser mais fácil trabalhar com este conteúdo, criamos uma n lista(listas) onde o output estava em formato utf-8.

```
# buscar info pubmed

def url_get_id(i):
    url_list= [ ]
    for id in i:
        url = "https://www.ncbi.nlm.nih.gov/nuccore/{}".format( id )
        url_list.append(url)
    return url_list
link_genebank= url_get_id(Ids)
print(link_genebank)

content_id = []
for url in url_get_id(Ids):
    r_id = requests.get(url)
    content_id.append(r_id.content)

from bs4 import BeautifulSoup
listas=[]

for c in content_id:
    soup_id = BeautifulSoup(c, 'html.parser')

    lines = soup_id.find_all('meta', {'name':"ncbi_uidlist"} )

    id = ""
    url = ""
    for line in lines:
        if 'content' in line.attrs:
            id = line.attrs['content']
    if id:
        url = "https://www.ncbi.nlm.nih.gov/sviewer/viewer.fcgi?id={}&db
    r2 = requests.get(url)
    r4= r2.content.decode('utf-8')
    listas.append(r4)

cc= ', '.join(listas)
er= cc.replace('//','')
final= er.split(', ')
```

De seguida, a estratégia utilizada passa pela utilização de expressões regulares extrair da lista referida anteriormente os IDs do NCBI e os Ids da PubMed. A partir daqui, criou-se um dicionário onde os IDs dos genes seriam as keys, e os IDs da PubMed (referentes a cada gene) os valores. No caso do IDs do gene não ter nenhum ID PubMed associado, o valor desde seria uma lista vazia.

Posto isto, criaram-se duas listas diferentes, uma lista para armazenar os IDs dos genes(id_ncbii) e outra lista para armazenar os IDs da pubmed(ID_PUB), no caso de não existir IDs de PubMed associado, será adicionado à lista a string "N/A". Como existe a

possibilidade de termos IDs PubMed repetidos, utilizamos um código para criar uma lista(*new_list_)* só com ids não repetidos.

```
#criar dicionário de ids ncbi e ids pubmed
output_dict = {}
for x in final:
    version = re.search(r'VERSION\s+(.*?)\s', x)
    pubmed = re.search(r'PUBMED\s+(.*?)\s', x)
    if version:
        versionf=version.group(1)
        output_dict.setdefault(version.group(1), [])
    if pubmed:
        output_dict[versionf].append(pubmed.group(1))

#Criar Listas só com ids ncbi e ids pubmed
id_ncbii = []
ID_PUB = []
for key, vals in output_dict.items():
    if vals:
        for val in vals:
            id_ncbii.append(key)
            ID_PUB.append(val)
    else:
        id_ncbii.append(key)
        ID_PUB.append("N/A")

new_list_ = []
seen = set()
for item in ID_PUB:
    if item not in seen:
        seen.add(item)
        new_list_.append(item)
```

Para extrair informação relativamente aos artigos na PubMed, usamos o modulo Entrez de biopython, este módulo permite através dos IDs da PubMed contidos na lista (*new_list_)* extrair informações como os títulos, os autores, a fonte, *affiliation* e o *doi.*

```
#Extrair informação de artigos

titles=[]
authors=[]
source=[]
affiliation=[]
database = 'PubMed'
email= 'rodrigoce9@gmail.com'
idlist= new_list_
counter = 0
for i in idlist:
    if i!= "N/A":
        handle = Entrez.efetch(db=database, id=i, rettype="medline", retmode="text")
        records = Medline.parse(handle)
        for info in records:
            titles.append(info.get("TI", ["N/A"]))
            authors_string = info.get("AU", ["N/A"])
            if len(authors_string) > 5:
                authors_h = authors_string[0:5]
                authors.append(authors_h)
            else:
                authors.append(authors_string)
            source.append(info.get("SO", ["N/A"]))
            affiliation_string= info.get("AD", ["N/A"])
            if len(affiliation_string) > 5:
                affiliation_h = affiliation_string[0:5]
                affiliation.append(affiliation_h)
            else:
                affiliation.append(affiliation_string)
            counter += 1
    else:
        titles.append(["N/A"])
        authors.append(["N/A"])
        source.append(["N/A"])
        affiliation.append(["N/A"])
        counter += 1
```

