Report: Phase-3: BioJoin Creative

BiS332 course by Professor Doheon Lee

Lana Abu Hassan (20170842), Surayouth Phuksawattanachai (20180813), Alexander Semenov (20226159)

Introduction

The project in this 3rd phase aims to develop an integrated biomedical database system, which may be used to infer associations among datasets through python implementation further.

Generally, not only do numerous novel drugs emerge each passing day, but new types of diseases also do. As a result, physicians might find it challenging to follow such developments in the field. Thus, an organized disease-drug database system is mandatory to help physicians catch up with the current trends in bio-pharmaceutical practice. Through a legitimate application of the database query systems, it will generously assist the medical practice.

In current times, databases that include drug, disease, and gene relations are common. However, this data might not be accessible in practical use due to its tremendous or uninterpretable information. As a result, our group aims to establish a database system that accommodates data from external sources to the current version we implemented in order to compile the data into an applicable one.

With this, we introduce the 'scoring' system, which may be implemented to suggest the best drug, disease, or other components for a particular system, in which several factors can refer to this 'scoring'.

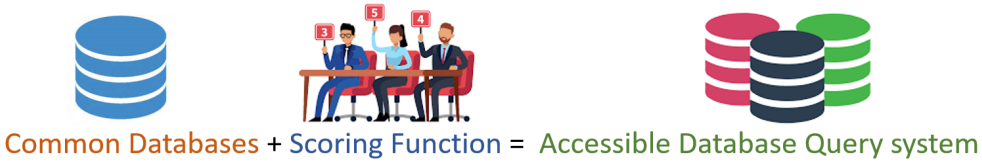


Figure 1: Scoring system schematics

For example, some candidate factors of a feasible drug are scored by their toxicity value. According to the Journal of Pharmaceutical Sciences and Drug Development, drug toxicity refers to the "level of damage that a compound can cause to an organism". Considering the drug inference query, since each drug has a different level of 'toxicity value', we may apply this information to select the least toxic drug to assign for a patient's prescription according to the disease they have.

In particular, the databases that we referred to in this project are from primary bioinformatician databases, which TAs and Comparative Toxicogenomics Database provided. As such, we will show the design of database architecture along with its corresponding relational schema and entity-relational diagram. Besides, we will show that the design contains all mandatory information and format while also having an effective design. Upon that, with additional implementation on SQL DDL/Query/DML, which we study during class, we may be able to acquire an appropriate database query system. Upon that, we will also explain why we select each procedure for each step of implementation

As a result, we can obtain a scoring function implemented-drug/disease/gene database system that provides practical information for patient diagnosis.

Objectives

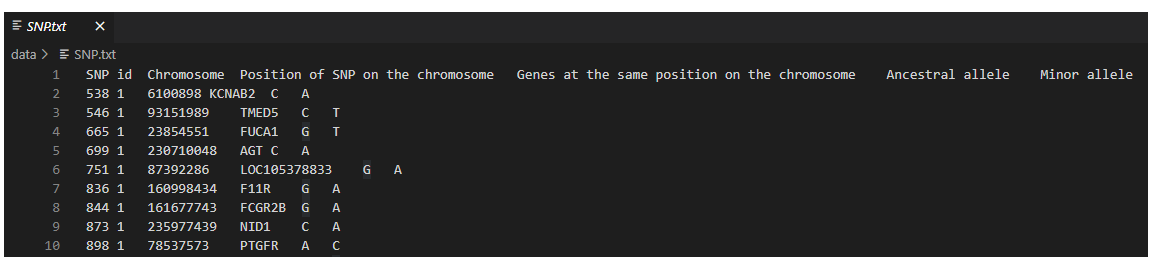
By implementing existing data, we aim to verify distinct relations, such as:

1. **Drug-Disease relation**
    a. Given a name of disease, verify its therapeutic drug name
    b. Given a therapeutic drug name, verify disease that can be treated with the drug
2. **Drug-Gene relation**
    a. Given a name of a therapeutic drug, verify affected genes
    b. Given a name of a therapeutic drug, verify the frequency of genes affected within a specific chromosome
3. **Disease-Gene relation**
    a. Given a particular chromosome, verify its associated diseases
4. **Interesting statistics**
    a. Statistics of disease distribution on each chromosome
    b. Statistics of top universal drugs

Database Architecture

Raw data files

In this phase 3 project, we also use the same raw data files that TA provided to us from primary bioinformatician databases. Namely, OMIM.txt, gene_OMIM.txt, Homo_sapiens_gene_info.txt and SNP.txt files. All the files are tab-separated csv files with a given header (see examples below).



