Accelerating Drug Discovery with a TypeDB Knowledge Graph

London, March 2022

Tomás Sabat Stöfsel, Vaticle
tomas@vaticle.com

Abstract

Systems biology produces a tremendous amount of heterogeneous data, which present challenges in their integration due to their complex nature and rich semantics. Since understanding the complex relationships in biomedical data is one of the key goals of biology, solutions that speed up the integration and querying of such data are necessary.

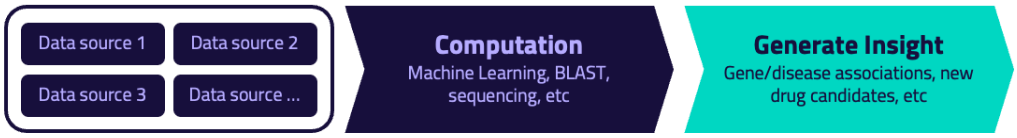
However, analysing large volumes of biomedical data through traditional database systems is troublesome and challenging. Here, we look at how TypeDB can be used to model complex biological relationships to accelerate the drug discovery process. We also present TypeDB - Covid, an open source biomedical knowledge graph for the biomedical community.

1 Integrating and Ingesting Biomedical Data in Traditional Systems

The rapid development and spread of analytical tools in biomedical science have produced a variety of information about all sorts of biological components (cells, tissues, diseases, proteins, pathways, drugs, etc) and their functions. Though important individually, their biological characteristics need to be understood in relation to the interactions they have with different biological components, which requires the integration of vast amounts of heterogenous data. However, as this data is highly complex and semantically rich, doing so can become very hard.

For example, some genes may be connected to multiple diseases, encoding many sequence-similar proteins, and could include provenance information. Identifying the relationships between such entities can be crucial in providing the biological context for hypothesis generation and validation.

A generalised workflow in the biomedical knowledge discovery workflow can be illustrated as follows:



Typically, the first step in producing any type of insight for drug discovery involves the integration of disparate biomedical data. These can be public or proprietary datasets, or data extracted through text mining methods from, for example, PubMed articles. The data sets can come in TSV and/or CSV formats (e.g. Uniprot, ENSEMBL, Drug Bank, etc) and as this is flat data, it is necessary to connect these datasets so that networks can be established. In doing so, it is not uncommon to end up with a few thousand lines of Python code.

This approach suffers from the fact that biological data is not always uniform. For example, some proteins may have been investigated experimentally, are well annotated, and are connected with other pieces of information. Other proteins, however, may just be 'hypothetical proteins' and have little to no information. Further, as biological research is unpredictable, new studies are constantly being published, which we may want to integrate into

our workflow. The result is a long and arduous process that is very time consuming and difficult to scale.

Once a certain set of data has been integrated (either in-memory or in a traditional relational database), this can then be ingested into a computational layer to produce some form of insight. This could be through the use of a sequencing algorithm to find sequence similarities between proteins or genes, for example, or by using machine learning to predict certain gene-disease associations.

However, in this workflow, any insight generated will not enjoy the biological contextuality insofar as it extends to the previously ingested data, unless we go through a complex integration process. This would mean, for example, that although the new insight may suggest a new drug candidate for a disease, we would be unable to understand how it would interact with other biological components (tissues, pathways, proteins, etc). Furthermore, even though this data could be represented in traditional relational databases, join queries (which connect separate tables) would make this computationally too expensive and complicated to design, especially as more complex join statements are done.

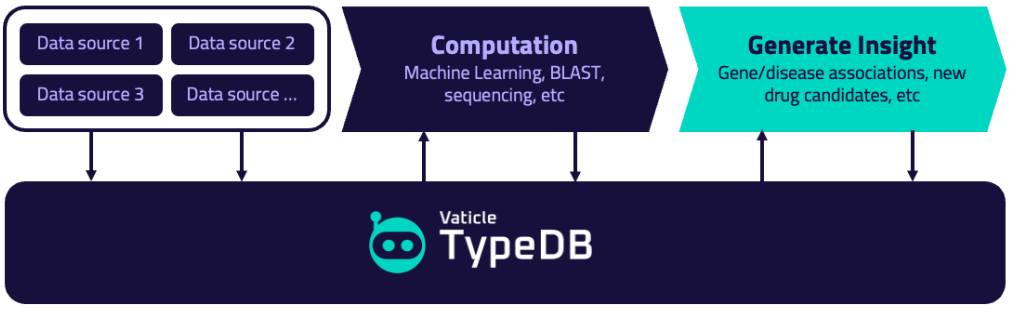
In summary, there are two major issues with this traditional workflow:

1. Data integration and ingestion: Using hard coded scripts to integrate flat data, whether in-memory or into a relational database, is time consuming and inflexible, especially when it comes to adding new data sources. Because of the data's inherent complexity, navigating it can be computationally too expensive.

2. Bringing biological context to newly generated insight: Due to the inflexible nature of the data integration process, it becomes difficult to associate the new data produced from a sequencing or machine learning algorithm. Therefore, this new data will lack the biological contextuality in its interaction with other biological components.

2 TypeDB as a Biomedical Knowledge Graph

TypeDB addresses the problems discussed above, and can be used as the unified representation of our knowledge. This means that architecturally, TypeDB will sit at the centre of our data pipelines:



TypeDB serves as the database that will represent all our disparate data sources throughout the entire drug discovery process. This will allow us to easily add new and unexpected data sources, integrate with generated insights, use complex traversal type queries, change the model if necessary, and much more.

If you're unfamiliar with TypeDB, it is an intelligent database in the form of a distributed knowledge graph which can organise complex networks of data. TypeDB's type system includes entities, relations, and attributes, while rules can be added to perform automated reasoning. TypeDB's schema is a type system that implements the principles of knowledge representation and reasoning, which produce a more expressive and useful system than traditional relational and NoSQL databases when managing large-scale linked data.

2.1 Ingesting Heterogeneous Biomedical Networks in TypeDB

In order to demonstrate the usefulness of TypeDB, please refer to TypeDB — Covid. This is an open source knowledge graph created in collaboration with GSK and Oxford Pharmagenesis for Covid research. The data ingested includes:

1. CORD-19: We have incorporated the original corpus which includes peer-reviewed publications from bioRxiv, medRxiv and others.
2. CORD-NER: The CORD-19 dataset released by the White House has been annotated and made publicly available. It uses various NER methods to recognise named entities on CORD-19 with distant or weak supervision.

3. Uniprot: We've downloaded the reviewed human subset, and ingested genes, transcripts, and protein identifiers.

4. Coronaviruses: This is an annotated dataset of coronaviruses and their potential drug targets put together by Oxford PharmaGenesis based on literature review.

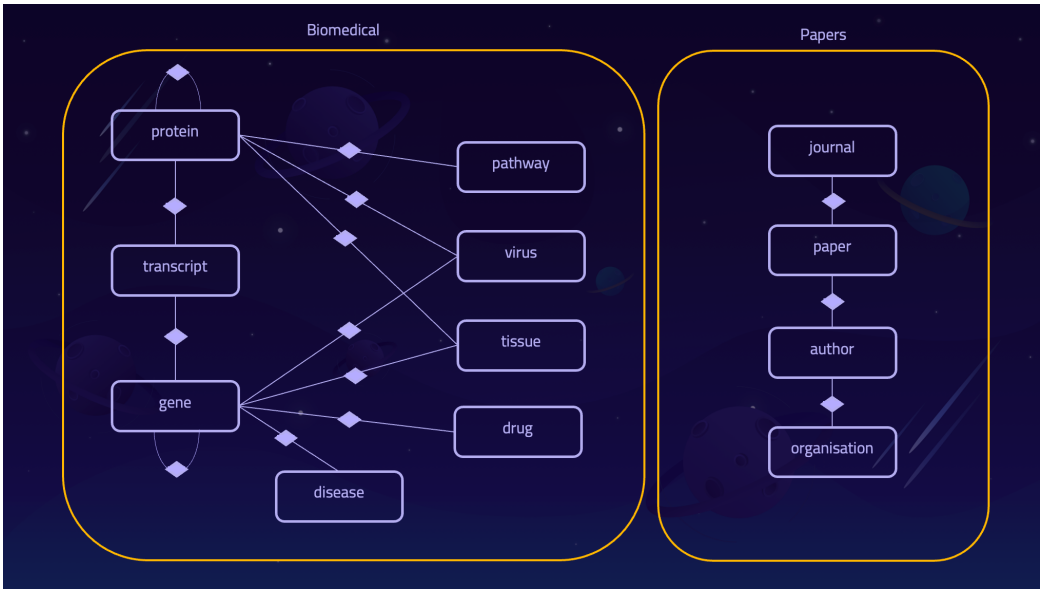
5. DGIdb: We've taken the Interactions TSV which includes all drug-gene interactions.