O *doi* estava contido na informação proveniente da lista *source*, e para ser extraído apenas o doi, utilizou-se expressões regulares, no caso de não existir nenhum *doi*, adicionou-se a *string* N/A. Posto isto, a informação necessária para preencher os atributos referentes à entidade “PubMed” foi recolhida.

```
#agrupar titles
titles = [ [title] for title in titles]

# agrupar dois
doi_list = []
for x in source:
    match = re.search("doi: (.*)", str(x))
    if match:
        doi_list.append(match.group(1))
    else:
        doi_list.append("N/A")
```

No que concerne à estratégia para a entidade autores e respetivos atributos realizou-se um dicionário que através dos autores extraídos com recurso ao *biopython*, tem como chaves, os IDs PubMed e como valor o nome dos autores. Referenciar, que como alguns artigos possuíam muitos autores, criamos uma lista onde continha apenas os 5 primeiros autores extraídos de cada artigo. A partir deste ponto, criaram-se duas listas diferentes, uma lista para armazenar os IDs da PubMed(*pubmed_list*) e outra lista para armazenar os nomes dos autores(*authors_list*). Como existe a possibilidade de termos nomes de autores repetidos, utilizamos um código para criar uma lista(*new_list_authors*) só com nomes de autores não repetidos. Assim, a informação necessária para preencher os atributos referentes à tabela "*Authors*" está totalmente recolhida.

```
#agrupar authors
id_authors_dict = {i: authors[counter] if i != "N/A" else ["N/A"] for counter, i in enumerate(idlist)}

pubmed_list = []
authors_list = []
for key, vals in id_authors_dict.items():
    if vals:
        for val in vals:
            pubmed_list.append(key)
            authors_list.append(val)

new_list_authors= []
seen = set()
for item in authors_list:
    if item not in seen:
        seen.add(item)
        new_list_authors.append(item)
```

Em relação à entidade "*affiliation*" raciocínio utilizado é idêntico ao descrito anteriormente para a entidade "*Authors*", com a particularidade de que caso um id pubmed não tenha associado nenhuma affiliation, será acrescentado à *lista affi_list* a string N/A. Desta forma, a informação para esta entidade, está recolhida.

```
id_affiliation_dict = {i: [single_affiliation_list[counter]] if i != "N/A" else ["N/A"] for counter, i in enumerate(idlist)}

pubmed_affi_list = []
affi_list = []
for key, vals in id_affiliation_dict.items():
    if vals:
        for val in vals:
            pubmed_affi_list.append(key)
            affi_list.append(val)
    else:
        pubmed_affi_list.append(key)
        affi_list.append(["N/A"])

new_list_affi= []
seen = set()
for item in single_affiliation_list :
    if item not in seen:
        seen.add(item)
        new_list_affi.append(item)
```

Relativamente à entidade CDS, a estratégia principal baseia-se na utilização de biopython. A função get_*CDS*_info, cria uma lista de listas em que cada uma é composta por: ID_CDS, Location e Translation (atributos pretendidos). Tendo em atenção, que existe uma relacionamento especifico entre esta entidade e a entidade gene, a estratégia utilizada foi a antecipada criação de um dicionário(*result_dict*) que permite associar cada atributo desejado a um determinado *ID_genebank* .

```
from collections import OrderedDict
def get_CDS_info(result_dict): #gets info for CDS [ID_CDS, location, Translation]
    database = 'nucleotide'
    email= 'rodrigoce9@gmail.com'
    Entrez.email = email
    cds_location_list = []
    several_location = []
    record_types={}
    processed_i = set()
    for i, value in result_dict.items():
        if i in processed_i:
            continue
        if value == 'N/A_CDS':
            cds_location = ['N/A_CDS', 'N/A', 'N/A']
            cds_location_list.append(cds_location)
        else:
            handle = Entrez.efetch(db=database, id=i, rettype="gb")
            records = list(SeqIO.parse(handle,"gb"))
            handle.close()
            for info in records:
                for i in info.features:
                    if i.type == "CDS":
                        cds_location = []
                        i_d = str(i.qualifiers["protein_id"])
                        translation =  str(i.qualifiers["translation"])
                        cds_location.append(i_d)
                        if isinstance(i.location, CompoundLocation):
                            for sub_location in i.location.parts:
                                several_location.append("[{} : {}]".format(sub_location.start, sub_location.end))
                            cds_location.append(several_location)
                            cds_location.append(translation)
                        else:
                            cds_location.append("[{} : {}]".format(i.location.start, i.location.end))
                            cds_location.append(translation)
                        cds_location_list.append(cds_location)
                processed_i.add(i)
            handle.close()
    return cds_location_list
```

Para a entidade Uniprot o primeiro passo, foi a obtenção dos links associados a uma determinada query de pesquisa,com recurso à mesma estratégia utilizada anteriormente para obtenção dos links ncbi, com a particularidade da criação de uma nova função *url_get_id_p*. Desta forma,  o atributo-*Link_Uniprot* foi conseguido.