SNP.txt

data > SNP.txt

```
SNP id  Chromosome  Position of SNP on the chromosome   Genes at the same position on the chromosome    Ancestral allele    Minor allele
538 1   6100898 KCNAB2  C   A
546 1   93151989    TMED5   C   T
665 1   23854551    FUCA1   G   T
699 1   230710048   AGT C   A
751 1   87392286    LOC105378833    G   A
836 1   160998434   F11R    G   A
844 1   161677743   FCGR2B  G   A
873 1   235977439   NID1    C   A
898 1   78537573    PTGFR   A   C
```

Figure 2: Example of SNP information text file - SNP.txt

Homo_sapiens_gene_info.txt

data > Homo_sapiens_gene_info.txt

```
#tax_id GeneID  Symbol  Synonyms    chromosome  map_location    description type_of_gene    Modification_date
9606    1   A1BG    A1B|ABG|GAB|HYST2477    19  19q13.43    alpha-1-B glycoprotein  protein-coding  20170402
9606    2   A2M A2MD|CPAMD5|FWP007|S863-7   12  12p13.31    alpha-2-macroglobulin   protein-coding  20170402
9606    3   A2MP1   A2MP    12  12p13.31    alpha-2-macroglobulin pseudogene 1  pseudo  20170402
9606    9   NAT1    AAC1|MNAT|NAT-1|NATI    8   8p22    N-acetyltransferase 1   protein-coding  20170403
9606    10  NAT2    AAC2|NAT-2|PNAT 8   8p22    N-acetyltransferase 2   protein-coding  20170402
9606    11  NATP    AACP|NATP1  8   8p22    N-acetyltransferase pseudogene  pseudo  20170312
9606    12  SERPINA3    AACT|ACT|GIG24|GIG25    14  14q32.13    serpin family A member 3    protein-coding  20170403
9606    13  AADAC   CES5A1|DAC  3   3q25.1  arylacetamide deacetylase   protein-coding  20170402
9606    14  AAMP    -   2   2q35    angio associated migratory cell protein protein-coding  20170403
```

Figure 3: Example of Gene information text file - Homo_sapiens_gene_info.txt

disease_OMIM.txt

data > disease_OMIM.txt

```
disease_OMIM_ID disease_name
1   $Deafness, Y-linke$
10  46XY sex reversa
100100  Prune belly syndrome
100300  Adams-Oliver syndrome 1
100800  Achondroplasia
101000  $Neurofibromatosis, type 2$
101200  Apert syndrome
101400  Saethre-Chotzen syndrome
101600  Craniofacial-skeletal-dermatologic dysplasia
101800  $Acrodysostosis 1, with or without hormone resistance$
101900  Acrokeratosis verruciformis
102200  $Acromegaly, somatic$
102370  Acromicric dysplasia
102500  Hajdu-Cheney syndrome
```

gene_OMIM.txt

data > gene_OMIM.txt

```
gene_symbol disease_OMIM_ID
1C7 609148
2E4 615637
3H11AG  610188
3H11AG  610189
3H11AG  611134
3H11AG  611755
3H11AG  615991
3M1 273750
3M2 612921
3M3 614205
3MC1    257920
3MC2    265050
3PK 617111
8D6 613646
```

Figure 4: Example of OMIM information text file - disease_OMIM.txt and gene_OMIM.txt

In addition to these data, we also introduced new data from http://ctdbase.org/downloads/
The followings are data we use to implement to our database:
- CTD_chemicals.xml (http://ctdbase.org/reports/CTD_chemicals.xml.gz)
- CTD_diseases.xml (http://ctdbase.org/reports/CTD_diseases.xml.gz)
- CTD_chemicals_diseases.xml (http://ctdbase.org/reports/CTD_chemicals_diseases.xml.gz)

