UTRECHT UNIVERSITY
Faculty of Science
Department of Information and Computing Sciences
MSc Computing Science

EVIDENCE BASES FOR DECISION SUPPORT IN PHARMACEUTICAL SCIENCES

A THESIS BY
Yue Shi
6904818

Contents

1 Introduction **1**

2 Related Work **3**

2.1 Decision Support System 3

2.1.1 Frame work of a DSSs 4

2.1.2 Application of DSS 4

2.2 Process Modeling 5

2.3 Process mining 10

2.3.1 Event Data 10

2.3.2 Process discovery 11

2.3.3 Conformance checking 12

2.3.4 Process enhancement 13

2.4 Process Querying 13

2.4.1 Process Model Repositories Query 13

2.4.2 Event Log and Process Model Querying 17

2.4.3 Graph Database and its Query Language 22

3 Problem Statement **25**

4 System Description **27**

4.1 Data model design 27

4.2 System function module design 30

4.2.1 Query 30

4.2.2 View 31

5 System Implementation **33**

5.1 Database 34

5.1.1 data model schema 34

5.1.2 instances of classes 36

5.2 Software System and User Interface 37

5.2.1 Query 38

5.2.2 Node detail and modification 39

5.2.3 View 41

6 Evaluation of the system **45**

7 Application **47**

7.1 System Configuration 47

7.2 Instruction of using the system 47

7.2.1 Query 48

7.2.2 Node and relationship modification 48

7.2.3 View 51

8 Conclusions and Future Work **53**

8.1 Future Work 53

Chapter 1

Introduction

Pharmaceutical Sciences encompass a wide variety of scientific disciplines concerned with the discovery, development, and manufacture of pharmaceutical goods. Before any medicine is released to the market, clinical studies of various compositions are done. This experimental information is crucial for medication development decisions. Pharmaceutical sciences are constantly evolving and decision support systems (DSS) are increasingly being utilized to support clinical decision-making related to drug therapy. The effectiveness of DSS depends heavily on the quality and reliability of the evidence that underpins it. Process querying is a method used to analyze data generated during the clinical drug trial process to identify patterns, trends, and potential issues that require further investigation.

The field of pharmaceutical sciences is constantly evolving, and healthcare professionals face increasingly complex decisions related to drug therapy. Decision support systems (DSS) have emerged as an important tool for supporting clinical decision-making in pharmacy practice. Clinical drug trials are indeed a process that involves multiple stages and requires close monitoring and evaluation to ensure the safety and efficacy of the drug being tested. If we think of a clinical trial as a process, it is divided into different steps, and each step will generate data records. Process models and process queries are critical components of decision support systems (DSS) in pharmaceutical sciences. The records of each step of the clinical trial, and the results of each step will become evidence to support decision makers in making decisions.

The motivation for building a pharmaceutical science decision support system with process querying is to provide a more efficient, effective, and evidence-based approach to decision-making in this complex and dynamic industry. It can provide a centralized repository for data or we can call it the evidence for decision support that can be accessed and utilized by all stakeholders involved in the decision-making process. This can lead to better collaboration, data sharing, and integration of diverse data sources.

This faces many challenges. Data sources in pharmaceutical science are heterogeneous and lack a unified framework to integrate them. Therefore, it is necessary to construct a data model of drug clinical trials to include as much data information as possible in pharmaceutical science and connect them. At the same time, this model preferably has the characteristics of a process model, which can be divided into a step-by-step process, and can clearly indicate the data used and the results produced by each step. The second difficulty is the design of the process query function. Existing process query languages have their own applicable database and process models. However, these process query languages and the adapted environments are mostly used in the fields of business process management and the like. Therefore, it is also difficult to select an existing process query language and successfully transplant it into our system.

Our main contribution is the design and implementation of an evidence bases for decision support in pharmaceutical sciences. It includes a data model that can store drug clinical trial data, and the data model is implemented in the neo4j graph database. It also includes a software system that can provide query functions and use the database view as a decision-making model. Also includes a user interface that provides data visualization capabilities.

The remainder of this thesis is organized as follows. In section 2 we will discuss the related work involved in the thesis. In this part we will introduce the background and application related to decision support system and background related to process model and process query. In section 3, we give a specific description of the problems that the thesis needs to solve. Section 4 introduces the design of the structure and function of the system. Section 5 describes how the system is implemented. Section 6 evaluates the usability of the system. Section 7 is the application section, which describes how to use this system. Finally the conclusion section, we summarized the thesis and discussed the directions in which the system can be improved in the future.

Chapter 2

Related Work

Three portions of background information on all the technologies relevant to our work will be covered in this discussion. In the first section, we will cover pertinent information on decision support systems and associated applications in the context of healthcare. The second part includes information related to process mining, and the third part contains information related to process query.

2.1 Decision Support System

A decision support system is an information system that assists in business-level or organizational-level decision-making activities, generally concentrated on middle and high-level management, supporting decision-making at the organization's internal management, operation, and planning levels. DSSs assist in decision-making on rapidly changing and challenging to predict problems, typically non-structured and semi-structured decision problems.[38] Our goal is to build a decision support system that supports decision makers in deciding whether a drug can be put on the market, so the framework of the decision support system will be the framework of our project.

It was known from the beginning that DSSs might be developed to help decision-makers at any level inside an organization. Over the years, many of the most intriguing DSS have been geared toward middle and senior management.

There are three major characteristics of a DSS[33]:

- DSS are created to facilitate decision-making.
- Decision making should be supported rather than automated by DSS.
- Decision-makers' needs are likely to change over time, and DSS should be able to adapt quickly.

Based on the three fundamental characteristics of DSSs, DSSs have some requirements that must be met. DSSs should be equipped with a knowledge base, the capacity to retain knowledge, the ability to articulate knowledge ad hoc in various customized ways and also in normalized documents, and the option to pick a preferred subset of stored knowledge for either demonstration or for extracting new knowledge, and it must be built to come into direct contact with a decision maker in a way that provides the user a versatile choice and sequential knowledge-management activities.

2.1.1 Frame work of a DSSs

A decision support system contains three fundamental components:[35]

- Database Management Subsystem contains a database with information pertinent to the issues for which the DSS was created. The software manages the database called a database management system (DBMS). A DBMS can be linked to the group's data warehouse and data marts. Users are kept apart from the physical structure and processes of the database by a database management system (DBMS). Additionally, it should be able to tell the user what kinds of data are accessible and how to get them.

- Model Management Subsystem: The roles of model bases management systems are comparable to that of database management systems. It has a model base containing models for different fields that give DSS analytical skills. The Model base Management System (MBMS), which controls the model base, is also included. An MBMS's function is to apply models to the data from the DBMS to transform it into knowledge. The MBMS should be able to guide the user in the model building because many of the issues that a DSS user will deal with may be unstructured.

- User Interface Subsystem: It encompasses every aspect of the interaction between a user and various DSS components. DSSs ought to have user-friendly interfaces since many of their users are managers who are not computer literate. These interfaces support the creation of models as well as interactions with them.

With the development of artificial intelligence technology, many technologies like knowledge base, natural language processing, and so on can be added to the decision support system to improve performance. Hence a new field was created called: Intelligent Decision Support Systems. Artificial intelligence (AI) techniques are utilized in decision support systems, which are collaborative computer-based systems that assist decision makers in dealing with complex, ambiguous, and unstructured challenges. For example, data mining and neural networks are the AI techniques that focus on knowledge discovery, whereas fuzzy logic and expert systems concentrate on knowledge in the form of rules. [37]

Besides the above-mentioned fundamental components, an Intelligent DSS has a Knowledge Management Subsystem. It is necessary to convert information into knowledge after it has been identified, gathered, and managed. This calls for categorization, evaluation, and synthesizing, all of which need human input. Technology cannot produce knowledge. Data mining, OLAP, machine learning, and artificial intelligence are some of the tools and technologies that convert and filter the information/knowledge for this stage of the knowledge management process.[39]

2.1.2 Application of DSS

DSS research may be roughly divided into three categories: application development, DSS theory development, and reference discipline research. The results or benefits of DSS research activities are DSS applications. As a consequence, it is crucial to regularly review DSS applications in order to track the development of the DSS field and to determine the course of future research.[14]

Decision support systems (DSSs) have made a significant and ongoing contribution to healthcare.

The first area DSSs contribute to is Quality. The foundation of healthcare decision-making continues to be quality. Most early studies used information technology to extract data from medical records[46], further refining and standardization resulted in the use of DSSs to evaluate the efficacy of medical decision-making[12]. The use of administrative discharge data by DSSs to analyze factors that affect patient care quality, such as the frequency of complications, readmission, and mortality, has advanced. Decision-makers identify clinical treatment procedures that produce desired outcomes using past discharge data. Patient satisfaction with services received while in the hospital is a subject of growing attention.[24]

Several examples of the application of DSSs in healthcare help improve quality. For example, using data mining to successfully transform vast amounts of datainin Personal Health Record (PHR) and other systems into quality improvements.[7] A collection of best practices known as evidence-based medicine may be developed by doctors using these kinds of information systems or decision support systems. Another determinant of the quality of healthcare is patient satisfaction. Examples of DSSs applications that can improve patient satisfaction include: CPOE integrated systems recommending less expensive drug options or minimizing test duplication.[9]This kind of DSSs can be cost-effective for healthcare through clinical interventions

Another area DSSs can contribute to is Risk Mitigation. Clinical DSSs are frequently used in strategies to decrease drug mistakes. For example, drug safety software protections for medicating, duplication of treatments, and checking drug-drug interactions are now incorporated into CPOE systems.[21] Electronic drug dispensing systems (EDDS) and bar-code point-of-care (BPOC) medication administration systems are other methods that aim to improve patient safety. Each procedure stage (prescribing, transcribing, dispensing, and administering) is automated and takes place inside a linked system when integrated to produce a ”closed loop.”[27]

We discussed the application of DSSs in the direction of healthcare in the last part. DSSs also have many applications in other fields. Another very representative field is Marketing. Marketing management support systems (MMSSs), marketing decision support systems (MDSSs), or intelligent MDSSs are terms used to describe systems that support marketing choices (IMDSSs).[18]They can help marketing management with a variety of tactical planning and decision-making processes. For example, by facilitating marketing tools through visualizations, calculations, rapid iterative development, and user guidance; and facilitating group scheduling with better-focused consultations, better common understanding, and greater consensus on strategic choices, a decision support system have the potential to improve strategic marketing planning. Action research was used to build and implement this decision assistance system in several South African businesses.[13]

In addition, DSSs also have many application cases in Investment, Forest Management, Network Security, Etc., permeating all aspects of our lives.

2.2 Process Modeling

The abundance of process modeling notations that are currently accessible is a clear indication of the importance of process modeling. It's possible that some companies merely make use of informal process models when it comes to structuring talks and documenting processes.

Through the provision of insight and the documentation of processes, process models contribute to the management of complexity. The configuration and operation of information systems must be led by clear and specific instructions. Cross-organizational procedures are only able to operate as intended if there is widespread consensus about the kinds of interactions that are necessary. As a direct consequence of this, companies in the modern day make extensive use of process models.[1]

This section does not intend to offer a comprehensive review of the process modeling notations that are currently in use. Only the notations that will be necessary for the rest of the discussion are discussed here.

The process model's representation and precision are determined by the discovery technique and the type of visualization used. Next we will discuss four different process models.

Directly-follows graph:

The essential representation of the process models is a Directly-Follows Graph (DFG). Each node in a DFG can be seen as an activity, and the arcs define the relationship between the activities. The directly-follows graph in a process model generally has a source and sink that stand in for the beginning and end activities. An arc in the DFG of a specific process denotes that the sink activity in the event log instantaneously follows the source activity.[43] Figure 1.1 shows the result of constructing the directly-follows graph based on traces in table 1.1.

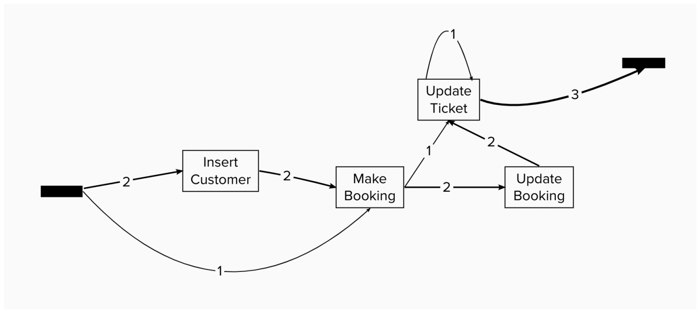


Figure 2.1: Directly-follows graph based on the event log in Table 1.1.

The graph provides a wealth of information, like repeating the pattern "Update Tickets." For all traces, they ended with the activity "Update Ticket."

BPMN

BPMN 2.0 (Business Process Model and Notation) is a popular framework that enables the creation of concise and intelligible process models. Subprocesses, data flows, and resources may all be incorporated inside a single BPMN diagram, in addition to the workflow perspective. Also, because the control flow view may be combined with data and resource perspectives derived from event logs, BPMN appeals to both process miners and business users.[23]

A business process is a series of actions or a flow of organizational operations with the purpose of carrying out work, as defined by the Business Process Model and Notation of the Object Management Group (BPMN v2.0 of OMG).[29] A process model built on the BPMN standard can be conceptualized as a directed typed attributed labeled graph. Following is an example of a simplified loan handling process model in BPMN notations.

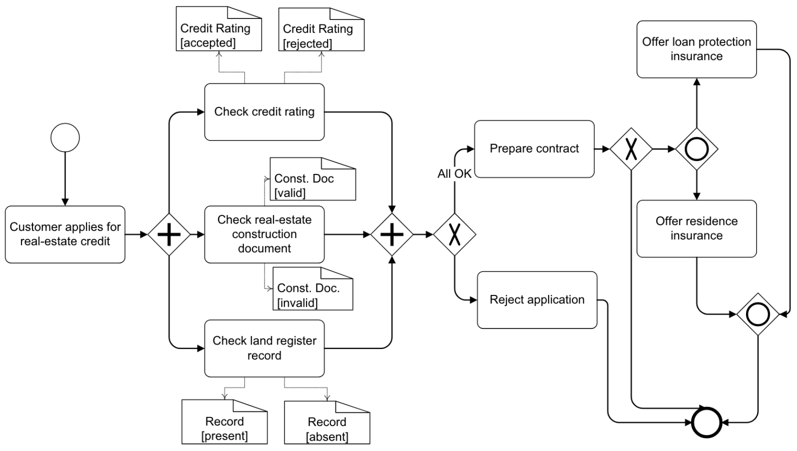


Figure 2.2: Banking Business Process in BPMN Notation[6]

”Event” is represented by a circle and refers to something that happens. There are two types of events: Start event and End event. The Start event acts as a trigger for the process; it is marked with a thin single line and can only be ”captured”, so it appears as a hollow icon. End event represents the result of the process, it is marked with a thick single line and can only be ”thrown”, so it is displayed as a solid icon. “Activity” is represented by a round triangle describes the kind of work that must be done. For example, the activity ”Customer applies for real-estate credit” in the figure. A diamond is used to represent the ”Gateway”, and the conditions shown define how pathways will branch and merge.

All the components we mentioned above belong to control objects. There is another type of objects in the figure called data objects like”Record(present)”. It is a mechanism to demonstrate how activities require or generate data.

The third type of components is flow objects. Control flow is represented by a solid line. It shows how activities are connected in the process. Data flow is represented by a dashed line. It shows how data, text and other artifacts are associated.

Petri Nets

The place/transition (PT) nets(Petri Nets) are the earliest and most well-researched process modeling language that supports concurrent modeling. A Petri net is a bipartite network with two elements: places and transitions, shown as white circles and rectangles. Places can have unlimited tokens, which are represented as black circles. If all sites connected to it as inputs have at least one token, the transition is activated.[30]

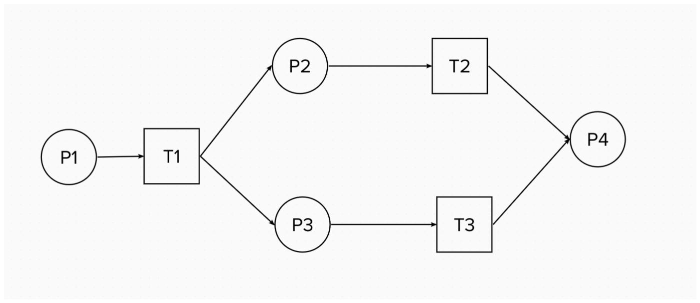


Figure 2.3: An example of the Petri Net

The simple Petri Net shown above can be described by a triplet N Æ (P, T, F). P is a finite set of places, and T is a finite set of transitions. The flow relation, abbreviated F, is a collection of directed arcs. So in the example of the Petri Net, we can see $P = \{P1, P2, P3, P4\}$, $T = \{T1, T2, T3\}$, and $F = \{(P1, T1), (T1, P2), (T1, P3), (P2, T2), (P3, T3), (T2.P4), (T3, P4)\}$ A transition is enabled if each input place of a transition has a sufficient number of tokens. When a transition is allowed, the transition fires, the tokens for the input place are consumed, and tokens are produced for the output place.