Recorremos à *webAPI* do Uniprot e atribuindo um field a cada atributo pretendido para esta entidade, conseguiu-se adquiri toda a informação desejada. Todavia esta informação aparecia  de uma forma pouca organizada e , como tal, com recurso a expressões regulares retirou-se a informação pretendida com a apresentação desejada e correto conteúdo. De salientar, que cada um destes atributos da  entidade Uniprot está associado a um determinado *ID_CDS*.

```
def get_field_for_id(ID_PROT, field):
    response = get_url("{}/uniprotkb/search?query={}&fields={}&size=1&format=tsv".format(WEBSITE_API,ID_PROT,field))
    return str(response.content)

try:
    def get_list_uniprot(ID_PROT, result_dict):
        results = []
        result = []
        tmp= []
        easy=dict_to_list(result_dict)
        for first_index, first_value in easy:
            tmp= []
            if first_value != 'N/A_CDS':
                for field in fields:
                    result = get_field_for_id(first_value, field)
                    tmp.append(result)
            else:
                result = ['N/A_Uniprot']
                tmp.append(result)
            results.append(tmp)
        uniprot_final_list=[]
        for index, values in enumerate(results):
            uniprot_list=[]
            n_a = re.search(r'(N/A_Uniprot)', str(values), re.DOTALL)
            if len (values)==1:
                if n_a:
                    uniprot_list.append(n_a.group(1))
                    tmp_list='N/A','N/A','N/A','N/A'
                    uniprot_list.extend(tmp_list)
            else:
                for i in values:
                    entry = re.search(r'b\'Entry\\n(.+?(?=\\n\'))', str(i), re.DOTALL)
                    function = re.match( r'b\'Function \[CC\]\\n.{9} (.+?(?=\\n\'))', str(i), re.DOTALL )
                    location_exist = re.search( r'b\'Subcellular location \[CC\]\\nSUBCELLULAR LOCATION: (.+?(?=\\n\'))', str(i),
                    location_notexist = re.search( r'b\'Subcellular location \[CC\]\\n\\n\'',str(i), re.DOTALL )
                    sequence = re.search(r'b\'Sequence\\n(.+?(?=\\n\'))', str(i), re.DOTALL)
                    if entry:
                        ent=entry.group(1)
                        uniprot_list.append(entry.group(1))
                    if function:
                        uniprot_list.append(function.group(1))
                    if location_exist:
                        uniprot_list.append(location_exist.group(1))
                    if location_notexist:
                        uniprot_list.append("N/A")
                    if sequence:
                        uniprot_list.append(sequence.group(1))
                        uniprot_list.append(len(sequence.group(1)))
                        uniprot_list.append(url_get_id_p(ent))

            if len(uniprot_list) < 6: #sometimes, there is no function associated
                uniprot_list.insert(2, 'N/A')
            uniprot_final_list.append(uniprot_list)

        return uniprot_final_list
    get_Uniprot=get_list_uniprot(ID_PROT,result_dict)
except:
    print("A informação foi apagada ")
```

Para a entidade *History*, fez-se o import do módulo *datetime* para extrair a hora e data do momento em que se fez a pesquisa, ou seja que se atribui uma query (atributo search). Associado a isso elaborou-se uma função que conta o número de proteínas, o número de IDs NCBI e o numero de IDs requeridos pelo o utilizador completando todos os atributos associados à entidade em questão.

```
query= input('escolha o que quer pesquisar: ')
data_e_hora_atuais = datetime.now()
```

```
def count_(genes):
    return len(genes)
count_(Ids)
count_(ID_PROT)
```

# 3.

Para povoar os atributos das entidades, é necessário fazer a conexão com a base de dados criada anteriormente (*SQLC.connect*), depois é necessário inserir a "ação" que pretendemos executar (*Insert*) indicando a entidade onde pretendemos executar essa ação selecionando os atributos que vão ser alterados e indicando os dados (*Values*) que se pretende adicionar. Evidenciar que os *values* referidos foram determinados na 2 fase deste projeto.