6. Human Protein Atlas: The Normal Tissue Data includes the expression profiles for proteins in human tissues.

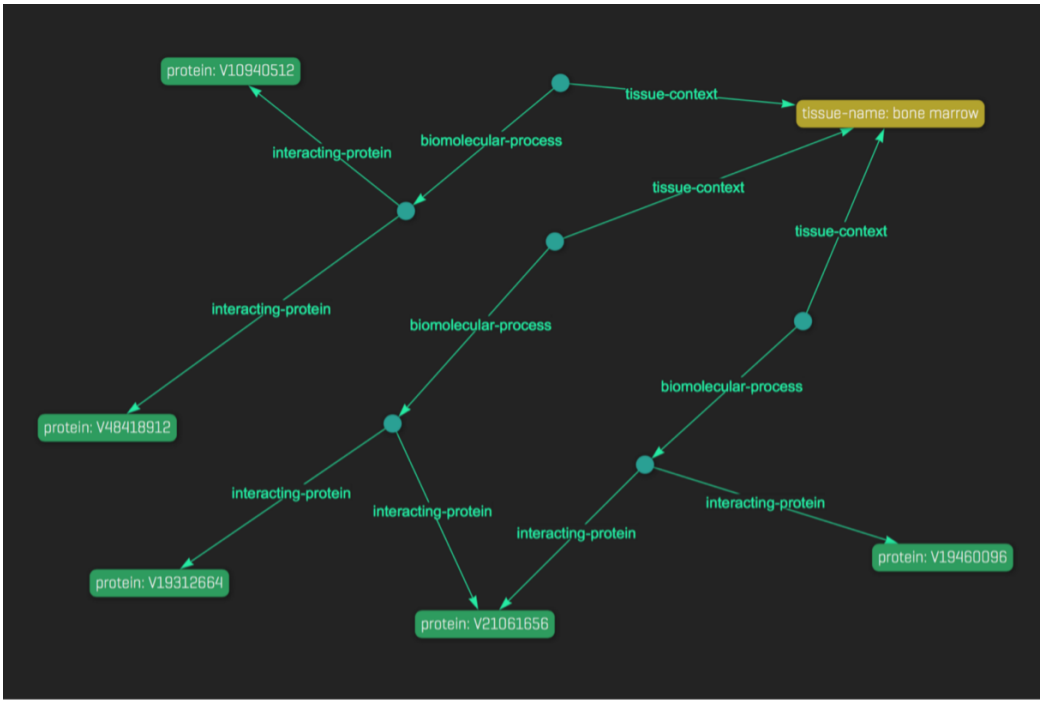
7. Reactome: This dataset connects pathways and their participating proteins.

8. Disgenet: We've taken the curated gene-disease-associations dataset, which contains associations from Uniprot, CGI, ClinGen, Genomics England and CTD, PsyGeNET, and Orphanet.

9. SemMed: This is a subset of the SemMed version 4.0 database



The schema for TypeDB — Covid represents biomedical components and papers. Below is a high-level overview of the main entities while the schema file can be found here. To load the source data into TypeDB, we wrote a migrator using the TypeDB Python driver.



The key benefits of using TypeDB in the creation of this knowledge graph are as follows:

Firstly, TypeDB's type system allows us to easily represent heterogeneous networks of biomedical data. Moreover, the simplicity and elegance of TypeDB's query language, TypeQL, allows even non-technicians to read and understand the model being created.