OpenSLEX

As mentioned in the previous section, Based on attributes like timestamp, event identifier, Etc., an *event* can be identified from other events. A *process* is the ordering of events with the overarching purpose of achieving a state, where a state is a defining feature of the system's current situation. A *trace* is a procedure that totalizes a sequence of events. A *behavior* is a collection of processes that may repeat itself several times to indicate that it has been or can be observed repeatedly.

Unlike the process model we talked about in the previous section, Open SQL Log Exchange (OpenSLEX) is a meta-model that provides a model not only in the process perspective but also in the event data perspective.

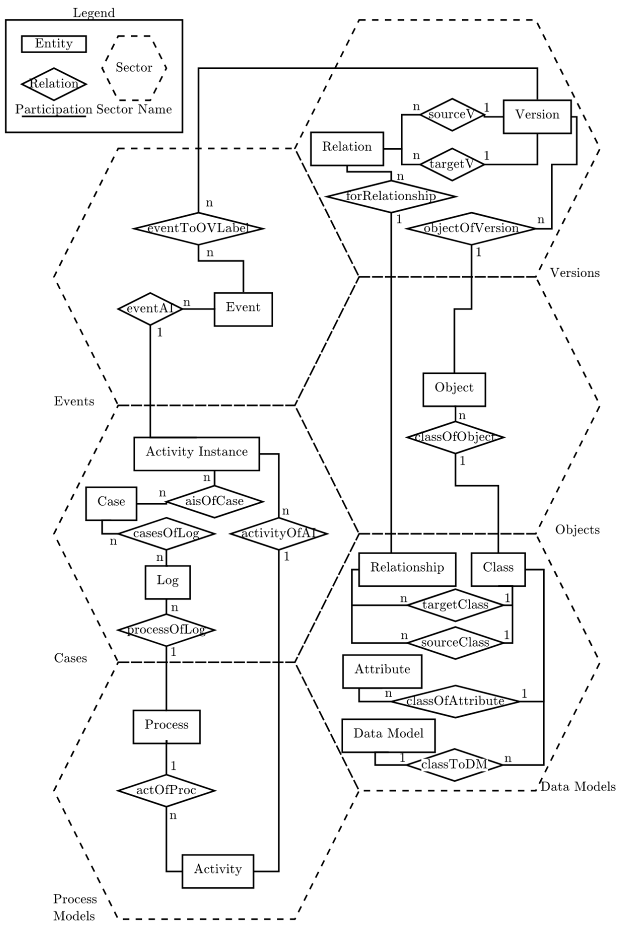


Figure 2.4: ER diagram of the OpenSLEX connected meta model.[17]

In figure 2.4, we can see that the *datamodel* and *process* are the highest abstract entities in the meta model. At the intermediate level of abstraction, each of these components can be divided into smaller parts. So we can see cases are formed with events, and objects can relate to several object versions. Both events and object versions show various states of a top-level abstraction (cases or objects) at different time points.

The figure shows a connected meta model which enables the connections between all the entities in the meta model. In this model, events can be connected to object versions, which have relationships. These connections are examples of relationships in a data model. This is comparable in database systems to understanding which events are related to each table row and utilizing foreign keys to convey table rows.

Process Modeling is an essential part of our project. But when building a process model, we did not choose an existing (discussed above) process model. For Direct-follows graph, BPMN, and Petri Nets, they just show what activities we need to take if we want to accomplish a task. These activities have not yet happened. For the OpenSLEX model, it considers both the process aspect and the event log aspect, but does not give the input required to complete an activity in the process, the output generated by this activity, and other entities related to this activity. information. Therefore, in our project, we will use our own designed database scheme to store the required information, which will be described in the problem statement chapter

2.3 Process mining

To facilitate the analysis of operational processes based on event logs, a family of techniques known as "process mining" links the fields of data science and process management. Process mining is to produce insight and guidance from event data. Process mining is an essential component of data science, driven by the availability of event data and a desire to improve processes.

Process mining provides a wide range of methods to understand better and improve processes, such as process discovery, conformance and compliance checking, performance analysis, process monitoring and prediction, and operational support. Although our work does not involve part of the technology of process mining, the data and models related to process mining are closely related. Most of these techniques rely on the existence of event logs.

2.3.1 Event Data

Events, which show the occurrence of a particular activity, are the minor units of event data we could use to undertake a process mining investigation.

An event log typically details the behavior of a certain process as it has been observed. Such activity may be captured and abstracted in a process model. Graphs are used in process models to show the causal connections between process activities.

Process discovery techniques applied to event logs result in a visual representation of a process. A process discovery algorithm like the α-algorithm produces a process model according to the event log.[1]

Three fundamental attributes define an event: timestamp, an activity name, and case identification.[42] The timestamp showed when the event occurred. The type of activity that was carried out can be inferred from the activity name. A case identifier, the last step, identifies the instance of the underlying process to which this event belongs. While the case identifiers give a mechanism to correlate events per process instance or trace, the timestamps of a collection of events give the collection some sequence.

Table 2.1: Event log example obtained from the database of a movie ticket selling system

Event	Case	CustomerID	BookingID	TicketID	Activity	Timestamp
1	1	100			Insert Customer	2020-10-20 15:50:05
2	1	100	101	011	Make Booking	2020-10-20 15:55:25
3	2	110			Insert Customer	2020-10-20 15:58:32
4	1		101		Update Booking	2020-10-20 15:58:55
5	1	100		012	Update Ticket	2020-10-20 16:01:30
6	1	100		013	Update Ticket	2020-10-20 16:03:49
7	3	111	103	031	Make Booking	2020-10-20 16:05:12
8	2		103	021	Make Booking	2020-10-20 16:07:29
9	2	110	102	022	Update Ticket	2020-10-20 16:09:36
10	3		102	032	Update Ticket	2020-10-20 16:12:18

Table 2.1 shows an example of an event log according to a movie ticket booking system. There are different kinds of events in the table corresponding to the system actions like making bookings(Make Booking), booking information update(Update Booking), customer inserted(Insert customer), and so on. Each event has a timestamp. Besides the three fundamental attributes of the event log, there are other attributes, such as CustomID in the table and BookingID in the table in various scenarios. Process mining techniques can make use of all of these attributes.

2.3.2 Process discovery

Process discovery techniques identify process models that describe the behavior of systems as observed in event data. Different process discovery algorithms use other representations, each having different characteristics.

α-algorithm

This algorithm is aimed at reconstructing causality from a set of sequences of events. It can create a workflow net from the event log based on the relationships between the different activities in the event log. The α-algorithm can be served as the foundation for numerous subsequent process discovery methods. In order to obtain the process model from the event logs, it is essential to get the relationship between event logs.

There are four types of relationships used in the α-algorithm.

- direct succession e1 $>$ e2 If and only if event e1 is promptly followed by event e2. For example, in figure2.1, the event "Make Booking" is followed by the event "Update ticket."
- causality e1 $\rightarrow$ e2 If and only if e1 $>$ e2, not e2 $>$ e1.
- parallel e1$||$e2 If and only if e1 $>$ e2, and e2 $>$ e1.
- unrelated e1e2 If and only if not e1 $>$ e2, and not e2 $>$ e1. For example, there is no trace e1e2 and no trace e2e1 in the event log base.

With these relationships, the α-algorithm can be defined as follow: An event log is first transformed by the alpha miner into directly-follows, sequence, parallel, and choice relations, which are then used to build a Petri net outlining the process model. The program starts by creating a footprint matrix. One may build a process model using the footprint matrix and the pattern previously illustrated. The first footprint-based matrix is found based on the four relations previously discussed. Places are found using the footprint-based matrix. In order to limit the number of locations, each site is designated by a pair of sets of activities.

Heuristic mining

A process model's control-flow perspective is mined using the Heuristics Miner Plug-in. It merely considers the sequence of events within a case to do this.[45] When creating a process model, heuristic mining algorithms consider the frequencies of events and sequences. The fundamental tenet is that infrequent pathways should not be included in the model.

The Heuristic miner makes some improvements based on the α-algorithm. It considers the events frequency and detects short loops in the process model. The first step of Heuristic mining is to build a directly-follows frequency matrix. Then the directly-follows frequency matrix can be used to build the dependency matrix. As we mentioned above, event e1 is followed directly by e2 can be defined as e1 > e2. The dependency between e1 and e2 can be calculated as

$$\frac{|a > b| - |b > a|}{|a > b| + |b > a| + 1}$$

This frequency-based measure expresses how convinced we are that two events A and B have a dependence relationship. A high value of this metric is to indicate there is a dependency relation between two activities. According to this metric, we can construct the dependency graph of the process model.

Genetic process mining

The genetic algorithm is a method of searching that imitates the process of evolution that occurs naturally in biological systems. These algorithms evaluate current points, use mutation to create new points, or combine existing points to locate a solution in the search space. These methods rely on randomization to discover new options and are not deterministic.[28]

This algorithm includes the following steps:
The first step of a generic algorithm is to create the initial population. For process mining, the initial population is the collection of some randomly created Petri nets.
Then we need to calculate fitness for all occurrences in a population. The fitness function can be defined as

$$fitness(\sigma, N) = \frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p})$$

Where m is the number of missing tokens, c is the number of consumed tokens, r is the tokens left over when something reaches the output place, and p is the number of produced tokens.
The next step is to keep the "best ones" and use the "survived" instances as parents to generate the next generation. For generating the "children," there are two essential concepts:

- cross-over: The process of merging various Petri nets (these Petri-nets are already good solutions)
- mutation: Adding some random changes for diversifying.

The Genetic Process Miner is far more noise resistant, allowing for gradual progress, and may be integrated with other algorithms.

2.3.3 Conformance checking

The goal of conformance checking is to compare a process model to an event log for the same process. It is used to determine whether the actual execution of a process, as recorded in the event log, is consistent with the process model. [2]

Rarely does the observed behavior of a process exactly match the predicted behavior in practice. Multiple factors could be responsible for this, including fraud, noise, and flexible approaches. Correctly relating event data to the process model used as a reference for addressing the conformance and compliance problems is one of the challenges. Using reliable statistics to give the user valuable diagnostics presents another challenge.

2.3.4 Process enhancement

There are different techniques for process enhancement. These techniques can be divided into two groups: *Process repair* and *Process extension.*[26] Repair is a type of enhancement that involves changing the model to represent reality better. For instance, the model may be adjusted to reflect that two actions depicted sequentially might occur in any sequence.

Extension, or adding a new perspective to the process model by cross-correlating it with the log, is another improvement. The inclusion of qualities that are related to the events is necessary for the expansion of process models so that they can accommodate diverse perspectives. Process extension techniques like Model-Aligned Event Logs[10] require replaying the event-log traces on the process model. So the missing parts of the process model can be found.

2.4 Process Querying

Process querying is an emerging concept that is constantly refined through an iterative process of adopting and solving practical problems for obtaining and trying to manipulate process models and process-related artifacts. In general, a process query is defined as follows:
Query processing is the procedure used to respond to an inquiry made to a database or information system. It typically entails understanding the query, searching the area where data is stored, and retrieving the results that fulfill the request.

Process querying aims to recognize foundational algorithms, assessment, and analytics over processes to encourage their centralized development and improvement for later reuse in practical situations involving the administration of processes and process-related objects such as resources, knowledge, and information.[41] Such core process-related computations are referred to as process querying methods. Process querying increases the impact of process querying methods in various scenarios exponentially and prevents such practices from being reinvented in different contexts.

In this part, we will discuss two different methods of process query. The first is toquery for process model, and the second is the query language that can query both event data and process model

2.4.1 Process Model Repositories Query

In this section we will discuss three different languages for query flow models. Different from the above query language for querying process data, most process models are mined from event data through process discovery technology. Process models are abstract.

BPMN-Q:Visual Business Processes Query Language

As mentioned in the previous section, BPMN is a popular framework for process models. It is pretty valuable for provide process model designers with a query engine for utilizing previously built business process models (by themselves or others).

BPMN-Q is a visual language based on notations from Business Process Modeling Notation(BPMN).[5] This language can be seen as an abstract syntax extension of BPMN as BPMN is a standard visual notation for modeling business processes.

Usually, we can know to query a subgraph isomorphism of a graph, which is an NP-complete problem.[19] in reality, the business process repository is huge, and the relationship between the components in the process model is very complex.

This query language is built on relational database management systems(RDBMSs) to address time and complexity challenges. It stores the graph-based models of the business process repository using a fixed mapping storage method. The core idea is to design a suitable matching schema for the process model and process repository.[36]

There are four tables to do the relational encoding to transfer the process models into the relational database[6]:

- BPModel(ModelID, ModelName, ModelDescription).
- BPElements(ModelID, ElementID, ElementName, ElementType).
- BPEdges(ModelID, EdgeID, SElementID, DElementID, EdgeType).
- BPPaths(ModelID, PathID, SElementID, DElementID, ElementList)

Elements are represented by tuples stored on a single table. The same is true for the edges, stored in a single table in the format of a tuple. Therefore, query the process model's subgraph is not like the traditional subgraph query. Each node has its type and set of properties. BPMN-Q query nodes can also be polymorphic, which means they can be matched to nodes of multiple kinds.

There are two phases for BPMN-Q queries. In the beginning, there is a filtering phase. All the candidates that match the input query structure will be identified in the repository. For example: if we would like to find a subgraph contains a set of nodes in size n called QE, and a set of edges in size m called QS.

```
SELECT DISTINCT $E_1.modelID, E_i.ElementID$
FROM BPElements as $E_1$,..., BPElements as $E_m$, BPEdges as $S_1$,..., BPEdges as $S_n$
WHERE
$\forall_{i=2}^{m}(E_1.modelID = E_i.modelID)$
AND $\forall_{j=1}^{n}(E_1.modelID = S_j.modelID)$
AND $\forall_{i=1}^{m}(E_i.ElementName = QE_i.ElementName)$
AND $\forall_{i=1}^{m}(E_i.ElementType = QE_i.ElementType)$
AND $\forall_{j=1}^{n}(S_i.EdgeType = S_J.EdgeType)$
AND $\forall_{j=1}^{n}(S_j.SElementID = E_{f(S_j.SElementID)}.ElementID$
AND $S_j.DElementID = E_{f(S_j.DElementID)}.ElementID)$;
```

Figure 2.5: Template for the filtering step of BPMN-Q queries in SQL.[6]

Each referenced table E_i in this template denotes a specific instance from the table BPElements and translates the data of a single element of the collection of query nodes QE. The information of one edge from the set of query edges is mapped to each referenced table S_j, which represents an instance from the database BPEdges. The mapping function between each QE element and its corresponding BPElements table instance E_i is f.

The second phase is to verify the candidates. This phase is optional. It is only applied when the query contains Path, Negative, and Negative Sequence Flow. The two processing phases can prune the not required part effectively and efficiently.

APQL:Semantic Process-Model Query Language

Several query languages like BPMN-Q to query the process model are based on the syntactic relationships between tasks. Unlike this kind of query language, A Process-model Query Language

(APQL) focuses on the semantic relationship between tasks. It is independent of the actual process modeling language, which means it can be applied to various modeling languages in practice. Also, the APQL is designed to exploit the semantics of the process model when querying.[40]

To discuss the syntax of the language, there are twenty basic predicates in the process model to capture the occurrences of process activities. The primary function of these predicates is to capture the semantics relationships between tasks in a process model like $posoccur(t, r)$. There is some execution of r when at least one occurrence of t happens. The exclusive and concurrent relationships between task occurrences are captured by the following two predicates. $exclusive(t1, t2, r)$ means it is never feasible for instances of t1 and t2 to co-occur during any execution of r. $concur(t1, t2, r)$: In every execution of r, if an instance of t1 happens, then an instance of t2 occurs, and vice versa. However, t1 and t2 are not causally coupled. Then, we take into account different kinds of interrelations between task occurrences. $Precedence(pred)$ or $succession(succ)$ can be used to describe the relationship in which one activity may happen right away or gradually come before or after another activity. It could be valid for any or all instances of the jobs in any or all process executions. There are 20 types of predictors in APQL.