Em relação à entidade *History*, definiu-se que a chave primária (ID_*search*) seria autoincrementada e, por isso, a "ação" explicada anteriormente para associar um valor a este atributo primário é ligeiramente diferente. Além disso, como a entidade em questão está relacionada (1 para N) com a entidade *Gene* foi necessário extrair os IDs autoincrementados, sendo que, apenas interessa o último *ID_search* obtido pois, como é o mais atual, é esse que vai estar relacionado com a entidade *Gene* que se vai povoar de seguida.

```
#Povoação "History"

DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    sql= "INSERT INTO History (search, Genes_number_input, Day, Genes_number_NCBI, Protein_number  )   VALUES (%s, %s, %s, %s, %s
    val=(query, numero_genes, data_e_hora_atuais, count_(Ids), count_(ID_PROT) )
    Cursor.execute(sql,val)

except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()

search_id =[]
DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    sql= "Select ID_search FROM History"
    Cursor.execute(sql)
    for row in Cursor:
        search_id.append(str(row))

except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()

div= ', '.join(search_id)
h= div.replace("(",'')
hh= h.replace(",)",'')
SEARCH_ID= hh.split(', ')
Hist= SEARCH_ID[-1]
```

De modo a comprovar o correto funcionamento do código, utilizou-se uma query de pesquisa escolhida aleatoriamente (Diabetes) e obteve-se a povoação da tabela History.

| ID_search | Search | Genes_number_input | Day | Genes_number_NCBI | Protein_number |
|---|---|---|---|---|---|
| 12 | diabetes | 15 | 2023-01-18 19:23:03 | 8 | 4 |

No que concerne à povoação da entidade *Gene* o processo é ligeiramente distinto, uma vez que, como estamos a povoar vário genes ao mesmo tempo, foi necessário usar um *ciclo for*, com recurso ao *enumerate*. Esta estratégia, permite trabalhar com todas as listas criadas na fase 2, separadamente, e a partir daqui associar a cada atributo necessário para povoar esta tabela.

```
#Povoação "Gene"-

des = (description)
DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    for index, value in enumerate(Ids):
        #print(value)
        #print(des[index])
        sql= "INSERT INTO Gene (ID_genebank, Description, Organism, sequence, Date_publish, length, Adenina, Citosina, Guanina, T
        val=(value, des[index], Organismos[index], seqss[index], dates[index], tam[index], Adenina[index], Citosina[index], Guani
#         print(type(des[index]))
        Cursor.execute(sql,val)
#         for values in des:
#             Cursor.execute(f"INSERT INTO Gene ( Description)  VALUES ('{values}')")
except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()
```

O resultado obtido para a povoação da entidade em questão com a query já referida está demonstrado na seguinte tabela, constituída pelos respetivos atributos.

| | ID_genebank | Description | Organism | date_publish |
|---|---|---|---|---|
| ▶ | M90830.1 | Xerocomus chrysenteron 18S ribosomal RNA ge... | Xerocomellus chrysenteron | 09-APR-2001 |
| | M94340.1 | Xerocomus chrysenteron 18S ribosomal RNA ge... | Xerocomellus chrysenteron | 09-APR-2001 |
| | V00883.1 | Rabbit (O. cuniculus) germ line gene coding for ... | Oryctolagus cuniculus | 14-NOV-2006 |
| | V01310.1 | Yeast (S. cerevisiae) gene HIS4 (histidine meta... | Saccharomyces cerevisiae | 18-APR-2005 |
| | X61243.1 | C.elegans repetitive DNA | Caenorhabditis elegans | 30-SEP-2005 |
| | X67017.1 | S.cerevisiae cyt2 gene for cytochrome c heme l... | Saccharomyces cerevisiae | 13-JUN-2006 |
| | Z15032.1 | S.cerevisiae gene for S.pombe cdc21+ homolog... | Saccharomyces cerevisiae | 25-JUL-2016 |
| | Z17674.1 | ATTS0248 Gif-SeedA+B Arabidopsis thaliana cD... | Arabidopsis thaliana | 10-NOV-1992 |