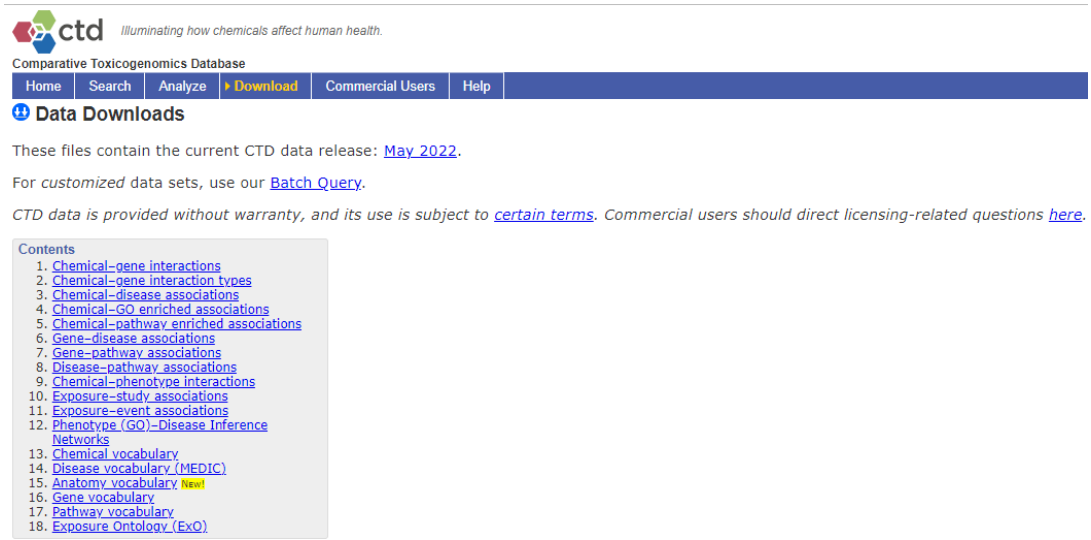


Figure 5: CTD databases for drug and diseases relational data

Tables creation : SQL DDL

Similarly, we adapt the structure of the database from phase 2 to our database. Despite that, we also make some adjustments for parameters. Here, we introduce the update version of the SQL DDL statement for tables creation:

dbSNP

snp_chr
snp_pos
gene_symb
anc_allele
min_allele
snp_id {PK}

→

```
CREATE TABLE SNP (
snp_id int NOT NULL,
snp_chr varchar,
snp_pos varchar,
gene_symb varchar,
anc_allele varchar,
min_allele varchar,
PRIMARY KEY (snp_id));
```

gene

gene_id
gene_syn
gene_chr
gene_pos
gene_sum
gene_type
gene_mod_date
popularity
snp_id {FK}
gene_symb {PK}

→

```
CREATE TABLE Gene (
gene_id int UNIQUE,
gene_symb varchar NOT NULL,
gene_syn varchar,
gene_chr varchar,
gene_pos varchar,
gene_sum text,
gene_type varchar,
gene_mod_date date NOT NULL,
popularity float,
snp_id varchar,
PRIMARY KEY (gene_symb));
```

disease

disease_name
alt_id
disease_id {PK}
gene_symb {PK}

→

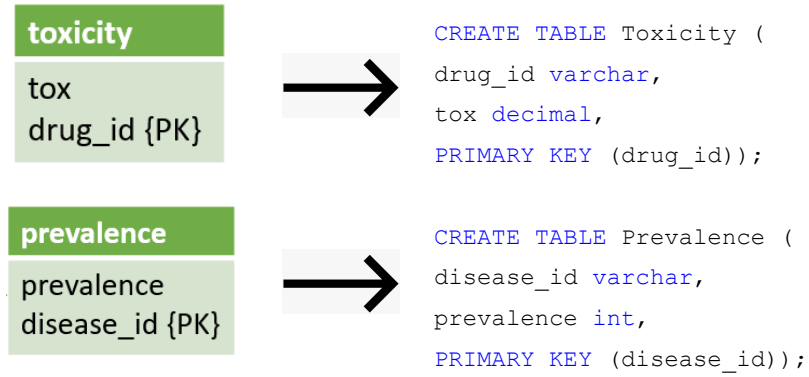
```
CREATE TABLE Disease (
disease_name varchar,
id varchar,
alt_id varchar,
gene_symb varchar,
PRIMARY KEY (id, gene_symb));
```

disease_drug

drug_name
disease_name
drug_id {PK}
disease_id{PK}

→

```
CREATE TABLE Disease_Drug (
drug_name varchar ,
drug_id varchar,
disease_name varchar,
disease_id varchar,
PRIMARY KEY (drug_id, disease_id));
```



In particular, the dbSNP table is the same as in bio join phase 2. However, the gene table is extended with a column of 'popularity' value (varying from 0 to 100 as a random float), which determines the searching trend of each gene. The OMIM table is revised into a disease table which changes omim_name into disease_name, omim_id into disease_id, keeping gene_symb and adding alt_id for alternative disease_name. Additionally, we introduce three new tables: the disease_drug table, prevalence table, and toxicity table.