The syntax of APQL is abstract, so it can avoid dedicating ourselves too early to certain keyword selections or the arrangement of different assertions. A basic query in APQL is a collection of Assignments with a Predicate. Those process models that meet the predicate are the outcome. When processing the Predicate, each variable is replaced with the relevant TaskSet that the Assignment assigned to it. A TaskSet may be created over other TaskSets by construction or application, or it may be an enumeration of tasks. It can also be defined using a TaskSetVar variable.

We can use an example to illustrate the query language. if we would like to choose every model of a process where task A occurs during some process executions and task B occurs during all process executions. This query can be represented by a grammar tree:

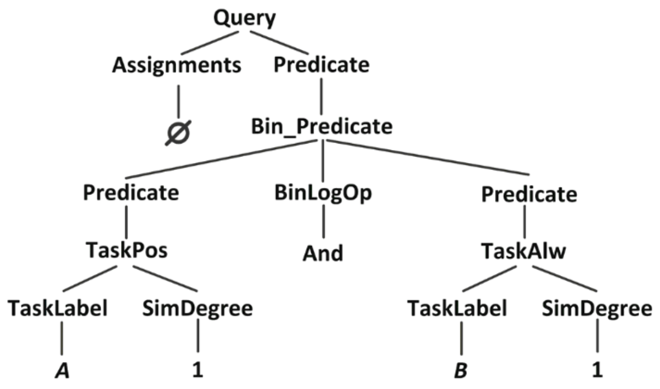


Figure 2.6: Grammar tree of the example APQL query[40]

A predicate can comprise a straightforward TaskPos with the semantics defined by the fundamental Predicate $posoccur$. It will select the process model where Task A occurs during some process executions.

a TaskAlw specifies what the fundamental Predicate $alwoccur$ means in its intended semantics. It will select the process model where Task B occurs during all process executions.

A TaskRel can be defined recursively as a binary or unary Predicate by using logical operators, with the intended meaning that all process models meeting that particular relation should be retrieved. In this query example, the logical operator is "And," which means the result needs to satisfy both predicates.

The APQL can also deal with more complex problems like Choosing all process models where task H's immediate predecessors are task B's immediate successors during some process execution. This question involves two task sets: task H's immediate predecessors and task B's immediate successors. So we need to use the *Assignment* part in the query.

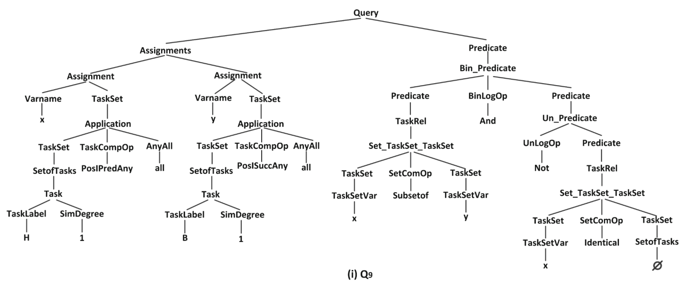


Figure 2.7: Grammar tree of the example APQL query[40]

As we said previously, a query can be divided into an assignment and predicate parts. In this query example, we will find the two task sets in the assignment part, and each task set has a variable name. When evaluating the predicate part, this variable name will be replaced by the actual task set.

APQL is distinct from other languages. Its abstract syntax and semantics are independent of process modeling languages (such as BPEL or BPMN). This allows APQL and its query evaluation mechanism to be implemented in other process modeling languages. APQL may describe all potential temporal-ordering connections (precedence/succession, concurrence, and exclusivity) between individual tasks.

Querying Workflow with VsTrails

VisTrails is a system to assist in involving data exploration via workflows.[8] The idea of the provenance of processes is a brand-new idea presented with VisTrails.

A most important feature of the VisTrails system is the sophisticated provenance structure. Similar to a database transaction log, VisTrails keeps track of how workflows are implemented. These modifications adequately identify the source of data products and provide insight into how processes develop over time.[16]

The query mechanism of Vistrails is not like the text text-based languages. As the workflows always display as graphs, using text-based language for querying needs to encode the graph to text. Vistrails adopts a query-by-example mechanism. Users construct queries precisely like they would construct individual pieces in a workflow model. Also, users can set a set of specific parameters to filter. The query results will be displayed visually.

The core part of querying is to compute the difference between different workflows. VisTrails has a visual difference feature that allows users to distinguish between two processes by coloring components and links differently depending on which workflow corresponded.

Workflow analogies mitigate these changes we mentioned above by dynamically modifying processes based on a change-based template. There are three steps to obtain the workflow analogies.

- Workflow Differences. The most basic method to compute the difference between workflows is to compare two different workflows. As we mentioned in the previous section, according to the sequence changes, it is easy in the Vistaril system to determine the distinction between the two linked workflows.

- Workflow Matching. Another important thing to get the workflows analogies is to find a relationship between the two starting workflows. This is not easy. Because the workflows are directed acyclic graphs in Vistaril, this problem is analogous to graph matching, which is an NP-Hard problem. It is hard to be estimated quickly and effectively within a subpolynomial factor.[20] Nevertheless because modules include well semantics, we can represent this probabilistically with a high likelihood of success. The key is to balance the regional compatibility between modules with global topology similarity.

- Applying the Analogy. After computing the difference and mapping the start workflows, the next step is translating the difference based on the matching. To accomplish this, we must interpret each unique modification using the derived matching. For example, if one change is to link components a and b, and the matching indicates that a and b in pipeline A are equal to components c and d in pipeline C, the change turns connecting modules c and d. The interpreted changes are then implemented in C, resulting in the creation of a new pipeline, D.

The ability of workflow systems to employ analogies and queries by example can enable knowledge reuse in building complicated processes. As a result, they may be used in real-world scenarios such as cosmology, environmental monitoring systems, bioinformatics, and medical treatment planning.

2.4.2 Event Log and Process Model Querying

In this section, we will discuss about the query language that can provide the functionalities to query both event log and process model.

Data-Aware Process Oriented Query Language

There are different approaches to query event data.

A Data-Aware Process Oriented Query Language called DAPOQ-Lang enables users to process and query data that is stored in a way that is consistent with the OpenSLEX meta model. Data models, objects, and object versions from databases are combined with events, logs, and processes from process mining as first-class citizens in OpenSLEX. It is feasible to design queries in the process mining area that are enhanced with data features with less complexity than in other general purpose query languages like SQL due to DAPOQLang's consideration of the same first-class citizens as OpenSLEX.[31]

Next, we will introduce this query language's essential elements and syntax.

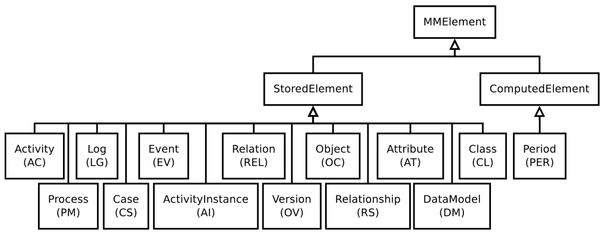


Figure 2.8: Hierarchy of DAPOQ-Lang types in UML. Arrows show subtype relationships.[31]

As we said before, all the components in the meta model can be seen as first-class citizens for the query language. They are called $MetaModelElements(MME)$ in the query language. It is the union of all the subtypes elements. There are two different types of elements in the model: the element at the next level of the MME.

- stored elements (StoredElement) The feature of such elements is that they can be found directly in the meta model, like activities, events, objects, and so on.
- computed elements (ComputedElements) Thtis type of elements are the elements determined on the basis of rest, temporal periods of cases, and temporal periods of events

The query language provides the functions operated on the MMEs, to obtain the specific subtypes elements in the model. Also, the results of the queries belong to MME sets. Based on the meta model, there are 57 essential query functions, and they are divided into five blocks. These functions formed the syntax of the query language.

The first block is the syntax to query terminal meta model elements. There are 13 functions to obtain the whole set of the corresponding type in the meta model. For example, *allRelationships* to query the stage of all class relationships, *allProcesses* to query the set of all processes.

The second block contains 14 functions. These functions accept a collection of elements of the same type as input and provide a set of elements linked to them of the type matching the function's return type. For example, $attributesOf(es)$ will return a set the set of attributes related to the input es.

The third block focuses on the elements with temporal properties. There are eight functions to deal with the period or duration query. For example, $Duration.ofDays$ to query the duration of the specified days.

The functions in the fourth block focused on the relationships between the time intervals, like Allen's interval algebra, which describes a calculus for temporal reasoning to define possible relations between time intervals.[3] The blocks' functions will compute and create periods in Allen's interval algebra format. For example, starts(a, b) describes if and only if a and b both start at the same time, but a is shorter.

The functions in the fifth block enable users to obtain the attribute values of elements in the meta model. For example, the process *getAttributeEvent* will returns the value for an attribute of an event.

We can use an example to show how this query language works. Suppose we would like to change a customer's address from "Fifth Avenue" to "Sunset Boulevard" due to an event that occurred between two dates.

```
def P1 = createPeriod("1986/09/17 00:00","2016/11/30 19:44","yyyy/MM/dd HH:mm")

casesOf(
  eventsOf(
    versionsOf(
      allClasses().where {name == "CUSTOMER"}
    ).where { changed([at:"ADDRESS", from:"Fifth Avenue", to:"Sunset Boulevard"])}
  ).where
  {
    def P2 = createPeriod(it.timestamp)
    during(P2,P1)
  }
)
```

Figure 2.9: An example of DAPOQ-Lang Query[31]

This query includes the functions like allClasses in the first syntax block, Caseof ,Eventsof in second syntax block, createPeriod in the third syntax block, ETC. We can use the grammar tree to show how this query to be evaluated.

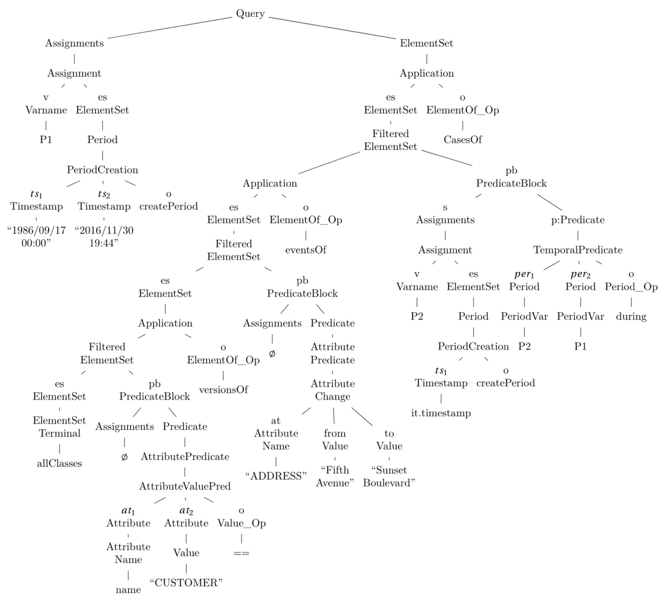


Figure 2.10: Grammar Tree of the DAPOQ-Lang Query example[31]

The grammar of DAPOQ-Lang query language is similar to APQL we mentioned in the previous section. It also includes two parts in the grammar tree. The assignment part shows how to create the period and it has a variable name to represent the period. When the query is being evaluated, the variable name will be replaced by the actual value of the period. Unlike the APQL can only select the elements based on constraints. DAPOQ-Lang supports the function of changing the value of the selected element like the *AttributeChange* component in the grammar tree.

Celonis PQL: A Query Language for Process Mining

The software architecture of Celonis includes Celonis PQL as a key element. This language is used by all Celonis apps to query data from a data model. Along with the actual data from the source systems, the data model also includes metadata such as schema details and the connections between the tables that use foreign keys.[44]

Celonis Process Query Language (Celonis PQL) is a domain-specific language for business users suited for a particular process data model. More than 150 expressions, spanning from process-specific operations to machine learning and mathematical operators, are covered by Celonis PQL. Although it has a syntax similar to SQL, it is tailored for process-related queries.[44]

Celonis PQL aims to provide a query language to perform process mining tasks(like process discovery, conformance checking, ETC.) based on the event data. This query language is based on a relational data model.

SQL serves as the basis for Celonis PQL, and there are some differences between the two query languages. These differences can reflect the features of Celoins PQL.

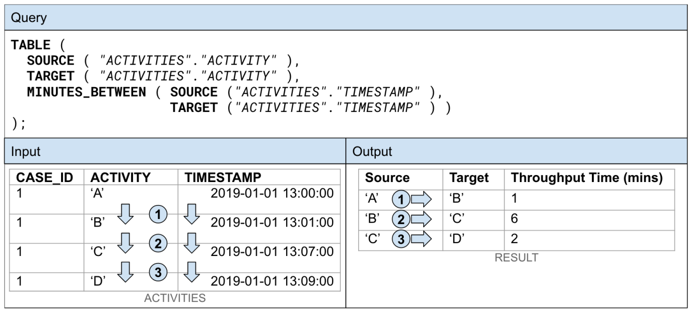
The varied language scope between SQL and Celonis PQL is a crucial distinction. Therefore, not all of the SQL operators are supported by Celonis PQL. This is because only the operators required for the intended use cases are implemented, and user requirements drive language development.

The absence of a data manipulation language and data definition language support is another significant deviation from SQL. There is no need to manually change and update the data using the query language in the process mining scenario because all updates should originate from the source systems.

Unlike SQL, Celonis PQL is domain-specific and provides a variety of process mining operators that are not offered in SQL. As a result, Celonis PQL effortlessly connects the process view with the data. There are four basic operators for process mining in Celonis PQL: SOURCE and TARGET, VARIANT, and CONFORMANCE.

SOURCE and TARGET operators can be applied to connect one event to another immediately after or eventually by figuring out the variance of the relevant timestamps. TARGET refers to the event's immediate aftermath, whereas SOURCE always refers to the actual occurrence. Therefore, an event and its subsequent event can be combined in the same row in a table using SOURCE and TARGET.

There is an example of throughput time computation using SOURCE and TARGET operators.


Query

```
TABLE (
  SOURCE ( "ACTIVITIES"."ACTIVITY" ),
  TARGET ( "ACTIVITIES"."ACTIVITY" ),
  MINUTES_BETWEEN ( SOURCE ("ACTIVITIES"."TIMESTAMP" ),
                    TARGET ("ACTIVITIES"."TIMESTAMP" ) )
);
```

Input

CASE_ID	ACTIVITY	TIMESTAMP
1	'A'	2019-01-01 13:00:00
1	'B'	2019-01-01 13:01:00
1	'C'	2019-01-01 13:07:00
1	'D'	2019-01-01 13:09:00

ACTIVITIES

Output

Source	Target	Throughput Time (mins)
'A'	'B'	1
'B'	'C'	6
'C'	'D'	2

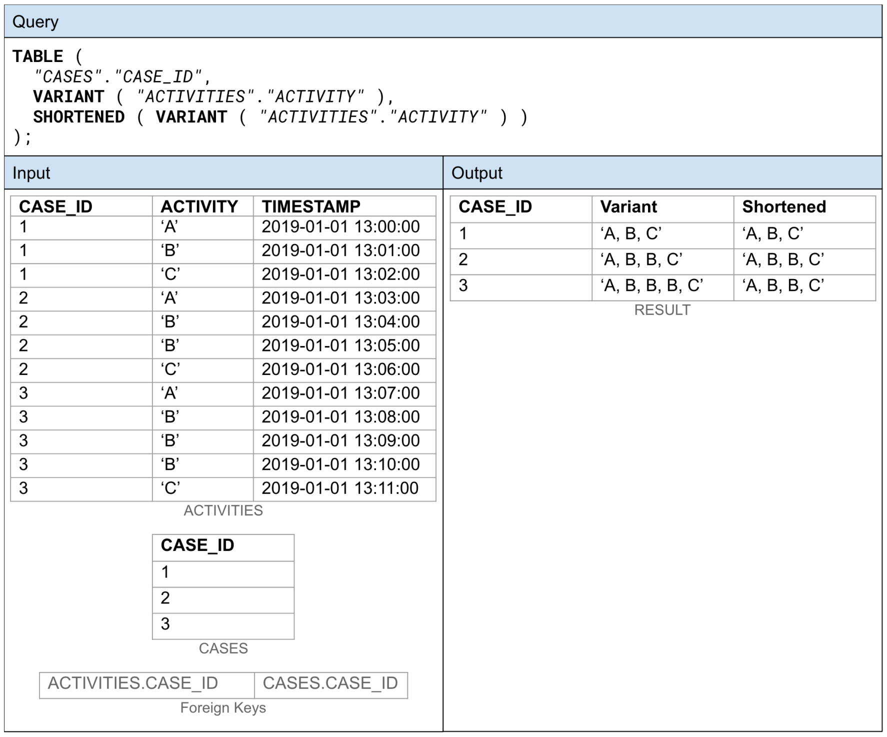
RESULT

Figure 2.11: an example calculation of throughput time using the SOURCE and TARGET operators

SOURCE and TARGET receive a column in a table as the input, MINUTES_ BETWEEN will operate on each element in the column. And the results will be displayed in a column.

