Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Life Sciences

Bachelorarbeit

Konzeption und Umsetzung einer webbasierten Oberfläche zur retrospektiven Identifikation von akuten pathologischen Patientenzuständen in der perioperativen und intensivmedizinischen Versorgung

Vorgelegt von

Ngoc Quynh Nhu Nguyen

[redacted]

[redacted]

Matrikelnummer: [redacted]

Studiengang: Medizintechnik

Betreuerin: Frau Prof. Dr. Petra Margaritoff

Klinischer Betreuer: Herr Dr. rer. medic. Dipl.-Ing. Robert Huhle

Dresden, 14. Juni 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AF	Atemfrequenz in min^{-1}
ANE	Klinik und Poliklinik für Anästhesiologie und Intensivtherapie
Arb.unit	arbitrary unit
bpm	beats per minute (Einheit der Herzfrequenz)
DB	Database (Datenbank)
DBP	Diastolic blood pressure (der diastolische Blutdruck)
HF	Herzfrequenz, 5-Minuten gleitender Mittelwert
instHF	Herzfrequenz, Schlag zu Schlag
KNIME	Konstanz Information Miner
MAP	mean arterial pressure: Der mittlere arterielle Druck
mmHg	Millimeter-Quecksilbersäule (Druckeinheit) (1mmHg= 133,332 Pa)
OCLIDA	Online Clinical Database Access
SQL	Standard Query Language
UKD	Universitätsklinikum Carl Gustav Carus, TU Dresden

Kurzfassung

In Intensivstationen und Operationssälen werden zur Überwachung des Gesundheitszustandes der Patienten eine Vielzahl von Signalen und Patienten-Parametern (z.B. Herzfrequenz, EKG) kontinuierlich und rund um die Uhr gemessen und bestimmt. Die Gesamtheit dieser Daten aller Patienten der Anästhesie werden in einer Datenbank gespeichert. Zur retrospektiven Untersuchung der Entstehung kritischer Patientenzustände, in Forschung und Qualitätsanalyse wird eine Möglichkeit benötigt, die gemessenen Signale und Parameter nach verschiedenen, selbst definierten Bedingungen zu filtern. Zur Lösung dieses Problems wird in dieser Arbeit die Erweiterung einer webbasierten grafischen Benutzeroberfläche für den Zugriff auf diese Datenbank beschrieben. Diese ermöglicht es Patienten-datensätze nach definierten pathologischen Parameterverläufen zu identifizieren. Die Oberfläche erlaubt die intuitive Kombination logischer Bedingungen, sowie die Definition von Vergleichswerten und Zeitintervallen durch den Anwender.

Die Erweiterung umfasst sowohl die grafische Benutzeroberfläche als auch die Anwendungslogik zur automatischen Umsetzung definierter Filterbedingungen in Datenbankabfragen, die Ausführung der so generierten Datenbankabfragen und die Darstellung der erhaltenen Ergebnisse. Der entwickelten Lösung liegt ein Anforderungskatalog zugrunde der in Abstimmung mit dem medizinischen Fachpersonal des Universitätsklinikums erarbeitet wurde. Die Lösung stellt eine webbasierte Benutzeroberfläche zur intuitiven Definition von Filterbedingungen und deren logische Verknüpfung dar. Der Benutzer kann beliebig viele Bedingungen definieren und das Zeitintervall einstellen, für das die Bedingungen ausgewertet werden sollen. Zurückgegeben wird eine Liste mit Patienten aus der ein Patient für den weiteren Datenzugriff für die weitere Auswertung ausgewählt werden kann.

Zur Optimierung der Ausführungsdauer der generierten Datenbankabfragen wurden zwei verschiedene Varianten umgesetzt. In der ersten Variante („Gesamtanfrage“; „ursprüngliche Variante“) werden die definierten Filterbedingungen in eine SQL-Abfrage übersetzt. In der zweiten Variante („Teilanfrage“; „optimierte Variante“) werden die definierten Filterbedingungen in mehrere SQL-Abfragen übersetzt, deren Ergebnisse am Ende zusammengeführt werden. In dieser Arbeit werden beide Varianten miteinander verglichen und diskutiert. Die dabei identifizierte Lösung mit der geringsten Ausführungsdauer wurde in die finale Version für den klinischen Einsatz implementiert. Diese erlaubt dem medizinischen Fachpersonal die Patientendatenbank nach Patienten mit pathologischen Parameterverläufen geschwindigkeitsoptimiert zu durchsuchen und Filterbedingungen für die Erkennung dieser Parameterverläufe über eine intuitive grafische Benutzeroberfläche zu definieren.

1 Einleitung

Die retrospektive Analyse von Patientendaten ist notwendig in der Forschung sowie der Qualitätssicherung. Die dazu zur Verfügung stehenden Systeme werden vorgestellt und daraus die Motivation und Aufgabenstellung der vorliegenden Arbeit abgeleitet.

1.1 Motivation

Auf der Intensivstation werden Patienten behandelt, die sich in einer kritischen medizinischen Verfassung befinden. Für diese Patienten werden eine Vielzahl von Signalen und Parametern (z.B. Herzfrequenz, EKG) rund um die Uhr gemessen bzw. bestimmt und in einer Datenbank abgespeichert. Aufgrund dieser guten Verfügbarkeit von Daten ist es grundsätzlich möglich, bei Intensivpatienten bestimmte Krankheitsbilder oder medizinische Auffälligkeiten, zu Forschungszwecken, automatisch durch die Auswertung der abgespeicherten Daten zu erkennen.

Für den Zugriff auf diese Datenbank wurde am UKD (Universitätsklinikum Carl Gustav Carus Dresden) in der Pulmonary Engineering Group der Klinik und Poliklinik für Anästhesiologie und Intensivtherapie die webbasierte Schnittstelle OCLIDA (Online Clinical Database Access) entwickelt [1]. Diese Schnittstelle kann über einen Browser bedient werden und erlaubt den gefilterten Abruf und die Visualisierung der Patienten-Monitoring-Daten ("Parameter"). Bislang fehlt die Möglichkeit Krankheiten durch die Analyse dieser Daten zu erkennen und die betreffenden Patienten zu ermitteln. In dieser Arbeit wird OCLIDA um die Funktionalität erweitert, Patienten-Monitoring-Daten, die auf eine Krankheit hinweisen, anhand von durch den Anwender, d.h. den Mediziner, definierten Bedingungen zu erkennen. Anhand von hohem Blutdruck lässt sich z.B. eine Hypertonie ersten Grades erkennen [2].

1.2 Hintergrund

Die Abbildung 1 enthält eine schematische Übersicht über das Patientenüberwachungssystem.

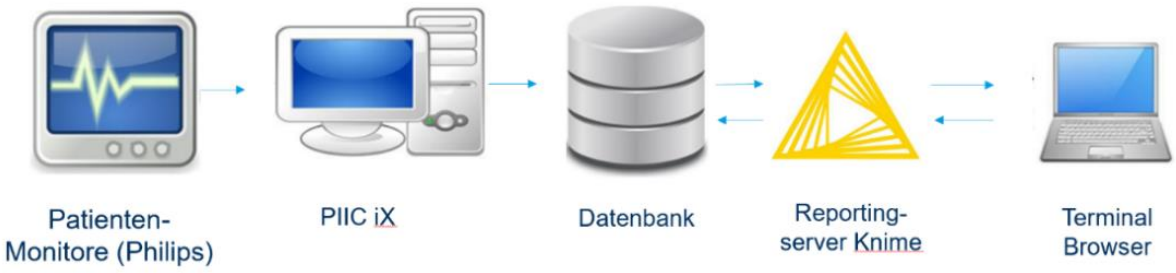


Abbildung 1: Architektur des Patientenüberwachungssystems (überarbeitet nach [1])

Patienten-Monitore: Es handelt sich dabei um Anzeigegeräte, die am Bett der Patienten stehen. Die Patienten tragen verschiedene äußere Sensoren, die jeweils über ein Kabel mit dem Monitor (Überwachungsgerät) verbunden sind. Hier werden laufend Parameter wie Herzfrequenz (EKG), Blutdruck, Körpertemperatur und Sauerstoffsättigung registriert und angezeigt. Die Werte werden in Form von Kurven oder Zahlen auf dem Bildschirm dargestellt und werden an den Stationsrechner weitergeleitet [3].

PIIC iX-Stationsrechner: Ein Stationsrechner akzeptiert Daten von mehreren Patienten-Monitoren. Über das Patientenüberwachungssystem "*IntelliVue Information Center iX*" von Royal Philips (PIIC iX) ist der Stationsrechner an eine Datenbank angebunden. Die zuvor über die Sensoren aufgenommenen Patienten-Parameter werden so an zentraler Stelle gespeichert und stehen damit innerhalb der gesamten Einrichtung für den späteren Abruf zur Verfügung [4].

Datenbank: Die Patienten-Datenbank beinhaltet alle von den Patienten aufgenommenen Monitoring-Daten. Sie ist der zentrale Informationsspeicher innerhalb des Krankenhauses, auf den z.B. jeder Arzt zurückgreifen kann, wenn er den Parameterverlauf eines Patienten ermitteln möchte. Die Patienten-Datenbank wird über das Datenbankmanagementsystem Microsoft SQL-Server betrieben. Sie ist Teil des PIIC iX Patientenüberwachungssystem und trägt den Bezeichner Philips.PatientData.

KNIME-Reportingserver: Für die praktische Arbeit mit einer Datenbank wird eine komfortable Benutzerschnittstelle benötigt. Der KNIME-Reportingserver ("KNIME-Server") hostet eine webbasierte Benutzerschnittstelle, OCLIDA, und stellt diese den Anwendern („Clients") auf Anfrage zur Verfügung.

Terminal Browser*:* Bei dem Terminal Browser handelt es sich um einen gewöhnlichen Webbrowser, der für die Darstellung und Interaktion mit der webbasierten Benutzerschnittstelle, OCLIDA, verwendet werden kann. Der Zugriff auf die Datenbank kann mittels KNIME-Server aus Sicherheitsgründen nur über einen Computer erfolgen, welcher dem lokalen Netzwerk des UKDs angehört.

1.3 Präzisierte Aufgabenstellung

In der Arbeit sollen die folgenden ingenieurtechnischen Fragestellungen untersucht werden:

1. Wie sollte eine grafische Benutzeroberfläche zur Definition von Filterbedingungen gestaltet sein, sodass diese vom medizinischen Fachpersonal ohne mathematisch-technisches Hintergrundwissen genutzt werden kann?
2. Wie sollten SQL-Abfragen für die schnelle Auswertung von logischen Filterbedingungen auf den Daten einer sehr großen medizinischen Datenbank gestaltet sein?

Anforderungen

Der Aufgabenstellung (siehe Anhang) sind folgende Anforderungen an die webbasierte grafische Benutzeroberfläche zu entnehmen:

1. Den Parametern sollte eine passende physikalische Einheit zugeordnet sein, d.h. wenn der Benutzer die Parameter ändert, wird die Einheit ebenfalls automatisch geändert.
2. Der Benutzer sollte Größenvergleiche mit selbstgewählten Schwellwerten definieren können.
3. Der Benutzer sollte die Größenvergleiche zu logischen Bedingungen kombinieren können. Bei der Abwesenheit von Klammern sollte die UND-Verknüpfung Vorrang vor der ODER-Verknüpfung haben.
4. Der Benutzer sollte mit Klammern arbeiten können.
5. Eingabefehler sollen bei der Eingabe erkannt werden.
6. Der Benutzer soll Filterbedingungen per Drag-and-Drop definieren können.
7. Zur Optimierung der Ausführungsdauer der Datenbankabfragen soll die Lösung in zwei Varianten angeboten werden, wobei Variante 1 auf der Ausführung einer einzigen SQL-Query basieren soll und Variante 2 auf der kombinierten Ausführung mehrerer SQL-Queries.
8. Für beide Varianten sollen Kennwerte zur Charakterisierung des Zeit- bzw. Speicherbedarfs ermittelt werden.

2 Materialien und Methoden

In diesem Abschnitt wird das Werkzeug *Knime Analytics Platform* [5] kurz beschrieben, welches für die Entwicklung der webbasierten Datenbankschnittstelle *OCLIDA* benutzt wurde. Weiterhin wird ein technischer Überblick über *OCLIDA* und die für diese Arbeit relevanten Teile des verwendeten Datenbankschemas gegeben. Als verwendete Methode wird der Entwicklungsprozess für die Benutzeroberfläche und die Datenfilterung (Filterung der vorhandenen Datenbasis durch die generierten Datenbankabfragen) vorgestellt und es werden die drei Ansätze für den Datenbankzugriff beschrieben.

2.1 KNIME Analytics Platform

Funktion

KNIME ist eine graphische Programmiersprache unter der GPL-Lizenz („Freie Software“) die sich vor allem an Anwender aus dem Bereich Data Science wendet. Typische Arbeitsvorgänge werden unterstützt: Daten können aus verschiedenen Datenquellen geladen und transformiert werden, Werkzeuge zur Datenanalyse mit Techniken des Data Minings können angewendet werden, für die Interaktion der Anwender mit dem Programm lässt sich leicht eine webbasierte grafische Benutzeroberfläche bereitstellen [5].

Ein KNIME-Programm ist dabei ein Workflow, d.h. ein Netzwerk aus mehreren Knoten. KNIME bietet viele vordefinierte, konfigurierbare Knoten, aber erlaubt auch die Definition eigener Knoten. Die Programmiersprache bietet Module zur Datenvorverarbeitung, Modellierung, der Analyse von Daten und sowie der Visualisierung der Ergebnisse [3]. Workflows können graphisch durch das Verbinden von Knoten entwickelt werden. Jeder Knoten hat eine Funktionalität und wird mit seiner Umgebung über Eingänge bzw. Ausgänge für den Datenempfang und -weiterleitung verbunden. Das Verbinden zweier Knoten erfolgt über das Setzen einer sog. Kante vom Ausgang des einen Knoten zum Eingang des anderen Knoten. Es gibt einen umfangreichen Satz vordefinierter Knoten mit nützlichen Funktionalitäten. So können mit bestimmten Knoten Datenbankverbindungen hergestellt werden, mit anderen Knoten SQL-Abfragen formuliert werden und schließlich mit wieder anderen Knoten die Spaltenbezeichner der abgerufenen Daten umbenannt werden. Weiterhin kann die Knotenfunktion auch in verschiedenen Programmiersprachen umgesetzt werden, z.B.: Java, JavaScript, Python. Daten können graphisch dargestellt werden und als Datei in einem bestimmten Ausgabeformat exportiert werden (z.B. csv-Format, MS Excel-Datei). Mit KNIME wird darüber hinaus eine leichte und schnelle Bereitstellung des definierten Workflows für die Benutzung durch die Anwender möglich. Aus dem Workflow kann automatisch eine Web-Anwendung generiert werden und an den KNIME-Server übertragen werden. Die Anwender können dann über die generierte Web-Oberfläche mit der

Anwendung interagieren. Als KNIME-Client arbeitet ein Browser der die Web-Oberfläche vom KNIME-Server abruft und dem Nutzer anbietet. Über diese Nutzerschnittstelle kann auf die einzelnen Funktionalitäten der einzelnen Knoten zugegriffen werden und vom Server erstellte Dateien können auf den Client heruntergeladen werden.

Die Entwicklungsumgebung KNIME ist in Abbildung 2 mit einem beispielhaften Workflow dargestellt. Das Bild soll einen Eindruck von der Benutzeroberfläche von KNIME vermitteln. Dargestellt ist der Verbindungsaufbau zu einer Datenbank und die Ausführung einer Datenbankabfrage über geeignete KNIME Knoten.

Die Benutzeroberfläche ist in drei Bereiche aufgeteilt (siehe Punkte 1,2 bzw. 3 in Abbildung 2). Unter Punkt 1 ist das Knotenrepositorium (Node Repository) dargestellt. Über das Node Repository lassen sich alle in KNIME verfügbaren Knoten abrufen und in den Workflow einfügen. Der Anwender kann die Knoten per Drag-&-Drop aus dem Knotenrepositorium in den Workflow ziehen.