The first table, the disease_drug table, consists of drug_name/id and disease_name/id, which refers to the disease and drug connected to each other. On the other hand, the toxicity and prevalence table is a table that is constructed from random-generated data. The toxicity value data is used to score the drugs, while prevalence data is used to score the diseases.

Database creation and filling

In order to construct the database, we reset the entire old database by deleting all tables and creating new tables from the new SQL DDL statements. After that, we then fill the table with data from both primary bioinformatician databases and Comparative Toxicogenomics Database. In this step, we implement fill.py to insert data into their corresponding table. As said above, the toxicity and prevalence table are filled from random-generated data for now. In the future this data can be exchanged to a real data without any additional editing to the software.

The following are functions for random data generation:

```
def generate_random_mock_toxicity(db_connection):
    """assigns a random toxicity percentage (0-100) to a drug"""
    …
    for drug in all_drugs_ids:
        random_tox_percent = round(random() * 100, 2)
        data.append([drug, random_tox_percent])
```

```
def generate_random_mock_prevalence(db_connection):
   """assigns a random Prevalence (1.000 - 1.000.000) to a disease """
   …
   for disease in all_diseases_ids:
       random_tox_percent = randrange(1000, 10 ** 6)
       data.append([disease, random_tox_percent])
```

As said in the beginning, the OMIM table is revised into a disease table. The table was extended with the data from the CTD database. Unfortunately, CTD diseases had different identification for the diseases called MESH. So, it became a challenge to merge both databases correctly. Nevertheless, the merge of those two databases was needed so we could map diseases with drugs (using MESH notation) and, on the other side, keep the information of associated genes (using OMIM notation). We were lucky that a mapping of disease entries was possible with alternative symbols in the diseases table from the CTD database (alternative symbols had OMIM notation). The merging procedure consisted of three steps:

1. Process all the OMIM identifications for given entry in CDT table (either it is main id or alternative id) and find corresponding data in OMIM table
2. Create separate entries if one MESH entry has multiple associated OMIM entries (each new disease has MESH, OMIM and gene symbol
3. Create MESH entires with no OMIM associations, leave the gene symbols empty

Here is a code snippet of the smart_merge_disease function:

```
def smart_merge_disease(disease_data, omim_data):
   """ from CTD diseases we obtain the OMIM id and
       from OMIM get the gene_sysmb to add to CTD diseses
   """
   full_disease_data = []
   for disease in disease_data:
       # if omim is in id (isted of mesh)
       if 'OMIM' in disease[1]:
           …
       # if omim in alternative ids
       if disease[2] is not None:
           alternatives = disease[2].split("|")
           omim_alternatives = [omim.split(':')[1] for omim in alternatives if 'OMIM:'
in omim]
             …
       else:
           # no omim in id or alternative ids
           single_disease_no_omim = disease.copy()
           single_disease_no_omim.append('None')# set gene symbol to None
```

The result of created and filled tables looks like this:

```
u20226159=> select * from disease limit 1;
          disease_name          |      id      | alt_id | gene_symb
--------------------------------+--------------+--------+-----------
 10p Deletion Syndrome (Partial) | MESH:C538288 |        | None
(1 row)

u20226159=> select * from disease_drug limit 1;
     drug_name     | drug_id |              disease_name             | disease_id
-------------------+---------+---------------------------------------+------------
 Vehicle Emissions | D001335 | HETEROTAXY, VISCERAL, 6, AUTOSOMAL | OMIM:614779
(1 row)

u20226159=> select * from gene limit 1;
 tax_id | gene_id | gene_symb |        gene_syn        | gene_chr | gene_pos |        gene_sum        |   gene_type    | gene_mod_date | popularity
--------+---------+-----------+------------------------+----------+----------+------------------------+----------------+---------------+------------
 9606   |       1 | A1BG      | A1B|ABG|GAB|HYST2477 | 19       | 19q13.43 | alpha-1-B glycoprotein | protein-coding | 2017-04-02    |      88.29
(1 row)

u20226159=> select * from snp limit 1;
 snp_id | snp_chr | snp_pos | gene_symb | anc_allele | min_allele
--------+---------+---------+-----------+------------+------------
    538 | 1       | 6100898 | KCNAB2    | C          | A
(1 row)

u20226159=> select * from prevalence limit 1;
  disease_id  | prevalence
--------------+------------
 MESH:C564334 |     286725
(1 row)

u20226159=> select * from toxicity limit 1;
 drug_id | tox
---------+-----
 C065545 | 8.16
(1 row)
```

Figure 6: Content of tables after data has been filled

ER diagram

During normalization, each table is normalized to 1NF by defining each attribute to contain the only atomic or indivisible value, in which the value of each attribute contains only a single value from its corresponding domain. Thus, each table consists of a primary key and no composite attributes.

