Automatic hierarchical data extraction from relational databases

Student: Diego Arenas Contreras
Supervisor: Prof. Peter Buneman

Master of Science
Data Science
School of Informatics
University of Edinburgh
2016

Abstract

The problem of extracting hierarchical data from a curated relational database is addressed in this project. It is assumed low or non existent prior knowledge about relationships in the database. The approach is using the metadata available of the database and also the use of reverse database engineering to make educated guesses about relationships between the data or suggesting to the designer paths of data integration. The objective is to get the hierarchical data to deliver the data in a NoSQL format. Methods like reverse engineering, metadata exploration, graph analysis, statistical analysis and automatic query construction are used in this project. Statistical methods are used as heuristics to find all possible candidate keys and to find relationships between tables that are not explicitly defined in the database.

Acknowledgements

I would like to thank my project supervisor professor Peter Buneman of the School of Informatics at The University of Edinburgh. For the possibility to work on this project and for his guidance during this period. This report marks the end of a year of personal and professional growth.

I would also like to thank my parents, Maggie and Raúl, for providing me their unconditional support. Thank you to my family, my friends and classmates. For all the good and bad things in live because without them I couldn't be here.

Diego Arenas C.
Edinburgh, Scotland, August 18, 2016.

Contents

Chapter 1

Introduction

Many relational databases encode hierarchical data. Can we extract one or many hierarchies automatically?

In this dissertation project report we provide heuristic methods to solve the problem of how to automatically extract hierarchical data from a relational database.

The system's heuristics will make use of the metadata of the relational database. The system will use the definitions of the database as they are stored in the database engine and also it will apply analysis using graph theory and statistical analysis. The statistical analysis can be used to explore any data source.

The solution is implemented using PostgreSQL database engine and Python programming language, but it can be implemented for most of the database engines in the market and also using any programming language that allows connections to the database engine.

The project is implemented in three connected phases or stages.

The first stage is the analysis of the metadata structures on the database engine. This exploration will lead to identify and classify the type of tables and relationships in the database. We will be looking for those kinds of relationships that may represent a hierarchical structure. The IS-A type relationship will be searched for the algorithms and heuristics of this project.

The second stage, is a graph analysis phase. This phase will make use of the data collected during the first phase. A directed graph is created using

the relationships between the tables as edges and the tables as nodes. In this phase is important to find the existence of loops, that would indicate recursion in the connections. In the presence of loops it is necessary a way of solving them using heuristics before to retrieve the hierarchical data from the database. The directed graph is also to visualize the database and gain a better understanding of the organization of the database. Some graph measures like degrees or connected components are calculated to explore in detail the database.

The last stage is statistical analysis. In this stage heuristics are based on data frequency calculations to provide enough information to make educated guesses about the content of the database. Exploratory data analysis with statistical measures are computed. The statistics will provide good understanding about the content in the database but also they are used to look for possible relations between tables that were not present in the definition of the database.

The use of metadata as a main resource is based on its ability to provide information at low processing times and low storage space requirements. We avoid any real-time query to the database due to performance reasons; instead, there is a pre-processing phase in the first stage. Several metadata tables are created and organised in ways to facilitate and optimise data exploration and analysis in the following stages.

The reason to make it automatically is that we can leave the system running in background mode and then analyse the results, also all the hypothesis about hierarchies will be formed considering all the information available.

In this report *RDBMS* or *database engine* will be terms to refer to the database administration software. And the term *database* or *iuphar* will be used to refer to the data bank where the data for this project is stored. The RDBMS will handle the database.

In section 2 we present the background section of the project, references to previous work and a the setting of the work it is attempted in this project. Section 3 describes the conceptual design work and the actual implementation. Also contains the problems or difficulties and the suggested solutions. In section 4 there are results and critical analysis about the project. And finally in section 5 there are remarks and observations, unsolved problems, and suggestions for further work.

Chapter 2

Background

2.1 Literature Review

The relational data model [Codd, 1970] present advantages to the graph data model, but relational databases do not naturally represent hierarchical dependencies between data.

There are ways to represent hierarchical data in relational data models but there is not a way to specify this relation. The relation must be inferred by the user. The explicit way of representing hierarchies in a relational database model is using constrain references. The link between two tables can be understood as a hierarchical relationship in some of the cases, that is why expert analysis is needed to identify this relationship.

Elicitation Process. Is process of finding the structure of the database through the analysis of its structure and content. Having the original structure it is possible identify hierarchical relationships between tables. The problem of elicitation in databases has been subject of research with different approaches and techniques. Techniques of reverse engineering [Lammari et al., 2007] has been suggested or simply Data Reverse Engineering [Henrard and Hainaut, 2001].

The problem of Generalization/Specification (G/S) [Smith and Smith, 1977] is an example of hierarchical representation of data in Codd's relational model. The authors mention that the relational schema support two types of abstraction, Aggregation and Generalization. We are interested in the latest concept as is the one used to represent IS-A relationships. Aggregation type

relations are hierarchical data representation, but they will be discarded for the purposes of this project as they represent simple relationships between tables and not necessarily semantic extensions of an object.

In [Lammari et al., 2007] the authors propose the MeRCI reverse engineering approach [Akoka et al., 1999] to extract generalization hierarchies from relational databases. The method involves the analysis of the DDL specifications then and optimization of the structures extracted and finally a conceptual schema. They define the applicability domain of an attribute or a set of attributes as the set of possible values that the attribute can take. The domain of an attribute A is denoted as Dom_A. They present three definitions to deal with attribute domains:

- *Mutual existence dependency* when $Dom_A = Dom_B$.
- *Exclusive existence dependency* when the intersection is empty, $Dom_A \cap Dom_B = \emptyset$ and,
- *conditioned existence dependency* when $Dom_A \subseteq Dom_B$. A set of rules are defined to check data dependencies and identify hierarchical structures from the relational model.

For [Lammari et al., 2007] the focus is in attribute domain. In this report we present a similar approach but based on the frequency of the attributes rather than in the domain. The definition of *attribute overlap* will be defined in the next section.

In [Karagiannis, 1994] the authors present a method to extract object oriented model from a relational model. They semantically enrich the object oriented from a relational database. Their model is a semantic extension of object oriented models. Their model captures semantic relationship from the relational model. They identify three dimensions of the relational model: Classification/Instantiation; Generalization/Specialization and Aggregation/Decomposition. And from there they create a object oriented model. In [Fong, 1997]

The approach in [Alhajj, 2003] is extracting the model from a legacy database. This particular idea is interesting for this project because is based on the use and content of the database rather in the original definition of the database. A legacy database has similarities with a curated database. Both

can have a disperse community of developers, the same dispersion applies to the documentation. In their approach user involvement is minimized, similar to what we intent with this work. They suggest improvements to the relational model analyzing relationships of the type many-to-many, in this report we present a way to identify and explore possible connections between tables that have a many-to-many relationship from a statistical approach.

In [Henrard and Hainaut, 2001] a database reverse engineering approach is presented. It is based in the analysis of the source code of the program. The approach is looking to reconstruct semantic connections between the data that are not represented in the relational data model. Data dependency is the name of the process and consists of the use of foreign keys for representing and analyzing connections inside the database. The approach includes a first step where the analysis participate in interviews and demonstrations to gain knowledge about the database. In this project the interaction are minimal or non-existent since the process is designed to be automated. The authors include two more phases. The analysis of the DDL code to create a *physical schema*. And finally a data structure conceptualization, consists in detecting non-conceptual structures to refine the final data model.