In der Mitte befindet sich das Bearbeitungsfenster (Nr. 2), in dem der Benutzer verschiedene Knoten zu einem Workflow verketten kann. Hier kann der Benutzer die Knoten konfigurieren und ausführen. Außerdem kann der Benutzer nach der Ausführung eines Knotens die Belegung der Ausgänge bzw. Eingänge dieses Knotens untersuchen. Die Knoten müssen in einer bestimmten Reihenfolge ausgeführt werden, sodass für alle ausgeführten Knoten alle Eingänge belegt sind. Im Folgenden wird der Beispielworkflow in Abbildung 2 näher beschreiben: Der Knoten *MySQL Connector* ist in der Lage eine Verbindung zu einer Datenbank aufzubauen. Er benötigt dazu keine Eingangsdaten, muss aber konfiguriert werden. Über seinen Ausgang wird den nachfolgenden Knoten die Datenbankverbindung zur Verfügung gestellt. Im Beispiel wird die Datenbankverbindung einem *DB Table Selector*-Knoten zur Verfügung gestellt. Über diesen Knoten wird eine Datenbankanfrage definiert. Die Datenbankanfrage wird von diesem Knoten nicht ausgeführt, kann aber über seinen Ausgang zu nachfolgenden Knoten weitergeleitet werden. Im Beispiel wird die Datenbankanfrage an einen *DB Reader*-Knoten weitergeleitet. Dieser Knoten führt die Datenbankanfrage aus und leitet die aus der Datenbank abgerufenen Daten über seinen Ausgang an die nachfolgenden Knoten weiter.

Auf der rechten Seite befindet sich das Fenster zur Knotenbeschreibung (Description, Nr. 3). Sobald ein Knoten angeklickt wird, erscheint in diesem Bereich eine ausführliche Beschreibung des Knotens. Beschrieben wird die Funktionalität des Knotens, seine Konfigurationsmöglichkeiten, sowie seine Schnittstelle, d.h. die am Eingang erwarteten Daten und die über den Ausgang zur Verfügung gestellten Daten.

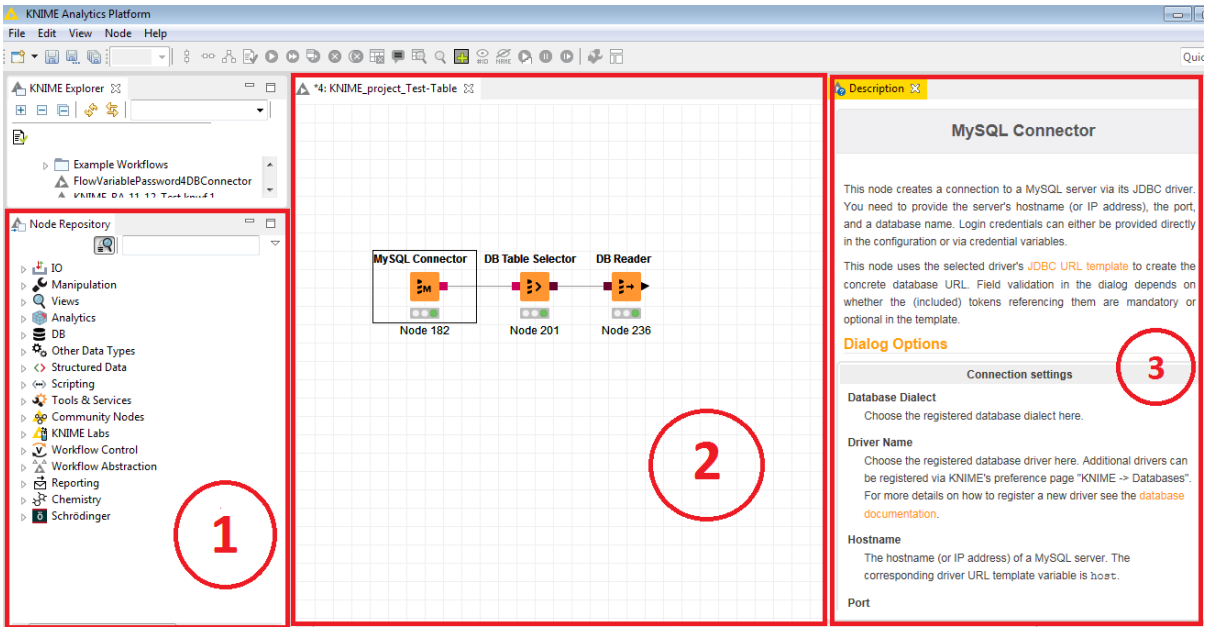


Abbildung 2: Übersicht über die Benutzeroberfläche von KNIME.

Vorteil gegenüber Implementierung mittels Webframeworks (z.B. Spring-Framework, Hibernate-Bibliothek)

- Der definierte Quellcode ist weniger komplex, sodass für die Arbeit mit KNIME geringere Programmierkenntnisse bzw. Programmiererfahrung schon ausreichen
- Verschiedene Programmiersprachen können innerhalb der Plattform verwendet werden, darunter Java, JavaScript, Python, R und Matlab. Diese Programmiersprachen können für die Entwicklung eigener Knoten genutzt werden.
- Die Einbindung dieser Programmiersprachen ist leicht möglich und die anfallenden Programmieraufgaben können leicht in einer geeigneten Programmiersprache gelöst werden. Dabei besteht aber immer eine klare Trennung zwischen den einzelnen Programmiersprachen.
- Der Arbeitsaufwand für die Entwicklung einer Benutzeroberfläche entfällt, da KNIME-Server den entwickelten Workflow automatisch in eine browserbasierten GUI umwandeln kann.

Nachteil gegenüber Implementierung mittels Webframeworks (z.B. Spring-Framework, Hibernate-Bibliothek)

- Die Entwicklung eines Workflows ist mit der kostenlosen Version möglich. Die Bereitstellung eines Workflows als Webanwendung ist allerdings mit hohen Lizenzgebühren verbunden, die für den KNIME-Server anfallen, jedoch für öffentliche Forschungsfragen kostenlos sind.
- KNIME bietet dem Entwickler die Möglichkeit an Knoten in einer von mehreren Programmiersprachen zu implementieren. Möchte man aber eine Bibliothek für diese Programmiersprache (z.B. JavaScript: jQuery, jQuery UI, plotly) verwenden, ist man von der

Unterstützung dieser Bibliothek durch KNIME abhängig.

- Es kann eine hohe Abhängigkeit von Standardknoten entstehen. Da diese in ihrem Verhalten an einen festen Satz von Konfigurationsmöglichkeiten gebunden sind, können kleinere Verhaltensänderungen, die nicht auf die Konfigurationsmöglichkeiten abgebildet werden können, zu einem hohen Implementierungsaufwand führen (Reimplementierung des Knotens, z.B. in einem Java-Knoten). Beispiel: Möchte man bei der Fehlersuche einen Standardknoten um Lognachrichten erweitern, müsste man den Standardknoten in einem programmierbaren Knoten (z.B. in einem Java-Knoten) reimplementieren.

2.2 OCLIDA

Das Programm OCLIDA ermöglicht den webbasierten Abruf, die Filterung, die Präsentation und den Export von Patientendaten aus der zentralen Datenbank des Universitätsklinikums Carl Gustav Carus Dresden (UKD). In dieser Datenbank sind neben bestimmten personenbezogenen Daten der Patienten auch Zeitverläufe biomedizinischer Größen wie Blutdruck und Herzfrequenz gespeichert. Bei OCLIDA handelt es sich technisch gesehen um einen umfangreichen KNIME-Workflow, der in der ANE/UKD in einer Diplomarbeit [1] mit dem Institut für Biomedizinische Technik, TU Dresden entwickelt wurde. Abbildung 3 zeigt die Struktur des OCLIDA-Workflows.

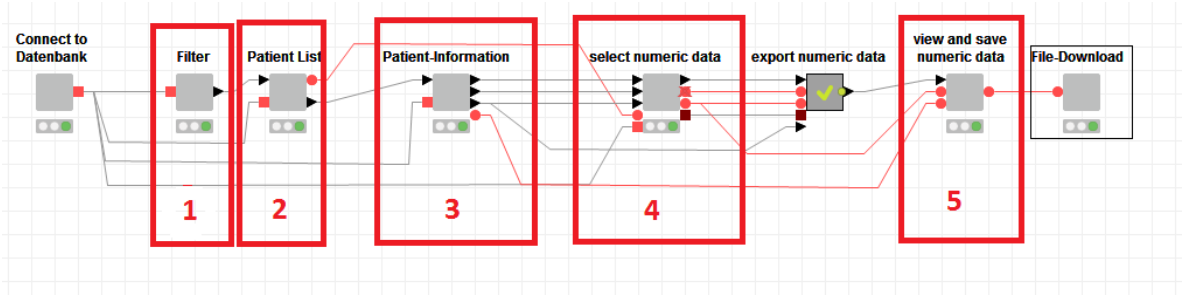


Abbildung 3: Struktur des OCLIDA-Workflows

Das Programm besteht aus 5 Schritten:

1. **Komponente „Filter“:** Diese Komponente implementiert die Filterung der Patienten nach verschiedenen Kriterien (siehe Abbildung 4). Durch den Filter wird eine Liste von Patienten erstellt. Die Patienten können gefiltert werden nach (1) persönlichen Daten, z.B. Vorname, Nachname, Alter, (2) Krankenhausaufenthaltsdatum, (3) Bettnummer und Gebäude. Dabei können mehrere Bettnummern und Gebäude gewählt werden.

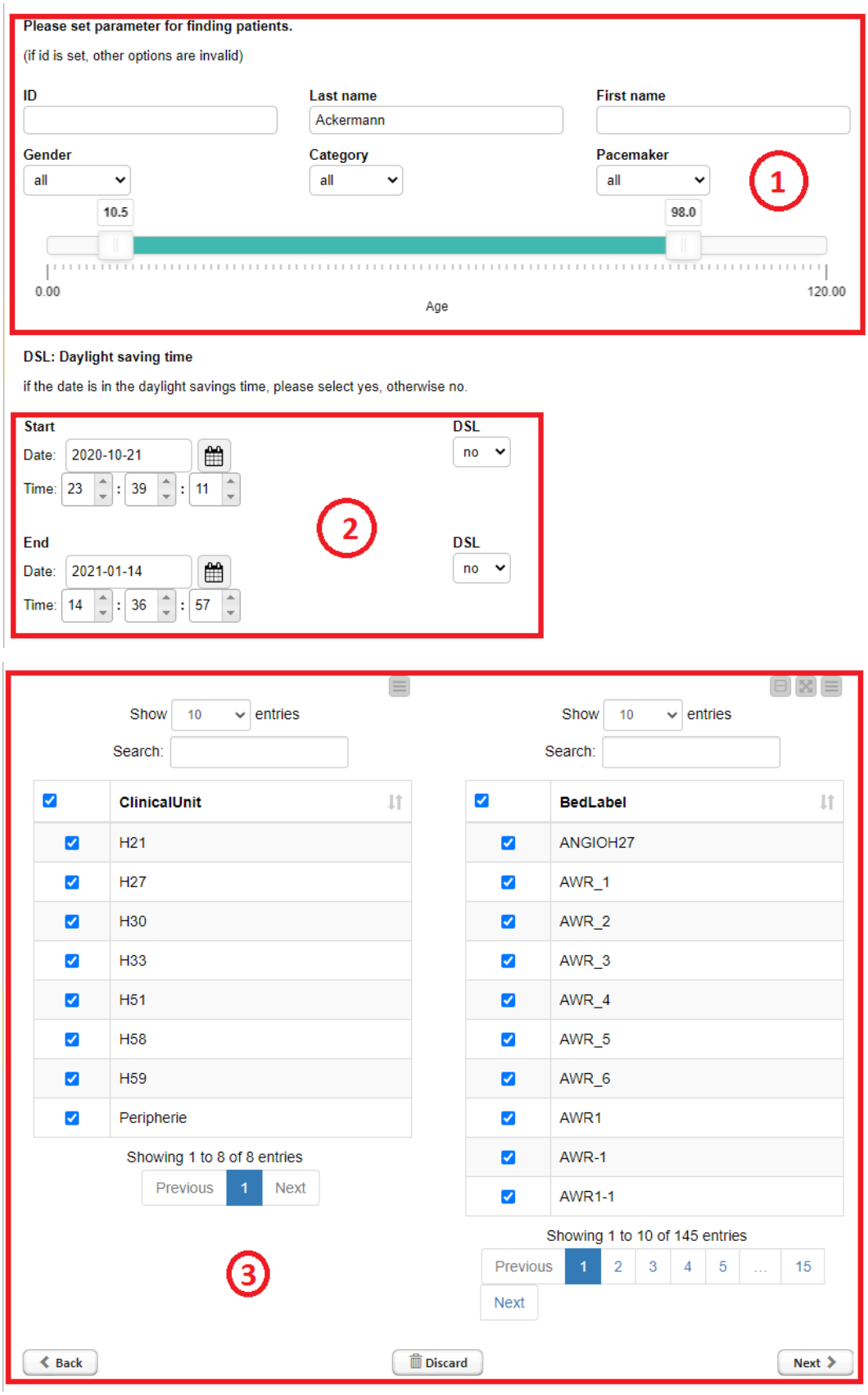


Abbildung 4: Filterung der Patienten nach verschiedenen Kriterien

2. **Komponente „Patient List“:** Über diese Komponente wird dem Benutzer eine Liste mit Patienten angezeigt, deren Daten die in der „Filter“-Komponente definierten Bedingungen erfüllen. Aus dieser Liste kann der Benutzer höchstens einen einzigen Patienten auswählen. Eine ähnliche Anzeige findet sich in der Abbildung 13.

3. **Komponente „Patient-Information“:** Über diese Komponente werden dem Benutzer verschiedene Informationen zum Patienten angezeigt, darunter die persönlichen Daten und bestimmte Aufenthaltsdaten (Bettnummer, Gebäudenummer), sowie eine Liste mit den Namen von den Parametern von denen für diesen Patienten gespeicherten Parameterverläufe vorhanden sind (siehe Abbildung 5).

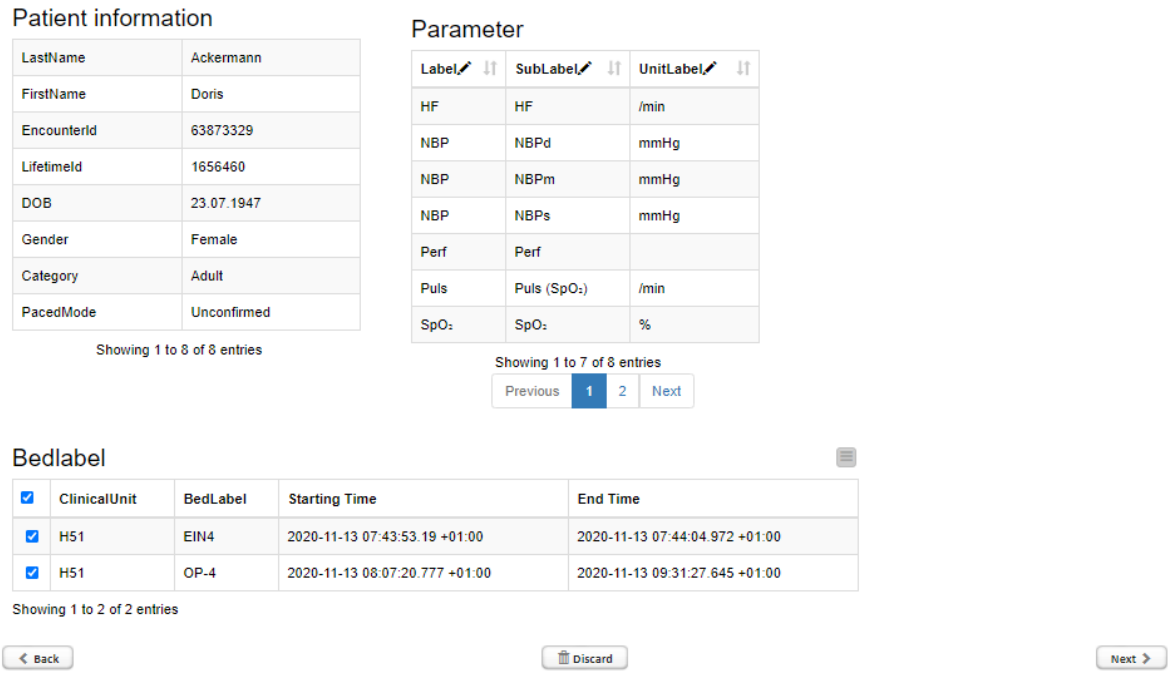


Abbildung 5: Auflistung von Patienteninformationen

4. **Komponente „Select numeric data (select signal)“:** Über diese Komponente kann der Benutzer (1) die darzustellenden Parameterverläufe auswählen, sowie (2) den Zeitraum für den die Parameterverläufe visualisiert werden sollen. Außerdem kann der Benutzer (3) den Zeitabstand zwischen 2 Parameterwerten (z.B. 1s, 3s, …) auswählen (siehe Abbildung 6).