| sequence | length | Adenina | Citosina | Timina | Guanina | Link | ID_search |
|---|---|---|---|---|---|---|---|
| GCAAGTCTGG...TTGACGGAAG | 604 | 25 | 18 | 26 | 28 | https://www.ncbi.nlm.nih.gov/nuccore/M90830.1 | 12 |
| AAAGATTAAG...AACCTGCGGA | 1759 | 25 | 20 | 26 | 27 | https://www.ncbi.nlm.nih.gov/nuccore/M94340.1 | 12 |
| AAGCTTCTCT...TGTGGAATTC | 1905 | 28 | 19 | 27 | 24 | https://www.ncbi.nlm.nih.gov/nuccore/V00883.1 | 12 |
| CTCGAGAAGA...TAGGAAAGAA | 4751 | 32 | 19 | 26 | 21 | https://www.ncbi.nlm.nih.gov/nuccore/V01310.1 | 12 |
| TTTTTTTACT...CTAACAATTT | 578 | 32 | 22 | 31 | 14 | https://www.ncbi.nlm.nih.gov/nuccore/X61243.1 | 12 |
| CCGGCCTTCT...TTTCCACCGG | 1416 | 30 | 23 | 29 | 17 | https://www.ncbi.nlm.nih.gov/nuccore/X67017.1 | 12 |
| TCTACTTCCA...CGTCTTGCAC | 228 | 26 | 16 | 34 | 22 | https://www.ncbi.nlm.nih.gov/nuccore/Z15032.1 | 12 |
| GAGCAATAAT...ATCTTTTACC | 720 | 24 | 22 | 30 | 21 | https://www.ncbi.nlm.nih.gov/nuccore/Z17674.1 | 12 |

Para a entidade *PubMed*, foi utilizado o mesmo raciocínio que na entidade Gene, notando que *a primary key*, é um ID autoincrementado (*ID_AI_PubMed*).

```
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    for index, value in enumerate(new_list_):
        #print(value)
        #print(des[index])
        sql= "INSERT INTO PubMed (ID_PumMed, title, Doi_number) VALUES (%s, %s, %s)"
        val=(str(new_list_[index]), str(titles[index]), str(doi_list[index]) )
#         print(type(des[index]))
        Cursor.execute(sql,val)
#         for values in des:
#             Cursor.execute(f"INSERT INTO Gene ( Description)  VALUES ('{values}')")
except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()

#retirar os valores AI de pubmed
ID_AI=[]
DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    sql= "Select ID_AI_PubMed FROM PubMed"
    Cursor.execute(sql)
    for row in Cursor:
        #print(row)
        ID_AI.append(str(row))
except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()


div_= ', '.join(ID_AI)
h_= div_.replace("(",'')
hh_= h_.replace(",)",'')
SEARCH_ID_= hh_.split(', ')
```

A povoação resultante está demonstrada na tabela abaixo. Referir que há determinados genes que podem não estar associados a artigos *PubMed* e portanto aparece a *string* N/A.

| | ID_AI_PubMed | Title | ID_PumMed | Doi_number |
|---|---|---|---|---|
| ▶ | 89 | [['N/A']] | N/A | N/A |
| | 90 | ['The nucleotide sequence of rabbit embryonic g... | 6271761 | N/A |
| | 91 | ['Molecular cloning and characterization of the S... | 1499554 | 10.1111/j.1432-1033.1992.tb17146.x. |
| | 92 | ['Molecular and genomic organization of clusters... | 1619649 | 10.1016/0022-2836(92)90131-3. |
| | 93 | ['Rate and mode differences between nuclear a... | 1382179 | 10.1093/oxfordjournals.molbev.a040760. |
| | 94 | ['Fission yeast cdc21+ belongs to a family of pr... | 1454522 | 10.1093/nar/20.21.5571. |
| | 95 | ['The nucleotide sequence of the HIS4 region of... | 7049842 | 10.1016/0378-1119(82)90055-5. |

É necessário ter em atenção que, como a entidade Gene está relacionada com a entidade *Pubmed* com cardinalidade de n para n, isto é, um gene pode ter vários artigos associados e vice-versa, foi necessário criar uma lista (*SEARCH_ID*) com todos os IDs auto

incrementados. A partir daqui foi possível saber o número do primeiro ID autoincrementado para depois a cada ID PubMed presentes na lista *ID_PUB*, pela ordem certa, atribuir o mesmo ID autoincrementado e, assim povoar o relacionamento *gene_pubmed*.

```
#Povação "Gene-Pubmed"
number_map = {}
next_number = int(SEARCH_ID_[0])
relation = []
for number in ID_PUB:
    if number not in number_map:
        number_map[number] = next_number
        next_number += 1
    relation.append(number_map[number])


DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:
    for index, value in enumerate(id_ncbii):

        sql= "INSERT INTO gene_PubMed (ID_genebank, ID_AI_PubMed) VALUES (%s, %s)"
        val=(id_ncbii[index], relation[index])

        Cursor.execute(sql,val)

except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()
```