Then, we confirm that every non-prime attribute within the table is dependent on the candidate key (primary key), which then normalizes to 2NF. Lastly, we normalized the table to 3NF by confirming that no attributes depend on other non-key attributes.

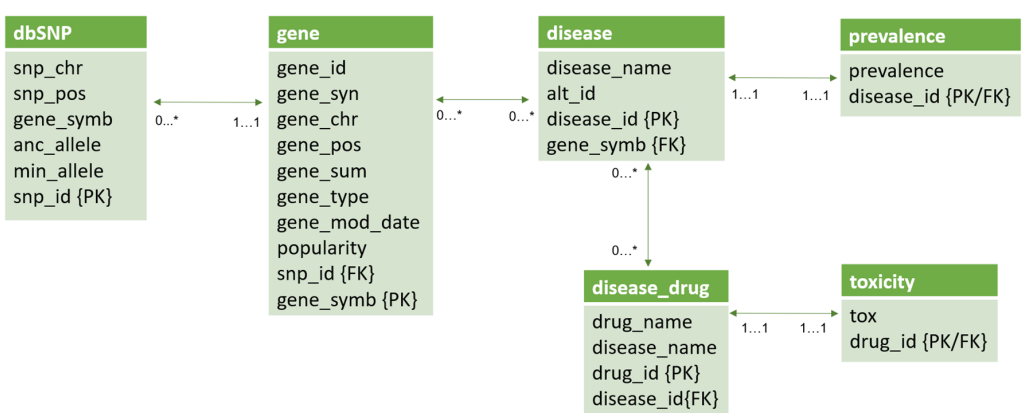


Figure 7: Combined ER diagram

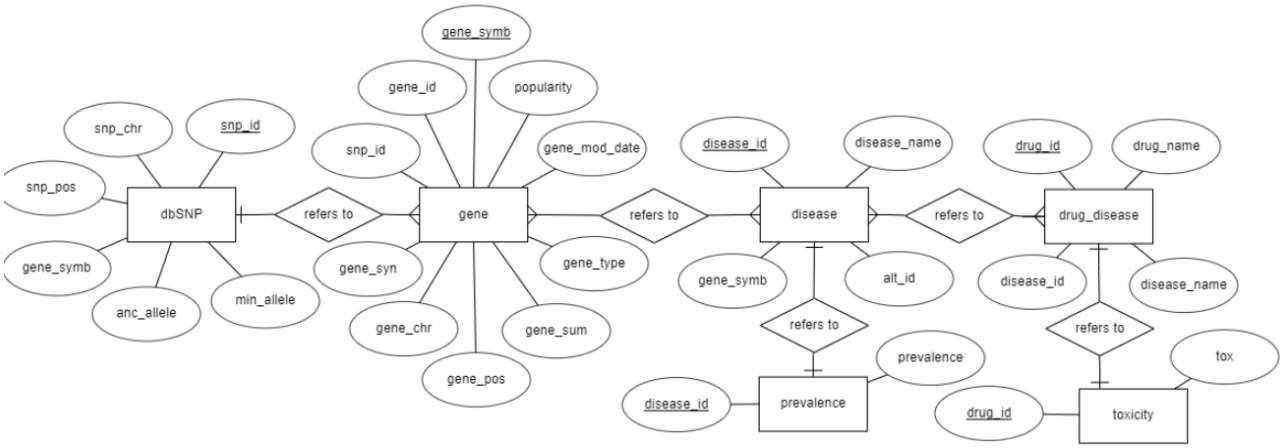
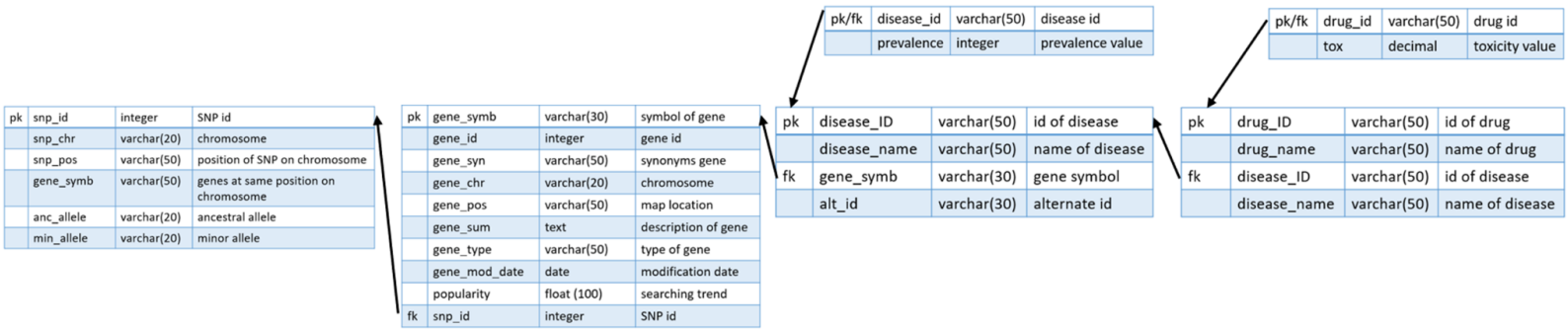