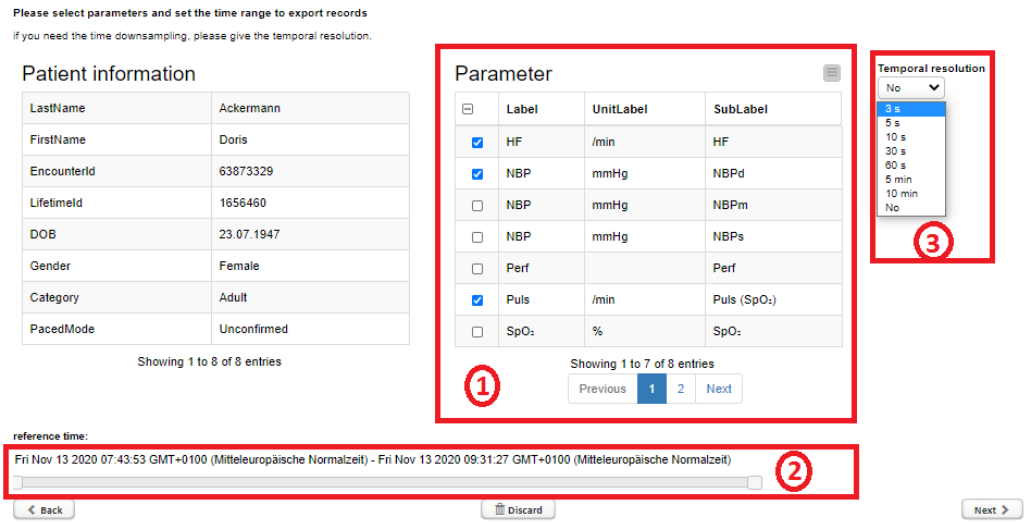


Abbildung 6: Auswahl der interessierenden Parameter und des Zeitbereichs

5. **Komponente „View and save numeric data“:** Über diese Komponente werden dem Benutzer die Parameterverläufe im ausgewählten Zeitraum graphisch dargestellt. Die Darstellung erfolgt über ein vergrößerbares Liniendiagramm (siehe Abbildung 7*)*. Weiterhin können die für den ausgewählten Zeitbereich vorhandenen Daten exportiert, d.h. heruntergeladen werden (csv- bzw. MS-Excel-Format). Die in dieser Komponente implementierte Funktionalität ist im Rahmen einer praktischen Vorarbeit zu dieser Arbeit entstanden.

Abbildung 7: Grafik (in ausgewähltem Zeitbereich)

2.3 Verwendetes Datenbankschema

Abbildung 8 enthält ein Entity-Relationship-Diagramm welches einen Ausschnitt aus dem in der Klinik verwendeten Datenbankschema beschreibt. Die in der Abbildung beschriebene Tabellenstruktur wird in der Arbeit benötigt und ist daher für das Verständnis der Arbeit wichtig.

External_Numeric: Dieser Entitätstyp beschreibt allgemeine Informationen zu einem Parameter, wie zum Beispiel dessen Namen und Einheit. Das Attribut „Id“ beschreibt die Identifikationsnummer des Parameters. Das Attribut „Sublabel“ beschreibt den Parameternamen (z.B.: HF (Herzfrequenz), AF (Atemfrequenz)). Das Attribut „UnitLabel“ beschreibt die Einheit des jeweiligen Parameters.

External_NumericValue: Dieser Entitätstyp beschreibt konkrete numerische Werte eines Parameters für einen Patienten zu einem bestimmten Zeitpunkt. Über das Attribut „NumericId“ wird eine External_Numeric Entität referenziert, also ein Parameter. Damit wird dem konkreten Wert ein Parametername und eine Einheit zugeordnet. Über das Attribut „PatientId“ wird einem numerischen Wert der Patient zugeordnet bei dem dieser Wert gemessen wurde. Das Attribut „TimeStamp“ beschreibt den Zeitpunkt an dem der numerische Wert in die Datenbank geschrieben wurde. Das Attribut „SequenceNumber“ beschreibt eine Sequenznummer die den numerischen Wert eindeutig identifiziert. Jeder Wert erhält (unabhängig vom Parameter) eine eigene Sequenznummer. Sequenznummern werden fortlaufend vergeben.

External_PatientStringAttribute: Dieser Entitätstyp beschreibt Patienteneigenschaften. Über das Attribut „Name“ wird die beschriebene Eigenschaft identifiziert. Mögliche Werte sind Firstname, Lastname, LifetimeId, EncounterId, MiddleName. Das Attribut „Value“ beschreibt den Wert der Eigenschaft. Über das Attribut „PatientId“ wird einer Eigenschaft ein Patient zugeordnet. Da ein Patient mehrere Eigenschaften hat, besitzen mehrere Entitäten dieses Entitätstyps dieselbe Patienten-Id. Das Attribut „TimeStamp“ beschreibt den Zeitpunkt an dem diese Eigenschaft des Patienten das erste Mal in der Datenbank gespeichert wurde.

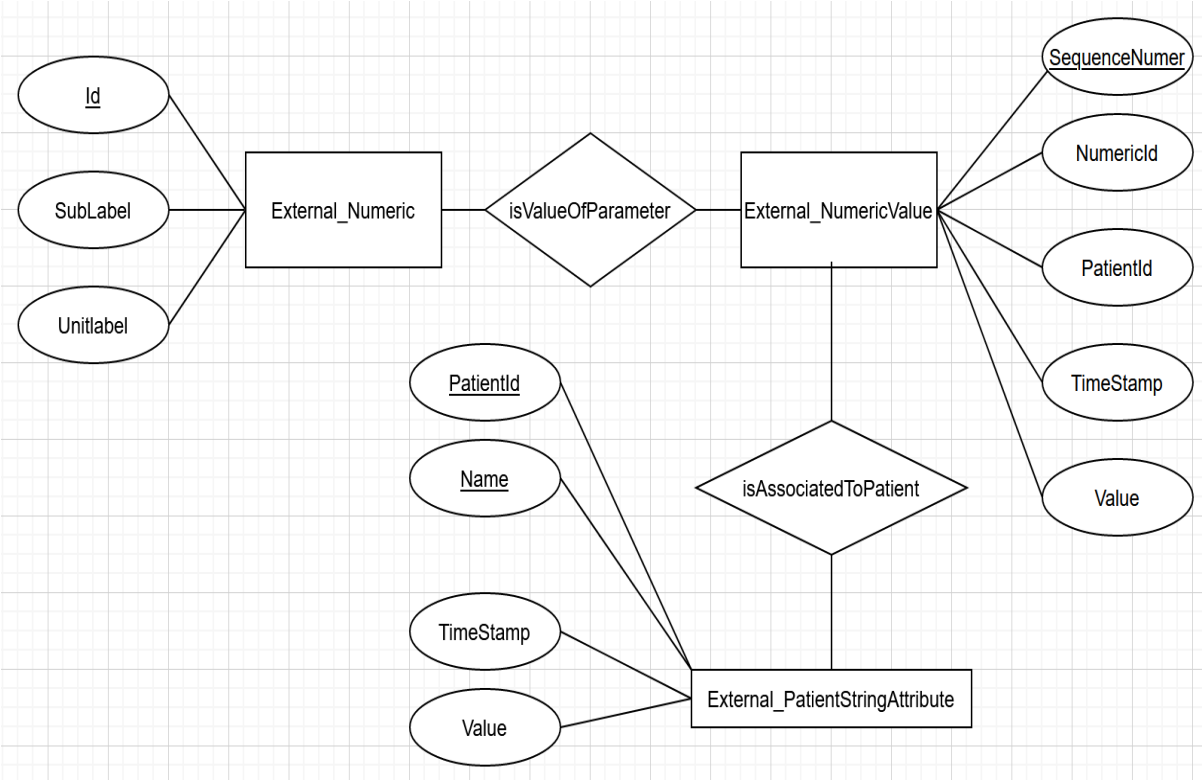


Abbildung 8: ER-Diagramm der für die Arbeit relevanten Teile des Datenbankschemas

2.4 Benutzeroberfläche

Die Benutzeroberfläche wird nach dem Ansatz „Versuch und Irrtum“ in mehreren Varianten entwickelt (siehe Abbildung 9).

Versuch: Dabei wird zunächst, ausgehend vom initialen Anforderungskatalog (Kapitel 1.3), eine Variante der Benutzeroberfläche skizziert bzw. unter Verwendung der im Abschnitt 2 beschriebenen Technologien implementiert (GUI-Prototyp; noch ohne Implementierung des Verhaltens).

Irrtum: Anschließend wird diese Variante mit einem Experten kritisch diskutiert. Die Einschätzung des Experten beruht auf langjähriger Erfahrung im Umgang mit der Nutzergruppe, also nicht-technischen Anwendern, insbesondere Medizinern. Es besteht eine Vertrautheit mit der Gedankenwelt und den Erwartungen der Nutzergruppe, was sich positiv auf die Verständlichkeit und Bedienbarkeit der Benutzeroberfläche auswirkt. Über diese Diskussionen werden Schwachstellen der diskutierten Variante offenbar.

Der Vorgang ist erst beendet, wenn die Diskussion die Eignung der Variante ergibt. Andernfalls wird bei „Versuch“ fortgesetzt und die Variante überarbeitet. So entstehen mehrere Varianten der Benutzeroberfläche, wobei jede Variante eine Weiterentwicklung einer Vorgängervariante darstellt. In dieser Arbeit wird lediglich die letztlich akzeptierte Variante präsentiert.

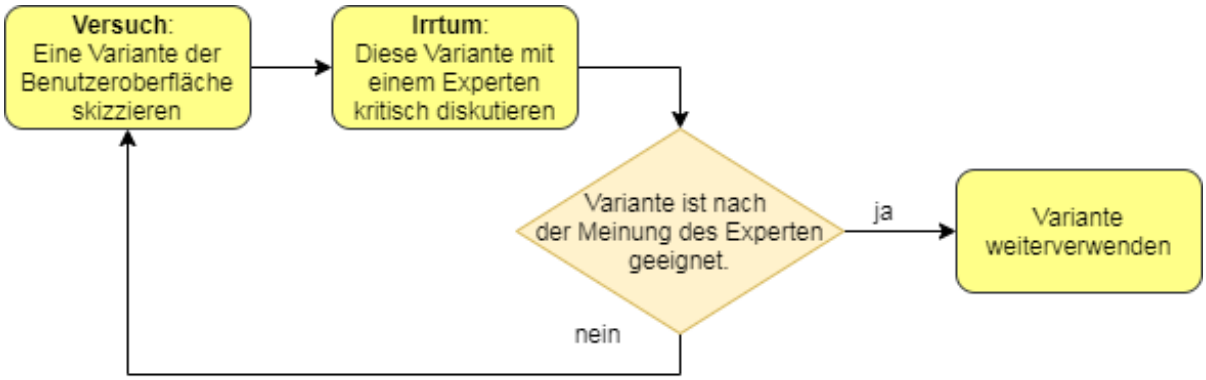


Abbildung 9: Übersichtliches Aktivitätsdiagramm für die Benutzeroberfläche

2.5 Datenfilterung

Die Abbildung 10 beschreibt das Vorgehen bei der Entwicklung der Funktionen zur Datenfilterung als ein Aktivitätsdiagramm. Der angewendete Entwicklungsprozess folgt einem iterativen Ansatz. Der Entwicklungsvorgang durchläuft die Phasen *Analyse*, *Entwurf* und *Implementierung* und schließt mit dem Test und Vergleich der Ergebnisse der beiden Iterationen ab.

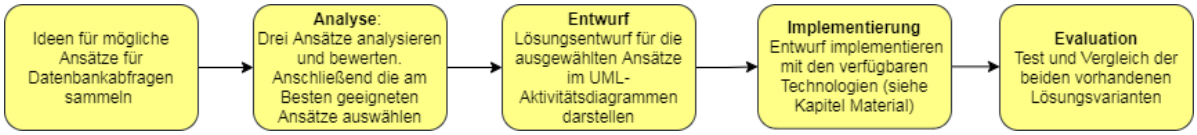


Abbildung 10: Aktivitätsdiagramm für den Entwicklungsprozess der Datenfilterung

2.5.1 Analyse

In der Analysephase werden verschiedene Ideen für mögliche Datenbankabfragen gesammelt. Es entstehen dabei drei konkrete Ideen. Anschließend werden Überlegungen zur Eignung dieser drei Varianten angestellt. Im Ergebnis wird eine Argumentation bezüglich der zu erwartenden Eignung einer jeden der drei beschriebenen Varianten erstellt und der am besten geeigneten Ansatz ausgewählt.

Erster Ansatz: Anfrage ohne definierte Zeitbedingung

In dem ersten Ansatz definiert der Benutzer lediglich Parameter-abhängige Filterbedingungen und keine Zeitbedingungen. Die definierten Parameter-abhängigen Filterbedingungen werden auf einem Zeitintervall von einem Jahr ausgewertet (Startdatum: Tag und Monat des heutigen Datums, Jahreszahl des heutigen Datums vermindert um eins, aktuelle Uhrzeit; Enddatum: heutiges Datum, aktuelle Uhrzeit). Dies entspricht einer Berücksichtigung des kompletten Datenbankinhalts, da die Datenbank lediglich alle Patientendaten der letzten 365 Tage speichert und ältere Daten automatisch gelöscht werden.

Zweiter Ansatz: Anfrage mit definierten Zeitbedingungen

Beim zweiten Ansatz definiert der Benutzer zusätzlich zu den Parameter-abhängigen Filterbedingungen auch ein Zeitintervall über dem die definierten Filterbedingungen auszuwerten sind.

Dritter Ansatz: Optimierte Anfrage mit definierten Zeitbedingungen

Dieser Ansatz stellt eine Optimierung des zweiten Ansatzes dar. Das Zeitintervall, über dem die definierten Filterbedingungen ausgewertet werden sollen, wird dabei in mehrere kürzere Zeitintervalle aufgespalten. Die Datenbankabfrage wird dann nacheinander über jedem Teilintervall ausgewertet. Die erhaltenen Teilergebnisse werden zusammengefasst.

Vergleich der Ansätze

Zur Auswahl der zu implementierenden Ansätze wurden alle drei Ansätze analysiert und verglichen. Die Betrachtung erfolgte unter der Fragestellung nach dem für den jeweiligen Ansatz zu erwartenden Aufwand bei der Anfrageauswertung. Eine allgemeine Struktur wurde erkannt: Jeder Ansatz besteht aus drei Schritten, (1) einer Überprüfung der Einzelbedingungen, gefolgt von (2) einer Prüfung auf gleichzeitige Erfüllung der Einzelbedingungen (beinhaltet Kreuzproduktbildung) und schließlich (3) aus der Aggregation aller Parameterdatensätze zum Endergebnis. Diese feinere Strukturierung war für die Beantwortung der oben genannten Fragestellung notwendig. Das Ergebnis des Vergleichs ist im Abschnitt 3.2.1 beschrieben.

2.5.2 Entwurf

Für die ausgewählten Ansätze wird eine Lösung entworfen. Das Verhalten der Ansätze wird in jeweils einem UML-Aktivitätsdiagramm dargestellt. Es werden unterschiedliche Abstraktionsebenen eingeführt und für die jeweilige Abstraktionsebene ein geeignetes UML-Aktivitätsdiagramm entworfen.

2.5.3 Implementierung

Die ausgewählten Ansätze werden anschließend auf Basis der Entwürfe implementiert, d.h. in praktikable Lösungsvarianten umgesetzt. Bei der Implementierung werden die im Abschnitt 2 beschriebenen Technologien verwendet.

2.5.4 Test und Vergleich der Ergebnisse

Abschließend werden die beiden erstellten Lösungsvarianten hinsichtlich ihrer Geschwindigkeit bei der Auswertung typischer Datenbankabfragen verglichen. Zum Vergleich werden die beiden Varianten bei der Ausführung von Filterbedingungen getestet; die Ergebnisse werden dokumentiert und

ausgewertet. Die beiden Varianten werden dabei auf dieselben Filterbedingungen angewendet. Testergebnis sind Kenntnisse bei welchen Konfigurationen welche Variante Geschwindigkeitsvorteile besitzt. Zur Bewertung des Vergleichs wird die Ausführungszeit der Abfrage herangezogen. Darauf aufbauend können Empfehlungen für die Verwendung der einen- oder der anderen Variante gegeben werden.