This project is highly based on referential integrity constraints [Date, 1981] and will explore and analyse the database and its content through the relationships between the tables, it will try to find connections that were not explicitly defined in the definition of the database. The system analyses the structure of the database using its metadata. Graph analysis and statistical analysis are used to find whether or not there exists a IS-A relationships.

The system considers the content of the database and not its design intentions. The use of the database may change over time, so it is necessary to consider the current state. This approach gives a flexible understanding of the use of the database.

The common approach in literature for early stages of database reverse engineering to use the DDL code to extract the database structure [Henrard and Hainaut, 2001] [Lammari et al., 2007] [Yeh et al., 2008] [Alhajj, 2003]. Our approach makes use of the metadata available in the database engine allowing to create connections to different databases while the system is running. The system will request all the necessary data and does not need to be fed with any file or script with the database definition.

2.2 Problem Statement

Several databases have a hierarchical nature but they are often represented in a relational model. One of the aims of this project is to see if the hierarchical representation can be extracted from the relational model and see if this process can be reversed.

How to extract hierarchical data from a relational database in an automated process is the problem of this MSc Dissertation Project.

The system uses the metadata of the relational model, graph analysis and statistical analysis. The use of this methods make the process reproducible for almost any database engine and programming language. It is also possible to create a distributed version using Hadoop-alike systems to process data, this discussion is included in section 5.

The **iuphar** database[1] is the database used to test this project. The iuphar database is a curated database with pharmacological data. It receives updates from contributors and it is possible that no single person knows the complete structure of the database. The objective of this project is to extract hierarchical data from this database.

The implementation of the analysis and data processing is on PostgreSQL and Python, but it can be implemented for any database engine that stores metadata about the databases and in any programming language able to establish a connection with the selected database engine.

The exploration of metadata structures can be done on different database engines like MySQL, SQL Server, DB2, Oracle Database, etc. allowing the implementation of this project idea for most of the database tools available in the market. The algorithms and methods implemented for PostgreSQL can be implemented for other database engines. The structures to store metadata will not change, they also can be implemented in any programming language.

2.2.1 Assumptions

It is assumed that the test database has an entity relation (ER) model approach and make use of referential integrity. This means that the references can be found in the definition of the database. Also it is assumed that a

[1] http://www.iuphar.org

primary key is the unique identifier for a tuple in the tables of the database.

2.3 Methods and techniques

Metadata Exploration. A revision of the PostgreSQL documentation was the first part of the project. All the important tables were identified with the aim of getting information about the database structure. Then new data structures were created to store this information and complementing it with calculations like number of records, number if distinct values per attribute and some classification of the table and references that will be explained in section 3.

Graph analysis. Reference constraints are used to create a directed graph representation of the database. The analysis of the graph will lead to an understanding of the structure. From this analysis we present a classification of the nodes in five categories and of the edges in three categories. The objective is identify those categories with hierarchical structures.

Statistical Analysis. A data frequency calculation will be computed for each attribute in the database. the data frequencies of each attribute will be used to compute a new measure called *attribute overlap*. The interpretation of this measure is the level of overlapping between two attributes from different tables. This will help to decide if two tables are maybe related and also is useful to decide the direction of the hierarchy analyzing the results of the data frequencies of each table.

2.4 Possible cases

The system will gather data and apply categories to differentiate tables and relationships. These classes or labels are based on the use of primary and foreign keys in the tables and also in the relative position of the table with respect to the other tables in the relational data model.

Depending on the use of attributes as primary or foreign keys, tables are classified in one out of five types suggested in this project.

Table type	Description
Root Table	Table with no parent table, it doesn't contain a reference to other table on top. There are references pointing to this table.
Linking Table	Represent an intersection between two tables. Also the PKs of this tables is formed by its FKs.
Foreign Relation	A table with FKs that are not part of the PK of the table.
Composed PK with FK	A table with FKs where some of them are part of the PK of the table.
Isolated Table	A table without references to other tables of the database.

The same process of classification is applied to the relationships between tables.

Reference type	Description
Exact key	When the referenced attribute is the PK in the referencing table.
Part of the key	When the referenced attribute is part of the PK in the referencing table.
Not key	When the referenced attribute id not part of the PK in the referencing table.

This classification of tables and references will allow to visualize the tables and its relationships in the database using directed graph representation. Tables are the nodes of the graph and the references are the edges between nodes; as the reference can be interpreted with a direction then it is possible to create a directed graph and to represent the different types using different colors.

In Figure 2.1 we can see and example of six tables and the relationships between them. The tables are ligand2synonym_refs, ligand2synonym, ligand, ligand_structure, ligand2meshpharmacology, peptide and reference. In the figure, **ligand** and **reference** tables are **Root Tables** because they are receiving references from other tables and these two tables do not have any dependency to any other table in the database. Table **ligand_structure** is an **Isolate Table** because it has no connections to other tables. **lig-**

and2synonym table is a **Foreign Relation** type, meaning the attributes with referential integrity not part of the primary key of the table; the reference to ligand table s showing a hierarchical relationship between these two tables. **ligand2synonym_refs** is a **Linking Table**, meaning its primary key is formed by its two foreign keys. And finally **ligand2meshpharmacology** table is a **Composed PK with FK** type, meaning its primary key is a combination of own attributes and attributes with referential integrity.

At the same time, we can use the example to explain the classification of the references. The relation between **peptide** and **ligand** table is an **Exact key** type, meaning the primary key of peptide is a foreign key referencing to the ligand's primary key. The relation between **ligand2synomym_refs** with **ligand2synonym** and reference tables is a **Part of the key** relationship, meaning the primary key of **ligand2synonym_refs** is composed by the foreign keys to **ligand2sysnonym** and reference tables. And finally, the relationship between **ligand2synonym** and **ligand** is of the type **Not key**, which means **ligand2synonym** is not using foreign keys as part of its primary key.

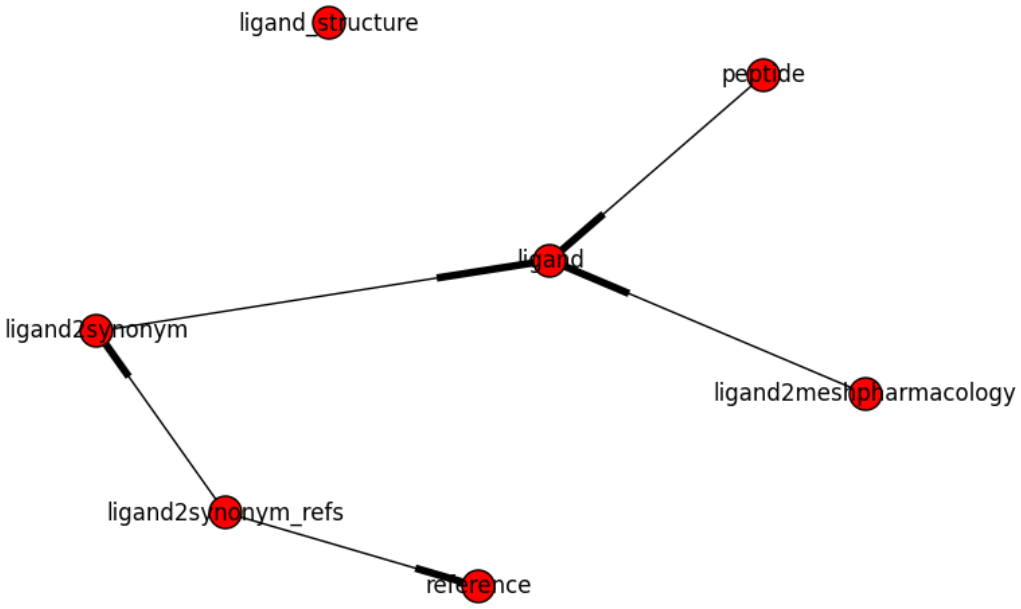

Figure 2.1: Example of tables and reference types.