Process mining requires the calculation of variations to be successful. Most process discovery algorithms employ them as input rather than the raw events and cases to significantly speed up the calculation, such as the Inductive Miner[25] or the Heuristics Miner[45]. When computing variations, Celonis PQL offers the VARIANT operator, which compiles all case events into a string that denotes the case's variant.

There is an example of the variant operator.



Query

```
TABLE (
  "CASES"."CASE_ID",
  VARIANT ( "ACTIVITIES"."ACTIVITY" ),
  SHORTENED ( VARIANT ( "ACTIVITIES"."ACTIVITY" ) )
);
```

Input

CASE_ID	ACTIVITY	TIMESTAMP
1	‘A’	2019-01-01 13:00:00
1	‘B’	2019-01-01 13:01:00
1	‘C’	2019-01-01 13:02:00
2	‘A’	2019-01-01 13:03:00
2	‘B’	2019-01-01 13:04:00
2	‘B’	2019-01-01 13:05:00
2	‘C’	2019-01-01 13:06:00
3	‘A’	2019-01-01 13:07:00
3	‘B’	2019-01-01 13:08:00
3	‘B’	2019-01-01 13:09:00
3	‘B’	2019-01-01 13:10:00
3	‘C’	2019-01-01 13:11:00

ACTIVITIES

CASE_ID
1
2
3

CASES

ACTIVITIES.CASE_ID	CASES.CASE_ID

Foreign Keys

Output

CASE_ID	Variant	Shortened
1	‘A, B, C’	‘A, B, C’
2	‘A, B, B, C’	‘A, B, B, C’
3	‘A, B, B, B, C’	‘A, B, B, C’

RESULT

Figure 2.12: VARIANT operator example with and without decreased self-loops

However, in other applications, it is not essential how frequently an activity is repeated, but rather whether or not there is a self-loop. In such circumstances, the VARIANT operator can be wrapped by the SHORTENED command, which limits the number of repetitions of self-loops. It is feasible to abstract from recurring actions in this way, reducing the number of different versions. An optional parameter specifies the maximum length of the self-loops. The maximum cycle length is set at 2 by default.

The Celonis PQL is a query langauge can deal with both event log side and process model side. The operator SOURCE and TARGET demonstrates the language's ability to handle event logs, and the VARIANT operator demonstrates that the language can also take care of the process model aspect.

Another significant process mining approach that links a process model to an event log is conformance checking, which is used in addition to process discovery.[11] Celonis PQL provides conformance checking capability through the CONFORMANCE operator. This operator takes a column of activities and a description of the process model as the input. It repeats the names of the activities from the input column. As an outcome, the activity table receives a temporary integer column. A row's value in this new column shows whether or not there is a conformance problem. Additionally, the process model's associated activities and the violation's kind are contained in this value.

In this section, we discuss the query principles of four query languages and a scientific workflow system. These query languages can be divided into two types. According to the model based on the query language, they can be divided into query according to relational data model and query according to semantics. For example, BPMN-Q and Celoins QL query languages are based on relational data model for query. The data is stored in the table. Although APQL and DAPOQ-

Lang are based on different data storage models, the same thing is that they both build their own complete grammatical structure for the query language.

2.4.3 Graph Database and its Query Language

A graph database is a storage system representing and storing data using graph topologies with nodes and edges.[32] A (labeled) property graph model is the most often used graph model in the area of graph databases.[34]

The property graph comprises linked entities (nodes) that may store an unlimited number of properties (attributes) represented as key-value pairs. Labels can be assigned to nodes and edges to indicate their various responsibilities in the application domain. The label is referred to as the type in specific techniques. Labels can also associate metadata—such as index or constraint information—with particular nodes. Relationships connect two nodes by providing directed, semantically meaningful connections (edges). A relationship always has a starting point and an ending point. Relationships, like nodes, can have any attributes. Relationships frequently have quantitative features like weight, cost, distance, ratings, or time intervals. Properties make nodes and edge more informative and valuable. Unique identification is assigned to each node and edge. Because connections are effectively stored, two nodes can share any number of different types of relationships without affecting performance. It is essential to note that, while they are directed, relationships may constantly be handled regardless of direction. The property graph model is concerned with data structures known as labeled and directed attributed multigraphs in graph theory.

We will use Neo4j, a successful graph database system, as an example to illustrate the storage mechanism of the graph database. Data is stored in Neo4j as nodes and relationships. Both nodes and relationships can have key-value attributes. Values can be either a primitive or an array of primitives of the same type. Nodes are frequently used to represent entities, although relationships may also be used for this purpose, depending on the domain. Internal unique identities for nodes and edges can be utilized for data search. Nodes cannot directly refer to themselves[22]. The semantics can be stated by introducing directed node relationships. Graph processing in Neo4j involves random mainly data access, which is inconvenient for Big Graphs. Graphs that cannot fit in main memory may need additional disk accesses, substantially impacting graph processing.

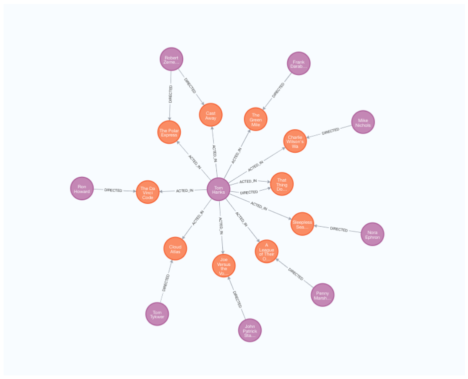


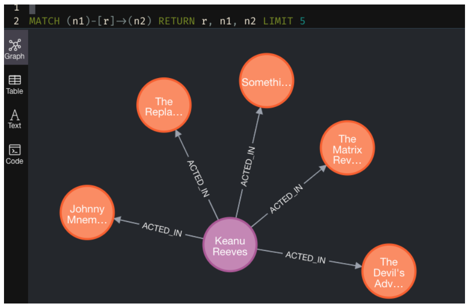
Figure 2.13: Graph Database Example Provide by Neo4j

Cypher: Graph Query Language

Query capabilities are essential for every DBMS. Of course, those employed in graph databases are derived from the accompanying graph model.[4]. Graph databases are frequently supplied with a declarative query language. Cypher, which works with the Neo4j database, is today's most well-known graph declarative query language. Cypher uses the Neo4j data model of property graphs, which we covered before. Cypher commands are ad hoc graph data queries loosely based on SQL syntax.

A Cypher query accepts a property graph as input and returns a table as output. These tables may offer bindings for parameters that observe specific patterns in a graph, along with some further processing. Queries are structured linearly by Cypher. Users can conceive of query processing as commencing at the beginning of the query text and moving linearly to the finish.[15] In Cypher, the projection is declared as RETURN after the query rather than at the beginning. Cypher extends the linear query stream to query composition. The projected table from the query part before WITH serves as the driving table for the query component after using WITH. The same projections as RETURN are permitted by the WITH clause, including aggregations.[15]

Cypher is unusual because it allows for the visible matching of patterns and relations. Cypher has an ASCII-art syntax in which (nodes)-[: ARE CONNECTED TO] rounded brackets represent →(other nodes) for circular (nodes) and -[:ARROWS]→ for relationships. When you create a query, you make a graph pattern out of your data. The MATCH clause in Cypher employs such a pattern and inserts new rows (synonymous with records) into the queried graph with bindings to the matched occurrences of the pattern. For example, if we use "MATCH (n1)-[r]→(n2) RETURN r, n1, n2 LIMIT 5" querying a graph, it will return a table with five node-to-node relationships.



```
MATCH (n1)-[r]→(n2) RETURN r, n1, n2 LIMIT 5
```

Figure 2.14: MATCH Query Example

Cypher has a powerful tracking language for altering the graph. Updating clauses employ the same visual graph pattern language as Cipher's rest and give the same straightforward, top-down semantic paradigm. CREATE is used to create new nodes and relationships, DELETE is used to remove entities, and SET is used to edit properties. For example, "CREATE (friend: Person name: 'Mark')" will create a node in the graph with the label Person and property name:'Mark.'

Cypher is a well-established powerful query language for the property graph model that is seeing increased acceptance in various businesses and initiatives. The language adds new capabilities, exceptional support for numerous graphs, and query composition. It is presently being published as a fully-specified standard that may be independently implemented utilizing multiple architectures and variable storage and query optimization algorithms under the auspices of the openCypher

Implementers Group.[15]

Chapter 3

Problem Statement

Think about we have an infinite set of attribute names $\mathcal{A}$, an infinite set of atomic values $\mathcal{V}$. An *attribute*is a pair $< a, v >$ where $a \in \mathcal{A}$, $v \in \mathcal{V}$. So the there is an infinite set of attributes $\mathcal{P}$ denoted as $\mathcal{P} = \mathcal{A} \times \mathcal{V}$.

Definition 1 An entity is a tuple

$$< id, P >$$

, where id is a string used as an identifier, and $P \subset \mathcal{P}$ contains the attributes of this entity.

An entity can be anything that exists in real life. For example, it could be a person, so the attributes of the person entity could be "name: Tom," "age: 18," ETC.

To record the start and end times of each activity, we assume there is an ordered time domain$\mathcal{T}$. A time interval between two time points t_start and t_end denoted as $[t\_start, t\_end]$, where $t\_start \in \mathcal{T}$ and $t\_end \in \mathcal{T}$.

Definition 2 An activity can be denoted as a tuple

$$< id, subactivities, inputDataset, outputDataset, Implements, period >$$

Where *id* is the identifier, *subactivities* is a sequence of activities. Both *inputDataset* and *outputDataset* are sequence of *Dataset*. *Implements* contains a process id to indicate which process this activity belongs to, and *period* is a time interval between two time points.

An activity could be a collection of subactivities during a specific period. For example, "Composition Analysis" could be an activity. It includes the events like "Sampling," "Spectral Analysis," "Impurity Analysis," Etc.

There is a big difference between *Activity* and *Process*. The *Activity* is a real-time object. It can be identified by the *id* and the specific period. *Process* is not a real-time object. It can only be determined by *id*.

Definition 3 An process p is a tuple denoted as

$$< id, subProcess >$$

where *id* is the identifier of an event, *subProcess* is a sequence of processes.

Each activity calls for a dataset to be provided as the source information, and each activity will produce a dataset to keep track of the outcomes of that action.

Definition 4 A dataset is a tuple

$$< id, version, name, type, host, port, username, password, directory >$$

. *id* is the identifier of the dataset. *version* is a number that can indicate how many times the dataset has changed. *name* is a display value to show what the dataset is about. *type* is to show what type of repository it is. The value of *type* could be: HDFS, MySQL, PgSQL, FS, MongoDB, ETC,. *host* is the IP address, localhost or domain name for user to retrieve the dataset. *username* and *passwaord* are the username and password to connect to the repository. According to the type of the dataset, if it is the FS or HDFS, *directory* is the full path of the directory. If the type of dataset is the MySQL or PgSQL, *directory* is the DB name.

Table 3.1: Model Classes in Database

Model Classes		
Object(type: Class)		

Entity{
type: Class
isA: Object
}
	Process{
	type: Class
	isA: Object
	id: string
	subProcess: a sequence of Processes
	
	}
		Activity{
		type: Class
		isA: Entity
		id:string
		subActivities: a sequence of Activities
		
		input: a sequence of Datasets
		output: a sequence of Datasets
		implements: Process
		period : [t1, t2]
		}

Dataset{
type: Class
isA: Entity
id: string
version: integer
type: HDFS, MySQL, PgSQL, FS
host: IP address, local host, or domain name
port: integer
username: string
password: string
directory: full path of the directory or the database name
}		

The information about the database that was just shown can store information on different processes, the outcomes of these processes, Etc.We hope to build a new data model that integrates the information of drug clinical testing with the concept of process model, and can store drug clinical trial information in the database. The information stored in the database will be used as evidence to provide decision support for users.

We also hope that the system can provide users with the function of process query about the new data model. We discussed in related work that process query can be divided into process model query and log query. We need to design and implement the query function of the system to solve the above problems.

As a decision support system, we need to provide users with the basic functions of the decision model: allows users to store decision models and obtain decision support

Chapter 4

System Description

Our system is can be divided into three parts. We designed a data model stored in the database to describe the various information needed in pharmaceutical sciences. A software part implements the main functions of the system. As a decision support system, it can provide two main functions for users. The first function is to query the data model and instances information. The second function is to use view as a decision model. The third part of the system is the user interface. It can visualize the data to the user, and at the same time accept the user's instruction and pass it to the software part.

In this chapter, we mainly introduce the design of the data model and the design of the system function.

4.1 Data model design

The model needs to include all the information contained in the clinical trials. The establishment of the model mainly refers to the information provided by the clinicaltrials.gov website. ClinicalTrials.gov is a Web-based resource that enables patients, their families, health care providers, researchers, and the general public quick access to information on officially and privately funded clinical trials on a variety of diseases and ailments. All the elements are defined with respect to classes in the data model.

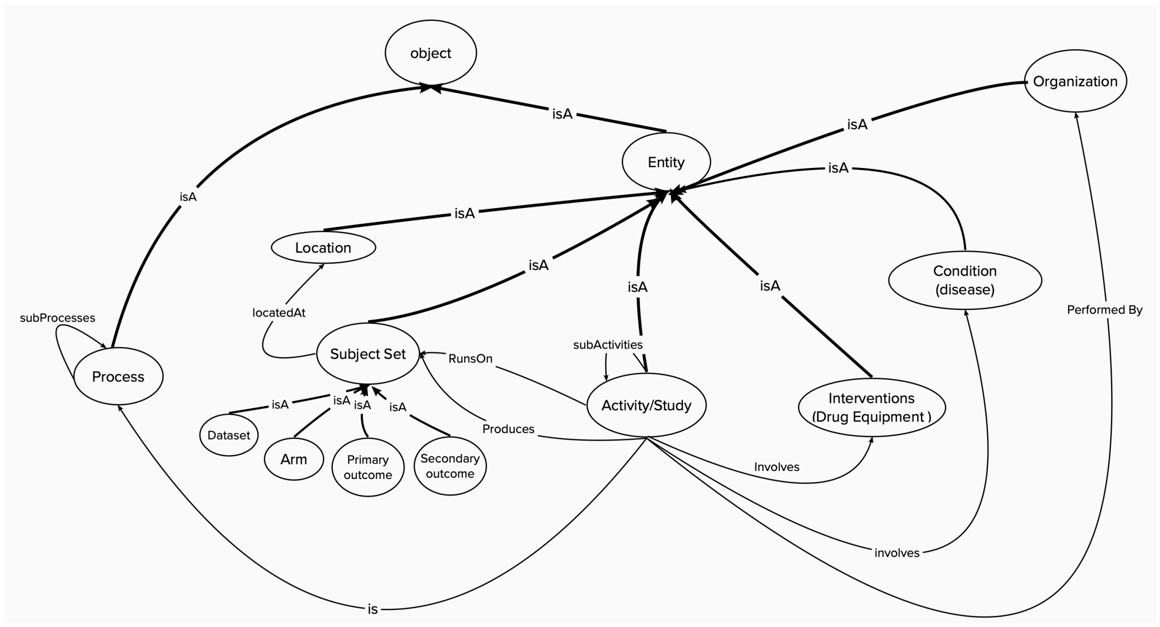


Figure 4.1: Data Model

In addition to the object, entity, process, dataset, and other parts discussed in the problem statement, in order to better describe the process of drug testing, some other parts have been added to the model.