Na tabela seguinta, mostra a correta associação entre um ID_genebank e um ID_AI_PubMed, ou seja, uma correta povoação do relecionamente gene_PubMed.

| | ID_genebank | ID_AI_PubMed |
|---|---|---|
| ▶ | M90830.1 | 93 |
| | M94340.1 | 89 |
| | V00883.1 | 90 |
| | V01310.1 | 95 |
| | X61243.1 | 92 |
| | X67017.1 | 91 |
| | Z15032.1 | 94 |
| | Z17674.1 | 89 |
| * | NULL | NULL |

Para a povoação da entidade *Authors* e *Afilliation* e, respetivos relacionamentos com a entidade PubMed de n para n, foi utilizado o mesmo raciocínio descrito anteriormente, alterando claro, apenas os respetivos atributos.

Os resultados obtidos para a povoação da entidade *authors* e *affiliation* estão representados nas 2 primeiras tabelas seguintes. Sendo as restantes, correspondentes ao relacionado gene_PubMed com Authors/ Affiliation.

| ID_Affiliation | Info |
|---|---|
| 67 | N/A |
| 68 | Institut fur Genetik und Mikrobiologie, Universit... |
| 69 | CNR International Institute of Genetics and Bio... |
| 70 | Department of Plant Pathology, University of C... |
| 71 | Department of Zoology, University of Oxford, UK. |
| NULL | NULL |

| ID_AI_PubMed | ID_affiliation |
|---|---|
| 89 | 67 |
| 90 | 67 |
| 95 | 67 |
| 91 | 68 |
| 92 | 69 |
| 93 | 70 |
| 94 | 71 |
| NULL | NULL |

| ID_Authors | Name |
|---|---|
| 274 | N/A |
| 275 | Hardison RC |
| 276 | Zollner A |
| 277 | Rodel G |
| 278 | Haid A |
| 279 | Naclerio G |
| 280 | Cangiano G |
| 281 | Coulson A |
| 282 | Levitt A |
| 283 | Ruvolo V |
| 284 | Bruns TD |
| 285 | Szaro TM |
| 286 | Coxon A |
| 287 | Maundrell K |
| 288 | Kearsey SE |
| 289 | Donahue TF |

| ID_AI_PubMed | ID_Authors |
|---|---|
| 89 | 274 |
| 90 | 275 |
| 91 | 276 |
| 91 | 277 |
| 91 | 278 |
| 92 | 279 |
| 92 | 280 |
| 92 | 281 |
| 92 | 282 |
| 92 | 283 |
| 93 | 284 |
| 93 | 285 |
| 94 | 286 |
| 94 | 287 |
| 94 | 288 |
| 95 | 289 |

A estratégia de povoação utilizada para a entidade Uniprot é bastante semelhante às restantes, todavia o caso particular é que poderiam existe genes que não tinham associado nenhum ID_Uniprot. Posto isso, e com o intuito de não ter a repetição no valor da chave primária (ter diversos ID_Uniprot = NA), a solução passou  por associar todos os genes que não têm referencia na Uniprot ao ID_Uniprot=N/A_Uniprot.

```
#Povoação "Uniprot"
# print(get_Uniprot) #(ID_Uniprot, Subcelular location, Function, Protein seq, length_aa)
DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done
a=["N/A_Uniprot"]
Cursor = DataBase.cursor()
try:
    sec = 'INSERT INTO Uniprot (ID_Uniprot, Subcelular_Location, Function, Protein_sequence, length_aa, Link_Uniprot) VALUES (%s,
    Cursor.execute(sec, (a[0],a[0],a[0],a[0],a[0],a[0]))
    for index, value in enumerate(get_Uniprot):
        if str(value[0]) == 'N/A_Uniprot':
            continue
        else:
            sql= "INSERT INTO Uniprot (ID_Uniprot, Subcelular_Location, Function, Protein_sequence, length_aa, Link_Uniprot) VALU
            val=(str(value[0]), str(value[1]), str(value[2]), str(value[3]), str(value[4]), str(value[5]))
            Cursor.execute(sql,val)

except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()
```