Chapter 3

Work Undertaken

As it was mentioned before, the process to address this problem consist of three phases. The first one is the exploration of the metadata. A graph analysis and finally a statistical approach using data frequencies of the data values to get more information about possible relationships among the data. The diagram is in Figure 3.1.

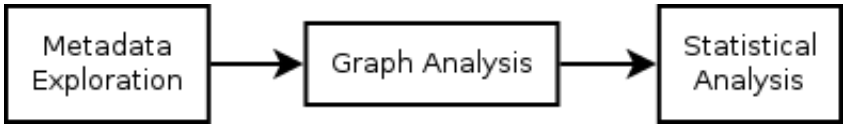

Figure 3.1: The three stages of the process.

Technological tools and libraries used in this project are:

- PostgreSQL 9.5 database engine[1]: As a database engine where the iuphar database is stored. Using PgAdminIII client of PostgreSQL is possible to execute and test queries against the database.
- SQL language: The required data structured are created as tables in the database. This will take advantage of the performance of the server where the database engine is installed.
- Jupyter notebook environment for Python development[2]: The Python code is written and tested in a notebook of the former iPython environment, allowing a faster development and testing of the code.

[1]https://www.postgresql.org
[2]http://jupyter.org

- Python programming language: The logic of the system is implemented in Python, auto-query generation is implemented in Python functions. The SQL commands are sent to the database using a Python library for this purpose. The graph analysis is computed using a Python library for this purpose.

- networkx[3] Python library: This library offers several functions to perform a graph analysis and also basic visualization tools.

- psycopg[4] Python library: This library is used to establish connections with PostgreSQL. Allows to send sentences and perform changes in the database with commits.

- Gephi[5]: A software for graph visualization and exploration.

The coding component of this project is implemented using two tools. Python is used to create the logic of the system. **Jupyter Notebook** is used to write Python code. Python is connected to the PostgreSQL database engine using the **psycopg** library. This library allows to send queries and sentences to the database.

Python is used to structure the process but is not intended to use it to perform any high demand data processing. The data processing tasks are implemented in SQL code and they will be sent to the database engine to perform these high demand data processing tasks. This design criteria is relying on the capacity of the server where the database engine is installed. It is expected that if the server is capable of storing considerable amount of data, will be capable enough to perform queries and analysis of its databases. This assumption applies to any size of database.

The SQL code is written in standard SQL[6], avoiding the use of specific functions implemented in specific database engines, so the code can run in different database engines. A new connection in the Python code has to be implemented to the specific new database.

The term **Metadata** can be understood as *data about data, that is not the original data*. This data about the data can be found in almost any database engine in the market.

[3]http://networkx.github.io
[4]http://initd.org/psycopg/docs/
[5]https://gephi.org
[6]http://www.contrib.andrew.cmu.edu/ shadow/sql/sql1992.txt

The following subsections will explain the work undertaken with each one of the tools selected for this project.

3.1 PostgreSQL

The first phase is the exploration of metadata in PostgreSQL. It is necessary to design and create structures with the table and attribute names, and with primary and foreign keys. This information is used during the graph analysis and statistical analysis phases, also will be used for auto-generation of the queries to explore the database and to export the results.

The first goal for this metadata exploration is to have enough information to generate a query using table name and all attribute names. Then, it is expected to have enough information to generate a simple join between two tables using single attributes as part of the JOIN instruction. Accomplishing this goals implies that the metadata collected has the required information about references between tables. The constraint references are crucial to identify to create a directed graph with the database tables.

PostgreSQL **Information Schema** contains the metadata about the database in the server. There are tables with information about table names, attribute names and its relationships. The constraints are also stored in this schema. The information schema can be accessed in PostgreSQL 9.5 in the **postgres** database, under **Catalogs**, then **PostgreSQL (pg_catalog)** and in **Tables** there are the relevant structures with metadata for this project. The tables that are be used for this project are the followings:

- pg_class
- pg_attribute
- pg_constraint

A data structure with every attribute of the database is created. It contains table name, attribute name, a flag indicating if the column is a primary key, a flag indicating if the column is a foreign key, and the foreign table and attribute names if a reference is found.

Using this basic structure with metadata it is possible to aggregate information, and to create other data structures to generate a directed graph, to infer information about the type of relationships between the tables and

to identify whether or not attributes are used as foreign or primery keys in their respective tables. The SQL code is in Appendix. A.

3.2 Metadata exploration

Having identified all the attributes and its corresponding tables, knowing if the column is a primary key or a foreign key. The following data structure is created to facilitate the process of identifying relationships.

Attribute	Description
relfilenode	Code of the table in PostgreSQL.
relname	Name of the table.
attname	Name of the attribute or column.
datatype	Type of the attribute.
attnum	Position number of the attribute in the table.
oid	ID of the attribute in PostgreSQL.
conname	Name of the primary key constrain.
pk	Flag to indicate if its a primary key.
fk	Flag to indicate if its a foreign key.
link	Flag to indicate if its a link table.
foid	ID of the referenced column.
fconname	Name of the constrain of the foreign key.
frelname	Name of the referenced table.
confkey	Array with position numbers of the attributes used in the constrain.
fattnum	Position number of the attribute in the referenced table.
fattname	Name of the attribute in the referenced table.

3.3 Graph analysis

The second phase is the graph analysis. The aim is to search for loops among the connections of the database. The references extracted from the metadata are enough to build a directed graph.

The nodes represent the tables of the database. The edges are the constrain references between tables.

This directed graph allows to run a visual check about relationships and also to apply graph measures to analyse the results. A graph representation with Gephi is in Figure 3.2.

Beside a visual check for loops in the graph, some measures for graph analysis are computed using the **networkx** Python library. Networkx contains several algorithms and measures for graph analysis and can be used for basic graph visualization with **matplotlib.pyplot** Python library. For a better visualization it is used the software **Gephi**.

A directed graph can be build using tables as nodes and references as edges. Information like the number of nodes, number of edges, number of connected components, average degree, number of cycles can be extracted using the networkx library. The measures are presented in the following table.

Measure	Value
Number of nodes:	159
Number of edges:	231
Average in degree:	1.4528
Average out degree:	1.4528
Is strongly connected?	False
Is directed acyclic graph?	True

The directed graph can be transformed into an undirected graph and calculate a couple of aditional measures like if whether or not there are cycles and how many connected components (CC) are in the graph.

There are 6 connected components. A big CC of 153 nodes, a second CC of two tables [celltype] and [celltype_isa]; and four isolated tables that forms a CC by themselves [version], [ligand_cluster_old], [ligand_physchem] and [ligand_structure].

There are 78 cycles in the undirected graph of the database. An example of cycle with 5 tables is presented in Figure 3.3.

Cycles are not present in the directed graph. It is possible to form hierarchies using the nodes. Duplication may occur due to references to more than one table. This situation implies at the moment of forming hierarchical structures to extract the data one or more path can be formed and it is the designer responsibility to decide what path to use.

