

Feito por:

PG51118 - Guilherme Sousa

PG51119 - Karyna Lysenko

PG50923 - Rodrigo Esperança

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 foi criar uma Base de Dados (BD) que apresentasse um largo conjunto de informação relativamente a um determinado número de genes associados a uma *query* de pesquisa escolhida pelo utilizador. Esta *query* pode ser nome de um organismo, de um gene ou de uma proteína específica. Além disso, o intuito desta base de dados foi apresentar uma otimização à existente do NCBI, de forma a que a BD permitisse fazer uma ligação direta entre as informações de NCBI e a Uniprot e que, fosse possível navegar na BD recorrendo a uma filtragem de genes por número de CDS.

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.

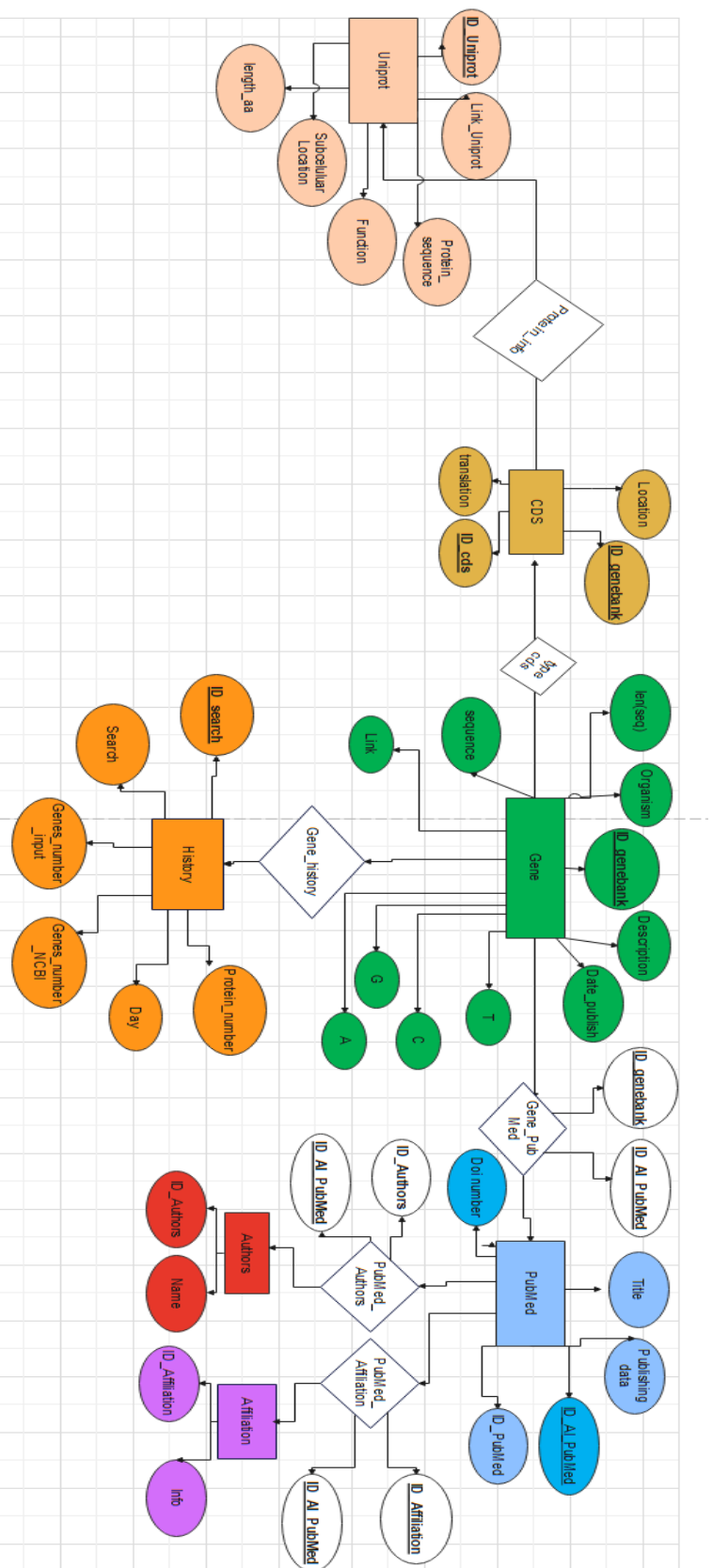
Segundo a informação pretendida, realizou-se a organização desta em entidades, representadas por diversos atributos com diferentes tipos de relacionamentos entre si, como é demonstrado nas tabelas abaixo. Tabelas que serviram de orientação para a realização das restantes fases do trabalho.