O resultado da povoação da entidade Uniprot está demonstrado abaixo, com a particularidade referida de poder existir um N/A_Uniprot que representa que um determinado gene não tem informação associada ao Uniprot.

| ID_Uniprot | Function | Subcelular_Location | Protein_sequence | length_aa | Link_Uniprot |
|---|---|---|---|---|---|
| N/A_Uniprot | N/A_Uniprot | N/A_Uniprot | N/A_Uniprot | N/A_Unipro | N/A_Uniprot |
| P00815 | N/A | N/A | MVLPILPLIDDLASWNSKKEYVSLVGQVLLDGSSLSN... | 799 | https://www.uniprot.org/uniprotkb/P00815/entr |
| P02099 | This protein functions as an embryonic globin, b... | N/A | MVHFTAEEKAAITSTWKLVDVEDAGAEALGRLLVVY... | 147 | https://www.uniprot.org/uniprotkb/P02099/entr |
| P30665 | N/A | Nucleus {ECO:0000250}. | MSQQSSSPTKEDNNSSSPVVPNPDSVPPQLSSPALF... | 933 | https://www.uniprot.org/uniprotkb/P30665/entr |
| Q00873 | Lyase that catalyzes the covalent linking of the ... | Mitochondrion inner membrane {ECO:0000269\|... | MMSSDQQGKCPVDEETKKLWLREHGNEAHPGATA... | 224 | https://www.uniprot.org/uniprotkb/Q00873/entr |
| NULL | NULL | NULL | NULL | NULL | NULL |

No caso da entidade CDS, como a chave primária é composta por ID_genebank e ID_CDS, os valores correspondentes podem aparecer repetidamente. Este aspeto, permite associar a designação N/A_CDS para diversos ID_genebank, o que significa que posso ter n genes associados a 1 ID_CDS=N/A_CDS.

```
#Povoação "CDS"

DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done
Cursor = DataBase.cursor()
try:
    for item in join_CDS:
        sql= "INSERT INTO CDS (ID_CDS, Translation, Location, ID_genebank, ID_Uniprot) VALUES (%s, %s, %s, %s, %s)"
        val=(str(item[1]),str(item[4]),str(item[3]),str(item[0]),str(item[2]))
        Cursor.execute(sql,val)
except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()
```

Como demonstrado na tabela seguinte o ID_CDS apareceu repetidamente com a designação N/A_CDS associada a diversos genes, o que mostra que a estratégia utilizada para povoar a entidade CDS correu como esperado.

| ID_CDS | Translation | Location | ID_genebank | ID_Uniprot |
|---|---|---|---|---|
| CAA24252.1 | ['MVHFTAEEKAAITSTWKLVDVEDAGAEALGRLLVV... | ['[223 : 316]', '[440 : 662]', '[1479 : 1608]'] | V00883.1 | P02099 |
| CAA24617.1 | ['MVLPILPLIDDLASWNSKKEYVSLVGQVLLDGSSLS... | [1331 : 3731] | V01310.1 | P00815 |
| CAA47407.1 | ['MMSSDQQGKCPVDEETKKLWLREHGNEAHPGAT... | [246 : 921] | X67017.1 | Q00873 |
| CAA78750.1 | ['STSKSQILQYVHKITPRGVYTSGKGSSAVGLTAYIT... | [<0 : >228] | Z15032.1 | P30665 |
| N/A_CDS | N/A | N/A | M90830.1 | N/A_Uniprot |
| N/A_CDS | N/A | N/A | M94340.1 | N/A_Uniprot |
| N/A_CDS | N/A | N/A | X61243.1 | N/A_Uniprot |
| N/A_CDS | N/A | N/A | Z17674.1 | N/A_Uniprot |
| NULL | NULL | NULL | NULL | NULL |