The element *Activity/Study* contains following attributes:

- nct_number: Each clinical trial registered on ClinicalTrials.gov is assigned a unique identification code called NCT number.
- official_title: The formal title of a protocol used to identify a clinical investigation or a brief title expressed in layman's terms.
- Status: The current recruiting status or the enhanced access status is shown by this attribute.
- study_phase: According to FDA standards, the stage of a clinical trial in which a medication or biological product is being studied (FDA). The phase is determined by the study's purpose, the number of participants, and other factors. The phases are as follows: Early Phase 1 (formerly known as Phase 0), Phase 1, Phase 2, Phase 3, and Phase 4. Not Applicable is used to describe studies that do not have FDA-defined stages, such as device trials or behavioral therapies.
- study_type: This attribute explains what a clinical trial is like. Interventional studies, commonly known as clinical trials, observational studies (including patient registries), and increased access are examples of study types.
- expanded_access: A means of obtaining a medicinal product that has not received FDA approval for people with significant illnesses or conditions who are unable to take part in research trials (FDA). Likewise known as compassionate usage. There are several sorts of increased access. Available: Patients who are not participants in the clinical research may be able to obtain access to the medication, biologic, or medical device under study. Extended access is presently

offered for this experimental treatment. No longer available: Extended access was formerly accessible for this intervention, but it is not now and won't be in the future. Temporarily not available: This intervention does not yet have expanded access, but it should do so in the future. Approved for marketing: The U.S. Food and Drug Administration has given the intervention public use approval.

- start_date: The precise day that the first participant in clinical research was signed up. The researchers "estimated" study start date is the day they anticipate the investigation to begin.

- end_date: The final data collection date for the primary outcome measures, secondary outcome measures, and adverse events in a clinical trial is the date of the final participant's visit, which is also known as the last visit date.

- has results: This attribute indicates whether the current experiment has results.

The central component of the data model is the study, to which many other components are connected. For instance, research may contain certain interventions as well as some diseases.

The *intervention* in the data model means an activity or procedure that is the subject of a clinical investigation. Drugs, equipment, treatments, vaccinations, and other goods that are either under development or currently on the market are interventions. Non-invasive methods like education or altering a diet and exercise routine can also be used in interventions. It has the following attributes:

- interventions_type: This attribute is used to describe the type of intervention. It could be drugs, medical devices, procedures, vaccines, and other products.

- name: This attribute used to identify the intervention

- description: Interpret and describe this intervention

The *disease* in the data model is the ailment, problem, syndrome, condition, or damage that is the subject of study. It can also refer to other aspects of health, such as longevity, quality of life, and health risks.

The subject set class is used in the data model to describe the input and output required by activities. In the problem description section, we describe the inputs and outputs of activities as datasets. This is a broader expression of the same idea. In our data model, we use arm and study outcome to more specifically describe the input and output of the study.

The *arm* in the data model means a group or subset of clinical trial participants who get a specific intervention/treatment, or no intervention, based on the study protocol. It has the following attributes:

- arm_group_label: This attribute used to identify different arm group

- arm_group_type: A basic briefly describe of the clinical trial arm. It specifies the function of the intervention provided to participants. Arms are classified as experimental arm, active arm, placebo arm, sham arm, or no intervention.

- description: Interpret and describe this arm group

A predetermined assessment used in clinical trials to assess the impact of an intervention or treatment on participants is stated in the protocol. a measurement or observation used in observational research to characterize the distribution of illnesses or features, or relationships between exposures, risk factors, or treatments. There are two different types of study outcomes: primary outcome and secondary outcome. The attributes of the primary outcome are:

- measure: The most significant intended outcome measure in a clinical trial protocol for evaluating the efficacy of an intervention/treatment. The majority of clinical trials have one primary outcome measure, but some have many.
- time_frame: The time frame is the precise period of time point or time window during which you will be gathering participant-specific data for each outcome measure.
- description: Interpret and describe the results.

The attributes of the secondary outcome are almost the same. The difference is the interpretation of the attribute *measure*. It is not as significant as the primary outcome measure for assessing the effectiveness of an intervention, but it is still worth considering.

4.2 System function module design

This decision support system has two main functions. One function is to query based on the attribute information of the object, and the other function is to create views as decision models.

4.2.1 Query

Each element in the data model has its own properties. The function of this part is to query the object and all information related to this object required in the database according to the combination of attributes entered by the user. This query function can not only query schema classes in the data model. At the same time, the instance of the class can also be queried as the target object.

This function can be divided into the following steps:

1. The user enters the desired combination of attributes in key-value format.
2. Convert the attributes set into a query statement and transmit it to the database to obtain the target data node
3. According to the identification of the target data node, retrieve all the properties of the target object and all other objects related to it in the database
4. Display the attributes and relationships information of the target data node to the user as the query result

For example, We want to query an instance of the primary study outcome. It needs to have specific properties: $time\_frame : 6months$ and $measure : variation\ in\ grades\ of\ mucositis$. The system receives the user's input, transforms the attribute information into a query, and sends it to the database. The database then sends back to the system all the information related to the queried instance, including the attribute and relationship details, and the system then demonstrates the above information to the user.

The design of the query function does not create a new query language, but only allows users to input key-value pairs of attribute information to query specific objects. When displaying the query results, not only the attribute information of the queried object will be displayed, but also its relationships information will be displayed to achieve the purpose of the process query. As we discussed before, process querying aims to recognize foundational algorithms, assessment, and analytics over processes to encourage their centralized development and improvement for later reuse in practical situations[41]. But in the face of the actual situation of this system, only one data model is involved in our database, and the structure of the data model is simple and clear. By returning the relationships of the characteristic node, we can achieve the purpose of our required process query. For example, if we query a specific study outcome, we can see which study produced such a result, or if we query a certain drug, we can see which clinical trials have been performed on this drug, etc. This functional design simplifies the input method, and it can be used without the need for users to learn a new query language, which is more convenient.

4.2.2 View

Database views are named queries that are kept in the database and may be used to save commonly performed, sophisticated queries. We design to store these views in the database to form our decision model. Next, we will introduce the decision model of the system in detail.

For each query of the database, we can regard it as a simple decision model. For example: Are there more than three trials for the drug Placebo? It can be disassembled into two parts. The first part: queries the number of all experiments involving the drug Placebo, and the second part: determines whether the number of experiments is greater than three. Therefore, a simple view in the system contains the following attributes:

- name: the name of the view, can be used as an identification.
- expression:
$$Q(query)\ + operation\ symbol +\ \{result\ set\}$$
  The operation symbol includes$>, <, =, \leq\ and \geq$

The expression for the above simple view example is Q(the number of all experiments involving the drug Placebo) > 3. This expression can be thought of as a boolean expression. When it is evaluated, it will return a true or false result. The result set can be the number in the above example or a collection of values of any attribute. At this time, we use the following method to evaluate the expression: the return value of the Q (query) part can be a set, which is different from the operation symbol to judge the relationship between the returned set and the result set. When the operation symbol is$\geq\ or >$, if the result set is a subset or proper subset of the set returned by the query, then the expression returns true, otherwise, it returns false. When the operation symbol is$\leq\ or <$, if the set returned by the query is a subset or proper subset of the result set, then the expression returns true, otherwise, it returns false.

A general decision model will eventually produce a true/false result, and when we evaluate a Boolean expression, it will also produce a true/false result. Having a simple view, we designed a more complex view to simulate a more complex decision model by using boolean operands *and*, *or*, *not*. This kind of view in the system contains the following attributes:

- name: the name of the view, can be used as an identification.
- expression: a boolean expression containing simple view as parameters.

Some decisions may require more than one simple view to construct, for example, experiments involving the drug Placebo require more than three items, and the organizations conducting these experiments must be certain specific organizations. In this way, we cannot use a simple view to build a decision-making model. But if we have simple view1 to describe the experiments involving the drug Placebo require more than three items, and simple view2 to describe the organizations conducting experiments involving the drug Placebo must be a subset of certain specific organizations. Using simple view1 and simple view2 as parameters, build a Boolean expression "simple_view1 and simple_view2". When evaluating this expression, we can get the desired result. If both simple view1 and simple view2 can be satisfied at the same time, then return true

Whenever the user loads the view function page or edits a certain view in the database, the system will evaluate all the views in the database according to their expression, and get a true or false result, providing decision-making assistance for the user.

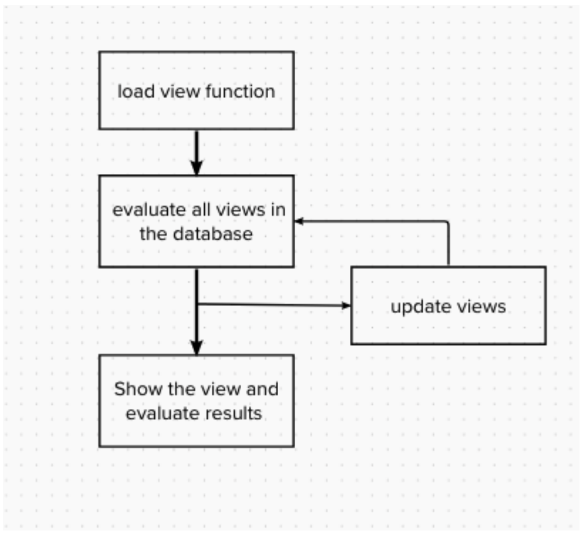

load view function
evaluate all views in the database
update views
Show the view and evaluate results

Figure 4.2: View Functional Flowchart

Chapter 5

System Implementation

We implemented a decision support system in pharmaceutical science to query and modify data and views. The system is divided into three parts: database, software system, and user interface. We use the Neo4j graph database for storage. The software system is partially implemented in the python environment. It consists of three parts. The relationship between each part of the system is shown in the figure below

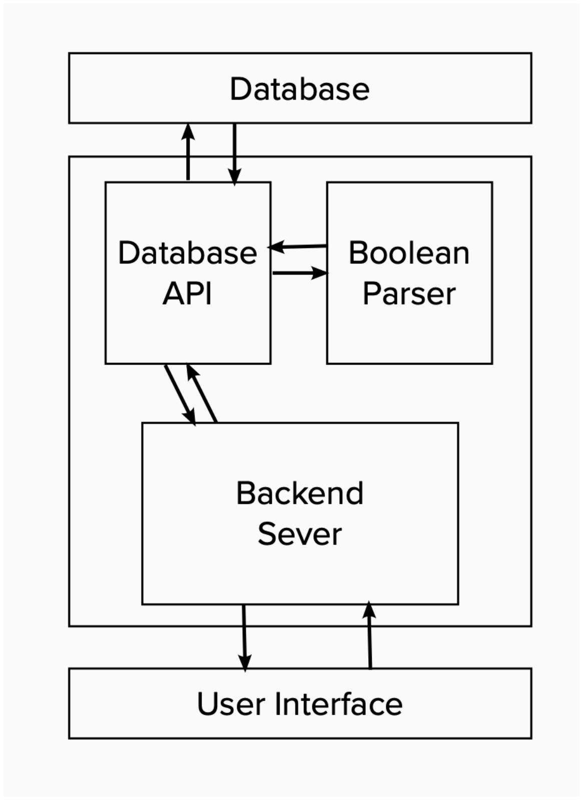


Figure 5.1: System Structure

5.1 Database

We use the neo4j graph database to implement the database part of the system. In a graph database, data is stored in the form of nodes and edges. Relationships and nodes can both have key-value characteristics. Values may be a single primitive or an array of similar primitives.

The nodes in the neo4j database have a special attribute label, which is used to implement different classes in the data model. A node in the neo4j graph database will contain the following information:

- identity: Each node in the database has a unique id. The identities in neo4j usually are a string of numbers.

- labels: Labels classify nodes into sets, with all nodes with the same label belonging to the same set. A node can have one or more labels.

- properties: Properties are key-value pairs used to store information on nodes.

The relationships specify the connections between a source and a destination node. A relationship has the following characteristics:

- connects a source and a target node.

- Has one direction.

- It must have a type (at least one type) to define (classify) the kind of connection.

- Properties (key-value pairs) can be used to store information in the connection

Cypher is a declarative query language for retrieving and manipulating data in graph databases like Neo4j. It is straightforward and expressive in design, allowing users to create complicated queries in a succinct and natural manner. Filtering, sorting, aggregating, and altering data, as well as adding, updating, and removing nodes and relationships in the graph database, are all supported by Cypher. It is an extremely effective tool for studying and altering complicated data connections in graph databases.

5.1.1 data model schema

The schema nodes in the database have two labels, one of which is **schema**, which can indicate that these nodes are all schema nodes, and the other label is the class name of these nodes. In addition, the properties will also store the names of other attributes of this object as keys, and the value part is the format or possible value of this attribute.

Model Classes Attributes			

Entity{
type: Class
isA: Object
}
	Subject set{
	Type: class
	IsA: entity
	}
		Arm{
		Type: Class
		IsA: subject set
		arm_group_type: String
		arm_group_label: String
		}
			Dataset{
			Type: Class
			Version: timestamp
			IsA: subject set
			}

Study/Trial{
Type: Class
IsA: Entity
NCT: String
Official Title: String
Status:
Active, Suspended, Terminated, or Completed
Study Phase: 1, 2, 3, or 4
Study type:
interventional study, observational study, or expanded access
Expanded Access:
Available, No longer available, Temporarily not available or Approved for marketing
Start Date: string
End Date: string
Has Result: True/False
}
	Location{
	Type: Class
	IsA: Entity
	LocationType:
	HDFS, MySQL, PgSQL, or FS
	Host:
	IP address, local host, or domain name
	port: integer
	username: string
	password: string
	directory: full path of the directory or the database name
	}
		Primary_outcome{
		Type: Class
		IsA: Subject set
		Time_frame: String
		Measure: String
		Description: String
		}
			Secondary_outcome{
			Type: Class
			IsA: Subject set
			Time_frame: String
			Measure: String
			Description: String
			}

Interventions/Treatment{
Type: Class
isA: Entity
Interventions Type:
Drug, Device, or Behavioral
Name: String
Description: string
}
	Condition(disease){
	Type: Class
	IsA: Entity
	Name: String
	}		

We can use the relationships in the neo4j database to realize the connection between schema nodes, and use type to define the name of the relationship.

When building this part of the database, we use the cypher language to create this part of the nodes and relationships in the database. For example, create a Condition schema node with its attributes in the database:

CREATE(a:schema:Condition) WHERE a.Name = 'String'

The above statement creates a node in the database with two labels, schema and condition, and creates an attribute $Name$ for this node after the SET statement, and the value is $String$. To create a relationship between two nodes in the database, we can use the following statement:

MATCH(a:schema:Entity)

MATCH(b:schema:Drug)

CREATE (b)- [:isA] $\rightarrow$(a)

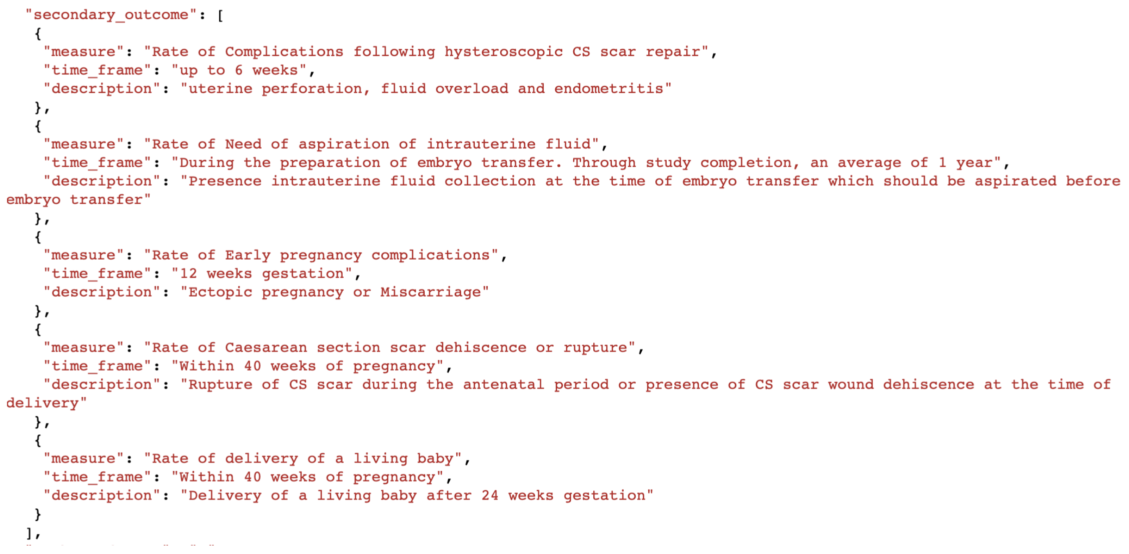
In the data model diagram of the previous section, we can see that there is a relationship between subclasses and parent classes between classes. We use the form of relationship in the database to show this relationship. In the above cypher statement example, we created a relationship of type isA between the point with label schema and Entity and the point with label schema and Drug. The direction is from the schema drug node to the schema Entity node. In this way, the connection between the subclass and the parent class between the nodes is established.