3 Ergebnis

In diesem Abschnitt wird ein Überblick über den Arbeitsablauf des Systems gegeben, die Benutzeroberfläche beschrieben, sowie Entwurf und Implementierung der beiden ausgewählten Lösungsvarianten für die Auswertung der Filterbedingungen beschrieben.

Wie in der Überblicksdarstellung in Abbildung 11 dargestellt beinhaltet die Arbeit mit dem System sowohl manuelle- als auch automatische Aktivitäten. Der Ablauf ist auf dieser Abstraktionsebene unabhängig von den beiden Varianten, die im Folgenden betrachtet werden.

Im Einzelnen besteht der Arbeitsablauf des Systems aus den folgenden Aktivitäten und arbeitet mit den folgenden Daten:

Filterbedingungen definieren (manuelle Aktivität): Der Benutzer definiert die Bedingungen und das Zeitintervall, in der die SQL-Abfrage für die Bedingungen ausgeführt sind. Er verwendet dazu die im Abschnitt 3.1 beschriebene Benutzeroberfläche.

Datenbankabfragen ausführen (automatische Aktivität): In diesem Schritt werden die definierten Filterbedingungen durch eine oder mehrere Datenbankabfragen ausgeführt. Die beiden implementierten Varianten unterscheiden sich in diesem Schritt. Ergebnis von Datenbankanfrage ist eine Tabelle mit Spalten für die Patienten-Id, sowie den Start- bzw. den Endzeitpunkt des Intervalls innerhalb dessen alle definierten Filterbedingungen für den jeweiligen Patienten erfüllt sind.

Patient auswählen und Detailinformationen anzeigen (manuelle Aktivität): In diesem Arbeitsschritt wird dem Benutzer eine Liste mit Patienten-Informationen (Nachname, Vorname, Id) im Webbrowser angezeigt. Der Benutzer kann einen einzigen Patienten auswählen. Danach wird ihm für den ausgewählten Patienten das Zeitintervall angezeigt in dem alle definierten Filterbedingungen erfüllt sind.

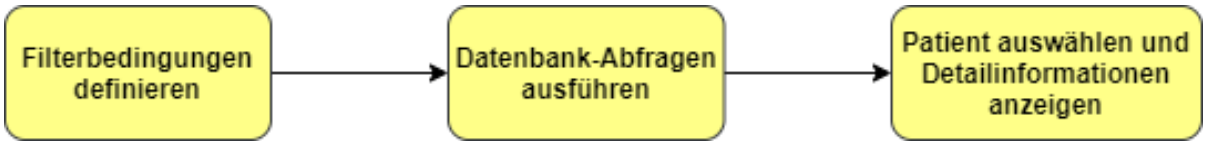


Abbildung 11: Überblick über den Arbeitsablauf des Systems

3.1 Benutzeroberfläche

Die Abbildung 12 zeigt die Benutzeroberfläche im Webbrowser. Diese Benutzeroberfläche ist einfach bedienbar.

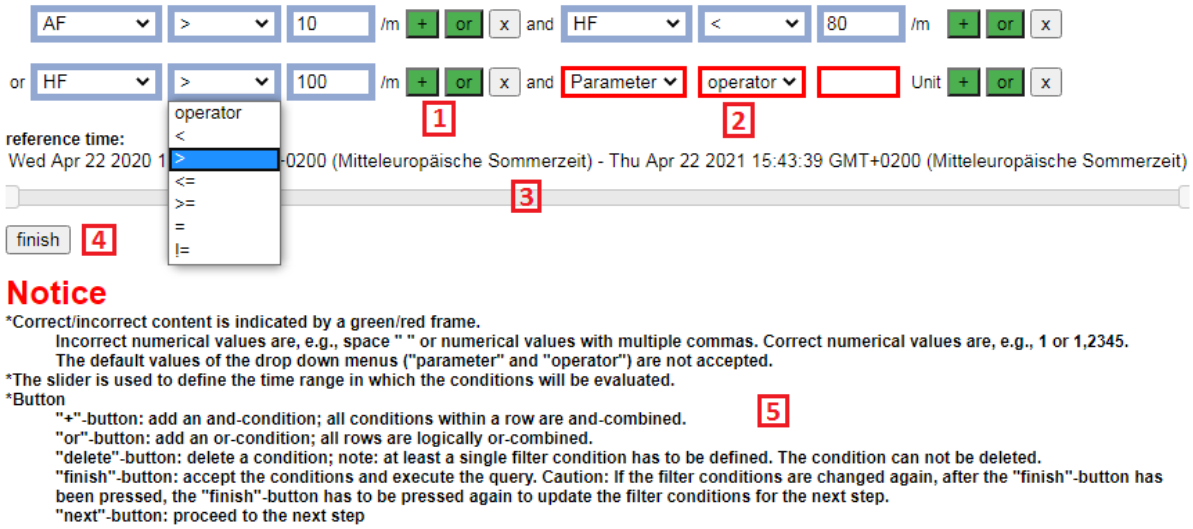


Abbildung 12: Benutzeroberfläche

Über Dropdown-Listen und Textfelder kann der Benutzer beliebige Filterbedingungen definieren. Eine Filterbedingung besteht aus drei GUI-Elementen. Die Filterbedingung (z.B. „HF>100“, siehe zweite Zeile links in Abbildung 12) besteht aus folgenden GUI-Elementen: Eine Dropdown-Liste zur Auswahl des Parameternamens („HF“), eine Dropdown-Liste zur Auswahl des Vergleichsoperatoren („>“) (siehe Nummer 1 in Abbildung 12) und schließlich einem Textfeld zur Eingabe des Vergleichswertes („100“). Neue Dropdown-Listen bzw. Textfelder und solche mit ungültigem Inhalt werden mit einer roten Umrandung gezeichnet (siehe Nummer 2 in Abbildung 12). In der Benutzeroberfläche werden dem Anwender einige Hinweise zum Auffinden ungültiger Eingaben angezeigt (siehe Nummer 5 in Abbildung 12). Dropdown-Listen bzw. Textfelder mit korrektem Inhalt werden mit einer blauen Umrandung gezeichnet. Zu einem ausgewählten Parameter wird automatisch die korrekte Einheit rechts neben das Textfeld für den Parameterwert geschrieben.

Die Filterbedingungen sind zeilenweise angeordnet. Alle Filterbedingungen einer Zeile werden mit dem logischen Operator UND verknüpft. Diese Verknüpfung ist implizit, sodass der Benutzer keinen logischen Operator auswählen muss. Zeilen werden implizit mit dem logischen Operator ODER verknüpft. Daher braucht man keine Klammer setzen. Der Benutzer kann beliebig viele Zeilen- und auch beliebig viele Bedingungen innerhalb einer Zeile definieren. Durch einen Klick auf eine mit „+“ beschriftete Schaltfläche kann der Benutzer eine neue Filterbedingung rechts neben einer existierenden Filterbedingung erstellen. Jeder Filterbedingung ist eine „+“-Schaltfläche (siehe Nummer

1 in Abbildung 12) zugeordnet, sodass neben jeder Filterbedingung eine neue Filterbedingung erstellt werden kann. Ein Klick auf eine mit „or“ beschriftete Schaltfläche erstellt eine neue Zeile (ODER-Verknüpfung). Jeder Filterbedingung ist ebenfalls eine eigene „or“-Schaltfläche zugeordnet. Schließlich kann eine Filterbedingung durch einen Klick auf die ihr zugeordnete „x“-Schaltfläche gelöscht werden (siehe Nummer 1 in Abbildung 12). Die Anwendung erlaubt die Suche nach Patienten die innerhalb bestimmter zeitlicher Grenzen die Filterbedingungen (wenigstens vorübergehend) in ihrer definierten logischen Verknüpfung erfüllen. Diese Funktionalität erfordert neben der Definition von Filterbedingungen, und ihrer logischen Verknüpfung, auch die Möglichkeit einen Zeitraum mit zeitlichen Grenzen zu definieren. Die Benutzeroberfläche bietet dem Anwender dazu einen sogenannten Time-Range-Slider an (siehe Nummer 3 in Abbildung 12). Die maximale Dauer des Zeitraumes beträgt ein Jahr und der früheste Beginn ist der Tag der Anfrage vermindert um ein Jahr. Diese zeitlichen Obergrenzen wurden gewählt da in der medizinischen Datenbank ohnehin alle Daten nach einem Jahr gelöscht werden. Nach der Definition aller Filterbedingungen und ihrer logischen Verknüpfung kann der Anwender mit einem Klick auf die mit „finish“ (siehe Nummer 4 in Abbildung 12) beschriftete Schaltfläche die Generierung der SQL-Query auslösen (siehe Abschnitt 3.2) und diese zur Ausführung an die Datenbank weiterleiten. Nach der erfolgreichen Ausführung der Anfrage werden dem Anwender die erhaltenen Ergebnisse angezeigt. Ein beispielhaftes Anfrageergebnis ist in Abbildung 13 dargestellt.

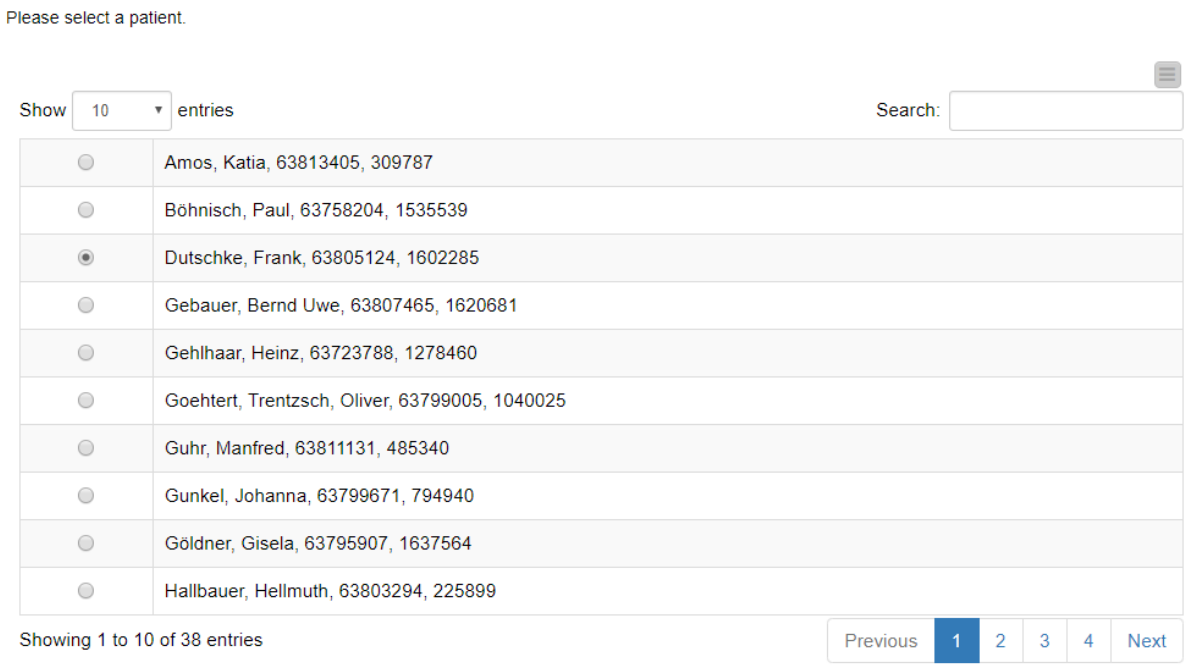


Abbildung 13: Ergebnis nach der Ausführung der SQL-Anfrage

3.2 Datenfilterung

In diesem Abschnitt werden die drei gefundenen Ansätze analysiert und verglichen. Zwei Ansätze werden zur Umsetzung ausgewählt und der Entwurf der zugehörigen Lösungsvarianten wird anhand eines Aktivitätsdiagramms beschrieben.

3.2.1 Vergleich der Ansätze

In der Tabelle 1 werden die drei Ansätze (Beschreibung siehe Abschnitt 2.5.1) hinsichtlich ihrer Verarbeitungsgeschwindigkeit analysiert. Algorithmische Eigenschaften die für eine hohe bzw. eine niedrige Verarbeitungsgeschwindigkeit sprechen werden tabellarisch dargestellt.

	Argumente für eine hohe Verarbeitungsgeschwindigkeit	Argumente für eine niedrige Verarbeitungsgeschwindigkeit
Erster Ansatz	Überprüfung der Einzelbedingungen („Vorfilterung“) ohne Berücksichtigung des vom Benutzer definierten Zeitbereiches führt zu einem Geschwindigkeitsgewinn da der Zeitaufwand für die Überprüfung der Zeitbedingungen entfällt. ➔ Vorfilterung ist schnell	Auswertung des zweiten Schritts dauert wesentlich länger, da der erste Schritt eine größere Datenmenge liefert, die im zweiten Schritt zu verarbeiten ist und da die Überprüfung der gleichzeitigen Erfüllung mehrerer Filterbedingungen relativ aufwändig ist. ➔ Datenmenge ist sehr groß
Zweiter Ansatz	In der Vorfilterung werden aufgrund der Auswertung der Zeitbedingung viele Daten ausgefiltert, sodass diese etwas länger dauert. Der zweite Schritt wird dafür schneller abgeschlossen, da die zu verarbeitende Datenmenge reduziert ist. Durch die Wahl eines kurzen Zeitbereiches kann der Nutzer damit die zu verarbeitende Datenmenge reduzieren und die Ausführungszeit der Anfrage verkürzen. ➔ Datenmenge kann reduziert werden	Die Vorfilterung dauert durch die zusätzliche Auswertung der Zeitbedingung länger. Für sehr lange Zeitbereiche in der Zeitbedingung ist gegenüber dem ersten Ansatz keine Verbesserung erkennbar, da in diesen Fällen auch eine große Datenmenge anfällt und der zweite Schritt damit (wie in der ersten Variante) sehr aufwändig wird. ➔ Vorfilterung ist langsamer als erster Ansatz und bringt für lange Zeitbereiche keinen Geschwindigkeitsvorteil
Dritter Ansatz	Dieser Ansatz ist eine Optimierung des zweiten Ansatzes. Es besteht noch der Nachteil der zusätzlichen Überprüfung von zeitlichen Bedingungen in der Vorfilterung, aber durch die Verringerung der in der Einzelanfrage zu verarbeitenden Datenmenge könnte sich in der Summe ein Geschwindigkeitsvorteil ergeben. ➔ Gesamtverarbeitungsaufwand könnte bei langen Zeitbereichen geringer sein als für den zweiten Ansatz.	Zusätzliche Zeitverluste durch häufigere Nachrichtenübertragungen. Auch ist kein Geschwindigkeitsvorteil für kurze Zeitbereiche zu erwarten. Ein Geschwindigkeitsvorteil kann nur entstehen, wenn sich die Ausführungszeiten mehrerer kurzer Anfragen nicht genau zur Ausführungszeit einer langen Anfrage addieren. Geschwindigkeitsvorteile sind denkbar, aber vor der Implementierung nicht leicht abzuschätzen.

Tabelle 1: Vergleich möglicher Lösungsansätze

Das in Tabelle 1 dargestellte Ergebnis der Analyse lässt einen Geschwindigkeitsnachteil des ersten Ansatzes gegenüber den anderen beiden Ansätzen erwarten, da bei der Überprüfung der gleichzeitigen Erfüllung mehrerer Filterbedingungen in jedem Fall eine hohe Datenmenge zu verarbeiten ist. Dieser Nachteil würde nur entfallen, wenn der Anwender die Filterbedingungen über einen langen Zeitraum (ein Jahr) auswerten wollte. Aufgrund der hohen in der Datenbank gespeicherten Datenmenge lassen solche langen Zeiträume aber ohnehin für den Nutzer nicht akzeptable Ausführungsdauern erwarten und werden in der Praxis kaum durchgeführt. Daher wird der erste Ansatz nicht in Software umgesetzt. Die Ansätze zwei und drei lassen insbesondere für kurze Zeiträume Geschwindigkeitsvorteile erwarten. Ansatz drei könnte bei längeren Zeiträumen zu kürzeren Ausführungszeiten führen als Ansatz zwei. Daher werden beide Ansätze in Software umgesetzt und die jeweils erzielbaren Ausführungszeiten für unterschiedliche Anfragen und Filterzeiträume miteinander verglichen. Im folgenden Abschnitt werden Entwurf und Implementierung der beiden Ansätze beschrieben. Entwurf und Implementierung des zweiten Ansatzes bilden dabei die erste Lösungsvariante. Entwurf und Implementierung des dritten Ansatzes bilden dabei die zweite Lösungsvariante.