A basic TypeQL schema where we model genes and proteins can look as simple as this:

```
define
gene sub entity,
owns name,
plays encode:encoding;
protein sub entity,
owns name,
plays encode:encoded;
encode sub relation,
relates encoded,
relates encoding;
name sub attribute, value string;
```

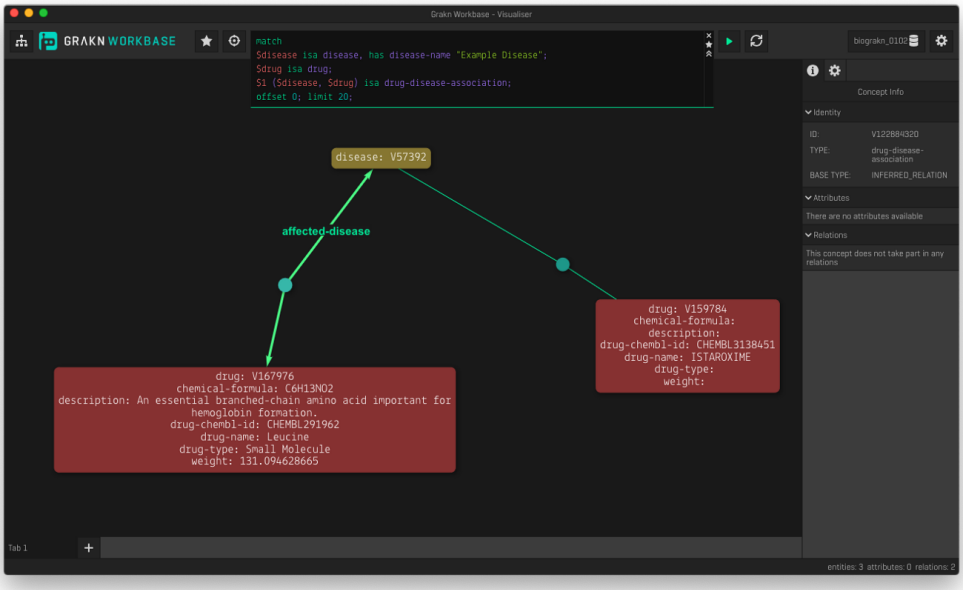
Here, we've defined two entity types, `gene` and `protein`, each with the `name` attribute. Both entities are connected through, respectively, the roles of `encoding` and `encoded` in an `encode` relation.

TypeDB's type system is highly flexible — we can change the schema at any point during the database's lifecycle. We therefore can maintain the consistency of our data, but unlike with a relational database, we can change the data whenever we need to without the need for expensive migrations.

In TypeDB's type system, entities, relations, and attributes are all first-class citizens. This brings great flexibility when modelling complex domains. A good example is the ability to model nested relations. This becomes useful when, for example, we're trying to build a model to localise a particular protein interaction in a tissue:

```
define
protein sub entity,
plays protein-interaction:interacting;
tissue sub entity,
plays process-localisation:locating;
protein-interaction sub relation,
relates interacting,
plays process-localisation:located;
process-localisation sub relation,
relates located,
relates locating;
```

This models the `tissue` entity as having a `process-localisation` relation with the `protein-interaction` relation — a nested relation. This level of expressivity is useful, as it means our model can be more closely guided by the needs of the application.



Second, TypeDB's ability to perform automated reasoning during query runtime enables us to reason over our data at scale. This allows us to find inferred information in our database — for example, inferred drug-disease associations.

To do this, we write rules in our TypeDB schema that enable it to reason over our database. One such rule is shown below. We're inferring relations between proteins and genes, where we know the transcript for which a gene encodes and the protein that it translates:

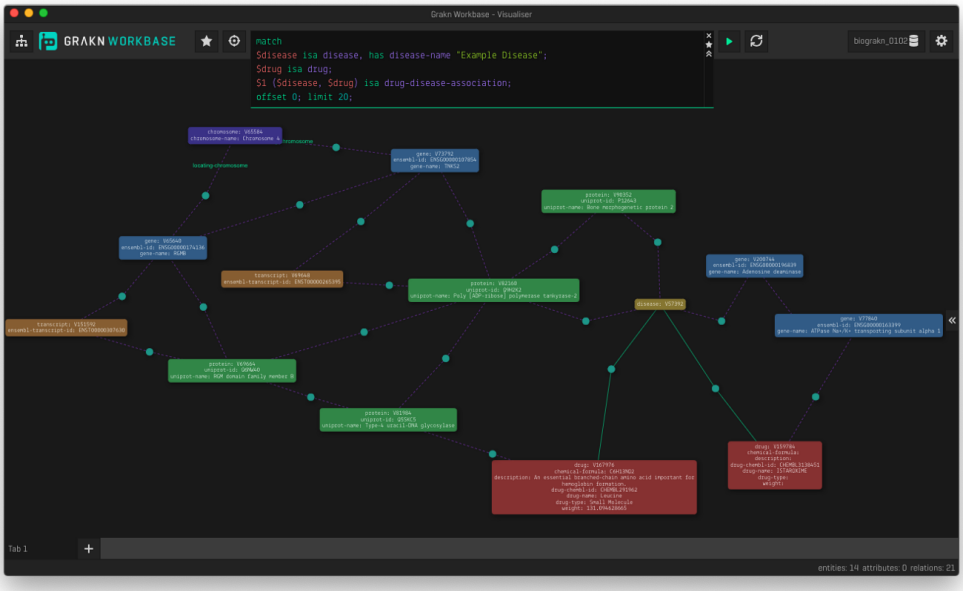
```
rule gene-transcript-protein:
when {
   $gene isa gene;
   $transcript isa transcript;
   $protein isa protein;
   ($gene, $transcript) isa transcription;
   ($transcript, $protein) isa translation;
} then {
   ($gene, $protein) isa encode;
};
```

This rule is useful if, for example, we had not ingested any data connecting

genes directly to proteins, but only connections between genes, transcripts and proteins. Leveraging the rule above, we would then be able to ask the question below and get an answer. With a traditional database we wouldn't get an answer.

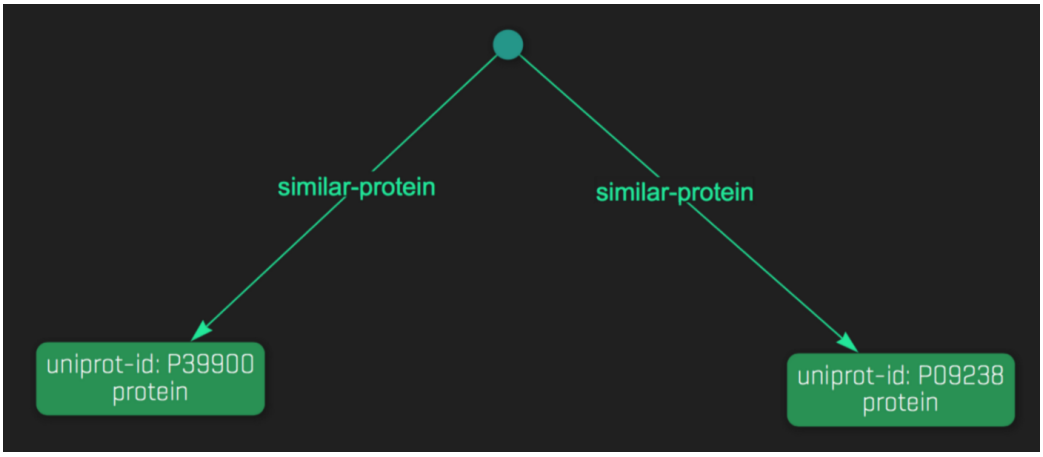
```
>>> match $g isa gene, has identifier "29851";
$p isa protein, has uniprot-id $up; ($g, $p) isa encode;
get $up;
>> {$up val "Q9Y6W8" isa uniprot-id;}
```

Of course, this is just a simple example. We can also use rules for more advanced hypothesis generation. For example, below is the visualised result of potential drug targets for a sample disease:



Both relations shown in the graph are inferred, i.e. they don't actually exist in the database. But if we fetch the explanations for those inferences, TypeDB will show the network that make up those inferences. For example, one of the inferences is due to the associated genes being located close to one another on Chromosome 4.

TypeDB's ability to perform automated reasoning allows us to work at a higher level of abstraction. As we saw in the example above, this means



directly asking for drug-disease associations, without worrying about all the possible interpretations of how they could be connected.