This statement first finds the two nodes to create a relationship through MATCH, and then creates a relationship through CREATE. Through the above two examples of cypher statements, we can implement the schema of the data model described above in the neo4j graph database.

5.1.2 instances of classes

The schema section describes the structure of the data in the database. The actual data is stored in the instances created by the classes. The label of each instance is the class to which it belongs. Store the attribute information of the instance in properties. Connections between instances are also represented by relationships in the database.

The instance information in the system is downloaded from the clinicaltrials.gov website. The raw data is in the format of a JSON file. However, the amount of data is huge. If we use cypher statements to insert node and relationship data one by one, the workload will be huge. Therefore, in order to import existing data into the database, we wrote a simple script to import data in python language combined with Apoc library. The Apoc library's Load JSON operations collect information from URLs or maps and convert it into one or more map values that Cypher may use. Nested data may easily be converted into graphs because to Cypher's support for deconstructing nested documents using dot syntax, slices, UNWIND. Each list may be broken down into individual rows using the UNWIND clause. These lists may consist of passed-in arguments, previously gathered results, or other list expressions.



```
  "secondary_outcome": [
   {
    "measure": "Rate of Complications following hysteroscopic CS scar repair",
    "time_frame": "up to 6 weeks",
    "description": "uterine perforation, fluid overload and endometritis"
   },
   {
    "measure": "Rate of Need of aspiration of intrauterine fluid",
    "time_frame": "During the preparation of embryo transfer. Through study completion, an average of 1 year",
    "description": "Presence intrauterine fluid collection at the time of embryo transfer which should be aspirated before
embryo transfer"
   },
   {
    "measure": "Rate of Early pregnancy complications",
    "time_frame": "12 weeks gestation",
    "description": "Ectopic pregnancy or Miscarriage"
   },
   {
    "measure": "Rate of Caesarean section scar dehiscence or rupture",
    "time_frame": "Within 40 weeks of pregnancy",
    "description": "Rupture of CS scar during the antenatal period or presence of CS scar wound dehiscence at the time of
delivery"
   },
   {
    "measure": "Rate of delivery of a living baby",
    "time_frame": "Within 40 weeks of pregnancy",
    "description": "Delivery of a living baby after 24 weeks gestation"
   }
  ],
```

Figure 5.2: Sample data provided by the clinicaltirals website

In a study data record provided by the clinical trials website, if we want to enter the secondary outcome into the database, we can execute the following cypher statement

CALL apoc.load.json("file path")

YIELD value

UNWIND value.clinical_study.secondary_outcome AS outcome

CREATE (n:Secondary_outcome)

SET n.time_frame = outcome.time_frame

SET n.measure = outcome.measure

SET n.description = outcome.description

The creation of instance information and relationship between instances in the database is done in the same way. Use Apoc.load.json to read the data first, and then create it through the CREATE clause

5.2 Software System and User Interface

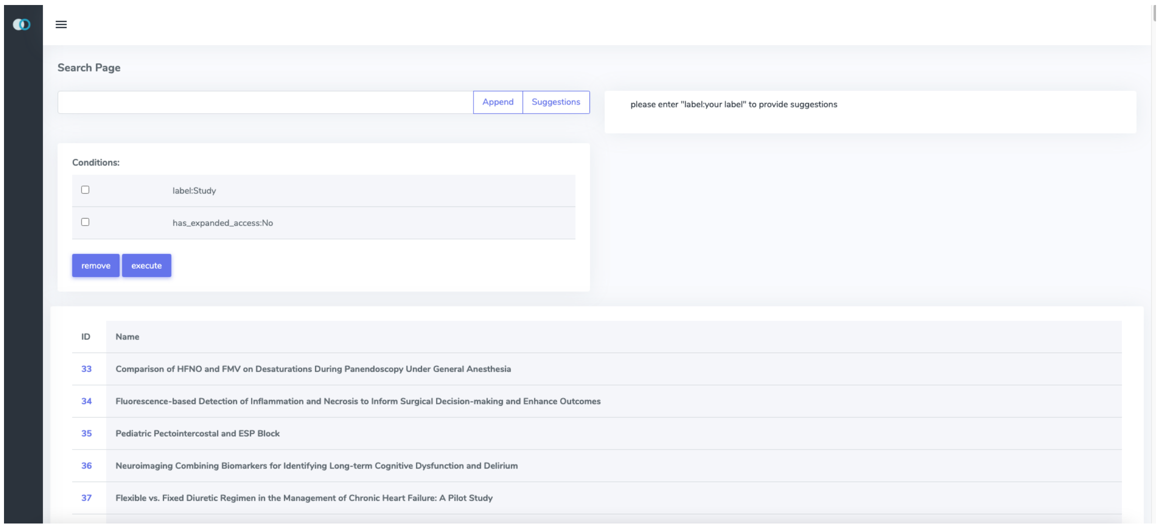
The software part mainly has three modules, all of which run in the python environment. The back-end part uses the back-end server of the flask, which is mainly responsible for data transmission with the user interface and other software modules.

The database API module is mainly responsible for the data transmission between the database and query, update, and other functions. This part mainly uses the Neo4j Python Driver library. When the system starts, the database API part will create a driver instance (provided by the neo4j library) to connect to the database. The process of collaboration between the database api part and the backend server is as follows:

1. When the system starts, create a neo4j driver instance to connect to the database.
2. The front-end will send a request to the back-end server according to the url port of the back-end server and the route required by the sent request.
3. When the backend server receives the request, according to different route it calls different APIs according to different requests, and passes the received data to the API.
4. After the API receives the data, it translates it into corresponding cypher statements according to different functions
5. The driver creates a session to execute the translated cypher statement
6. The API obtains and parses the results after running the session, and returns the data required by the user to the backend server
7. The backend passes the data to the frontend, and finally displays it to the user

The boolean praser is mainly used to process the expression in the view and obtain a true or false result. As we described in the previous section, there are two types of view expressions. A simple view is a query statement, an operation symbol, and a result set. A complex view expression is a Boolean expression composed of simple view names as variables. The boolean parser is only responsible for evaluating Boolean expressions in complex views. When calling the boolean parser, in addition to passing the boolean expression, the parameters and values contained in the boolean expression also need to be transmitted together. The boolean parser part is mainly implemented by the boolean-parser library and the eval() function in python

5.2.1 Query



Search Page

please enter "label:your label" to provide suggestions

Conditions:

☐	label:Study
☐	has_expanded_access:No

ID	Name
33	Comparison of HFNO and FMV on Desaturations During Panendoscopy Under General Anesthesia
34	Fluorescence-based Detection of Inflammation and Necrosis to Inform Surgical Decision-making and Enhance Outcomes
35	Pediatric Pectointercostal and ESP Block
36	Neuroimaging Combining Biomarkers for Identifying Long-term Cognitive Dysfunction and Delirium
37	Flexible vs. Fixed Diuretic Regimen in the Management of Chronic Heart Failure: A Pilot Study

Figure 5.3: User Interface for Query Function

The query interface mainly includes four parts. There is an text input filed for user to entering attributes key-value pairs. A table called conditions is used to store all the attributes entered by the user.Pressing the remove button in this part will delete all the attributes selected by the checkbox in the table. Pressing the execute button will send all the attributes in the table to the software part for query function. On the right side of this part is a text box. When the user presses the suggestion button, a prompt message will be given in this text box. The bottom part is the result of the query, which will display the id and name attributes of the nodes that match the user input attributes to the user. The ID part is also a link, and the user clicks the ID in the table to jump to the detailed information page of this node.

When the user presses the execute button, the attributes key-value pairs to be queried in the condition table are sent through the javascript *http get* function. This fucntion needs the url and port number of the backend server and route as parameter. The route will pass the request to the corresponding processing module in the backend. For example, if our backend server is deployed at 127.0.0.1, the port number is 2000, and the route for processing query requests is "/query", then the parameter of the get function is http://127.0.0.1:2000/query.

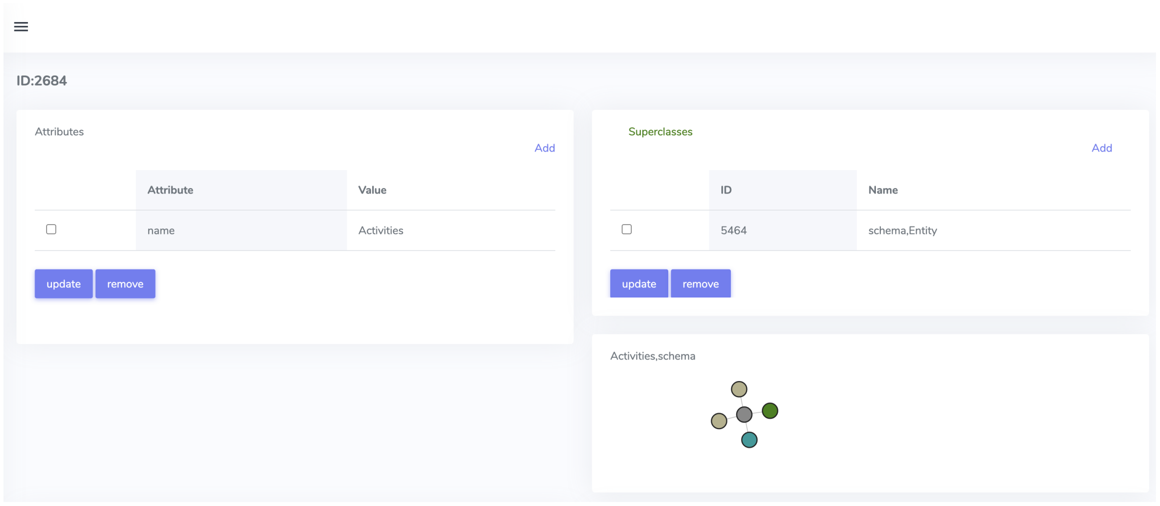
The query module in the backend server will call the query function in the database API and forward the data sent by the frontend. The driver will create a session, and we need to write the attribute name (key) and attribute value in the data into a cypher query statement. Run the cypher query statement through the session, get the result and send it back to the user interface. Then close the session. The function of this part is mainly used in "MATCH" in cypher. The MATCH clause lets you define the patterns Neo4j should look for in the database. The idea of writing the conditions (key-value pairs) sent by the user to the backend into the cypher query statement is as follows: In this way, we use a loop to write all the properties that need to be queried into the MATCH query statement of cypher.

```
Input: key_value table
1 query = " MATCH(n) WHERE"
2 for key, value in table: do
3     if key == "label": query = query + "label(n) =" + value
4     else query = query + "n." + key + "=" + value
5 end
6 query = query + "RETURN n"
7 session.run(query)
```

5.2.2 Node detail and modification

When this user interface is opened, a node id is received as a parameter in the front end. When the user interface is loaded, the back end will pass all the information of the node with the id to the front end. This page will contain the attribute and relationship information of the object queried by the query. After loading into this page, the system can provide users with the function of modifying the current object attributes and relationship information.

In this user interface, the attribute and relationship information of this node will be displayed, and if it is a shcema node, its superclass, subclass and instance information will also be displayed.



	Attribute	Value
☐	name	Activities

	ID	Name
☐	5464	schema,Entity

Figure 5.4: Node detail 1

The above is the part of the node detail user interface, and the information displayed by the nodes of different colors in the graph part is as follows:

- Gray nodes: Gray node: the node displayed on the current page
- Green node: The superclasses of the current node
- Brown node: The subclasses of the current node
- Blue node: The node whose relationship direction is inward with the current node
- Beige node: The node whose relationship direction is outward with the current node

Each node in the figure is a link, when the user clicks on one of the nodes will be redirected to the information page of the node.

Attributes update and remove

In the attribute table of the current node, our design allows users to click on the value of the attribute in the table to change it by entering the value. In the attribute table of the current node, our design allows users to click on the value of the attribute in the table to change it by entering the value. The name of the attribute cannot be changed, but the user can delete the node attribute by selecting the checkbox on the left of the attribute and pressing the remove button below. There is an Add button on the upper right of the table, and after the user presses it, a blank new row will be added to the table, and the user can enter a new attribute name and value to add a new attribute to the current node. After adding attributes to the table or modifying attribute values, you need to press the update button below to send the update information to the backend. The back-end processing flow of deleting or updating node attributes is the same.

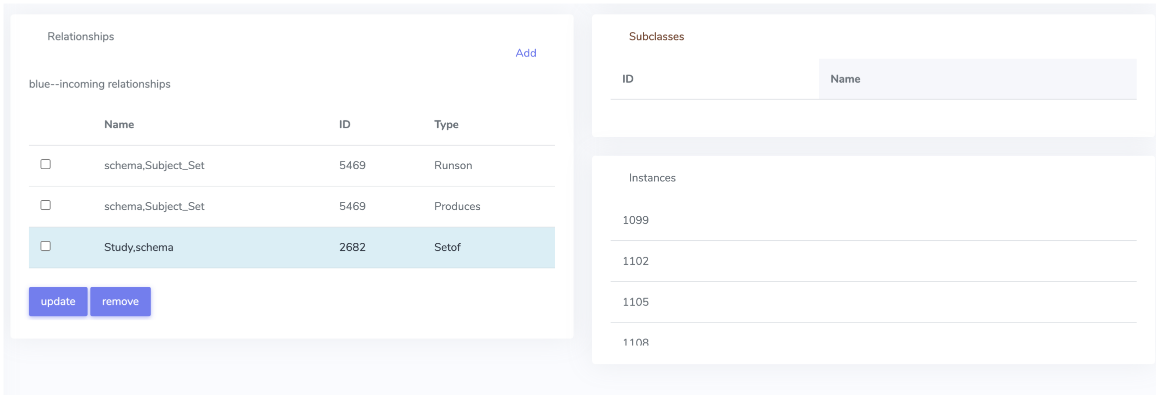
The process of forwarding the user request to the backend by the front end is the same as the forwarding process of the previous query function. The difference is that the route of the parameter part of the front-end get request is different, and the route of this part is http://127.0.0.1:2000/update_attribute. The front end sends the updated attribute table and the ID of the current node to the update_attribute module of the back end through this route. The module calls the database API, and the corresponding update_attribute part of the database API writes the attribute table and node ID as a cypher query, and the driver creates a session to process the query, and finally the attributes are successfully updated.

```
  Input: id, attributes_table
1 query = " MATCH(n) WHERE id(n) = id"
2 for for key, value in attributes_table: do
3 |   query = query + "SET n." + key + "=" + value
4 end
5 query = query + "RETURN n" session.run(query)
```

The cypher statement algorithm for constructing and updating attributes is also building a string of cypher query statements and hand it over to the session for processing. To deal with the delete attributes task we only need to replace the SET part with DELETE

Relationships and Superclasses update and remove

Our design only allows users to change the relationship with the current node relationship is superclass or the direction is outgoing. And the system only allows nodes to modify the relationship to existing nodes in the database.


Relationships

Add

blue--incoming relationships

	Name	ID	Type
☐	schema,Subject_Set	5469	Runson
☐	schema,Subject_Set	5469	Produces
☐	Study,schema	2682	Setof

Subclasses

ID	Name

Instances

1099
1102
1105
1108

Figure 5.5: Node detail 2

The design of the superclasses and relationships tables of the user interface is roughly the same as that of the attributes table. You can modify the data by clicking the corresponding row. The Add button on the top right of the table can add a new blank row to the table, and you can delete the relationship (or superclasses) by selecting the checkbox and pressing the remove button. You can also update the information through the update button.