3.2.2 Entwurf

In diesem Kapitel werden die Entwürfe beider Lösungsvarianten als Aktivitätsdiagramme dargestellt. Außerdem wird der Aufbau der Datenbankabfrage, und die daraus folgenden Phasen ihrer Auswertung, beschrieben.

Lösungsvariante 1

Die Beschreibung erfolgt anhand eines Aktivitätsdiagrammes welches die Arbeitsschritte der Anwendung und nötige Nutzerhandlungen beinhaltet (siehe Abbildung 14). Zur Veranschaulichung sind zu den meisten Aktivitäten und übertragenen Objekten in Kommentarblöcken Beispiele gegeben. Dieses Aktivitätsdiagramm besteht aus vier Aktivitäten:

1. **Filterbedingungen und Gesamtzeitintervall definieren:** In diesem Schritt definiert der Anwender die Filterbedingungen und das Zeitintervall, nach dem die in der Datenbank vorhandenen Patienten-Monitoring-Daten zu untersuchen sind. Die Definition erfolgt über die grafische Benutzeroberfläche, welche bereits im Abschnitt 3.1 vorgestellt wurde.
2. **Generiere Datenbankabfrage:** In diesem Arbeitsschritt wird der im vorherigen Schritt vom Anwender über die Benutzeroberfläche definierte logische Ausdruck in eine Datenbankabfrage übersetzt. (Datenbankabfrage: Objekt *<<Data>> Datenbankabfrage)*
3. **Datenbankabfrage ausführen:** Die im vorherigen Arbeitsschritt generierte Datenbankabfrage wird vom Datenbankmanagementsystem des Klinikums (Microsoft SQL-Server) auf der Datenbank

des Klinikums ausgeführt. Als Abfrageergebnis wird eine Liste mit den IDs, Nachnamen und Vornamen derjenigen Patienten zurückgegeben die – innerhalb des vom Nutzer vorgegeben Zeitintervalls – mit ihren Monitoring-Daten wenigstens vorübergehend den vom Nutzer definierten logischen Ausdruck erfüllen (zurückgegebenes Ergebnis: Objekt *<<Data>> Liste der Patienten-ID, Nachname, Vorname*).

4. **Patient auswählen und Detailinformationen anzeigen:** In diesem Arbeitsschritt wird das im letzten Arbeitsschritt erhaltene Ergebnis der Datenbankabfrage dem Anwender präsentiert. Es werden also Informationen zu verschiedenen Patienten angezeigt. Klickt der Anwender auf einen Patienten werden weitere Informationen angezeigt: Frühester und spätester Zeitpunkt an dem der Patient mit seinen Monitoring-Daten den logischen Ausdruck erfüllt (angezeigte, bzw. ausgewählte Informationen: Objekt *<<Data>> Ausgewählte Patienteninformation, Startintervall, Endintervall"*).

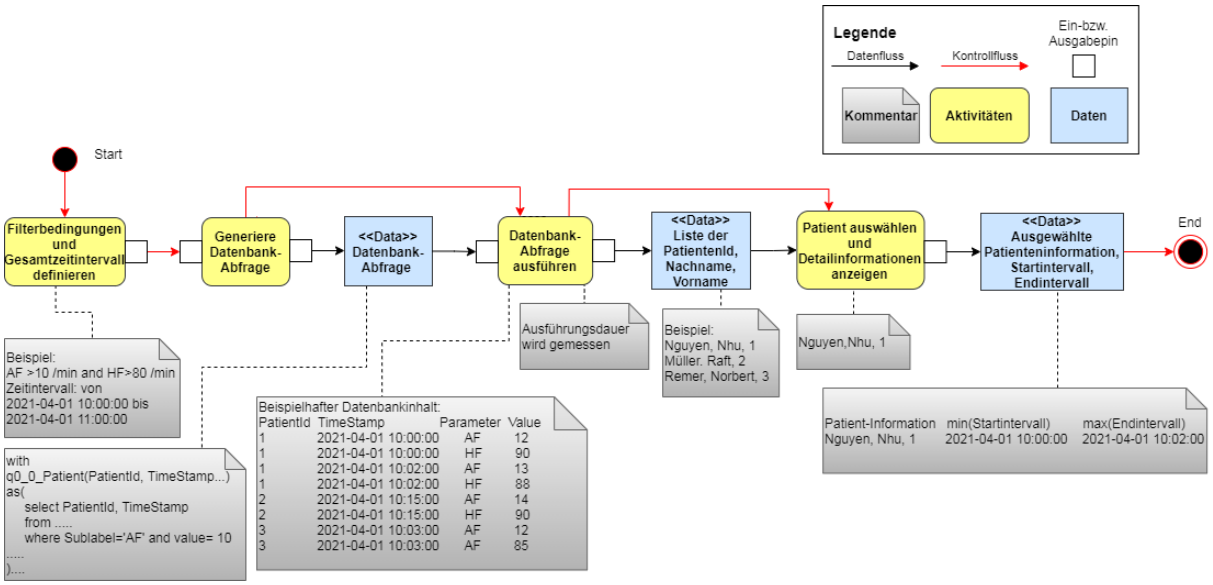


Abbildung 14: Aktivitätsdiagramm für die erste Lösungsvariante

Lösungsvariante 2

Die zweite Lösungsvariante stimmt in vielen Arbeitsschritten mit der ersten Lösungsvariante überein:

- Es wird die gleiche grafische Benutzeroberfläche zur Definition der logischen Formel benutzt
- Die von der Datenbank zurückgegebene Ergebnistabelle besitzt das gleiche Format wie in der ersten Lösungsvariante: (Patienteninformation, Startintervall, Endintervall)
- Der Nutzer kann abschließend ebenfalls aus einer Liste einen Patienten auswählen und sich so Detailinformationen anzeigen lassen

Folgende Unterschiede bestehen zwischen beiden Lösungsvarianten:

Lösungsvariante 1	Lösungsvariante 2
Das vom Anwender definierte Zeitintervall wird unverändert in der Datenbankabfrage verwendet.	In der Datenbankabfrage wird (zur Verringerung des Rechenaufwandes) ein kürzeres Zeitintervall verwendet („Teilintervall“)
Es wird nur eine einzige Datenbankabfrage generiert und von der Datenbank ausgewertet.	Es werden mehrere Datenbankabfragen generiert und nacheinander von der Datenbank ausgewertet. Jede Datenbankabfrage wird mit einem eigenen Teilintervall generiert. Für jedes Teilintervall wird eine Datenbankabfrage generiert und ausgeführt.
Keine Teilergebnisse müssen zusammengeführt werden.	Die für die einzelnen Teilintervalle erhaltenen Abfrage-Ergebnisse („Teilergebnisse“) müssen schrittweise zu einem Gesamtergebnis zusammengeführt werden.

Tabelle 2: Vergleich der Lösungsvarianten 1 und 2

Die in der Tabelle 2 beschriebenen Unterschiede finden sich im Aktivitätsdiagramm der Abbildung 15 wieder: In der Aktivität „Aufspaltung in Teilintervalle“ werden die zu verwendenden Teilintervalle bestimmt und in einer Liste gespeichert (Objekt „<<Data>> Teilintervalle“). Die zeitliche Länge der Teilintervalle kann über die Systemkonfiguration eingestellt werden, aber nicht durch den Anwender verändert werden. Die Aktivitäten „Vervollständige Datenbankabfrage mit Zeitintervall“ und „Führe Teil-Datenbankabfrage aus“ werden mehrfach ausgeführt. Für jede Ausführung wird ein anderes Teilintervall aus dem Objekt „<<Data>> Teilintervalle“ verwendet. Nach jeder Ausführung wird das erhaltene Teilergebnis mit den aus den vorherigen Durchläufen erhaltenen Teilergebnissen zusammengeführt (Aktivität „Sammle Teilergebnisse (Patientenliste)“ und Objekt „<Data> Patientenliste (Gesamtergebnis)“). Vor jeder Ausführung wird überprüft ob bereits alle Teilintervalle abgearbeitet wurden (Kontrollknoten „Alle Teilintervalle abgearbeitet?“). Ist dies der Fall, wird keine weitere Datenbankabfrage ausgeführt und stattdessen dem Nutzer das Gesamtergebnis angezeigt (Aktivität „Anzeige der vollständigen Patientenliste“).

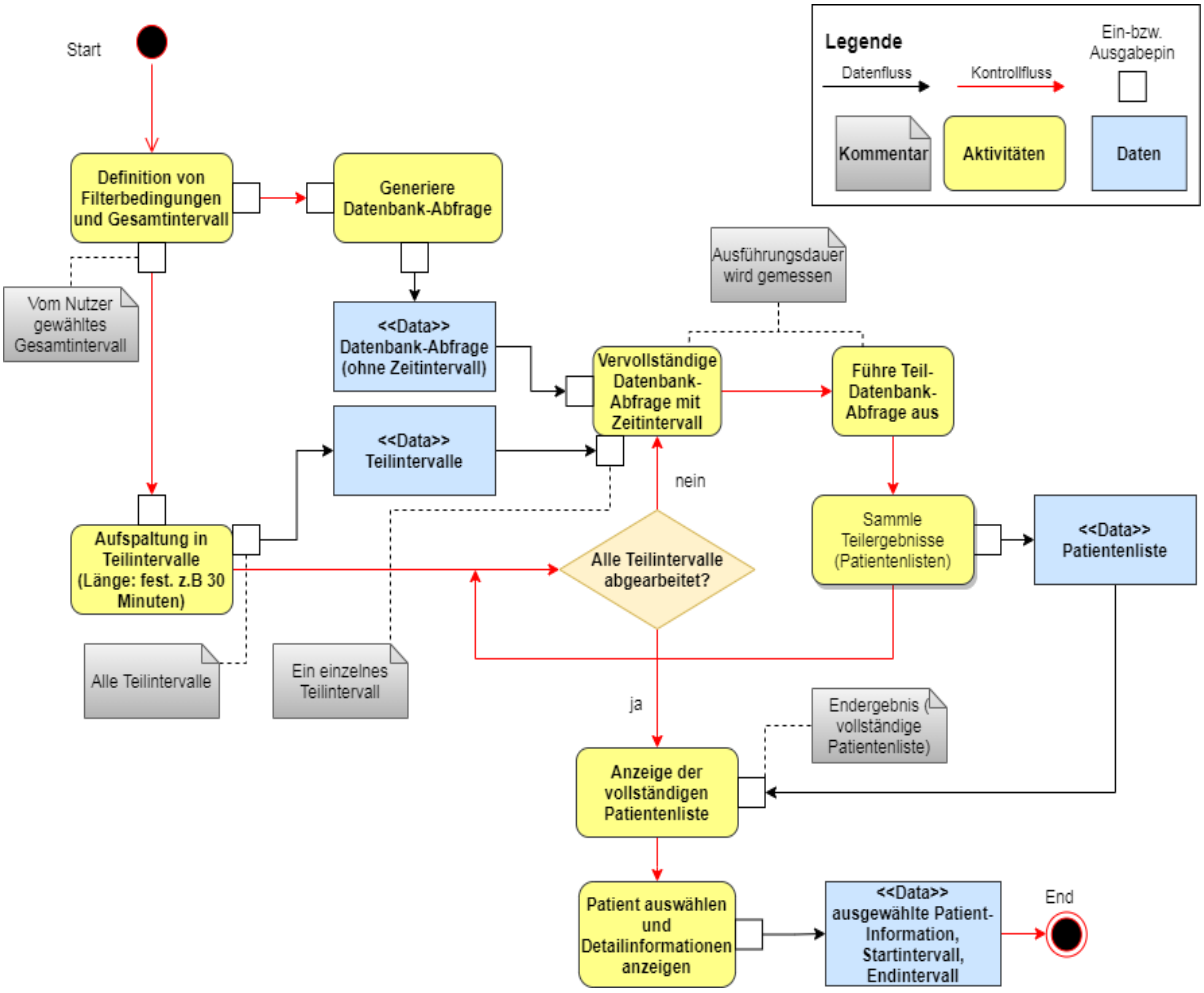


Abbildung 15: Aktivitätsdiagramm für die zweite Lösungsvariante

Entwurf der Datenbankabfrage

In beiden Varianten werden die Datenbankabfragen aus den vom Nutzer über die grafische Benutzeroberfläche definierten Filterbedingungen generiert. Wie aus den obigen Aktivitätsdiagrammen ersichtlich, gibt es beim Generieren der Datenbankabfragen Unterschiede zwischen den beiden Varianten. Diese Unterschiede betreffen aber nur das verwendete Zeitintervall. Konzeptionell sind die generierten Datenbankabfragen in beiden Varianten gleich. Im Folgenden wird daher die generierte Datenbankabfrage auf konzeptioneller Ebene für beide Varianten beschrieben.

Abbildung 16 zeigt eine schematische Darstellung (Aktivitätsdiagramm) für die Auswertung der Datenbankabfragen durch die Datenbank. Diese Auswertung lässt sich in drei Phasen unterteilen:

1. **Vorfilterung** (Phase 1): In dieser Phase wird die einzelne Filterbedingung auf die vorhandenen Daten (d.h. alle in der Datenbank vorhandenen Datenpunkte) angewendet. Dabei werden als Ergebnis nur Datenpunkte weitergegeben deren Parameter dem in der Filterbedingung definierten Parameter entspricht und deren Messwert die in der Filterbedingung definierte

Vergleichsbedingung (bestehend aus Vergleichsoperator und Vergleichswert) erfüllt. Außerdem wird überprüft ob der Datenpunkt innerhalb des vom Benutzer eingegebenen Zeitintervalls gemessen wurde. Alle Datenpunkte die alle genannten Bedingungen erfüllen werden in die zweite Phase weitergegeben.

2. **Filterung zur Überprüfung der zeitlichen Nähe und der Patienten-ID** (Phase 2): In dieser Phase wird für alle Filterbedingungen einer Zeile (d.h. für alle UND-verknüpften Filterbedingungen) überprüft, ob es Zeitpunkte gibt an denen sie näherungsweise gleichzeitig erfüllt sind. Außerdem wird überprüft ob sie zu ein- und demselben Patienten gehören. Als Ergebnis werden für jede Zeile die Namen der Patienten (und Informationen zu den zugehörigen Zeitstempeln) weitergegeben für die Datenpunkte gefunden wurden die die genannten Bedingungen erfüllen. In die Überprüfung werden alle möglichen Kombinationen von den in Phase 1 zu den jeweiligen Filterbedingungen der Zeile gefundenen Datenpunkte miteinbezogen.
3. **Endergebnis-Tabelle** (Phase 3): In der dritten Phase werden die für die einzelnen Zeilen erhaltenen Teilergebnisse (Patienten-ID, Zeitinformationen), aus der Überprüfung der zeitlichen Nähe, zu einem Gesamtergebnis zusammengefasst.