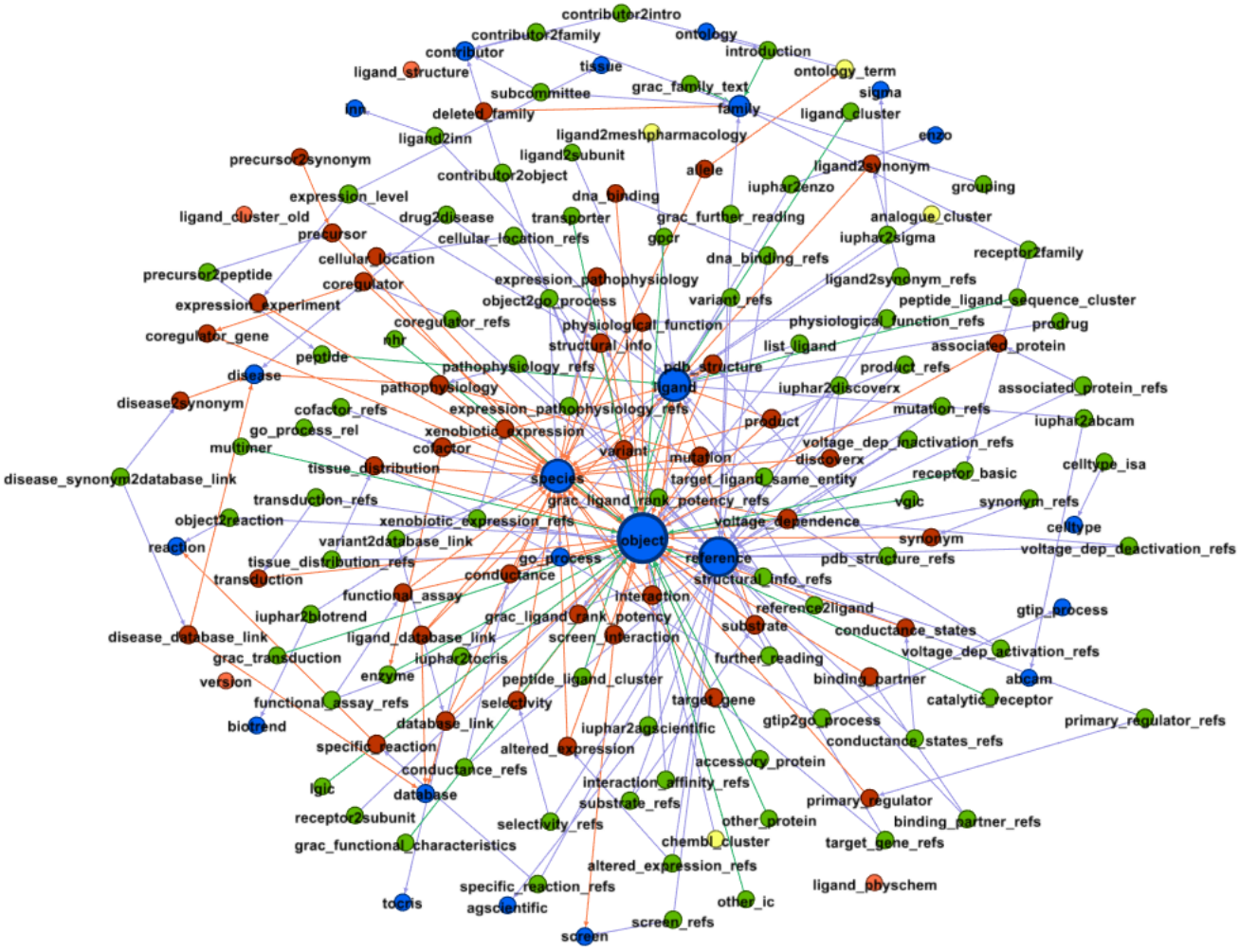

Figure 3.2: Graph representation of the iuphar database with Gephi.

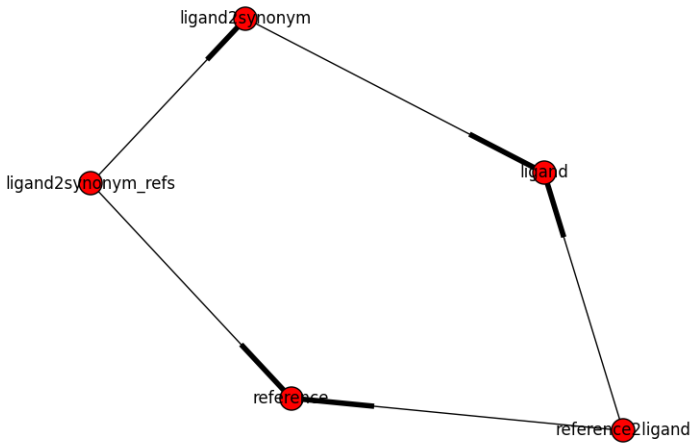

Figure 3.3: Example of a cycle in the undirected graph of the database.

3.4 Statistical analysis

The third phase is the statistical analysis. There are two main groups of analysis in this phase. The first is analyzing the metadata and the other one is analyzing the data, meaning the data values stored in each attribute.

It is necessary to run a couple of pre-processing tasks to prepare the data. These tasks are the computation of ***data frequencies*** and the ***rate of joins***. These two new data structures will contain enough information to suggest two novel implementations of use of metadata in data exploration processes.

These implementations are extensions of the idea of extracting hierarchical data from a relational database but are not bounded to this specific problem, they can be useful on any data exploration problem whenever is required to provide sound information about the database.

3.4.1 Data frequencies

From the data gathered in the first stage of the project, it is simple to autogenerate queries to calculate data frequencies of the data values of each attribute of each table in the database and store it with the following structure.

Attribute	Description	Type
entity	Table name in the database.	VARCHAR
field	Attribute name.	VARCHAR
data_value	Value that the attribute can its value from. Each data value appears only once per field and entity.	VARCHAR
frequency	Number of times the data_value was found in that field in that entity.	INTEGER
percentage	Calculation of the frequency of the data_value over the total number of records of the table.	DOUBLE PRECISION

It will be necessary to get tuples with table and attribute names of every attribute in the database. This pairs are stored in a list to auto-generate queries to send to the database and calculate data frequencies. An extract of the code is in Listing 1.

The utility of having the frequencies computed is that through simple rules it is possible to pointed out information about the data as insights for the designer.

"**70%** of the **10** records are **null values** in **full_name** attribute of the **accessory_protein** table."

"**100%** of the **12938** records in the **ontology_id** attribute in the **allele** table, have the value '**1**'."

"**91%** of the records in the **species_id** attribute in the **altered_expression** table, have the value **2**."

"The second most frequent value in the **database_id** attribute of the **database_link** table is '**42**' with **7985** records that represents **13%** of the total."

"'**enzyme**' is the most frequent value in the **type** attribute in **family**

table with **303** records and **43%** of the total."

"The attribute **database_id** of the **disease_database_link** table has **3** values: '**1**', '**38**' and '**56**' with **37%**, **38%** and **25%** respectively of the **2250** records of the table."

Creating rules using thresholds and selecting important attributes and tables it is possible to generate a pool of information that can be used to explore and analyze the database. The sentences generated can be used directly for presenting insights. And also can be used to present specific information in screen to the designer about the database.