Identity	Name	Tipo e dominio	Multivalued	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
	ID_genebank	VARCHAR	N	N	S
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

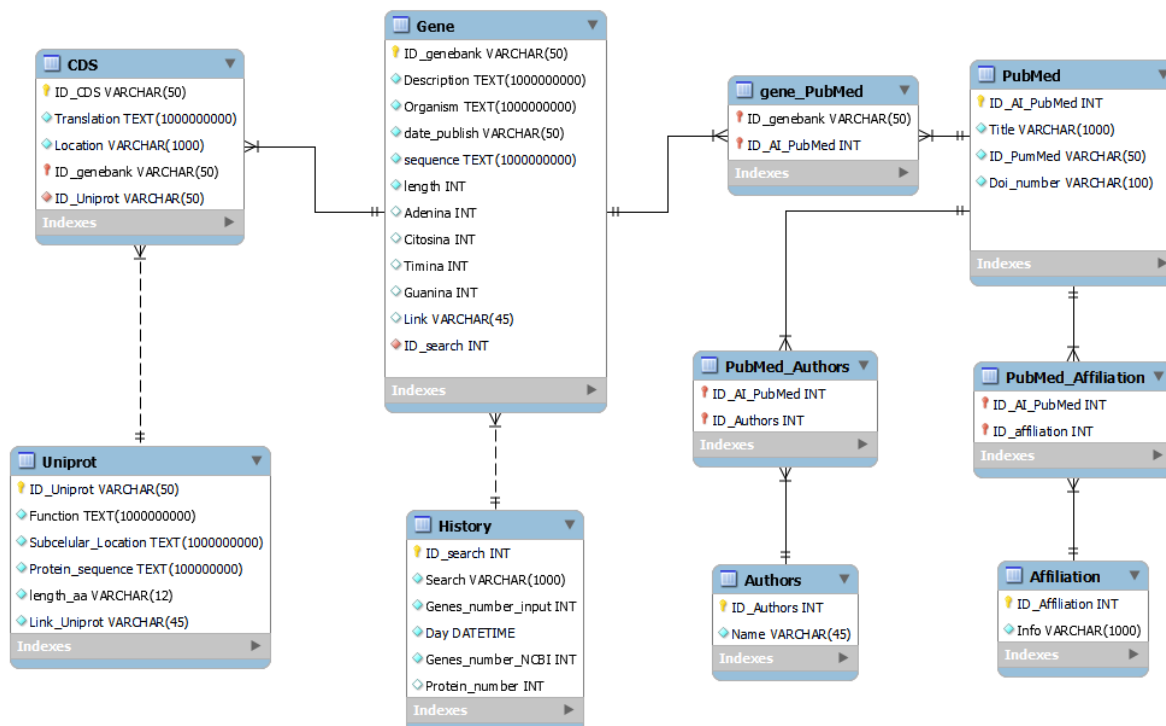
Identity	#	Relationship	Identity	#
Gene	1...N	gene_PubMed	PubMed	1...N
	1...1	Type_CDS	CDS	1...N
	1...N	Gene_History	History	1...1
PubMed	1...N	PubMed_Authors	Authors	1...N
	1...N	PubMed_Affiliation	Affiliation	1...N
CDS	1...N	Protein_info	Uniprot	1...1

Note-se que a BD apresenta uma particularidade que foi tida em conta em todo o trabalho. Sempre que as entidades “Pubmed”, “CDS”, ou “Uniprot”, foi-lhes atribuída a chave primária: *N/A*, *N/A_CDS*, *N/A_Uniprot*, respetivamente.