Para a povoação da entidade *History* funcionar corretamente para sucessivas pesquisas, foi necessário limpar todos os dados das restantes entidades e relacionamentos já referidos. Permitindo assim ficar com um histórico de todas as pesquisas feitas, sem ter problemas associados à repetição de chaves primárias repetidas.

```
#Limpar todos os dados exceto o historico

DataBase = SQLC.connect(
    host ="geo.di.uminho.pt",
    user ="bioinformatica",
    password ="20221207",
    database ="AP_db_KRG"
)
DataBase.autocommit = True # allows the change be done

Cursor = DataBase.cursor()
try:

        a= "delete from PubMed_Affiliation"
        Z= "delete from PubMed_Authors"
        b= "delete from Affiliation"
        c= "delete from Authors"
        d= "delete from gene_PubMed"
        e= "delete from PubMed"
        f= "delete from CDS"
        g= "delete from Uniprot"
        h= "delete from Gene"

        Cursor.execute(a)
        Cursor.execute(Z)
        Cursor.execute(b)
        Cursor.execute(c)
        Cursor.execute(d)
        Cursor.execute(e)
        Cursor.execute(f)
        Cursor.execute(g)
        Cursor.execute(h)

except mysql.connector.Error as e:
    print("Erro na escrita na base de dados: {}".format(e) )
finally:
    DataBase.close()
```

A povoação da entidade *History* está representada abaixo, com duas *querys* aleatórias inseridas pelo utilizador bem como os restantes atributos, referenciando a data e hora quando esta pesquisa foi realizada.

| | ID_search | Search | Genes_number_input | Day | Genes_number_NCBI | Protein_number |
|---|---|---|---|---|---|---|
| ▶ | 12 | diabetes | 15 | 2023-01-18 19:23:03 | 8 | 4 |
| | 13 | irs1 | 19 | 2023-01-18 20:03:51 | 14 | 4 |

# Conclusões:

De modo a verificar a correta povoação da nossa base de dados, tendo em atenção as diversas entidades, atributos e relacionamentos, realizaram-se alguns testes no MySQL.

```
11 •  SELECT Uniprot.ID_Uniprot, Gene.ID_genebank from Uniprot
12    join CDS on Uniprot.ID_Uniprot = CDS.ID_Uniprot
13    join Gene on CDS.ID_genebank = Gene.ID_genebank;
14
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | ID_Uniprot | ID_genebank |
|---|---|---|
| ▶ | N/A_Uniprot | M90830.1 |
| | N/A_Uniprot | M94340.1 |
| | N/A_Uniprot | X61243.1 |
| | N/A_Uniprot | Z17674.1 |
| | P00815 | V01310.1 |
| | P02099 | V00883.1 |
| | P14747 | X54964.1 |
| | P30665 | Z15032.1 |
| | Q00873 | X67017.1 |

```
11 •  SELECT Gene.Citosina, PubMed.ID_PumMed  FROM Gene
12    JOIN gene_PubMed ON Gene.ID_genebank = gene_PubMed.ID_genebank
13    JOIN PubMed ON gene_PubMed.ID_AI_PubMed = PubMed.ID_AI_PubMed
14    WHERE Gene.Citosina > 21;
15
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | Citosina | ID_PumMed |
|---|---|---|
| | 22 | 1646387 |
| | 22 | 1619649 |
| | 23 | 1499554 |
| ▶ | 22 | N/A |

```
10 •  SELECT Gene.ID_genebank, PubMed.ID_PumMed from Gene
11    join gene_PubMed on Gene.ID_genebank = gene_PubMed.ID_genebank
12    join PubMed on gene_PubMed.ID_AI_PubMed = PubMed.ID_AI_PubMed;
13
14
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | ID_genebank | ID_PumMed |
|---|---|---|
| ▶ | M94340.1 | N/A |
| | Z17674.1 | N/A |
| | V00883.1 | 6271761 |
| | X67017.1 | 1499554 |
| | X61243.1 | 1619649 |
| | M90830.1 | 1382179 |
| | Z15032.1 | 1454522 |
| | V01310.1 | 7049842 |
| | X54964.1 | 1646387 |

Tendo em conta a povoação de cada entidade, verificou-se que os resultados obtidos estão corretos, o que permite concluir que, de forma geral, a nossa base de dados está operacional.

## Melhoramentos Futuros;

1. Pretende-se aumentar o número de genes possíveis a ser trabalhados, ou seja, passar de 20 para 100/200 (relacionado com a paginação do NCBI).
2. Pretendemos explorar a entidade *History* onde esta passa a ser uma entidade n para n e, por isso, não seja necessário apagar os dados antes de uma nova pesquisa.
3. Otimizar e simplificar o código para funcionar com todas as *querys* que nele se colocam e, ao mesmo tempo, ser ainda mais fácil a sua interpretação.