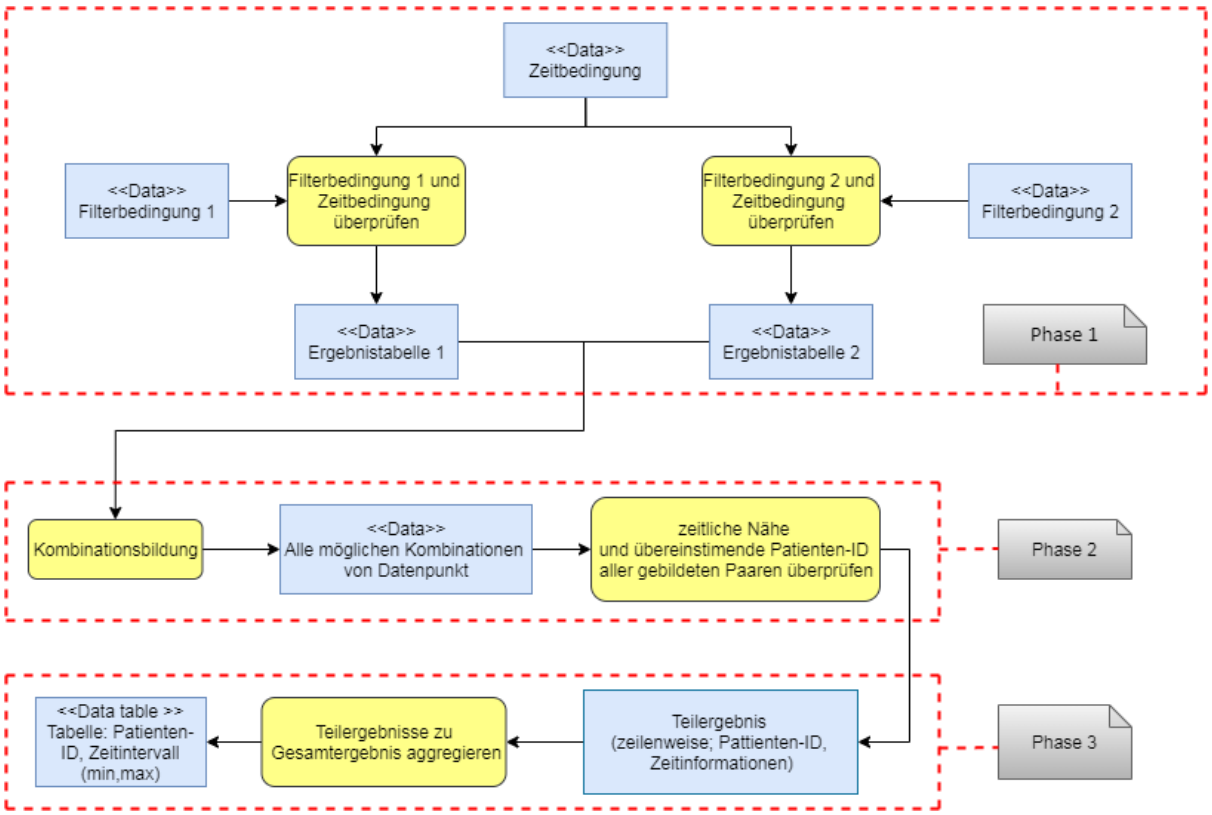


Abbildung 16: Schema zur Auswertung der Datenbankabfragen

3.2.3 Implementierung

In diesem Abschnitt wird die Implementierung des Mechanismus zur Definition und Auswertung von Filterbedingungen in KNIME beschrieben. Abbildung 17 gibt einen Überblick und zeigt die Struktur der obersten Ebene der entwickelten Komponenten-Hierarchie des KNIME-Workflows.

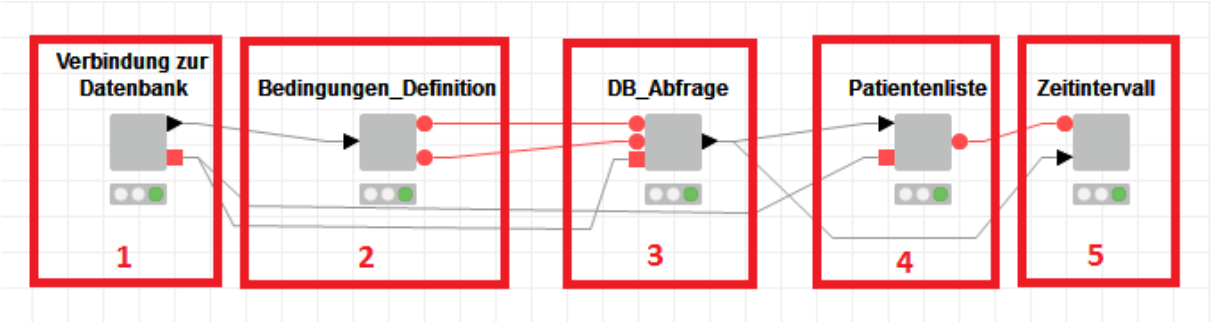


Abbildung 17: KNIME-Workflow des entwickelten Mechanismus

Der Workflow besteht aus 5 Komponenten:

1. **Komponente „Verbindung zur Datenbank“:** Diese Komponente übernimmt den Verbindungsaufbau zur Datenbank. Dies ist notwendig, da das Bestehen einer Datenbankverbindung eine Voraussetzung für das Absenden von Datenbankabfragen und deren Auswertung durch die Datenbank ist. Die Abbildung 18 zeigt den inneren Aufbau der Komponente. Über den Knoten „Microsoft SQL-Server Connector (legacy)“ wird eine Verbindung zwischen dem Rechner auf dem der KNIME-Workflow ausgeführt wird (Entwickler-Rechner, bzw. KNIME-Server Host im operativen Betrieb) und der Datenbank hergestellt. Der Knoten erwartet die Eingabe von Namen/Port des Hostrechners der Datenbank, sowie von Authentifizierungsinformationen. Ausgabe des Knotens (hellroter Pin) ist eine Objekt-Repräsentation der eingerichteten Datenbankverbindung. Alle Knoten an die dieses Objekt weitergeleitet wird sind in der Lage über die eingerichtete Verbindung Kontakt mit der Datenbank aufzunehmen und Datenbankabfragen abzusenden. Bereits innerhalb dieser Komponente wird eine Datenbankabfrage definiert und ausgeführt. Die Definition geschieht über den Knoten „Database Table Selector (legacy)“, die Weiterleitung der Anfrage an die Datenbank durch den Knoten „Database Connection Table Reader (legacy)“. Bei dieser Datenbankabfrage handelt es sich um die Ermittlung der Namen der Parameter, und der zugehörigen Einheiten, zu denen in der Datenbank Parameterwerte gespeichert sind. Diese Information wird benötigt für die Anzeige der Auswahlmöglichkeiten im Dropdown-Menü zur Parameter-Auswahl in der GUI zur Definition der Filterbedingungen. Als Ergebnis wird eine Liste von Parameter-Namen zurückgegeben. Diese wird durch den Knoten „Sorter“ alphabetisch sortiert und schließlich über den Ausgang der Komponente weitergegeben.

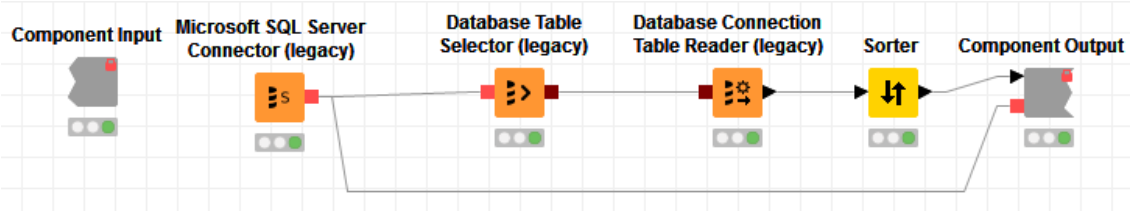


Abbildung 18: Innerer Aufbau der Komponente „Verbindung zur Datenbank“

2. **Komponente „Bedingungen_Definition“:** In der ersten Lösungsvariante besteht diese Komponente nur aus einem einzigen Knoten: „Generic JavascriptView“. Dieser Knoten enthält die Implementierung der Benutzeroberfläche für die Definition der Filterbedingungen (siehe Abschnitt 3.1). Außerdem enthält er den Code zur Generierung der Datenbankabfrage aus den vom Benutzer eingegebenen Filterbedingungen. Der Knoten ist in der Skriptsprache JavaScript implementiert. Der Knoten erhält als Eingabe eine Liste mit den im Parameterauswahl-Dropdownmenü anzuzeigenden Parameternamen, sowie eine Liste mit den zugehörigen Einheiten. Über den Ausgabeport wird die generierte Datenbankabfrage als eine Flow-Variable weitergegeben (siehe Abbildung 19).

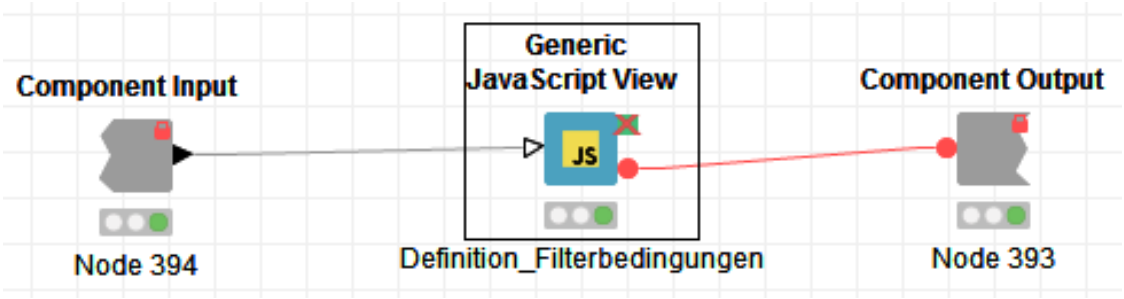


Abbildung 19: Die Komponente "Bedingungen_Definition" der ersten Lösungsvariante

Die Benutzeroberfläche wird nach folgendem Ansatz implementiert:

Zur Initialisierung wird ein JavaScript-Tabellenelement (<table></table>) angelegt. Das dynamische erweitern der Tabelle um neue Filterbedingungen bzw. neue Zeilen geschieht durch das dynamische Erzeugen und Einfügen von Tabellenzellen (<td></td>) bzw. Tabellenzeilen-Elementen (<tr></tr>). Dazu wird nach einem Klick auf die Schaltflächen „+“ bzw. „or“ eine der jeweiligen Schaltfläche zugeordnete Funktion aufgerufen. Zum Löschen von Filterbedingungen werden alle zugehörigen Tabellenzellen (Dropdownmenüs, Labels, Schaltflächen) nach Klick auf eine Schaltfläche „x“ aus dem Tabellenelement entfernt. Die Generierung der Datenbankabfrage erfolgt nach einem Klick auf die Schaltfläche „finish“ in einer eigens dafür definierten Funktion. In dieser Funktion werden nacheinander die zu den drei Phasen (Vorfilterung usw., siehe Unterabschnitt „Entwurf der Datenbankabfrage“) gehörenden Teile der Abfrage generiert. Die Funktion läuft beim Generieren eines jeden der drei Teile, Zeile für Zeile, und innerhalb einer Zeile,

Filterbedingung für Filterbedingung, durch die Tabelle und generiert dabei die für den jeweiligen Teil benötigten Ausdrücke. Im Folgenden ist eine beispielhafte Abfrage dargestellt:

```
WITH
  q0_0_Patient(p0_0_PatientId,t0_0_TimeStamp,s0_0_SubLabel,v0_0_Value)
  AS
    (
    select PatientId, TimeStamp,SubLabel, Value
    from External_Numeric,External_NumericValue
    where External_Numeric.SubLabel='AF' and External_NumericValue.Value > 10
    and TimeStamp between '2020-04-22 15:43:39.355 +02:00' and '2021-04-18
15:43:39.355 +02:00'
    and External_NumericValue.NumericId = External_Numeric.Id
    ),

   q0_1_Patient(p0_1_PatientId,t0_1_TimeStamp,s0_1_SubLabel,v0_1_Value)
   AS
    (
   select PatientId, TimeStamp,SubLabel, Value
   from External_Numeric,External_NumericValue
   where External_Numeric.SubLabel='HF' and External_NumericValue.Value < 80
   and TimeStamp between '2020-04-22 15:43:39.355 +02:00' and '2021-04-18
15:43:39.355 +02:00'
   and External_NumericValue.NumericId = External_Numeric.Id
    ),

    q1_0_Patient(p1_0_PatientId,t1_0_TimeStamp,s1_0_SubLabel,v1_0_Value)
    AS
   (
   select PatientId, TimeStamp,SubLabel, Value
   from External_Numeric,External_NumericValue
   where External_Numeric.SubLabel='HF' and External_NumericValue.Value > 100
   and TimeStamp between '2020-04-22 15:43:39.355 +02:00' and '2021-04-18
15:43:39.355 +02:00'
   and External_NumericValue.NumericId = External_Numeric.Id
   ),

     frow0(PatientId,TimeStamp)
   AS
   (
   select p0_0_PatientId, t0_0_TimeStamp
   from q0_0_Patient
   inner join q0_1_Patient ON p0_0_PatientId=p0_1_PatientId
         and (t0_0_TimeStamp >= DATEADD(second, -30, t0_1_TimeStamp))
         and (t0_0_TimeStamp < DATEADD(second, 30, t0_1_TimeStamp))
   ),

    frow1(PatientId,TimeStamp)
   AS
   (
   select p1_0_PatientId, t1_0_TimeStamp
   from q1_0_Patient
   ),

   frowUnion(PatientId, TimeStamp)
   AS
   (
   SELECT PatientId, TimeStamp
```

```
   FROM frow0
         UNION

   SELECT PatientId, TimeStamp
   FROM frow1

         )
SELECT PatientId, min(TimeStamp) AS startinterval,CASE WHEN count(*)>0 THEN
max(TimeStamp) END as endinterval
FROM frowUnion
GROUP BY PatientId
```

Code-Listing: Beispiel Datenbankabfrage

Diese SQL-Abfrage wurde folgendermaßen generiert und wird von der Datenbank wie folgt ausgewertet: Für jede Filterbedingung einer jeden Zeile wird ein WITH-AS SQL-Ausdruck („Vorfilterausdruck“) generiert. Die Tabelle „External_Numeric“ enthält für jeden Parameter eine Zeile und ordnet ihm darin u.a. ein SubLabel und eine Id (NumericId) zu. In der Tabelle „External_NumericValue“ sind alle Messwerte aller Parameter aller Patienten gespeichert. Dieser Ausdruck liefert aus der Datenbank-Tabelle „External_NumericValue“ alle Werte des jeweiligen Parameters (siehe Code-Listing: `where External_Numeric.SubLabel='AF' and External_NumericValue.NumericId = External_Numeric.Id`) die mit ihrem Wert der jeweiligen Vergleichsbedingung (siehe Code-Listing: `and External_NumericValue.Value > 10`) genügen und innerhalb des vom Nutzer definierten Zeitbereiches aufgenommen wurden (siehe Code-Listing: `and TimeStamp between '2020-04-22 15:43:39.355 +02:00' and '2021-04-18 15:43:39.355 +02:00'`). Die so erhaltenen Werte bilden eine temporäre Tabelle und können in die weiteren Ausdrücke der SQL-Abfrage über den Namen des WITH-AS Ausdrucks wie gewöhnliche Tabellen einbezogen und verwendet werden.

Anschließend wird für jede Zeile eine SQL-Subquery „frowx“ generiert, wobei x der Zeilennummer entspricht (siehe Code-Listing: `frow0` und `frow1`). In dieser Subquery wird für jede Zeile überprüft ob alle in dieser Zeile enthaltenen Filterbedingungen für einen Patienten näherungsweise gleichzeitig erfüllt sind. Diese Überprüfung ist nötig da alle Filterbedingungen einer Zeile implizit mit dem logischen Operator UND verknüpft sind und damit zur Erfüllung der Filterbedingung gemeinsam, d.h. zum gleichen Zeitpunkt, erfüllt sein müssen. Die Subquery „frowx“ greift auf die Ergebnisse der WITH-AS Vorfilter-Ausdrücke der Filterbedingungen der jeweiligen Zeile zu und bildet sukzessive das Kreuzprodukt aus den Ergebnissen eines Vorfilterausdrucks mit den Ergebnissen des nächsten Vorfilterausdrucks. Diese sukzessive Kreuzproduktbildung wird für die Vorfilterausdrücke aller Filterbedingungen der jeweiligen Zeile durchgeführt. Die Länge des Ausdrucks hängt damit stark von der Anzahl der Filterbedingungen in dieser Zeile ab. Bei jeder Kreuzproduktbildung wird die Zuordnung zu einem gemeinsamen Patienten überprüft (siehe

Code-Listing:`p0_0_PatientId=p0_1_PatientId`), sowie die zeitliche Nähe der beiden Messwerte ((`t0_0_TimeStamp >= DATEADD(second, -30, t0_1_TimeStamp)) AND (t0_0_TimeStamp < DATEADD(second, 30, t0_1_TimeStamp)`)). Die näherungsweise Überprüfung der Gleichzeitigkeit erfolgt nach folgendem Ansatz: Für die möglichen Kombinationen der Datenpunkte (Kombinationsbildung durch hintereinander gelegte INNER JOINs) wird überprüft ob die Zeitstempel aller beteiligten Datenpunkte (also den aus der Vorfilterung für die jeweiligen Filterbedingungen erhaltenen Abtastwerte mit Zeitstempel) innerhalb eines Zeitfensters von einer Minute liegen. Konkret wird der Zeitstempel des jeweiligen Datenpunktes der ersten Filterbedingung (im Beispiel: `t0_0_TimeStamp`) der Zeile als ein Referenzzeitpunkt für die gesamte Kombination verwendet. Nun wird für die Datenpunkte aller weiteren Bedingungen überprüft ob der Referenzzeitpunkt innerhalb des Zeitfensters liegt welches mit dem Zeitpunkt „Zeitstempel des zu überprüfenden Datenpunktes der jeweiligen Filterbedingung minus 30 Sekunden“ (im Beispiel ausgedrückt als „`DATEADD(second, -30, t0_1_TimeStamp)`”) beginnt und mit dem Zeitpunkt „Zeitstempel des zu überprüfenden Datenpunktes der jeweiligen Filterbedingung plus 30 Sekunden“ (im Beispiel ausgedrückt als „`DATEADD(second, 30, t0_1_TimeStamp)`“) endet (siehe Code-Listing):

```
(t0_0_TimeStamp >= DATEADD(second, -30, t0_1_TimeStamp))
and (t0_0_TimeStamp < DATEADD(second, 30, t0_1_TimeStamp))
```

Schließlich werden über die SubQuery „frowUnion“ die Teilergebnisse der Zeilen-basierten frowx-Subqueries zu einem Gesamtergebnis vereinigt. Dieses Gesamtergebnis enthält die IDs (mit Timestamp) der Patienten die die Filterbedingungen mindestens einer Zeile vorübergehend (d.h. zumindest für einen einzigen Zeitpunkt), gleichzeitig erfüllen.