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



Após o desenho do modelo conceitual 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 script com o propósito de obter os links associados ao NCBI a partir de uma determinada query e, destes links, com auxílio de expressões regulares, extrair os respectivos IDS genebank. Isto foi feito com a função *url_get(i)*. De notar 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 podem aparecer repetidos (funcionamento normal da própria Web BD), 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 módulo 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 (variável 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 (quando disponível) a descrição, o organismo, a sequência, o tamanho de sequência, calcular a percentagem dos nucleótidos e a 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 passo foi obter a informação relativa à entidade PubMed, onde através dos links dos IDs já “processados” referentes ao NCBI, extraiu-se o conteúdo utilizando o módulo *Beautifulsoup* do *bs4*. Todavia, para ser mais fácil trabalhar com este conteúdo, criamos uma 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={}&dt
    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 para extrair de *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, atribuiu-se 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 haver 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 módulo Entrez de Biopython. Este módulo permite, através dos IDs da PubMed contidos na lista (*new_list_*), extrair informações como o título, 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*, utilizaram-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”, o 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 de buscar informação 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 um relacionamento específico 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 *web API* do Uniprot e atribuindo um *field* a cada atributo pretendido para esta entidade, conseguiu-se adquirir toda a informação desejada. Todavia esta informação aparecia de uma forma pouco organizada e, como tal, com recurso a expressões regulares retirou-se a

informação pretendida com a apresentação desejada e o conteúdo organizado. 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\n'))', str(i), re.DOTALL)
                    function = re.match( r'b'Function \[CC\]\n.{9} (.+?(?=\n\n'))', str(i), re.DOTALL )
                    location_exist = re.search( r'b'Subcellular location \[CC\]\nSUBCELLULAR LOCATION: (.+?(?=\n\n'))', str(i), re.DOTALL )
                    location_notexist = re.search( r'b'Subcellular location \[CC\]\n\n\n', str(i), re.DOTALL )
                    sequence = re.search(r'b'Sequence\n(.+?(?=\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 ")
```

Neste ponto, toda a informação pretendida para as entidades “Uniprot” e “CDS” foi recolhida com sucesso. Sendo que estas duas entidades relacionam-se entre si, decidiu-se juntar as informações correspondentes em listas. Com isto, a função **join_ids_CDS** recebe como argumentos duas listas (**lista**, **uniprotID**) com resultado final de lista de listas [**IDgenebank**, **ID_PRO**, **ID_Uniprot**], onde:

- **lista** corresponde a: **dict_list = dict_to_list(result_dict)**
- **uniprotID** corresponde ao resultado de: **get_list_uniprot(ID_PROT, result_dict)**

De seguida, a função **join_lists(list1, list2)** junta a informação da lista anterior [**IDgenebank**, **ID_PRO**, **ID_Uniprot**] com a lista criada para a entidade “Uniprot”, com um resultado final [**IDgenebank**, **ID_PRO**, **ID_Uniprot**, **Location**, **Translation**]

Os scripts das funções **join_ids_CDS** e **join_lists** não estão apresentados no presente relatório de forma a simplificar o conteúdo do mesmo.

Por último, 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 com uma *query* específica (atributo “Search” contém informação da *query*). Associado a isso, elaborou-se uma função que conta o número de proteínas, o número de IDs NCBI e o número 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*). Seguidamente, deve-se inserir a “ação” que se pretende executar (*Insert*) indicando a entidade onde pretendemos executar essa mesma ação e, ainda, 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 auto incrementada 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 *ID_search* auto incrementados, sendo que, apenas interessa o último pois, como é o mais atual, é esse que vai estar relacionado com a entidade “Gene” que se vai povoar de seguida.

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.