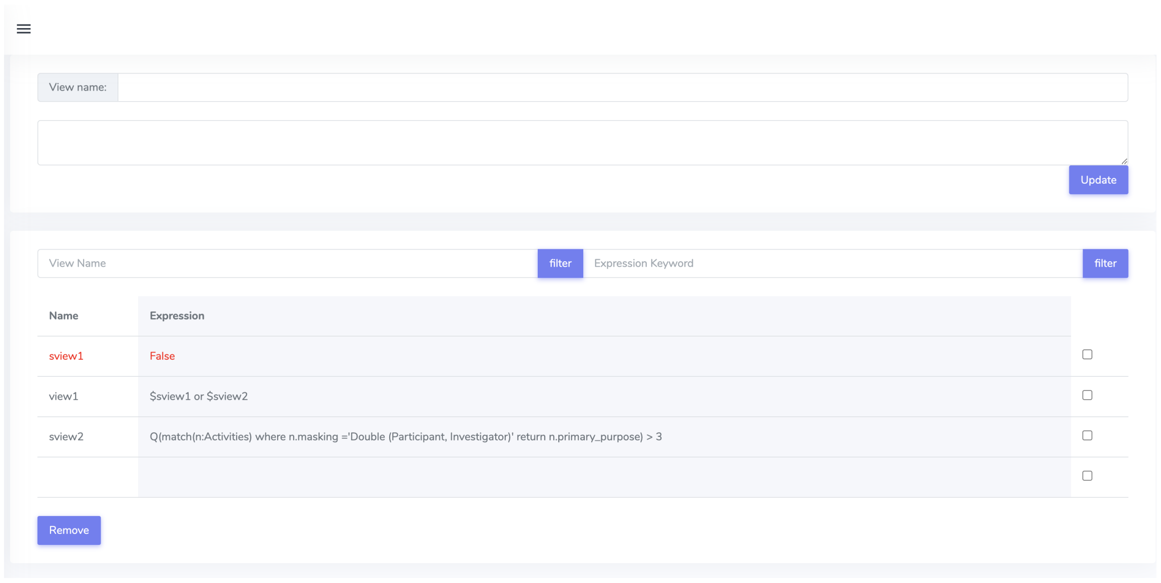
The process of passing the request from the front end to the back end is no different from the previous description. The processing method of the backend database api after receiving the request is different from that of node attribute update. The backend uses the node id in the received table as an index to find nodes in the database through the MATCH clause. If the node does not exist, it will directly send a prompt message of failure to update the relationship (or superclasses) to the front end. And it prompts that the class does not exist on the footer of the current information page. When updating a node's superclasses, we directly create an isA relationship between two nodes. When updating the node relationship, we create the relationship between nodes according to the type column in the table.

```
  Input: id, relationship_table
1 query = " MATCH(n) WHERE id(n) = id"
2 for for neighbor_id, type in relationship_table: do
3     query += "MATCH(m) WHERE id(m) =" +neighbor_id
4     query += "CREATE(n)-[:"+type+"]→ (m)"
5     session.run(query)
6 end
```

The difference from the previous algorithm idea is that when updating data, the entire table data is written into a cypher statement before execution. The relationship creation needs to run one by one cypher statement until the entire table is traversed.

5.2.3 View

The views in the system are also stored in the neo4j graph database. Therefore, the implementation of the view function will also call the database API. The difference is that in order to parse the view expression, the boolean parser in the system is also required.



Name	Expression	
sview1	False	☐
view1	$sview1 or $sview2	☐
sview2	Q(match(n:Activities) where n.masking ='Double (Participant, Investigator)' return n.primary_purpose) > 3	☐
		☐

Figure 5.6: View Dashboard

The interface of the view part is mainly divided into two parts. The upper part contains two text input fields for entering the name and expression of the view. There is also an update button to update the entered information to the database. If the input view name already exists, update the view information. If the entered view name does not exist, create a new view. If the input is an expression of a complex view, and the expression contains a variable (simple view) that does not exist in the database, then a corresponding prompt message will appear in the footer of this part, and the update fails.

The bottom half of the page shows all views and their expressions in the current database. If the view's expression evaluates to False, it will be displayed in red. The lower part of the page also provides users with filter functions, including two text input fields and two buttons. Allows users to filter based on the name of the view and the keywords contained in the expression.

The function of filtering by view name and view expression keyword is implemented through the following steps:

1. Store the keyword to be used for filtering in a variable.
2. Traverse the entire table, if the view name (or expression) contains the keyword we stored, keep it, otherwise remove it from the table.

The filtering function is implemented on the front end and does not communicate with the software system.

There is a checkbox at the far right of each view in the table, and a remove button at the bottom of the table. After pressing the remove button, the view selected by the checkbox will be deleted from the page and database.

When the view dashbaord is loaded, the software part will run as follows:

1. The user interface makes a request to the backend server
2. The backend server forwards the corresponding request to the database API

3. The neo4j driver creates a session and sends a request to the database to obtain all view information, including the name and expression of the view

4. Evaluate simple view expressions

5. Call the boolean parser in the API to get the evaluation result of expression for each complex view

6. Send the information of views and the result of expression evaluate to the user interface

Whenever views in the database are updated, it may cause the evaluate value of expressions in other views to be changed. Therefore, whenever the view update is successful, the view table will be reloaded once to get the latest results.

The view is also stored in the neo4j database in the form of nodes in the system. Whether it is a complex view or a simple view, it will have the same label "view". There are two attributes in the node's attributes, one is name, which is used to store the name of the view, and the other is the expression used to store the view by expression. The method of modifying the properties of the view nodes in the software system is not different from the method of modifying the properties of the nodes described above. It is processed by constructing a cypher statement used in conjunction with MATCH and SET. In this part we mainly introduce how to implement the evaluation of view.

The expression of the view returned from the database is in the form of string. In the previous section we introduced that the expression of a simple view consists of three parts.

$$Q(query)\ + operation\ symbol +\ \{result\ set\}$$

The operation symbol includes$>, <, =, \leq\ and \geq$

For the query part of the expression, we directly use the cypher query statement to implement it. Therefore, when evaluating the expression of a simple view, we first extract the query part of the expression. To extract the query part, we can use the regular expression (.+?) in python to extract the expression to match. Combined with python's findall function, for example expression = "Q(query) > {result set}"

```
Output: string between "(" and ")"
import re
expression = "Q(query) > {result set}"
result = re.findall(r"((.+?))",expression)
```

In the same way, we can construct a regular expression to extract the operation symbol and result set. After extracting the query part, we hand it over to the driver for execution, and store the returned result set in the **set** data type of python. Also store the extracted result set in the **set** data type. The set data type in python supports direct comparison with operation symbols ($>, <, =, \leq\ and \geq$) and returns True or False. In this way, our simple view completes the evaluation.

After completing the evaluation of all simple views, we can evaluate complex views. The expression of the complex view contains the name of the simple view as a parameter, and the entire expression is a Boolean expression.

The evaluation of complex views can be divided into the following steps

- Use regular expressions to extract the parameters in the expression (the name of the simple view)
- According to the evaluated result of the simple view, assign values to these extracted parameters.
- Evaluate boolean expressions with the eval() function in python.

Chapter 6

Evaluation of the system

An essential component in assessing a system's performance is its response time. A system's response time is the amount of time it takes to react to a user's request. It is a crucial element in assessing a system's quality and directly affects user satisfaction.We imported the part of the data provided by the clinicaltrials website into the system as a test.

Our front-end and back-end communicate through javascript post/get. In order to test the response time, get the current time before sending the request, get the current time in the function that gets the response after successful sending, and calculate the response time by subtraction. Each result in the table is the average of ten tests.

```
var sendDate = (new Date()).getTime();
$.get({
success: function(){
var receiveDate = (new Date()).getTime();
var responseTimeMs = receiveDate - sendDate;
}
});
```

The three functions in the table below are tested based on the secondary_outcome schema node. It has the most instances in the database at 1613

action	response time(ms)
query	120
reload node detail page	140

Next, we will explain how each function is tested and how to obtain the results:

- Query: We have performed query tests on all label type nodes in the database, and took the longest response time as the result. When querying the node whose label is secondary_outcome, the returned node id is the most, and the response time is the longest.
- For the test of reload node detail page, we still choose the schema node of secondary_outcome as the test node. Because this node has the most instances, the amount of data transmission between the front and back ends is the largest

We tested the three functions of update(and remove) attribute, relationship and superclasses of the secondary_outcome schema node. Since their working mechanism is the same, the response time of the test is also exactly the same. For columns named 1, 5, 10 means, the response time for processing update (or delete) 1, 5, and 10 rows of data.

Action	1	5	10
update node detail	32 ms	70 ms	94 ms
remove node detail	32 ms	68 ms	90 ms

The response time of reload view dashboard and remove views depends on the number of views in the database. The columns named 1 view, 5 views, and 10 views mean the response time when the database contains 1 view, 5 views, and 10 views. Our test results are as follows:

Action	1 view	5 views	10 views
reload view dashboard	37 ms	90 ms	112 ms
remove views	30 ms	74 ms	96 ms

The test of view update does not depend on the number of view nodes. Each view update action can only operate on one node. We tested the average response time for 10 times. The average response time is 32ms.

Users seldom contribute more than 10 pieces of data using the system's user interface under normal conditions; the quantity of data typically varies from 1 to 5 to 10, and the system's reaction time does not significantly increase. Thus, the system's functionality meets user demands.

Chapter 7

Application

7.1 System Configuration

The following environment needs to be completed in advance to use the system:

- Install the neo4j graph database, version 4.2.1.
- Install the python operating environment, the python version is 3.9.
- Install the python flask package

The system contains a file named configuration. It includes the following information.

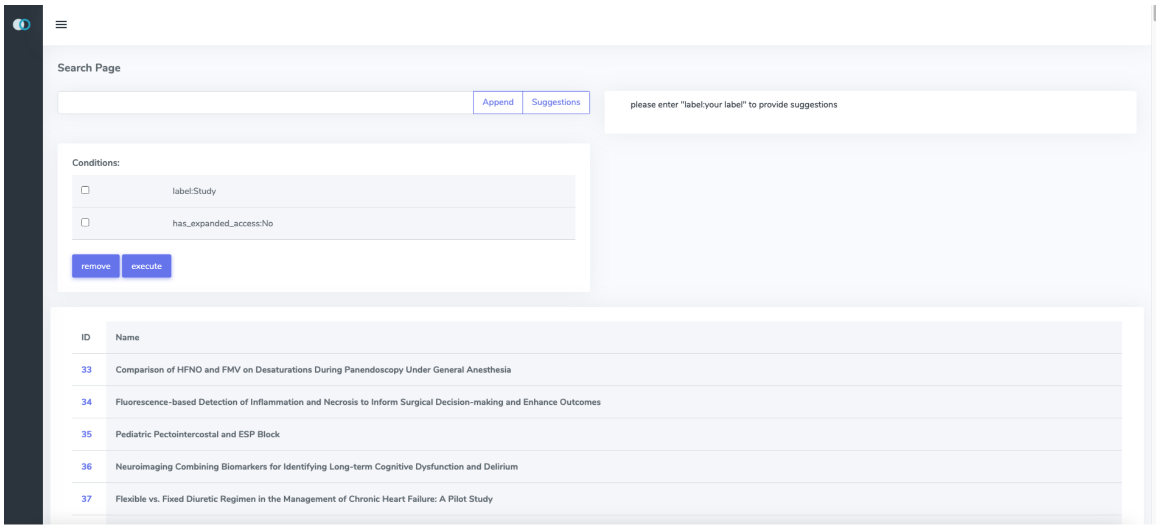
- flask server: http://localhost:2000
- neo4j server: bolt://localhost:7687
- ner4j user name: neo4j
- neo4j password: 123

The above environment configuration is the default configuration of the system. Users can modify the environment configuration of the system by changing the content of the configuration file. The user needs to restart the system after modifying the configuration file.

7.2 Instruction of using the system

In this part we will introduce the usage process of each function

7.2.1 Query



please enter "label:your label" to provide suggestions

☐	label:Study
☐	has_expanded_access:No

ID	Name
33	Comparison of HFNO and FMV on Desaturations During Panendoscopy Under General Anesthesia
34	Fluorescence-based Detection of Inflammation and Necrosis to Inform Surgical Decision-making and Enhance Outcomes
35	Pediatric Pectointercostal and ESP Block
36	Neuroimaging Combining Biomarkers for Identifying Long-term Cognitive Dysfunction and Delirium
37	Flexible vs. Fixed Diuretic Regimen in the Management of Chronic Heart Failure: A Pilot Study

Figure 7.1: User Interface for Query Function

Using the query function of the system is divided into the following steps:

- Enter the attribute name and value to be queried in the input box on the upper left of the user interface in the format of attribute:value
- Click append button next to the input box, and add the entered attribute to the table below
- If you want to get the prompt information of the input attribute, you can enter label: label_name in the input box, press the suggestion button next to the input box, and all the attribute information that can be queried by this type of node will appear in the information box on the right
- The Condition table contains all the attribute information entered by the user. There is a checkbox on the right side of each attribute. The user can select the checkbox of the undesired attribute and press the remove button below to remove it in the table delete
- Press the execute button below the table, all the attributes in the condition table will be sent to the backend for query
- The queried node information is displayed in the form of node id and node name attributes in the bottom table. If the queried node has no name, it will be displayed as null. Each node ID in the table is a link, clicking on the ID will open the information interface of the node

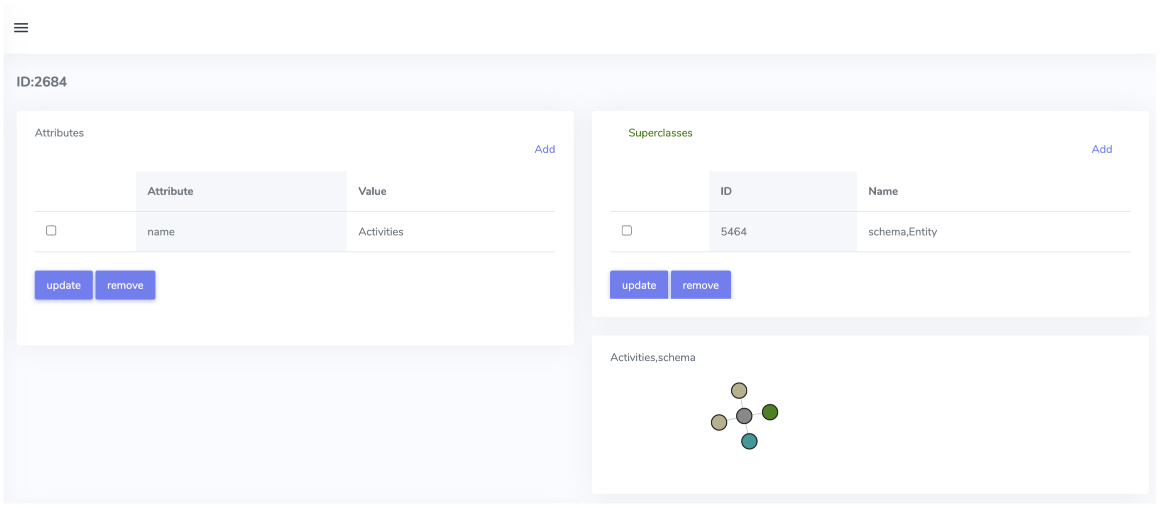
7.2.2 Node and relationship modification

This user interface mainly displays the attribute and relationship information of a node in the database. It has the following functions:

- Add, modify and delete the properties of node

- Add, modify and delete the outgoing relationships of the node

- Add, modify and delete the superclasses of the node



ID:2684

Attributes

	Attribute	Value
☐	name	Activities

Superclasses

	ID	Name
☐	5464	schema,Entity

Activities,schema

Figure 7.2: Node detail 1

The top of the page shows the id of the current node, and below the id is the attribute table of the node. The right side of the attribute table is the superclasses table of the current node. There is a graph card under the superclasses table. This figure takes the current node as the core, showing the relationship nodes of the current node, the superclass and subclass of the current node.