2.2 Bringing Biological Context to Newly Generated Insight

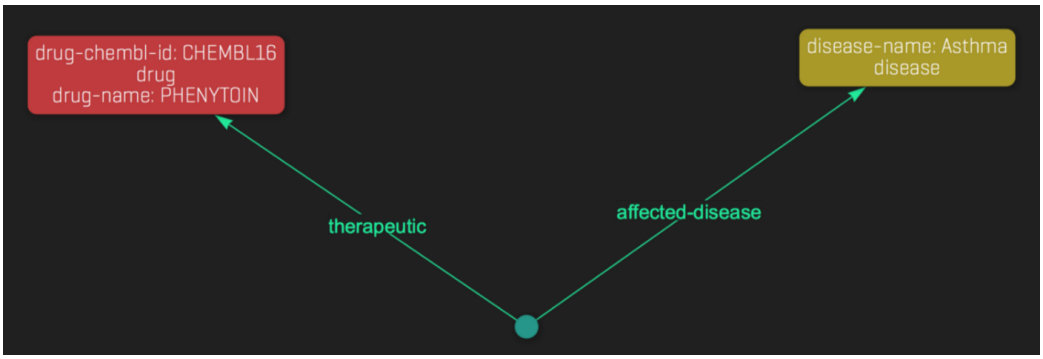
Having created our model in TypeDB, we can also look at how we can connect it with other software components relevant to drug discovery. That could include text mining tools to annotate medical papers or machine learning frameworks for predicting gene-disease associations.

What's particularly interesting in TypeDB, especially in drug discovery/repurposing, is the ability to learn and make predictions over highly contextualised data. This would differ from an approach where we ingest flat data into a learner. For example, we could fetch a subgraph from TypeDB into something like KGCN to make link predictions between genes and diseases. Those predictions can then be put back into the knowledge graph, and can serve as evidence for hypothesis generation.

Here, the key is to leverage the use of TypeDB as the unified representation of our knowledge. This way, we can provide important biological context to any new insight that we generate. Below are a few examples of such insights that we could generate:

First, if we used a sequencing algorithm which found similarities between sequences of `proteins`, we could insert these as sequence similarity relationships between two protein entities. As an example: below you can see how

we would model a sequence similarity between proteins `P09238` and `P39900`.



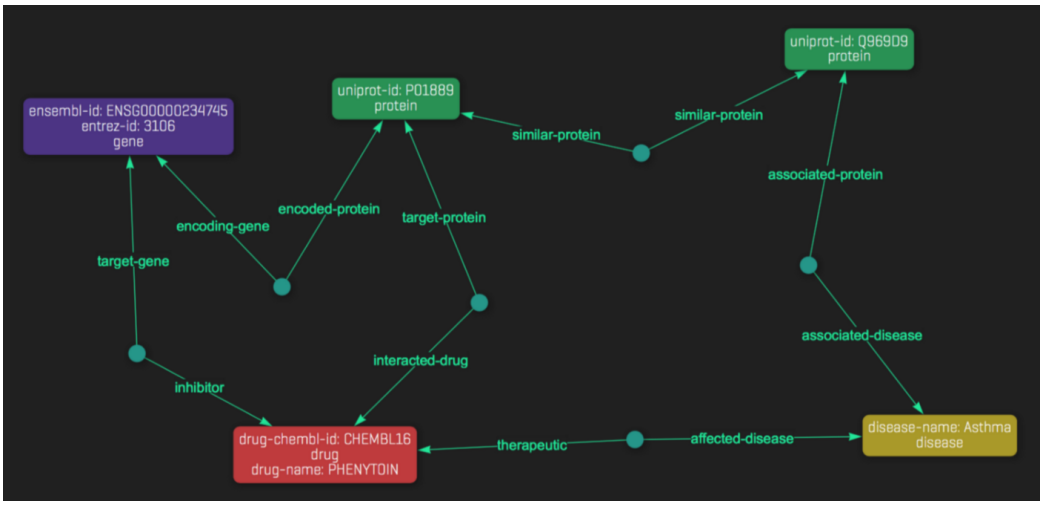
Second, we can then define TypeDB rules in the schema to find new insights in the data. Below is an example of such a rule. This particular rule creates a new `drug-disease-association` relation when TypeDB finds two proteins with a sequence similarity, where one protein is a target for a disease and the other relates to a drug.

```
rule drug-disease-association-when-seq-similarity:
when {
 $disease isa disease;
 $protein isa protein;
 $protein2 isa protein;
 $protein != $protein2;
 $drug isa drug;
 (associated-disease: $disease, associated-protein: $protein) isa
    protein-disease-association;
 (similar-protein: $protein, similar-protein: $protein2) isa
    protein-similarity;
 (target-protein: $protein2, interacted-drug: $drug) isa
    drug-protein-interaction;
} then {
 (affected-disease: $disease, therapeutic: $drug) isa
    drug-disease-association;
};
```