```
#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]
```

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 vários 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, 1
        val=(value, des[index], Organismos[index], seqss[index], dates[index], tam[index], Adenina[index], Citosina[index], Guan
        # 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...	re/M90830.1 12
AAAGATTAAG...AACCTGCGGA	1759	25	20	26	27	https://www...	re/M94340.1 12
AAGCTTCTCT...TGTGGAATTC	1905	28	19	27	24	https://www...	re/V00883.1 12
CTCGAGAAGA...TAGGAAAGAA	4751	32	19	26	21	https://www...	re/V01310.1 12
TTTTTTTACT...CTAACAATTT	578	32	22	31	14	https://www...	re/X61243.1 12
CCGGCCTTCT...TTCCACCGG	1416	30	23	29	17	https://www...	re/X67017.1 12
TCTACTTCCA...CGTCTTGAC	228	26	16	34	22	https://www...	re/Z15032.1 12
GAGCAATAAT...ATCTTTTACC	720	24	22	30	21	https://www...	re/Z17674.1 12

Para a entidade “PubMed” foi utilizado o mesmo raciocínio que na entidade “Gene”, notando que a *primary key*, é um ID auto incrementado (*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 auto incrementado para depois a cada ID PubMed presentes na lista *ID_PUB*, pela ordem certa, atribuir o mesmo ID auto incrementado e, assim povoar o relacionamento “gene_PubMed”.

```

#Povoação "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 seguinte, mostra a correta associação entre um *ID_genebank* e um *ID_AI_PubMed*, ou seja, uma correta povoação do relacionamento “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 “Affiliation” 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	Naderio 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 podem existir genes que não têm associado nenhum ID_Uniprot. Posto isso, e com o intuito de não ter a repetição no valor da chave primária (ter diversos ID_Uniprot = N/A), a solução passou por associar todos os genes que não têm referência 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_Uniprot	N/A_Uniprot
P00815	N/A	N/A	MYLPILPLIDDLASWNSKKEYVSLVGQVLLDGSSLSN...	799	https://www.uniprot.org/uniprotkb/P00815/entry
P02099	This protein functions as an embryonic globin, b...	N/A	MVHFTAEBKAAITSTWKLVDVEDAGAEALGRLLVY...	147	https://www.uniprot.org/uniprotkb/P02099/entry
P30665	N/A	Nucleus (ECO:0000250).	MSQQSSPTKEDNNSSSPVVPNPDSYPPQLSSPALF...	933	https://www.uniprot.org/uniprotkb/P30665/entry
Q00873	Lyase that catalyzes the covalent linking of the ...	Mitochondrion inner membrane (ECO:0000269)...	MMSSDQQKCPVDEETKQLVLRHGNHAPGATA...	224	https://www.uniprot.org/uniprotkb/Q00873/entry
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 se pode ter N genes associados a 1, ou seja, *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	[MVLPLILPLIDDLASWNSKKEYVSLVGQVLLDGSSLS...	[1331 : 3731]	V01310.1	P00815
CAA47407.1	[MMSSDQQGKCPVDEETKLLWLRHGNHAEHPGAT...	[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.