3.4.2 Attribute overlap

The second novel implementation is the attribute overlap measure. Its a measure of the level of intersection of two single attributes from different tables. This calculation is applied to every attribute from every table in the database.

The calculation turns into a simple sum when the data from the frequencies table is used. Because the intersection between two tables through a single attribute can be computed using a simple join of the unique data values in the attribute

$$\frac{|\{t \in A | \exists s \in B \cdot t\,[a] = t\,[b]\}|}{|A|} \leq 1$$

This process is computed using the data frequency of the values, a join of a single attribute per table is computed using the data value field. This approach assure the uniqueness of the data values in the structure also the attribute overlap is calculated simply adding the values of the frequencies and adding the values of the percentages. The operation is calculated using LEFT and RIGHT JOIN for every table-attribute.

For example, considering tables S1 and S2 in Table 3.1. Table S1 with 4 records and 4 distinct values and table S2 with 100 records and 3 distinct values. The attribute overlap between this two tables is that 50% of the 4 records in sid of table S1 are present or contained in table B, and that 99% of the 100 records in sid of table S2 are present or contained in table S1.

S1		
data_value	frequency	percentage
3.1415	1	0.25
42	1	0.25
64	1	0.25
128	1	0.25

S2		
data_value	frequency	percentage
1.618	1	0.01
42	89	0.89
64	10	0.1

Table 3.1: Tables S1 and S2.

The previous results are showing that there is a potential relation between these two tables, because 99% of the records in S2 can be mapped from table S1. The next step is to check whether or not the values in both tables are unique. There are three possible cases: 1) the values in both tables are unique; 2) the values in one of the tables are unique and; 3) the values in both tables are not unique. In 1) it can be suggest that there is an IS-A relationship between the tables. In 2) it can be a case of data dependency where the table with unique records can be referenced by the other table, the relationship can exists. And in 3) the relation using those attributes exists, there must exists a third table or more tables with unique records, such that these two tables are related to a parent table where their data values are unique; if we allow this connection it will create a cartesian product.

The attribute overlap is good enough to identify relationships between tables without creating a cartesian product, meaning a multiplication of the records because the attribute overlap is always calculated with unique data values and its their frequency and percentage the measures that are summed up.

The Python code of this process is presented in Listing 2 and the call to the function simulating a cartesian product between every table-attribute pair is presented in Listing 3.

The exact percentage of intersection between two attributes from different tables is calculated adding the percentage field from the data frequency table. This optimises computing time performance because is not performing the join operation between the tables, it is simulating the join using unique values from each table. The worst case time will be equal to the actual performance of the JOIN operation between the tables when every data value from each table is unique.

Adding up percentages is an efficient way of calculating the attribute overlap measure or the intersection rate between two tables. The metadata collected about references in the definition of the database plus the attribute overlap calculation allows us to have enough information to decide whether or not a relationship between two tables correspond or not to a IS-A relationship or to a relationship that implies a hierarchical structure between the tables. If the tables have a high attribute overlap in one of the directions and both attributes have unique records, then we can say that there is a potential hierarchical relationship between them. It will be labour of the designer to decide if the relationship is real or not.

The simulation of a cartesian product using all the attributes in the database allows to spot possible relationships between tables that were not defined in the definition of the database schema. But also allows to perform this checking against external data sources. And it is not restricted to structured data but unstructured data can be scanned with this approach.

Access to attrite overlap measures of the complete database is usually not provided by any data analytical tool. The benefits for data exploration when the designer knows little or have no insights about the content of the database. This process can be a powerful tool for data analysis finding new relations in the database.

When working with one database seems trivial for the designer to know about the content of the database. But this analysis can extend its capabilities to external datasets to the repository. Understanding the term *repository* as the set of internal databases, datasets and data sources of the organization.

The attribute overlap process is designed to run automatically, it autogenerates the necessary SQL code to extract and analyze all the data in a specific database or repository. It can be extended to analyse unstructured data sources.

The results of the attribute overlap gives an understanding of how possible is that two tables can be related to each other. The following result is from a tuple that represent a real relationship in the database:

"The table **altered_expression** can be related to **altered_expression_refs** through the attributes **altered_expression_id** and **altered_expression_id** respectively, because **99.7%** of the **1742** records of the first table are in the second table and **100%** of the **2360** records of the second table are present

in the first table."

Potential problem. There is a risk of finding spurious connections. For example, when a pair of attributes have more or less the same number of records and both are sequences. In this case when attribute overlap will be high in both directions. Additional information must be provided to the designer to make an educated guess about the possible relationship found.

A problem with this approach is the spurious relationships that can be found. The following is an example of this case because the field *display_order* is present in several tables and it is used to sort the data when its presented.

"The table **contributor2family** can be related to **contributor2object** through the attributes **display_order** and **display_order** respectively, because **99.2%** of the **913** records of the first table are in the second table and **100%** of the **3164** records of the second table are present in the first table."

When a table is empty, there is no way of getting information with the statistical analysis. But as the objective of this project is to extract hierarchical data from a relational database, empty tables result of no interest for the project. Empty tables can be identified in the statistical analysis phase.

3.4.3 Primary key checking

Sometimes, even databases with no hierarchical structure may have a composed primary keys in its tables. This primary key will be formed from two or more references to other tables. It can be the case that one of the columns of the primary key will contain the others, meaning the values of a single column are unique, or near to be unique, and is enough to identify a the tuple. In this case, when one attribute is enough to represent the uniqueness of the tuple, it may be understood as a hierarchy, because that single attribute comes from the referenced table.

$$\frac{|N_a|}{|A|} \leq 1$$

If the result of the number of distinct values of the attribute A, N_a, divided by the number of records in attribute A, $|A|$, is equal to 1, then A is

a candidate key. This also can be true for values near to 1, considering some inputs as mistakes of special cases where the data needs to be duplicated.

The same calculation is performed to establish if there is a potential hierarchical relationship in the previous section for the attribute overlap measure and to calculate the uniqueness of the data values of an attribute.

3.5 Summary of the process

The benefits of this approach are listed bellow:

- The approach allows to identify hierarchical relationships, classify them and make suggestions to the designer.
- Calculates potential candidate keys for each table and identify potential relationships between tables.
- The data frequency calculations turns very easy to present automatic insights from the database or repository.
- Automatic semantic interpretation from the stored data.
- Rapid analysis of the level of integration between data sources.
- Access to potential relationship between all the data in the database.
- Efficient way to calculate possible relationships between tables.
- Enable to perform the this analysis with more than one database, but with external data sources, with structured or unstructured formats.
- It is an automated process that can run in the background and provide useful information about the databases.
- Allows to identify many-to-many relationships and then search for tables that can make a link between them. This would cause a cartesian product using a direct query between those two tables.

```
def insertDataPercentage(tname, aname, data_value, fnumber,
 ↪  pnumber):
    cur.execute("INSERT INTO frequencies (entity, field,
 ↪  data_value, frequency, percentage) VALUES (%s, %s, %s, %s,
 ↪  %s);", (tname, aname, data_value, fnumber, pnumber))
    conn.commit()

def getDataDistribution2(table, attribute):
    cur.execute("select "+attribute+" \
        , count(a.*) as n \
        , count(a.*) * 1.0 / t.n as percentage \
        from "+table+" as a  \
        inner join \
        (select count(*) as n from "+table+") as t \
        on 1 = 1 \
    group by "+attribute+", t.n;")
    rows = cur.fetchall()
    for row in rows:
        insertDataPercentage(table, attribute, row[0], row[1],
 ↪  row[2])
        print table, attribute, row[0], row[1], row[2]

cur.execute("select distinct relname, attname from summary
 ↪  order by 1,2;")
rows = cur.fetchall()
for row in rows:
    getDataDistribution2(row[0], row[1])
```