With this rule defined, we can now directly query for drug-disease-association relations, even though we may not have any such data instances in the database. Due to the rule we defined, TypeDB will infer and find candidate drugs. This is what such a query would look like:

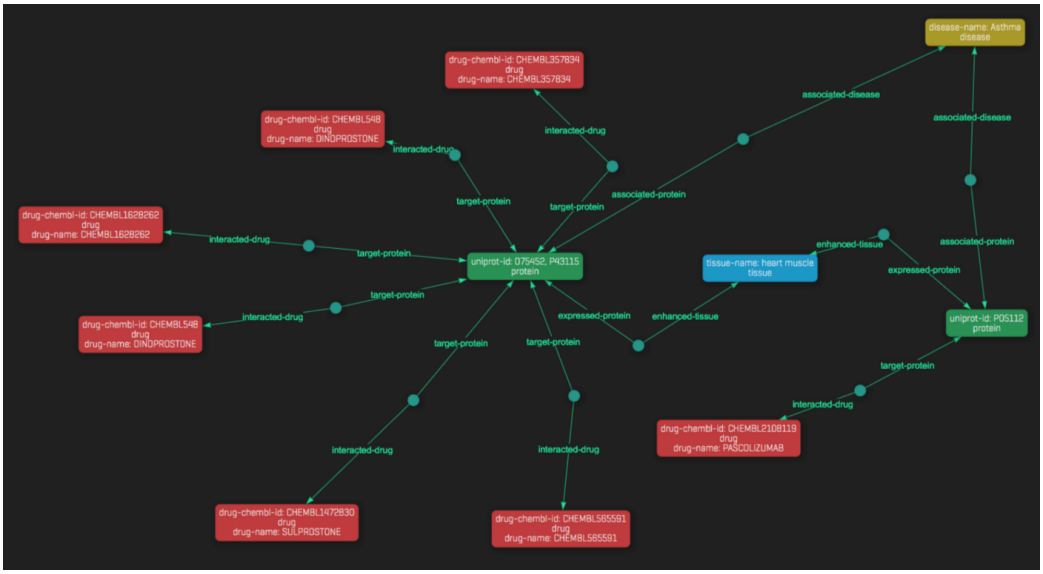
```
match
$disease isa disease, has disease-name "Asthma";
$drug isa drug;
$r (affected-disease: $disease, therapeutic: $drug) isa
    drug-disease-association;
```

This looks for potential candidate drugs against Asthma, and below is what it would return. We see that the drug `PHENYTOIN` is connected to Asthma. Remember that no direct associations actually exist in the data — these relations were created (inferred) by TypeDB.