Figure 8: Combined ER diagram with symbolic description

Relational Schema



pk/fk	disease_id	varchar(50)	disease id
	prevalence	integer	prevalence value

pk/fk	drug_id	varchar(50)	drug id
	tox	decimal	toxicity value

pk	snp_id	integer	SNP id
	snp_chr	varchar(20)	chromosome
	snp_pos	varchar(50)	position of SNP on chromosome
	gene_symb	varchar(50)	genes at same position on chromosome
	anc_allele	varchar(20)	ancestral allele
	min_allele	varchar(20)	minor allele

pk	gene_symb	varchar(30)	symbol of gene
	gene_id	integer	gene id
	gene_syn	varchar(50)	synonyms gene
	gene_chr	varchar(20)	chromosome
	gene_pos	varchar(50)	map location
	gene_sum	text	description of gene
	gene_type	varchar(50)	type of gene
	gene_mod_date	date	modification date
	popularity	float (100)	searching trend
fk	snp_id	integer	SNP id

pk	disease_ID	varchar(50)	id of disease
	disease_name	varchar(50)	name of disease
fk	gene_symb	varchar(30)	gene symbol
	alt_id	varchar(30)	alternate id

pk	drug_ID	varchar(50)	id of drug
	drug_name	varchar(50)	name of drug
fk	disease_ID	varchar(50)	id of disease
	disease_name	varchar(50)	name of disease

Figure 9: Relational schema with primary and foreign key assignment

Tools

Similarly, we also work on this project using the following platforms and libraries:

- Python 3.8.13
- Psycopg2 (PostgreSQL adapter for the Python programming language)
- Pandas
- Progressbar2 4.0.0

Likewise, we also implemented other python native build-in modules and libraries

Database manipulation software : SQL DML Statements

Overview

The python program in phase 3 keeps the earlier code from phase 2 and modifies additional code to the existing one. As such, the command-line interface still maintains some part of the previous version within itself. The command-line interface is illustrated in the lower figure.

```
What template do you want to try out?

                    1. Find all gene information
                    2. Find all gene symbols located in the chromosome
                    3. Find all diseases associated with the SNP
                    4. Find all SNP IDs associated with the disease
                    5. Drug-Diseases related templates
                    6. Drug-Genes related templates
                    7. Diseases-Genes related templates
                    8. Statistics
                    q. Quit
```

Figure 10: Example of Command-line Interface

Usage of the software

1. **Drug-Diseases related templates:**
   a. **Find drug to treat given disease:**
   Input: Disease Name
   SQL Statement behind the scenes:

```
SELECT disease_drug.drug_name, toxicity.tox
FROM disease_drug JOIN toxicity
ON disease_drug.drug_id = toxicity.drug_id
WHERE disease_drug.disease_name = {disease_name};
```