Listing 1: Python code example of data frequency calculation.

```
def insertJoins(table1, table2, att1, att2):
    cur.execute("select f1.entity \
        , f2.entity as entity2 \
        , f1.field \
        , f2.field as field2 \
        , count(distinct f1.data_value) as num_values \
        , sum(f1.frequency) as sum_freq \
        , sum(f1.percentage) as sum_perc \
        , count(distinct f2.data_value) as num_values2 \
        , sum(f2.frequency) as sum_freq2 \
        , sum(f2.percentage) as sum_perc2 \
        from (select entity, field, data_value, frequency,
↪   percentage  \
                from frequencies  \
                where entity = '"+table1+"'  \
                and field = '"+att1+"') as f1 \
        left join  \
        (select entity, field, data_value, frequency,
↪   percentage  \
                from frequencies  \
                where entity = '"+table2+"' \
                and field = '"+att2+"') as f2 \
        on f1.data_value = f2.data_value \
        group by 1,2,3,4;")
    rows = cur.fetchall()
    for row in rows:
        if row[1] <> None:
            cur.execute("INSERT INTO joins (entity, entity2,
↪   field, field2, num_values, sum_freq, sum_perc, num_values2,
↪   sum_freq2, sum_perc2) VALUES (%s, %s, %s, %s, %s, %s, %s,
↪   %s, %s, %s);",
↪   (row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],
↪   row[8], row[9]))
            conn.commit()
```

Listing 2: Function that calculates the attribute overlap between two tables using single attributes.

```
cur.execute("select distinct entity, field from frequencies
 ↪  order by 1,2;")
rows = cur.fetchall()
rows2 = rows
for r1 in rows:
    for r2 in rows2:
        if r1[0] <> r2[0] and r2[0] <> '':
            insertJoins(r1[0],r2[0],r1[1],r2[1])
```

Listing 3: Cartesian product using every attribute of every table in the database.

Chapter 4

Evaluation

4.1 Results from iuphar database

From the exploration of the iuphar database we can inform that the database has 159 tables and 231 references between them. There are 86 out of 159 tables of the type **Linking table** (54.09%), 43 **Foreign Relation** type (26.68%), 22 **Root tables** (13.84%), 4 **Composed PK with FK** (2.53%), and 4 **Isolated tables** (2.53%). With respect to the references, there are there 131 out of 231 refereces of the type **Part of the key** (56.77%), 80 **Not key** type (34.5%) and 20 **Exact key** type (8.73%).

A tree visualization of dependencies between tables is presented in Figure 4.1. The tree starts from a Root table and the levels represent direct connections to the upper table. The number of levels of each path is finite in the iuphar database, because it does not have loops in its connections. Repetition occurs when a table is referencing more than one table in the database. There is one table per line. The table type, the reference type and the number of records of the table is in parenthesis.

The **Exact key** reference type is one of interest. This type represents the existence of a hierarchical relationship between the two tables. This connection is representing a IS-A relationship, a case of specialization. There are 20 of these reference type in the database and can be seen in Figure 4.2

Another case of interest is when there is a single attribute referencing to

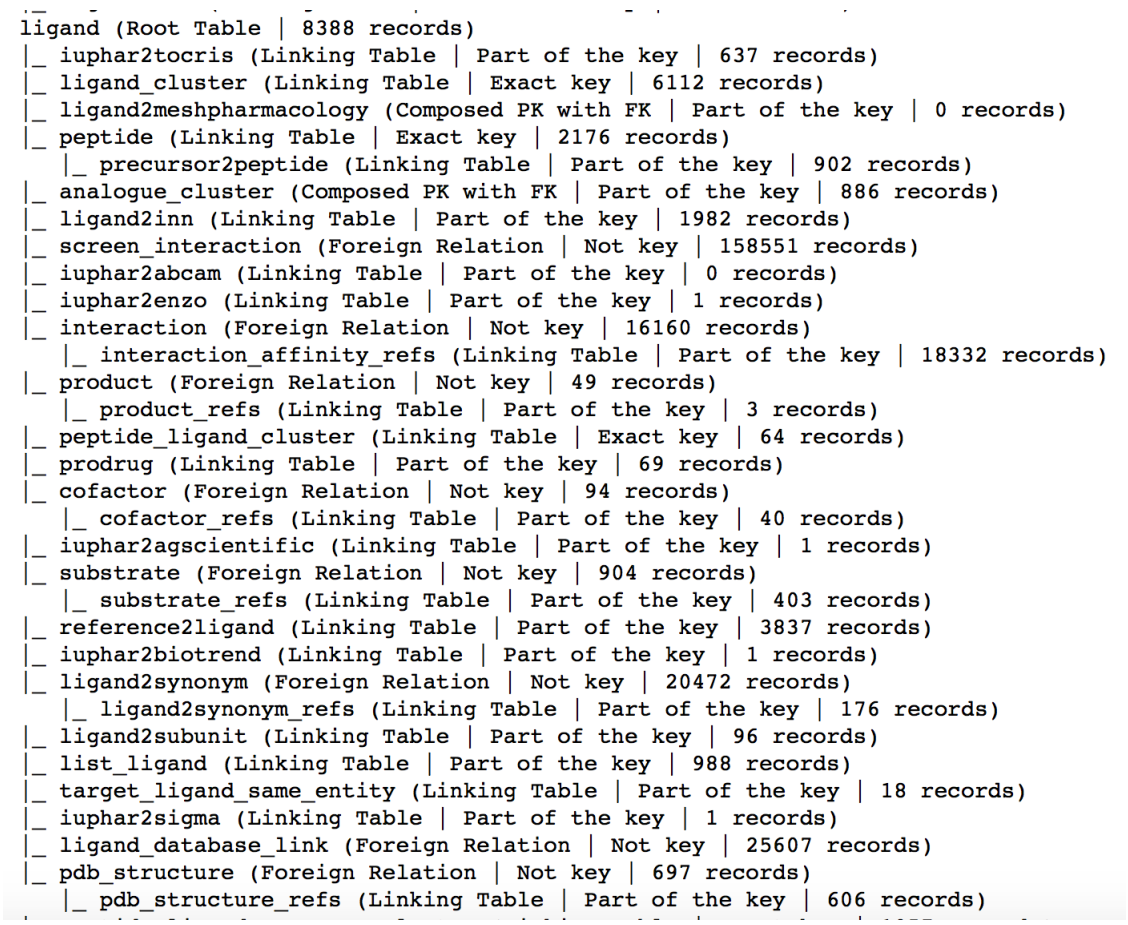

Figure 4.1: Example of hierarchy list from the database analysis.

other table. Like in the case of the **ligand** and **peptide** tables. **peptide** table has its own primary key and one of its attributes is referencing to the **ligand** table. From this relationship we can say there is a hierarchical relationship between the tables. The system is able to identify this relationship, obtain the data through an automatic query and present it in a hierarchical way. The system is able to identify all the path of relationships between table, then analysing the metadata and statistical measures is able to guess that a hierarchical relationship exists. The hierarchies can be extracted in both directions from the system, once a hierarchy is identify it is possible to extract the data from both directions. It can be seen in Figure 4.3.