Der unterste SELECT-Ausdruck ist unabhängig von den vom Nutzer eingegebenen Filterbedingungen und ermittelt für jeden Patienten den frühesten- und spätesten Zeitpunkt an dem die Filterbedingungen mindestens einer Zeile erfüllt sind. Die übrigen Zeitpunkte werden für diesen Patienten verworfen und nicht zurückgegeben.

Die zweite Lösungsvariante unterscheidet sich von obiger Beschreibung durch folgende Punkte: Anstatt das vom Benutzer definierte Zeitintervall in die WHERE-Bedingungen der Vorfilterausdrücke aufzunehmen, werden stattdessen nur Platzhalter eingefügt („START“, „END“). Zur Auswertung der Filterbedingungen über den Teilintervallen wird im weiteren Verlauf des Workflows für jedes Teilintervall eine eigene Datenbankabfrage generiert. Dazu werden in einer Kopie des generierten SQL-Ausdrucks die beiden Platzhalter durch den Start- bzw. Endzeitpunkt des jeweiligen Teilintervalls ersetzt. Abbildung 20 zeigt den in der Komponente „Bedingungen_Definition“ enthaltenen Teil des Workflows. Diese Abbildung enthält auch detaillierte Informationen zum Verständnis der Bedeutung der einzelnen Unterknoten.

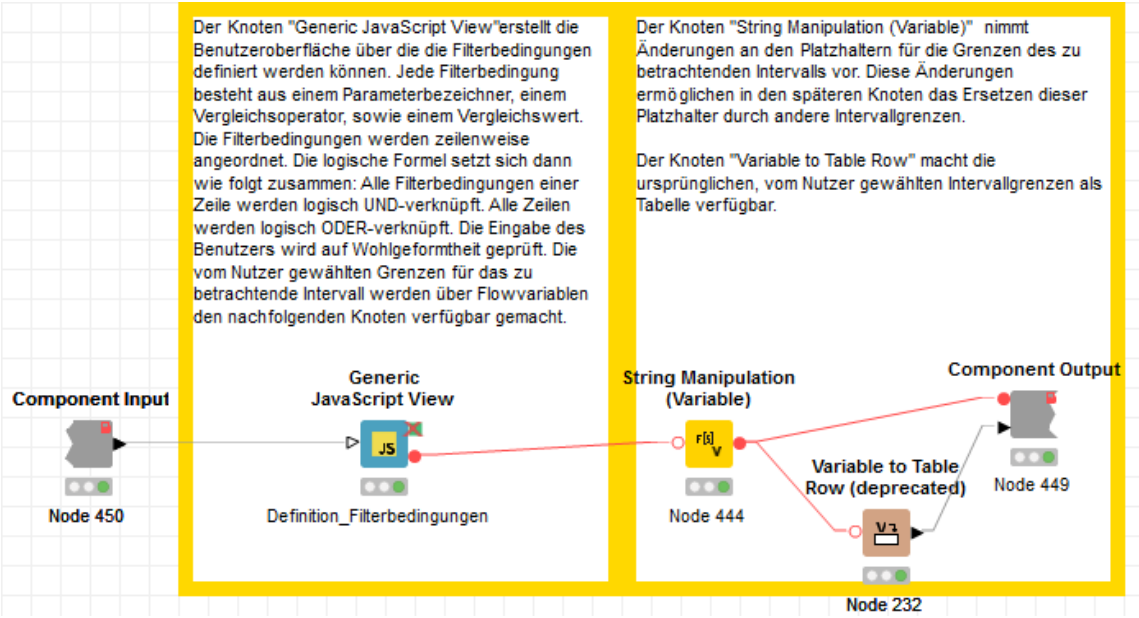


Abbildung 20: Die Komponente "Bedingungen_Definition" für die zweite Lösungsvariante

3. **Komponente „Datenbankabfrage“:** In dieser Komponente wird die zuvor generierte Datenbankabfrage ausgeführt.

   In der ersten Lösungsvariante wurde zuvor eine gültige Datenbankabfrage generiert. Diese wird anschließend über den Knoten „Database Reader (legacy)“ ausgeführt. Die Ergebnisliste (*Id*, *Startintervall*, *Endintervall* der Patienten die wenigstens vorübergehend den vom Benutzer definierten logischen Ausdruck erfüllen) wird durch den Knoten „GroupBy“ nach der Patienten-Id sortiert.

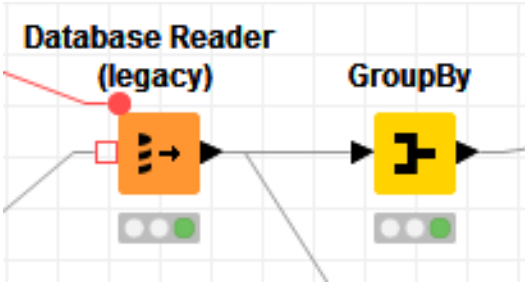


Abbildung 21: Die Komponente „Datenbankabfrage“ für die erste Lösungsvariante

Bei der zweiten Lösungsvariante besteht diese Komponente aus 2 Unterkomponenten: „Zeitintervall“ und „DB Abfrage“ (siehe Abbildung 22).

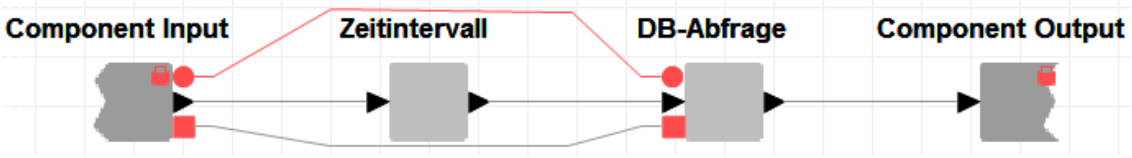


Abbildung 22: Die Komponente „Datenbankabfrage“ für die zweite Lösungsvariante

In der Komponente „Zeitintervall“ wird das Zeitintervall in mehrere kürzere Intervalle aufgespalten. Danach wird die Liste der Teilintervalle zur Komponente „DB-Abfrage“ weitergleitet. Die innere Struktur dieser Komponente ist in Abbildung 23 dargestellt.
Der Knoten „DB-Abfrage“ führt zu jedem Teilintervall eine eigene Datenbankabfrage aus. Der Knoten „Chunk Loop Start“ bekommt am Eingang die Liste mit den Teilintervallen, wird für jedes Teilintervall einmal ausgeführt und stellt bei jeder Ausführung am Ausgang dem Knoten „Table Row to Variable“ das jeweilige Teilintervall bereit. Dieser erstellt für den Start- und Endzeitpunkt aus dem Teilintervall jeweils eine Flow-Variable. Das Vorhandensein dieser beiden Flow-Variablen erlaubt dem nachfolgenden Knoten „Database Reader (legacy)“ den Start- und Endzeitpunkt automatisch an die Stelle der Platzhalter einzufügen. Die so angepasste Datenbankabfrage sendet der Knoten zur Datenbank und leitet das erhaltene Ergebnis zum Knoten „Loop End“ weiter. Dieser sammelt die erhaltenen Teilergebnisse und gibt schließlich nach dem Durchlauf aller Iterationen das so erhaltene Gesamtergebnis an den Knoten „GroupBy“ weiter, in dem es wie bei der ersten Lösungsvariante sortiert wird. Die Definition dieses Ablaufs findet sich in Abbildung 23.

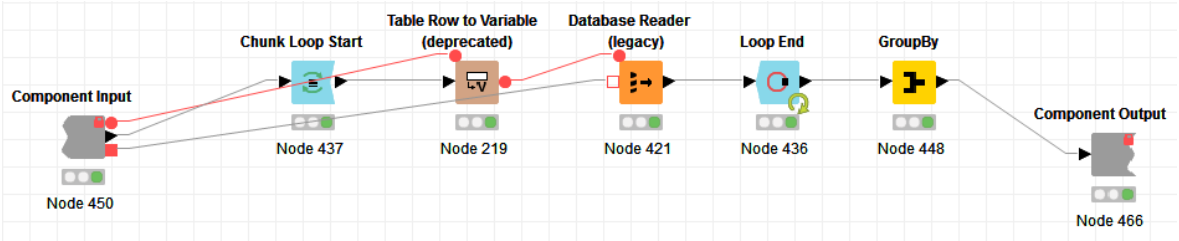
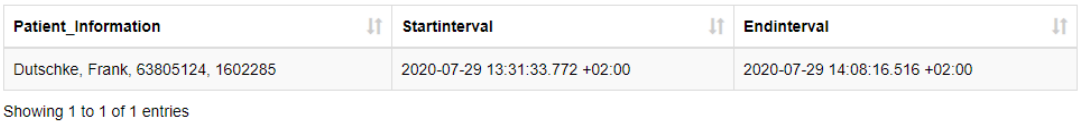


Abbildung 23: Die Komponente "DB-Abfrage" für die zweite Lösungsvariante

4. **Komponente „Patient List“:** Über diese Komponente wird dem Benutzer eine grafische Übersicht des erhaltenen Gesamtergebnisses dargestellt: Es wird eine Liste der Namen und IDs der zum Gesamtergebnis gehörenden Patienten angezeigt. Diese Komponente wurde auf dem bestehenden System OCLIDA übernommen [1]. Aus dieser Liste kann der Benutzer höchstens einen einzigen Patienten auswählen (siehe Abbildung 13).

5. **Komponente „Zeitintervall“:** Über diese Komponente werden dem Benutzer Detailinformationen zum ausgewählten Patienten angezeigt (siehe Abbildung 24). Die Spalte „Patient_Information“ zeigt dabei den Nachnamen und den Vornamen des Patienten, sowie eine lebenslang gültige Identifikationsnummer (LifetimeId) und eine nur für diesen Krankenhausaufenthalt gültige Identifikationsnummer (EncounterId). Die Spalten „Startinterval“ und „Endinterval“ zeigen den frühesten- bzw. spätesten Zeitpunkt an dem der Patient den vom Benutzer definierten logischen Ausdruck erfüllt.



Patient_Information	Startinterval	Endinterval
Dutschke, Frank, 63805124, 1602285	2020-07-29 13:31:33.772 +02:00	2020-07-29 14:08:16.516 +02:00

Showing 1 to 1 of 1 entries

Abbildung 24: Detailinformationen für den ausgewählten Patienten

3.2.4 Testergebnis

Die Zeitdauer für die Auswertung eines gegebenen logischen Ausdrucks, nach den beiden Lösungsvarianten, wurde getestet. Dazu wurden mehrere Testfälle definiert. Die Testfälle werten einen festen logischen Ausdruck über einem festen Zeitintervall (mit einer Länge von 6h) aus und unterscheiden sich dabei in der Anzahl der Teilintervalle in die das gegebene Zeitintervall aufgespalten wird. Für jedes Teilintervall wird der logische Ausdruck dabei über eine eigene Datenbankabfrage ausgewertet (siehe Abbildung 15) Der Testfall „Anzahl der Teilintervalle = 1“ entspricht der Anwendung der ersten Lösungsvariante, alle anderen Testfälle sind Anwendungen der zweiten Lösungsvariante, wobei die Anzahl der Teilintervalle variiert wird. Die Tests wurden für zwei verschiedene logische Ausdrücke durchgeführt. Es folgt eine Darstellung der für beide Testausdrücke in den beiden Varianten erhaltenen Ergebnisse. Im folgenden Kapitel werden die erhaltenen Ergebnisse diskutiert, sowie konkrete Empfehlungen für die Verwendung der einen- oder anderen Variante in unterschiedlichen Abfragesituationen gegeben.

Erster Testausdruck: Eine Bedingung

Der erste logische Ausdruck besteht aus einer einzigen Filterbedingung:

Bedingung: *HF > 80*

Diese Filterbedingung wird über einem Zeitintervall von 6 Stunden ausgewertet:

12.02.2021 06:00:00.000 +02:00 – 12.02.2021 12:00:00.000 +02:00

Die Ergebnisse sind in Tabelle 3 dargestellt. Diese Tabelle stellt der Anzahl- bzw. der zugehörigen Dauer der Teilintervalle die gemessene Ausführungsdauer des logischen Ausdrucks in der Datenbank gegenüber. Eine Diskussion der erhaltenen Ergebnisse erfolgt im folgenden Kapitel.

Anzahl der Teilintervalle	Teilintervalle [Minuten]	Ausführungsdauer[s]
180	2	39,28
72	5	15,5
36	10	8,43
24	15	6,16
12	30	3,44
6	60	2,23
3	120	1,51
2	180	1,31
1	360	1

Tabelle 3: Testergebnisse für den ersten logischen Ausdruck

Zweiter Testausdruck: Zwei UND-verknüpfte Bedingungen

Der zweite logische Ausdruck besteht aus zwei Filterbedingungen, die mit dem logischen Operator UND verknüpft wurden:

Bedingung: *HF>80 UND HF<100*

Das verwendete Zeitintervall, sowie die Parametrierung der Testfälle nach der Anzahl der Teilintervalle entsprechen der obigen Darstellung für den ersten Ausdruck mit einer einzelnen Filterbedingung. Tabelle 4 zeigt die für den zweiten Ausdruck erhaltenen Ergebnisse.

Anzahl der Teilintervalle	Teilintervalle [Minuten]	Ausführungsdauer[s]
180	2	60,25
72	5	24,6
36	10	18,72
24	15	16,2
12	30	16,16
6	60	16,08
2	120	23,9
2	180	23,95
1	360	20,02

Tabelle 4: Testergebnisse für den zweiten logischen Ausdruck

4 Diskussion

Die initialen Teilanforderungen an die zu implementierenden Lösung wurden wie folgt erreicht:

1. Den Parametern sollte eine passende physikalische Einheit zugeordnet sein, d.h. wenn der Benutzer die Parameter ändert, wird die Einheit ebenfalls automatisch geändert.

   Diese Anforderung ist erfüllt. Wenn der Benutzer über die Dropdown-Liste für die Parameterauswahl den Parameter-Wert ändert, wird die zugehörige Einheit automatisch aktualisiert. Solange der Benutzer noch keinen Parameter ausgewählt hat wird im Textfeld für die Einheit der Text „Unit“ angezeigt.

2. Der Benutzer sollte Größenvergleiche mit selbstgewählten Schwellwerten definieren können.

   Diese Anforderung ist erfüllt. Der Benutzer für jede Filterbedingung über eine Dropdown-Liste einen Vergleichsoperator wählen und in ein Eingabefeld einen Vergleichswert eintragen.

3. Der Benutzer sollte die Größenvergleiche zu logischen Bedingungen kombinieren können. Bei der Abwesenheit von Klammern sollte die UND-Verknüpfung Vorrang vor der ODER-Verknüpfung haben.

   Diese Anforderung ist erfüllt. Vom Benutzer wird nicht verlangt seine Formeln über Klammern zu strukturieren. Eine derartige Strukturierung ist in der Benutzeroberfläche implizit durch Zeilen implementiert: Alle Filterbedingungen einer Zeile werden zunächst mit dem logischen UND-Operator verknüpft. Dies wird für alle Zeilen durchgeführt. Anschließend werden die für die Zeilen erhaltenen Ergebnisse mit dem logischen ODER-Operator verknüpft.