   Output: Top 5 chemicals/drugs according to their toxicity

```
What template do you want to try out?

                    1. Find drug to treat given disease
                    2. Find diseases that can be treated with your drug
                    q. Quit

Enter your choice: 1
Please provide a disease name: Liver Diseases
Use these drugs:
ToxVal | Drug Name
-----------------
99.98  | 7-methyl-5-(1-((3-(trifluoromethyl)phenyl)acetyl)-2,3-dihydro-1H-indol-5-yl)
99.98  | argemone oil
99.98  | calcium propionate
99.98  | Thiamine
99.93  | 4-amino-1,8-naphthalimide
-----------------
```

Figure 11: Therapeutic drug corresponding to a given disease

b. **Find diseases that can be treated with your drug**

Input: Chemical or drug name

SQL Statement behind the scenes:

```
SELECT disease_drug.disease_name, prevalence.prevalence
FROM disease_drug JOIN prevalence
ON disease_drug.disease_id = prevalence.disease_id
WHERE disease_drug.drug_name = {drug_name};
```

Output: Top 5 diseases according to their prevalence

```
What template do you want to try out?

                        1. Find drug to treat given disease
                        2. Find diseases that can be treated with your drug
                        q. Quit

Enter your choice: 2
Please provide a drug name: Ibuprofen
These disease can be treated:
Prevalence | Diseases
-----------------
999360.0   | Graft Occlusion, Vascular
998891.0   | Carcinoma, Ductal, Breast
998741.0   | Anterior Cruciate Ligament Injuries
996922.0   | Squamous Cell Carcinoma of Head and Neck
996453.0   | Hyperalgesia
------------------
```

Figure 12: Disease corresponding to a given Therapeutic drug

2. **Drug-Genes related templates**

a. **Find genes that are affected by given drug**

Input: Chemical or drug name

SQL Statement behind the scenes:

```
SELECT G.gene_symb, G.popularity
FROM gene G WHERE G.gene_symb
IN(SELECT D.gene_symb from disease D JOIN disease_drug DD
   ON D.id = DD.disease_id WHERE DD.drug_name = {drug_name});
```

Output: Top 5 genes according to their popularity

```
Please provide a drug name: Ibuprofen
These genes are affected:
Popularity | Gene Symbols
-----------------
100.0      | PRKCH
99.89      | ADH1B
99.75      | MST1R
99.75      | MDD1
99.74      | ICOS
99.51      | NOP56
99.45      | UVSSA
99.37      | HGF
99.36      | NUP214
99.19      | ISG15
------------------
```

Figure 13: Genes affected by a given drug

b. **Find chromosomes that are affected by given drug**

Input: Chemical or drug name

SQL Statement behind the scenes:

```
SELECT gene.gene_chr FROM gene WHERE gene.gene_symb
IN (SELECT disease.gene_symb from disease WHERE disease.id
    IN (SELECT disease_drug.disease_id FROM disease_drug
        WHERE disease_drug.drug_name = {drug_name}));
```

Output: Top 3 chromosomes consisting the most affected genes

```
What template do you want to try out?

                        1. Find genes that are affected by given drug
                        2. Find chromosomes that are affected by given drug
                        q. Quit

Enter your choice: 2
Please provide a drug name: Ibuprofen
These chromosomes are most commonly affected:
Chromosome 1: 74 affected genes
Chromosome 2: 52 affected genes
Chromosome 6: 52 affected genes
-----------------
```

Figure 14: Number of genes affected by a drug within each chromosome

3. **Diseases-Genes related templates**

a. **Given chromosome number find associated diseases**

Input: Chromosome number

SQL Statement behind the scenes:

```
SELECT D.id, D.disease_name, R.prevalence
FROM disease as D JOIN (SELECT disease_id, prevalence
FROM prevalence WHERE disease_id IN (SELECT disease_id FROM disease JOIN
gene ON disease.gene_symb = gene.gene_symb
WHERE gene.gene_chr = {chr_nr})) as R ON R.disease_id = D.id;
```

Output: Top 5 diseases

```
Please provide a chromosome number: 5
These disease are assosiated with chromosome '5':
-------------------------------------------------
Heparin Cofactor II Deficiency
2-Hydroxyglutaricaciduria
Graft Occlusion, Vascular
Lymphomatoid Granulomatosis
Cardiomyopathy, Familial Hypertrophic, 13
------------------
```