An interesting result from the system is the capability to infer hierarchical relationships between the tables. For example, when a table is referencing to two other tables, we can analyse the results from the *primary key checking* process and suggest that as one of the attributes contains the other, there may exists a hierarchy. When the primary key checking is near to one, means that few records will be repeated in the hierarchy, most of the data values will belong to one class. An example of a three level hierarchy can be seen in Figure 4.4 where we can see the relationship between the tables object, receptor_basic and receptor2family. receptor is a specialization of object and receptor2family is referencing to receptor_basic and family but when the primary key checking is analyzed the hierarchy seems to be using the relationship to receptor_basic because less than 2% (58 out of 2912) of the data values are repeated in the hierarchy.

The results can be exported in JSON-alike format and also in a a JSON schema [1]. The JSON schema is defined using the metadata from the database. The hierarchy, attribute names and types are required. An example is presented in Figure 4.5. The figure is showing the output of the relation between the **peptide** and **ligand** tables. The query is auto-generated using the metadata collected from the database. In this case, the heuristic found an IS-A relationship leading to the generation of this result.

[1] http://json-schema.org

	relname name	frelname name	reference_type text
1	accessory_protein	object	Exact key
2	catalytic_receptor	object	Exact key
3	enzyme	object	Exact key
4	gpcr	object	Exact key
5	grac_family_text	family	Exact key
6	grac_functional_characteristics	object	Exact key
7	grac_transduction	object	Exact key
8	introduction	family	Exact key
9	lgic	object	Exact key
10	ligand_cluster	ligand	Exact key
11	multimer	object	Exact key
12	nhr	object	Exact key
13	other_ic	object	Exact key
14	other_protein	object	Exact key
15	peptide	ligand	Exact key
16	peptide_ligand_cluster	ligand	Exact key
17	peptide_ligand_sequence_cluster	ligand	Exact key
18	receptor_basic	object	Exact key
19	transporter	object	Exact key
20	vgic	object	Exact key

Figure 4.2: List of IS-A relationships found.

```
ligand.ligand_id : 1
ligand.name : flesinoxan
ligand.pubchem_sid : 135650267
ligand.radioactive : False
ligand.old_ligand_id : 2846
ligand.type : Synthetic organic
ligand.approved : False
ligand.approved_source :
ligand.iupac_name : 4-fluoro-N-[2-[4-[(3S)-3-(hydroxymethyl)-2,3-dihydro-1,4-benzodioxin-8-yl]piperazin-1-yl]ethy
l]benzamide
ligand.comments :
ligand.withdrawn_drug : False
ligand.verified : False
ligand.abbreviation :
ligand.clinical_use :
ligand.mechanism_of_action :
ligand.absorption_distribution :
ligand.metabolism :
ligand.elimination :
ligand.popn_pharmacokinetics :
ligand.organ_function_impairment :
ligand.emc_url :
ligand.drugs_url :
ligand.ema_url :
ligand.bioactivity_comments :
ligand.labelled : False
ligand.in_gtip : None

        ligand2synonym.ligand_id : 1
        ligand2synonym.synonym : DU-29,373
        ligand2synonym.from_grac : False
        ligand2synonym.ligand2synonym_id : 26505
        ligand2synonym.display : True


        ligand2synonym.ligand_id : 1
        ligand2synonym.synonym : (&plus;)-flesinoxan
        ligand2synonym.from_grac : False
        ligand2synonym.ligand2synonym_id : 21786
        ligand2synonym.display : True

ligand.ligand_id : 2
ligand.name : quinpirole
```

Figure 4.3: Example of the output of a hierarchical relationship between peptide and ligand tables.

Figure 4.4: Example of the output of a hierarchical relationship between peptide and ligand tables.

Figure 4.5: JSON schema definition example.

Chapter 5

Conclusion

The objective of finding one or many hierarchical relationships in a relational database was accomplished using heuristics analysing the metadata of the database, a graph representation and the data itself. Measures like data frequencies and attribute overlaps were calculated to provide more information and to support the findings of the hierarchical relationships between the tables.

It is possible with system to identify explicit and implicit relationships and to extract the hierarchical data from the relational model.

The objective of finding the explicit relationships was accomplished. The primary key checking process allowed the discovery of, implicit, hierarchical relationships. A full scan of candidate keys was computed for all the attributes in the database to provide insights about the database content. Also it is possible to find relationships created by use that were not included in the original design of the database.

The extensions implemented like the data frequency and the attribute overlap measures helped not only to find relationships in the database but also to explore and analyze the content and structures of the data sources or repositories. The process can be applied to several databases at the same time and find relationships among different repositories.

5.1 Further work

5.1.1 Big Data implementation

It is possible to extend this implementation for a Big Data environment allowing to analyze databases of gigabytes or terabytes with the same principles of analyzing its metadata, graph representation and statistical measures.

In the case of a schemaless database the third phase of statistical analysis will replace the first phase of metadata exploration. The computation of the frequencies of the data values and its attribute overlap analysis will provide the necessary insights about relationships between the data structures. Same results are expected from the analysis of an unstructured data source.

The statistical analysis can make use of a map-reduce implementation. To process large volumes of data in a distributed way. The data frequency calculation is a counting function of the key part of a **key-value** representation of the data. This process can be applied to each field of a table or property if the data is in a document format. The data value will be the key part. Using several reducers the counting task can be distributed and could process terabytes of data in reasonable time.

Finally the output of this process will deliver kilobytes of data, and the process is required to run once. It will not require to reprocess the same amount of data. The metadata with the attribute overlap between two single attributes from two different tables, makes easy to analyse the potential connections between the data sources.

5.1.2 Beyond single attributes

The approach in this project considers single attribute comparison. But it can be extended to consider any number of attributes. The reason is because all the tables in the iuphar database uses a single attribute as reference to another table. But it can be the case that more than one attribute is used as a single reference constraint. One approach is to reduce multiple attributes to a single one and continue the process as it is defined in the system. Another approach is to include a list of the attributes and create the code to autogenerate the queries in a recursive way.

5.1.3 Optimization to the process

An optimization of the data frequencies calculation process in the statistical analysis phase is to make a distinction by the length of the data values and storing them in different data structures. This allows to create indexes for the data value column, with text values now its impossible, and speed up the data processing. Text type attributes are less expected to appear because they will represent annotations or plain texts, a different kind of analysis could be performed on this data values like applications of natural language processing tasks.

The formation of sentences using metadata of the database can be extended using more than one table in the sentences formation. Sentences using in this report are limited to one table at the time. The system could learn about the preferences of the designer presenting more information according to selection criteria in the use of the system.

The implementation of the systems included the calculation of data frequency for each attribute. It can be extended a distinct of string and numeric attributes. This would allow to calculate additional measures for numeric attributes like average, median, min, max, range, percentiles, etc. Additional evaluation of data distribution comparison can be included in the system to say if two data distribution are similar to see the similarity between two numerical variables. Also suggest that two numerical variables are amplified by an order of magnitude but they behave in a similar way.