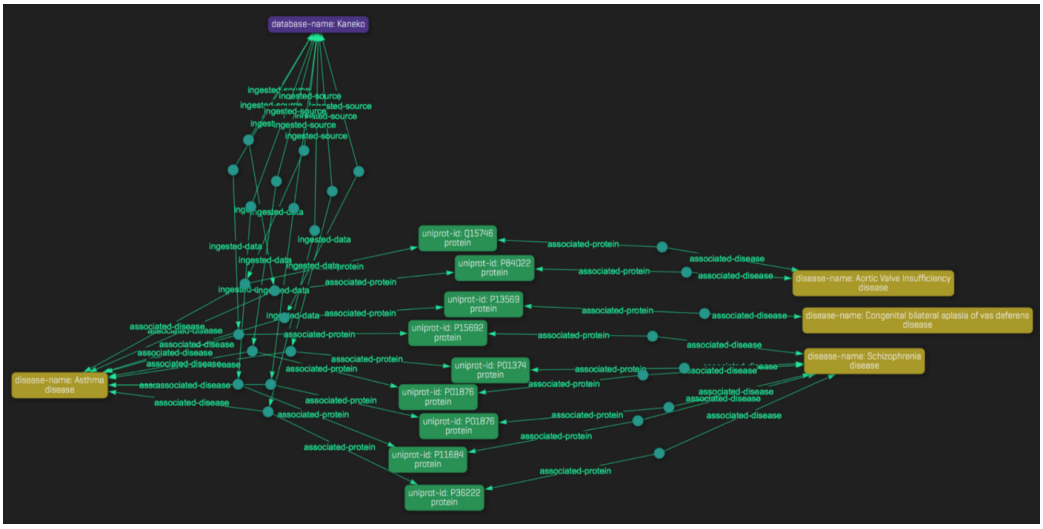
If we want to explore why TypeDB returned this answer, we can press explain on the relation, and the graph will expand (see below). We can then see that protein `Q969D9` is associated to Asthma, and has a sequence similarity with `P01889`, which has a relation with the drug `PHENYTOIN`. But as none of our original datasets included protein-drug associations, this relation is also inferred through another rule, which states that if a disease is associated with a gene (data from DGIdb), that disease should also be associated with the proteins which that gene encodes. Therefore, as `PHENYTOIN` has been reported to be an inhibitor to gene with entrez-id `3106`, the protein it encodes, `P01889`, also becomes associated with it. Exploring such transitive relations will be of great interest to drug discovery research.

Third, when exploring the biological context of new insights and comparing the network of neighbourhoods of biological components, traversal type queries are important. These can reveal paths connecting components



that may not have been anticipated initially. Such queries are easily done using TypeQL, but would be computationally too expensive for a traditional databases. Below we demonstrate a query that asks for connections between asthma, the heart muscle, proteins, and drugs:

```
match
$disease isa disease, has disease-name "Asthma";
$tissue isa tissue, has tissue-name "heart muscle";
$drug isa drug; $protein isa protein;
(associated-disease: $disease, associated-protein: $protein) isa
    protein-disease-association;
(expressed-protein: $protein, enhanced-tissue: $tissue) isa
    tissue-enhancement;
(target-protein: $protein, interacted-drug: $drug) isa
    drug-protein-interaction;
```



Fourth, we may also want to query and identify proteins which are related to a disease from one particular study, and explore how they relate to diseases from other studies. We illustrate such a query below:

```
match
$asthma isa disease, has disease-name "Asthma";
$protein isa protein; $disease2 isa disease;
$database isa database, has database-name "Kaneko";
$disease2 != $asthma;
$pda (associated-disease: $asthma, associated-protein: $protein)
    isa protein-disease-association;
$di (ingested-source: $database, ingested-data: $pda) isa
    data-ingestion;
$pda2 (associated-disease: $disease2, associated-protein: $protein)
    isa protein-disease-association;
```

This asks for proteins which are common to Asthma and have been reviewed by Kaneko et al. (2013), and are also associated to other diseases. We can create this query easily by asking for all `protein-disease-association` relations that are associated with the `Kaneko` database entity:

3 Conclusions

Recent advances in omics technologies have created a wealth of high throughput genome-wide scanning data. Therefore, the field of life sciences must keep innovating and exploring new techniques for effective and scalable analysis of such data. In this article, we looked at how using TypeDB to represent biomedical data can accelerate the drug discovery process. In summary, there are two areas where TypeDB accelerates this process:

1. **Ingesting and Integrating Biomedical Data into TypeDB**

   Due to the unpredictable nature of biomedical research, and the dynamic and constantly changing requirements of biomedical communities, TypeDB's type system facilitates the rapid ingestion and integration of new data. Its schema language, TypeQL, through a type-based hierarchical model and hyper relations, gives a level of expressivity to model non-uniform biomedical data closely guided by the needs of its intended application.

2. **Bringing Biological Context to Newly Generated Insight**

   New insights and data generated through sequencing or machine learning algorithms need to be understood in their biological context to advance the knowledge discovery process. TypeDB allows for the easy ingestion of such new data. What's more, to leverage this biological context, we can use rules to find inferred relations between unconnected biological components, which may suggest new candidate drugs. The ease of doing traversal type queries is also essential in this process.