Figure 15: Chromosome-related diseases

4. Statistics

a. Statistics: Count number of diseases for each chromosome

```
SELECT count(*) FROM disease JOIN gene
ON disease.gene_symb = gene.gene_symb WHERE gene.gene_chr = {chromosome};
```

```
Chromosome-Diseases stats:
Chromosome 1: 524
Chromosome 2: 385
Chromosome 3: 336
Chromosome 4: 220
Chromosome 5: 249
Chromosome 6: 294
Chromosome 7: 240
Chromosome 8: 178
Chromosome 9: 188
Chromosome 10: 228
```

```
Chromosome 11: 357
Chromosome 12: 313
Chromosome 13: 111
Chromosome 14: 156
Chromosome 15: 157
Chromosome 16: 204
Chromosome 17: 322
Chromosome 18: 90
Chromosome 19: 250
Chromosome 20: 148
Chromosome 21: 47
Chromosome 22: 115
Chromosome X: 325
```

Figure 16: Statistics of number of gene-related diseases

b. Statistics: Drugs that can treat the most diseases

```
SELECT disease_drug.disease_name, disease_drug.drug_name
FROM disease_drug;
```

```
The most universal drugs:
Num of Diseases it can treat | Drug name
---------------------------------------
5789 | bisphenol A
5542 | Benzo(a)pyrene
5461 | Tetrachlorodibenzodioxin
5364 | Valproic Acid
5015 | Acetaminophen
4533 | Aflatoxin B1
4452 | Tobacco Smoke Pollution
4399 | sodium arsenite
4268 | Ethinyl Estradiol
4254 | Estradiol
```

Figure 17: Statistics of top universal drugs

Discussion

With all the design of SQL DDL/DML proposed above, our group ultimately succeeded in constructing a functional database query system. Considering the 'scoring' function, we may look into three main topics: drug-disease, drug-gene, and disease-gene association.

For drug candidate evaluation, there were two possible factors that we were contemplating, including toxicity value and bioavailability. We eventually select toxicity value as the candidate in order to prevent posing toxic drugs for the prescription.

There were five proposed factors for evaluating disease candidates: prevalence, mortality, case fatality, incidence, and attack rate. In the end, we choose to apply prevalence for disease evaluation as it implies the number of patients with a given disease, which in turn determines the necessity of disease diagnosis.

We were contemplating two factors for gene candidate evaluation: its popularity (based on searching trends) and mortality statistics. We choose to utilize popularity because the data is more comprehensible than mortality statistics.

In the end, it is important to repeat, that the current used values for the ranking do not correspond to reality due to the lack of data. As far as the data is available to us, it is no problem to switch the tables. No additional changes in the software are necessary.

Further information and source code

For more technical information and the access to the source code of the application, please visit our GitHub repository: https://github.com/Tsatsch/Biojoin

Conclusion

In summary, we successfully extended the BioJoin Basic integrated biomedical database system with the feature of exploring the disease-drugs associations. The software aims to assist the medical practice and provide the newest available information for the public using primary bioinformatician databases and trusted sources. Consequently, it provides practical information for patient diagnosis. With the scoring system, we can select the most relevant results for the user to provide the best value.

We plan to extend our database for future perspectives by involving other drugs and disease sources. In addition, we want to investigate significant diseases, drugs, and gene factors to improve our output data filtering. Using multiple factors will give the user the advantage of working with default evaluation and make their own factors weighting.

References

[1] National Library of Medicine, National Center for Biotechnology Information. ***dbSNP.*** https://www.ncbi.nlm.nih.gov/snp/. Retrieved on 13rd April 2022.

[2] National Library of Medicine, National Center for Biotechnology Information. ***Gene.*** https://www.ncbi.nlm.nih.gov/gene. Retrieved on 13rd April 2022.

[3] Johns Hopkins University. ***OMIM.*** https://omim.org/. Retrieved on 13rd April 2022.

[4] ***Entity–relationship model***. Entity–relationship model - Wikipedia. Retrieved on 13rd April 2022.

[5] Supreya Saxena. ***Relation Schema in DBMS.*** Relation Schema in DBMS - GeeksforGeeks. Retrieved on 13rd April 2022.

[6] Alvaro Monge. ***Basic SQL statements: DDL and DML.*** Database Design - DDL & DML (csulb.edu). Retrieved on 13rd April 2022.

[7] DI Biological Laboratory. ***Data Downloads | CTD.*** http://ctdbase.org/downloads/ Retrieved on 1st June 2022.

[8] Riley, A.L. (2010). ***Drug Toxicity.*** https://doi.org/10.1007/978-3-540-68706-1_1131 Retrieved on 1st June 2022.