- Gray nodes: The node displayed on the current page

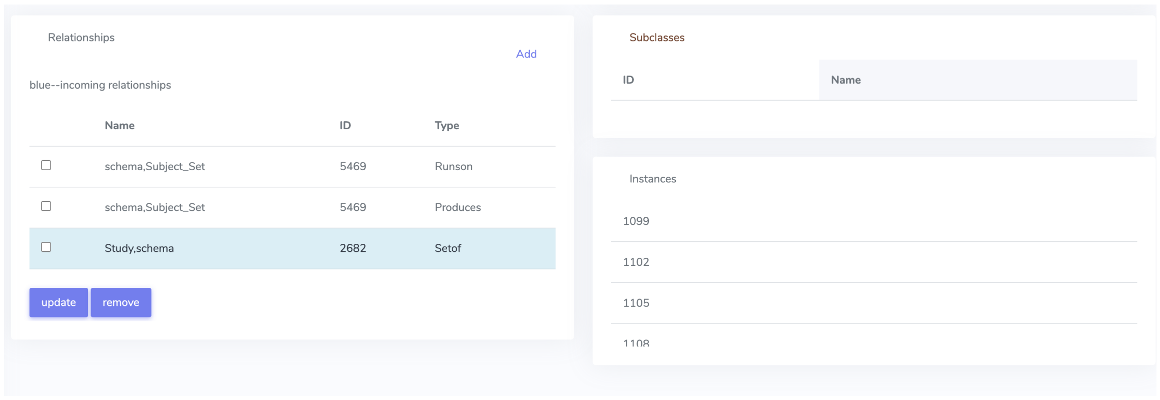
- Green node: The superclasses of the current node

- Brown node: The subclasses of the current node

- Blue node: The node whose relationship direction is inward with the current node

- Beige node: The node whose relationship direction is outward with the current node

Each node in the figure is a link. After clicking, the current page will be redirected to the information page of the clicked node.



Relationships

blue--incoming relationships

	Name	ID	Type
☐	schema,Subject_Set	5469	Runson
☐	schema,Subject_Set	5469	Produces
☐	Study,schema	2682	Setof

Subclasses

ID	Name

Instances

1099

1102

1105

1108

Figure 7.3: Node detail 2

The lower part of the page contains a table of node relationship information, a table of node subclasses information and node Instances table.

Node properties, superclasses and relationship information have the same function button design. The steps to delete the attributes of nodes, superclasses and relationship information are the same: There is a checkbox on the left side of each row in the table. After selecting the checkbox of the row you want to delete, press the remove button below to delete the attributes (or relationships, superclasses) of the selected node in the database.

The steps to modify and add node attributes, relationships and superclasses are roughly the same, but the details are different. To add a new attribute(relationship or superclasses), the user need to press the Add button on the upper right of the table. After pressing, a blank row will be added to the list, and the user can enter the information to be added. To modify the information displayed in the table, the user can directly click on the part to be modified in the table to modify. After modifying or adding information, the user needs to press the update button below to pass the updated information to the backend.

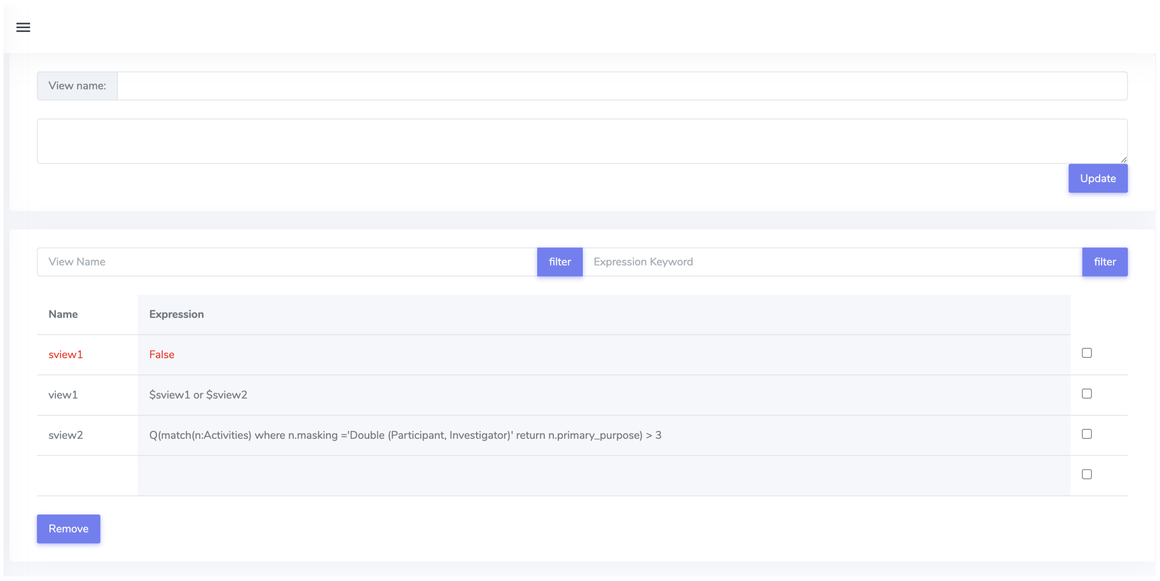
Some of the different details are:

- The attribute names already displayed in the table cannot be modified, only the value of the attribute can be modified

- The superclasses table can only be operated on when the current node is a schema node. Instances node cannot operate on superclass

- If the id entered in the relationships table and superclass table does not exist in the database, the update failure will be displayed after the user presses the update button

7.2.3 View



Name	Expression	
sview1	False	☐
view1	$sview1 or $sview2	☐
sview2	Q(match(n:Activities) where n.masking ='Double (Participant, Investigator)' return n.primary_purpose) > 3	☐
		☐

Figure 7.4: View Dashboard Interface

The user interface of view can provide users with the functions of adding, modifying and deleting views.

The view part's interface can be split into two sections. Two text input areas for the view's name and expression may be seen in the upper section. The user can enter the view name and expression and press the update button to update the view information. Update the view data if the input view name already exists. Create a new view if the view name you specified does not already exist. A matching prompt message will show in the footer of this section and the update will fail if the input is an expression of a complex view that contains a variable (simple view) that is not present in the database.

All views and their expressions in the current database are displayed in the page's bottom half. The expression of the view will be highlighted in red if it evaluates to False. Users can filter by view name and view expression. There are two input boxes and two buttons on the top of the view dashboard table, which are used to filter by view name and view expression respectively. The user enters the keyword to be filtered and presses the corresponding button. The table will display views that contain the keyword in the view name (or expression).

Each view in the table has a checkbox located right of it, and there is a delete button at the bottom. The checkbox-selected view will be erased from the page and database after pressing the remove button.

Chapter 8

Conclusions and Future Work

We provide an evidence base for making decisions assistance in pharmaceutical sciences in this thesis. Initially, ideas like process models, process inquiries, and decision support systems are investigated. We then define the data model that our system needs and the functionality that the system must implement.

We created and implemented the data model in the Neo4j graph database by fusing the idea of the process model with the data utilized in pharmacological clinical trials. All information stored in the data model can be used as evidence for decision support for users. The decision support system we design and implement includes a software system and a user interface in addition to the data model and database. The system can provide the query function based on node attributes, the visualization function of node information, and modification of node relationship and attributes. The system also provides users with a mechanism to use the view of the database as a decision-making model. At the same time, the decision model composed of this view can be evaluated.

We use response time as an indicator. From the time when the user makes an action on the user interface, until the entire system completes the action and transmits the data from the back end to the front end, a simple test is carried out on the various functions of the system. The test results prove that the system can meet the user's needs.

8.1 Future Work

This system can continue to be improved in the future. Our data model is only based on clinical trials of drugs, but making decisions about the drug development process will involve more and more responsible experimental information. Therefore the data model can be extended in the future.

The current system only supports users to query nodes through the combination of attributes. In the future, the system can provide the similarity query between the name and value of the attribute when querying. If the data model is extended, due to many data sources and different data source structures, there may be a problem that the attribute names of different nodes are different but very similar, and these attributes describe related information, but they will not be queried. Querying based on attribute similarity may be a solution

Bibliography

[1] Wil van der Aalst. "Process Modeling and Analysis". In: *Process Mining: Data Science in Action*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 55–88. ISBN: 978-3-662-49851-4. DOI: 10.1007/978-3-662-49851-4_3. URL: https://doi.org/10.1007/978-3-662-49851-4_3.

[2] Wil M. P. van der Aalst. "Conformance Checking". In: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 191–213. ISBN: 978-3-642-19345-3. DOI: 10.1007/978-3-642-19345-3_7. URL: https://doi.org/10.1007/978-3-642-19345-3_7.

[3] James F Allen. "Maintaining knowledge about temporal intervals". In: *Communications of the ACM* 26.11 (1983), pp. 832–843.

[4] Renzo Angles and Claudio Gutierrez. "Survey of graph database models". In: *ACM Computing Surveys (CSUR)* 40.1 (2008), pp. 1–39.

[5] Ahmed Awad. "BPMN-Q: A language to query business processes". In: *Enterprise modelling and information systems architectures–concepts and applications* (2007).

[6] Ahmed Awad and Sherif Sakr. "Querying graph-based repositories of business process models". In: *International Conference on Database Systems for Advanced Applications*. Springer. 2010, pp. 33–44.

[7] David W Bates and Atul A Gawande. "Improving safety with information technology". In: *New England journal of medicine* 348.25 (2003), pp. 2526–2534.

[8] Louis Bavoil et al. "Vistrails: Enabling interactive multiple-view visualizations". In: *VIS 05. IEEE Visualization, 2005.* IEEE. 2005, pp. 135–142.

[9] Stacy Calloway, Hameed A Akilo, and Kyle Bierman. "Impact of a clinical decision support system on pharmacy clinical interventions, documentation efforts, and costs". In: *Hospital pharmacy* 48.9 (2013), pp. 744–752.

[10] Josep Carmona, Boudewijn van Dongen, and Matthias Weidlich. "Conformance Checking: Foundations, Milestones and Challenges". In: *Process Mining Handbook*. Ed. by Wil M. P. van der Aalst and Josep Carmona. Cham: Springer International Publishing, 2022, pp. 155–190. ISBN: 978-3-031-08848-3. DOI: 10.1007/978-3-031-08848-3_5. URL: https://doi.org/10.1007/978-3-031-08848-3_5.

[11] Josep Carmona et al. "Conformance checking". In: *Switzerland: Springer.[Google Scholar]* (2018).

[12] Enrico Coiera. "Clinical communication: a new informatics paradigm." In: *Proceedings of the AMIA Annual Fall Symposium.* American Medical Informatics Association. 1996, p. 17.

[13] Elizabeth Daniel, Hugh Wilson, and Malcolm McDonald. "Towards a map of marketing information systems: an inductive study". In: *European Journal of Marketing* (2003).

[14] Sean B Eom et al. "A survey of decision support system applications (1988–1994)". In: *Journal of the Operational Research Society* 49.2 (1998), pp. 109–120.

[15] Nadime Francis et al. "Cypher: An evolving query language for property graphs". In: *Proceedings of the 2018 International Conference on Management of Data.* 2018, pp. 1433–1445.

[16] Juliana Freire et al. "Managing rapidly-evolving scientific workflows". In: *International Provenance and Annotation Workshop.* Springer. 2006, pp. 10–18.

[17] Eduardo González López de Murillas, Hajo A Reijers, and Wil MP Van Der Aalst. "Connecting databases with process mining: a meta model and toolset". In: *Software & Systems Modeling* 18.2 (2019), pp. 1209–1247.

[18] Mike Hart. "Systems for supporting marketing decisions". In: *Handbook on Decision Support Systems 2.* Springer, 2008, pp. 395–417.

[19] Juris Hartmanis. "Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)". In: *Siam Review* 24.1 (1982), p. 90.

[20] Johan Hastad. "Clique is hard to approximate within n/sup 1-/spl epsiv". In: *Proceedings of 37th Conference on Foundations of Computer Science.* IEEE. 1996, pp. 627–636.

[21] Pieter J Helmons et al. "Drug-drug interaction checking assisted by clinical decision support: a return on investment analysis". In: *Journal of the American Medical Informatics Association* 22.4 (2015), pp. 764–772.

[22] Judith Hurwitz et al. *Big data for dummies.* Vol. 336. John Wiley & Sons Hoboken, NJ, 2013.

[23] Anna A Kalenkova et al. "Process mining using BPMN: relating event logs and process models". In: *Software & Systems Modeling* 16.4 (2017), pp. 1019–1048.

[24] Rajiv Kohli and Frank Piontek. "DSS in healthcare: Advances and opportunities". In: *Handbook on Decision Support Systems 2* (2008), pp. 483–497.

[25] Sander JJ Leemans, Dirk Fahland, and Wil MP Van Der Aalst. "Discovering block-structured process models from event logs-a constructive approach". In: *International conference on applications and theory of Petri nets and concurrency.* Springer. 2013, pp. 311–329.

[26] Massimiliano de Leoni. "Foundations of Process Enhancement". In: *Process Mining Handbook.* Ed. by Wil M. P. van der Aalst and Josep Carmona. Cham: Springer International Publishing, 2022, pp. 243–273. ISBN: 978-3-031-08848-3. DOI: 10.1007/978-3-031-08848-3_8. URL: https://doi.org/10.1007/978-3-031-08848-3_8.

[27] Charles D Mahoney et al. "Effects of an integrated clinical information system on medication safety in a multi-hospital setting". In: *American Journal of Health-System Pharmacy* 64.18 (2007), pp. 1969–1977.

[28] Ana Karla A de Medeiros, Anton JMM Weijters, and Wil MP van der Aalst. "Genetic process mining: an experimental evaluation". In: *Data mining and knowledge discovery* 14.2 (2007), pp. 245–304.

[29] Business Process Model. “Notation (BPMN) version 2.0”. In: *OMG Specification, Object Management Group* (2011), pp. 22–31.

[30] Tadao Murata. “Petri nets: Properties, analysis and applications”. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580.

[31] Eduardo Gonzalez Lopez de Murillas, Hajo A Reijers, and Wil MP van der Aalst. “Data-Aware Process Oriented Query Language”. In: *Process Querying Methods*. Springer, 2022, pp. 49–83.

[32] Jaroslav Pokorný. “Graph Databases: Their Power and Limitations”. In: *Computer Information Systems and Industrial Management*. Ed. by Khalid Saeed and Wladyslaw Homenda. Cham: Springer International Publishing, 2015, pp. 58–69. ISBN: 978-3-319-24369-6.

[33] Daniel J. Power. “Decision Support Systems: A Historical Overview”. In: *Handbook on Decision Support Systems 1: Basic Themes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 121–140. ISBN: 978-3-540-48713-5. DOI: `10.1007/978-3-540-48713-5_7`. URL: `https://doi.org/10.1007/978-3-540-48713-5_7`.

[34] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. ” O’Reilly Media, Inc.”, 2015.

[35] Andrew P Sage. *Decision support systems engineering*. Wiley-Interscience, 1991.

[36] Sherif Sakr. “Storing and querying graph data using efficient relational processing techniques”. In: *International United Information Systems Conference*. Springer. 2009, pp. 379–392.

[37] Pedro A. C. Sousa, João Paulo Pimentão, and Rita Almeida Ribeiro. “Intelligent decision support tool for priorotizing equipement repairs in critical/disaster situations”. In: 2006.

[38] Ralph H Sprague Jr. “A framework for the development of decision support systems”. In: *MIS quarterly* (1980), pp. 1–26.

[39] Ahmad Tariq and Khan Rafi. “Intelligent decision support systems-A framework”. In: *Information and Knowledge Management*. Vol. 2. 6. Citeseer. 2012, pp. 12–20.

[40] Arthur HM Ter Hofstede et al. “APQL: A process-model query language”. In: *Asia-Pacific Conference on Business Process Management*. Springer. 2013, pp. 23–38.

[41] Wil Van Der Aalst. *Process mining: data science in action*. Vol. 2. Springer, 2016.

[42] Wil Van Der Aalst. “Process mining: Overview and opportunities”. In: *ACM Transactions on Management Information Systems (TMIS)* 3.2 (2012), pp. 1–17.

[43] Wil MP Van Der Aalst. *A practitioner’s guide to process mining: limitations of the directly-follows graph*. 2019.

[44] Thomas Vogelgesang et al. “Celonis PQL: A query language for process mining”. In: *Process Querying Methods*. Springer, 2022, pp. 377–408.

[45] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. “Process mining with the heuristics miner-algorithm”. In: *Technische Universiteit Eindhoven, Tech. Rep. WP* 166.July 2017 (2006), pp. 1–34.

[46] Robert S Wigton. “Use of linear models to analyze physicians’ decisions”. In: *Medical decision making* 8.4 (1988), pp. 241–252.