Para mais detalhes sobre os scripts, ler o ficheiro *READ_ME.txt* do repositório.

```

#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 *queries* 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:

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.

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.

De forma a testar a BD, o script foi corrido várias vezes com várias *queries* e com escolhas de nº de genes diferentes. Com isto, os próximos prints são de *queries* de MySQL onde a *query* do script é “diabetes” para “20” genes. Assim sendo, a próxima imagem ilustra a entidade “CDS” povoada com esses *inputs*.

ID_CDS	Translation	Location	ID_genebank	ID_Uniprot
CAA24252.1	[MVHFTAEEKAAITSTWKLVDVEDAGAEALGRLLVV...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	V00883.1	P02099
CAA24617.1	[MVLPLILIDDLASWNSKKEYVSLVGQVLLDGSSLS...	[1331 : 3731]	V01310.1	P00815
CAA28945.1	[MDTVNNYRVLEHKAAGHDGTLTDGGLLIFKPAFP...	[311 : 1379]	X05328.1	P07250
CAA38712.1	[MSSDAIRNTEQINAAIKIENKTERPQSSTTPIDSK...	[906 : 2721]	X54964.1	P14747
CAA47407.1	[MMSSDQQGKCPVDEETKLLWREHGNEAHPGAT...	[246 : 921]	X67017.1	Q00873
CAA78750.1	[STSKSQILQYVHKITPRGVYTSKGSSAVGLTAYIT...	[<0 : >228]	Z15032.1	P30665
CAA79566.3	[MDFTENEEEYEHVTHIERRVPFQGYRTYVASTYIS...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	Q03613
CAA79568.2	[MTSRMQPKSGVYSVKADFTKIFEDVPKVPITSASV...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	Q03609
CAA79569.3	[MMFHSLLGLLYFLYVTLAEDCIDRPCVECRHYIKQD...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	Q03610
CAA79570.2	[MVTQDLFLLLLLLTSCAAQIDGLRDKRQTGSGGRV...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	O62504
CAB63418.2	[MSTAEPAPDPTNPSTSGLAPTTNGIGSPPTASAA...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	Q03611
CAE18055.3	[MCFQLQEYLTGPVVLKFRPYRNVFIGISSVAIFFAIL...	[[223 : 316], [440 : 662], [1479 : 1608], [10...	Z19157.1	Q7YWM4
N/A_CDS	N/A	N/A	M90830.1	N/A_Uniprot
N/A_CDS	N/A	N/A	M94340.1	N/A_Uniprot

A próxima imagem ilustra o resultado de um *COUNT* onde 1 ID_genebank onde o gene tem mais do que 1 CDS associado.

```

12 • SELECT Gene.ID_genebank, COUNT(CDS.ID_CDS) from Gene
13   join CDS on Gene.ID_genebank = CDS.ID_genebank
14   group by Gene.ID_genebank
15   HAVING COUNT(CDS.ID_CDS)>1;
16

```

ID_genebank	COUNT(CDS.ID_CDS)
Z19157.1	6

A próxima imagem ilustra o resultado de um *SELECT* de ID_genebank onde 1 onde o gene não tem CDS's.

```

9
10 • SELECT Gene.ID_genebank, CDS.ID_CDS from Gene
11   join CDS on Gene.ID_genebank = CDS.ID_genebank
12   where CDS.ID_CDS= 'N/A_CDS';
13
14

```

ID_genebank	ID_CDS
M90830.1	N/A_CDS
M94340.1	N/A_CDS
X61243.1	N/A_CDS
Z17674.1	N/A_CDS

A próxima imagem ilustra o resultado de um *SELECT* de ID_genebank que tem atribuído apenas 1 ID_CDS.

```

10 • SELECT Gene.ID_genebank, CDS.ID_CDS from Gene
11 join CDS on Gene.ID_genebank = CDS.ID_genebank
12 where CDS.ID_CDS != 'N/A_CDS'
13 group by Gene.ID_genebank
14 having count(CDS.ID_CDS) = 1;
15

```

Result Grid			Filter Rows:	Export:	Wrap Cell C
	ID_genebank	ID_CDS			
▶	V00883.1	CAA24252.1			
	V01310.1	CAA24617.1			
	X05328.1	CAA28945.1			
	X54964.1	CAA38712.1			
	X67017.1	CAA47407.1			
	Z15032.1	CAA78750.1			

Perspectivas Futuras:

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 estruturar o código para funcionar com todas as *queries* que nele se colocam e, ao mesmo tempo, ser ainda mais fácil a sua interpretação.