Appendix A

Basic Metadata Structures in SQL

```
-- metadata of tables and attributes in the database. PK
   and FK are identified.
drop table if exists tempo;
create table tempo as
select  c.relfilenode
        , c.relname
        , a.attname
        , col.data_type
        , a.attnum
        , pk.oid
        , pk.conname
        , case when pk.conrelid is not null then 'pk' else
          '' end as pk
        , case when fk.conrelid is not null then 'fk' else
          '' end as fk
        , case when pk.conrelid is not null and fk.conrelid
            is not null then 'link' else '' end as link
        , fk.oid as foid
        , fk.conname as fconname
        , ft.relname as frelname
        , fk.confkey
        , fa.attnum as fattnum
        , fa.attname as fattname
from    pg_class as c inner join pg_attribute as a
```

```
        on c.relfilenode = a.attrelid
        and relkind = 'r'
        and a.attnum > 0
        and c.relnamespace = 2200
        left join pg_constraint as pk
        on c.relfilenode = pk.conrelid
        and pk.contype = 'p'
        and pk.conrelid = a.attrelid
        and a.attnum = any(pk.conkey)
        left join pg_constraint as fk
        on c.relfilenode = fk.conrelid
        and fk.contype = 'f'
        and fk.conrelid = a.attrelid
        and a.attnum = any(fk.conkey)
        left join pg_class as ft
        on fk.confrelid = ft.relfilenode
        left join information_schema.columns as col
        on c.relname = col.table_name
        and a.attname = col.column_name
        left join pg_attribute as fa
        on fk.confrelid = fa.attrelid
        and fa.attnum = any(fk.confkey)
order by relfilenode, a.attnum;

-- drop table summary;
drop table if exists summary;
create table summary as
select  c.relfilenode
        , c.relname
        , a.attname
        , col.data_type
        , a.attnum
        , pk.oid
        , pk.conname
        , case when pk.conrelid is not null then 'pk' else
          '' end as pk
        , case when fk.conrelid is not null then 'fk' else
          '' end as fk
        , case when pk.conrelid is not null and fk.conrelid
            is not null then 'link' else '' end as link
```

```
        , fk.oid as foid
        , fk.conname as fconname
        , ft.relname as frelname
from    pg_class as c inner join pg_attribute as a
        on c.relfilenode = a.attrelid
        and relkind = 'r'
        and a.attnum > 0
        and c.relnamespace = 2200
        left join pg_constraint as pk
        on c.relfilenode = pk.conrelid
        and pk.contype = 'p'
        and pk.conrelid = a.attrelid
        and a.attnum = any(pk.conkey)
        left join pg_constraint as fk
        on c.relfilenode = fk.conrelid
        and fk.contype = 'f'
        and fk.conrelid = a.attrelid
        and a.attnum = any(fk.conkey)
        left join pg_class as ft
        on fk.confrelid = ft.relfilenode
        left join information_schema.columns as col
        on c.relname = col.table_name
        and a.attname = col.column_name
where   c.relname not in ('t1','t2','tempo','summary','
   summary_v2','summary_v3','summary_v4','table_relations')
order by relfilenode, a.attnum;

-- temporal structure to calculate number of PK, FK and
   hubs.
drop table if exists summary_v2;
create table summary_v2 as
select  relfilenode
        , relname
        , sum(case when pk = 'pk' then 1 else 0 end) as
          num_pk
        , sum(case when fk = 'fk' then 1 else 0 end) as
          num_fk
        , sum(case when link = 'link' then 1 else 0 end) as
           num_link
        , count(distinct frelname) as num_ftables
```

```
        , count(*) as features
from    summary
group by 1,2
order by num_fk, num_pk, num_link, features desc;

-- Classification of tables in the database according to
   their dependences.
drop table if exists summary_v3;
create table summary_v3 as
select  relfilenode
        , relname
        , case when num_fk = 0 and s2.frelname is null then
            'Isolated Table'
                when num_pk > 0 and num_fk = 0 then 'Root
                    Table'
                when num_pk = num_link and num_fk =
                    num_link and num_pk > 0 then 'Linking
                    Table'
                when num_pk > num_fk and num_fk = num_link
                    then 'Composed PK with FK'
                when num_pk <= num_fk then 'Foreign
                    Relation'
                else '' end as table_type
        , num_pk
        , num_fk
        , num_link
        , num_ftables
        , features
        , coalesce(f.n, 0) as num_references
from    summary_v2 as s
        left join
        (select distinct frelname from summary
        where frelname is not null) as s2
        on s.relname = s2.frelname
        left join
        (select frelname
                , count(distinct relname) as n
        from    summary
        where   frelname is not null
        group by frelname) as f
```

```
        on s.relname = f.frelname;

-- Aggregation of the tables with dependences in order to
   calculate the path to the origin tables.
drop table if exists summary_v4;
create table summary_v4 as
select distinct s.relname
        , s.frelname
        , s3.table_type as referencing_table_type
        , s4.table_type as referenced_table_type
        , case when s3.num_link = 1 and s3.num_pk = s3.
           num_link then 'IS-A'
                when s3.num_pk > s3.num_link and s3.
                   num_link > 0 then 'Part of the key'
                when s3.num_link = 0 then 'Not key'
                when s3.num_link > 1 and s3.num_pk = s3.
                   num_link then 'Part of the key'
                --when s3.num_link > 1 and s3.num_pk = s3.
                   num_link then 'Part of the full key'
        end as reference_type
from    summary as s left join summary_v3 as s3
        on s.relname = s3.relname
        left join summary_v3 as s4
        on s.frelname = s4.relname
where   s.fk = 'fk'
order by 1,2;


-- Query that returns the dependencies between the tables
   in
drop table if exists table_relations;
create table table_relations as
with recursive rel(relname, frelname, referenced_table_type
   ) as (
        select distinct relname, frelname,
           referenced_table_type
        from summary_v4 as r
        where referenced_table_type = 'Root Table'
        union all
```

```
        select distinct r.relname, r.frelname, r.
            referenced_table_type
        from summary_v4 as r, rel
        where r.frelname = rel.relname
        )
        select distinct * from rel limit 1000;
```

Bibliography

[Akoka et al., 1999] Akoka, J., Comyn-Wattiau, I., and Lammari, N. (1999). Relational database reverse engineering: Elicitation of generalization hierarchies. In *Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November 15-18, 1999, Proceedings*, pages 173–185.

[Alhajj, 2003] Alhajj, R. (2003). Extracting the extended entity-relationship model from a legacy relational database. *Inf. Syst.*, 28(6):597–618.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

[Date, 1981] Date, C. (1981). Referential integrity. In *VLDB*, volume 81, pages 2–12.

[Fong, 1997] Fong, J. (1997). Converting relational to object-oriented databases. *SIGMOD Record*, 26(1):53–58.

[Henrard and Hainaut, 2001] Henrard, J. and Hainaut, J.-L. (2001). Data dependency elicitation in database reverse engineering. In *Software Maintenance and Reengineering, 2001. Fifth European Conference on*, pages 11–19. IEEE.

[Karagiannis, 1994] Karagiannis, D., editor (1994). *Database and Expert Systems Applications, 5th International Conference, DEXA '94, Athens, Greece, September 7 - 9, 1994, Proceedings*, volume 856 of *Lecture Notes in Computer Science*. Springer.

[Lammari et al., 2007] Lammari, N., Comyn-Wattiau, I., and Akoka, J. (2007). Extracting generalization hierarchies from relational databases: A

reverse engineering approach. *Data & Knowledge Engineering*, 63(2):568–589.

[Smith and Smith, 1977] Smith, J. M. and Smith, D. C. P. (1977). Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst.*, 2(2):105–133.

[Yeh et al., 2008] Yeh, D., Li, Y., and Chu, W. C. (2008). Extracting entity-relationship diagram from a table-based legacy database. *Journal of Systems and Software*, 81(5):764–771.