4. Der Benutzer sollte mit Klammern arbeiten können.

   Diese Anforderung ist nicht erfüllt, die Arbeit mit Klammern ist in der entwickelten Lösung nicht notwendig. Die Entscheidung diese Anforderung nicht zu erfüllen wurde bewusst- und nach Beratung mit dem technischen Fachpersonal des Krankenhauses getroffen. Die Entscheidung wurde aufgrund der geringeren Übersichtlichkeit, des höheren Benutzeraufwandes und der höheren Komplexität der zu generierenden Datenbankabfragen bei der Arbeit mit Klammern getroffen.

5. Eingabefehler sollen bei der Eingabe erkannt werden.

   Diese Anforderung ist erfüllt. Korrekt eingegebene Bedingungen werden blau umrandet und falsch eingegebene Bedingungen werden rot umrandet. Es wurden Regeln für korrekte Eingaben definiert, diese werden dem Benutzer bei der Eingabe von Filterbedingungen angezeigt.

6. Der Benutzer soll Filterbedingungen per Drag-and-Drop definieren können.

   Diese Bedingung ist nicht erfüllt. Eine weiterführende Diskussion erfolgt weiter unten in diesem Abschnitt.

7. Zur Optimierung der Ausführungsdauer der Datenbankabfragen soll die Lösung in zwei Varianten angeboten werden, wobei Variante 1 auf der Ausführung einer einzigen SQL-Abfrage basieren soll und Variante 2 auf der kombinierten Ausführung mehrerer SQL-Abfragen.

   Diese Anforderung ist erfüllt. Die beiden unterschiedlichen Lösungsvarianten sind im Abschnitt 3.2.2 beschrieben.

8. Für beide Varianten sollen Kennwerte zur Charakterisierung des Zeit- bzw. Speicherbedarfs ermittelt werden.

   Diese Anforderung ist teilweise erfüllt. Zur Charakterisierung des Zeitbedarfs wurden mit beiden Lösungsvarianten verschiedene Versuche und Zeitmessungen durchgeführt. Die Ergebnisse sind im Abschnitt 3.2.4 dargestellt und werden weiter unten in diesem Kapitel diskutiert. Der Speicherbedarf zur Ausführung der Datenbankabfragen konnte nicht charakterisiert werden, da die zugehörigen Messungen nicht über KNIME-Knoten realisiert werden konnten, sondern im Datenbankserver hätten vorgenommen werden müssen.

Eine Drag-and-Drop basierte Benutzeroberfläche wurde nicht implementiert. Es wurde eine Skizze einer alternativen Benutzeroberfläche mit Drag-and-Drop Funktionalität analysiert und dabei wurden verschiedene Nachteile gegenüber einer Schaltflächen-basierten Lösung festgestellt:

- Es ist für den Anwender relativ aufwändig per Drag-and-Drop GUI-Elemente von einem Bereich der Benutzeroberfläche in einen anderen Bereich zu bewegen.
- Bei den von uns betrachteten Anwendungsfällen möchte der Anwender mehrere Filterbedingungen definieren und hätte damit eine größere Anzahl an GUI-Elementen zu bewegen.
- Bei durchgängiger Handhabung der Benutzeroberfläche per Drag-and-Drop hätte auch das Löschen existierender Filterbedingungen per Drag-and-Drop möglich sein müssen. Insbesondere das Löschen per Drag-and-Drop (der Nutzer würde das zu löschende GUI-Element auf ein Papierkorb-Symbol ziehen) wird von den Anwendern als ein vermeidbarer Zeitaufwand empfunden der über eine Schaltflächen-basierte Benutzeroberfläche effizienter implementiert werden konnte.

In der Diskussion mit dem Fachpersonal haben sich diese Nachteile bestätigt, sodass die Anforderung nach einer Drag-and-Drop fähigen Benutzeroberfläche fallen gelassen wurde und eine zweite Skizze ohne Drag-and-Drop entwickelt wurde.

Das Durchführen einer Vorfilterung ist mit verschiedenen Vorteilen verbunden. Als Vorfilterung werden die in den WITH-AS Ausdrücken definierten Unterabfragen bezeichnet. Eine Beispielanfrage findet sich in Code-Listing. Wie in Tabelle 1 beschrieben entstehen durch die Vorfilterung Geschwindigkeitsvorteile gegenüber den alternativen Ansätzen. Ein weiterer Vorteil der Vorfilterung

ist die erhöhte Übersichtlichkeit der SQL-Abfrage. Wie in der Beispielanfrage in Code-Listing zu sehen, ist die SQL-Abfrage aufgeteilt in mehrere Unterabfragen. Alle Unterabfragen sind klar voneinander getrennt. Die Unterabfragen werden aufeinander aufbauend ausgeführt, d.h. die Ergebnistabellen der zuerst ausgeführten Unterabfragen werden in den nachfolgenden Unterabfragen verwendet. Die innere Struktur der SQL-Abfrage wird so deutlich. In der SQL-Abfrage können Phasen erkannt- und klar voneinander abgegrenzt werden (siehe Kapitel 3.2.2). Das Konzept der Vorfilterung führt somit zu einer Verbesserung der Verständlichkeit der SQL-Abfrage. Das gleiche Verhalten (mit den gleichen Geschwindigkeitsvorteilen) hätte man möglicherweise in der SQL-Abfrage auch ohne klar abgetrennte Unterabfragen, sondern ausschließlich über die geeignete Verschachtelung mehrerer „INNER JOIN"-Ausdrücke erreichen können. Aufgrund der wesentlich schlechteren Verständlichkeit der SQL-Abfrage wurde diese Richtung nicht weiterverfolgt.

Die zweite Phase, d.h. die Überprüfung der zeitlichen Nähe, ist notwendig. In der zweiten Variante wird für jede Zeile der vom Nutzer definierten Filterbedingungen nach Kombinationen von Datenpunkten gesucht , die zum Einen die jeweiligen Filterbedingungen erfüllen und zum anderen näherungsweise zu einem gleichen Zeitpunkt aufgenommen wurden. Die näherungsweise Betrachtung ist nötig, da die exakte Gleichzeitigkeit nicht notwendigerweise erfüllt ist. Die im klinischen Einsatz verwendeten Messmethoden und Sensoren können sowohl manuell als auch automatische Messungen durchführen und weisen darüber hinaus meist unterschiedliche Messintervalle auf. Beispielsweise erfolgt die Aktualisierung der Atemfrequenz nur einmal pro Minute. Biologische Teilsysteme arbeiten in unterschiedlichen Frequenzbereichen.

Je nach Anzahl der UND-verknüpften Filterbedingungen sind unterschiedliche Lösungsvarianten vorzuziehen. Diese Entscheidung kann anhand der im Abschnitt 3.2.4 beschriebenen Ergebnisse diskutiert werden (siehe Tabelle 3 und Tabelle 4). Die erste Lösungsvariante besteht im Versuch „Anzahl der Teilintervalle = 1". Die zugehörige Ausführungsdauer geht aus der Tabelle 3 und Tabelle 4 hervor: Es ist die Ausführungsdauer jeweils in der untersten Zeile, d.h. in der Zeile mit „Anzahl der Teilintervalle = 1". Die Ergebnisse für die zweite Lösungsvariante hängen von einem Parameter ab, der Anzahl der Teilintervalle. Die übrigen Zeilen der Tabelle 3 und der Tabelle 4 stellen daher das Testergebnis für die zweite Lösungsvariante dar. Aus den Tabellen wird deutlich, dass sich die Entwicklung der Ausführungsdauern der generierten Datenbankabfragen nach der Anzahl der definierten Filterbedingungen unterscheidet. Dabei fällt auf, dass keine Lösungsvariante als die für alle Fälle grundsätzlich bessere genannt werden kann. Allerdings können die Vor- und Nachteile der jeweiligen Lösungsvarianten klar benannt werden.

Für den Fall der Definition einer einzigen Filterbedingung hat die Lösungsvariante eins einen klaren Geschwindigkeitsvorteil gegenüber der Lösungsvariante zwei. Die Aufteilung des Gesamtzeitintervalls in mehr- oder weniger viele Teilintervalle führt zu Zeitverlusten die mit der Anzahl der Teilintervalle steigen. Aus den Ergebnissen wird deutlich, dass darüber hinaus, für den betrachteten Fall der einzelnen Filterbedingung, die Aufteilung in mehrere Teilintervalle zu keinen den Zeitverlusten gegenüberstehenden Zeitgewinnen führt. Die Ausführungsdauer steigt linear mit der Anzahl der Teilintervalle. Diese Tendenz wird aus Tabelle 3 klar deutlich.

Daraus kann die Schlussfolgerung gezogen werden, dass die Lösungsvariante eins für den Fall der Definition einer einzigen Filterbedingung vorgezogen werden sollte.

Ein weniger klares Bild zeigt sich bei der Betrachtung der Ergebnisse für den Fall der Definition zweier mit dem logischen Operator UND verknüpften Filterbedingungen. Aus Tabelle 4 wird deutlich das die Ausführungsdauer für die zweite Lösungsvariante länger oder kürzer sein kann als die Ausführungsdauer für die erste Lösungsvariante, in Abhängigkeit von der Anzahl der verwendeten Teilintervalle. Die Versuche können also bezüglich der Anzahl der Teilintervalle in zwei Gruppen eingeteilt werden: Eine Gruppe verwendet eine Anzahl von Teilintervallen für welche die Ausführungsdauer geringer ist als für die erste Lösungsvariante. Die andere Gruppe verwendet eine Anzahl von Teilintervallen für welche die Ausführungsdauer höher ist als für die erste Lösungsvariante. Dabei gibt es einen Versuch der mit seiner gegebenen Anzahl von Teilintervallen als das Optimum betrachtet werden kann. Die Gruppe mit der niedrigeren Ausführungsdauer beginnt (einschließlich) bei einer Teilintervall-Anzahl von 6 und endet (einschließlich) bei einer Teilintervall-Anzahl von 36.

Diese Ergebnisse lassen die folgende Schlussfolgerung zu: Für den Fall der Definition zweier (oder mehrerer) Filterbedingungen ist die zweite Lösungsvariante insofern der ersten Lösungsvariante überlegen als das die Chance besteht eine Anzahl von Teilintervallen zu wählen, die zu einem Geschwindigkeitsgewinn bei der Ausführung der Datenbankabfrage führt. Die erzielbaren Geschwindigkeitsvorteile sind nicht unerheblich (im Beispiel eine Reduktion von 20 Sekunden auf 16 Sekunden, d.h. um 20%). Für den betrachteten Fall ist die zweite Lösungsvariante aber insofern der ersten Lösungsvariante unterlegen, als das Unsicherheit darüber besteht, wie die Anzahl der Teilintervalle tatsächlich zu wählen ist um den – grundsätzlich erzielbaren – Geschwindigkeitsgewinn tatsächlich zu erzielen. Neben der Chance auf einen nicht unerheblichen Geschwindigkeitsgewinn besteht nämlich auch das Risiko eines dramatischen Geschwindigkeitsverlustes, denn mit steigender Anzahl der Teilintervalle geht die Entwicklung der Ausführungsdauern wieder in den bereits beim ersten Fall, also bei der Definition einer einzigen Filterbedingung, beobachteten linearen Anstieg der Ausführungsdauern über.

Die unterschiedliche Entwicklung der Ausführungszeiten bei unterschiedlicher Anzahl der Filterbedingungen dürfte in der für die beiden Fälle unterschiedlichen Arbeitsweise der Datenbankabfragen begründet liegen. Für den Fall der Definition einer einzigen Filterbedingung besteht nicht die Notwendigkeit der Überprüfung der näherungsweisen gleichzeitigen Erfüllung von Filterbedingungen. Die zweite Lösungsvariante hat aber gerade bei dieser Überprüfung ihre Vorteile, denn für diese Überprüfung wäre die Ausführung eines INNER JOINs nötig, d.h. die Bildung eines Kreuzproduktes. Für die zweite Lösungsvariante stellt aber gerade die Notwendigkeit der Ausführung eines Kreuzproduktes einen hinsichtlich des Vergleichs mit der ersten Lösungsvariante vorteilhaften Faktor dar. Aufgrund des Fehlens dieser Kreuzproduktbildung im ersten Fall (eine Filterbedingung) kann die zweite Lösungsvariante ihren an sich vorhandenen Vorteil nicht zur Anwendung bringen und zeigt daher für den ersten Fall schlechtere Ergebnisse als die erste Lösungsvariante.

Diesem vorteilhaften Faktor der zweiten Lösungsvariante (der im Unterabschnitt „Mathematische Beschreibung der Zeitaufwände“ genauer diskutiert wird) steht auch ein paralleler, nachteiliger Faktor gegenüber. Dieser wird bei der Betrachtung der Ergebnisse für den Fall der Definition zweier Filterbedingungen (siehe Tabelle 4) deutlich: Die Aufspaltung des Gesamtzeitintervalls in mehrere Teilintervalle ist mit einer höheren Anzahl von Datenbankanfragen verbunden. Für jede Datenbankabfrage entsteht aber, über den Zeitaufwand für die eigentliche Auswertung jeder einzelnen Anfrage hinaus, ein unvermeidbarer Zeitaufwand für die Hinführung dieser Anfrage zu dieser Auswertung. Diese Hinführung umfasst z.B. den Zeitaufwand für den Transport der Anfrage zur Datenbank, für das Einlesen der Anfrage innerhalb des Datenbank-Management-Systems, sowie für das Reservieren der zur Auswertung benötigten Ressourcen. Dies erklärt die Entwicklung der Ausführungszeiten in Tabelle 4: Für eine relativ geringe Anzahl von Teilintervallen überwiegt der vorteilhafte Faktor. Ab einem bestimmten Punkt überwiegt aber der durch die Erhöhung der Anzahl der Teilintervalle erfahrene Geschwindigkeitsnachteil dem dadurch parallel erfahrenen Geschwindigkeitsvorteil. Dieser Punkt, d.h. diese Anzahl von Teilintervallen, ist das Optimum. Ab diesem Optimum führt die Erhöhung der Anzahl der Teilintervalle zu einem linearen Anstieg der Ausführungszeit, da die wachsende Zahl an Teilintervallen mit einem zusätzlichen Zeitverlust verbunden ist, der aber für jedes hinzugekommen Teilintervall konstant ist.

Mathematische Beschreibung der Abfragedauer

Im Folgenden erfolgt eine mathematische Analyse der Zeitaufwände für die Auswertung der Datenbankabfrage. Bei der ersten Lösungsvariante muss das Kreuzprodukt über den Daten eines relativ langen Zeitintervalls (des Gesamtzeitintervalls) gebildet werden. Bei der zweiten Lösungsvariante muss das Kreuzprodukt über den Daten eines relativ kurzen Zeitintervalls gebildet werden (des Teilintervalls). Allerdings muss dies mehrfach geschehen, nämlich einmal für jedes Teilintervall. Wenn *a* die Anzahl der aus der Vorfilterung für die erste Filterbedingung sich ergebenden Datenpunkte darstellt und *b* die Anzahl der aus der Vorfilterung für die zweite Filterbedingung sich ergebenden Datenpunkte, so müssen bei der Kreuzproduktbildung a*b Kombinationen von Datenpunkten gebildet werden. Wenn nun *k* die Anzahl der Teilintervalle darstellt, und man annimmt, dass sich die Anzahl der Datenpunkte *a* und *b* antiproportional gegenüber der Anzahl der Teilintervalle verhält, also mit steigender Anzahl kleiner wird (weil bei steigender Anzahl der Teilintervalle auch die Länge des betrachteten Zeitintervalls kleiner wird, sodass die Vorfilterung für alle Filterbedingungen eher weniger Datenpunkte liefern wird), so kann man über den Ausdruck $\frac{\mathrm{a}}{\mathrm{k}} * \frac{\mathrm{b}}{\mathrm{k}}$ die Anzahl der Kombinationen beschreiben die bei einer Kreuzproduktbildung im Rahmen der Auswertung der Datenbankabfrage für ein einzelnes Teilintervall gebildet werden und über den Ausdruck $\frac{\mathrm{a}*\mathrm{b}}{k^2} * \mathrm{t}_0$ den dafür benötigten Zeitaufwand. Der Zeitaufwand zur Auswertung für alle Kombinationen die - insgesamt - bei der Auswertung aller Teilanfragen (es müssen insgesamt k Teilanfragen ausgewertet werden) gebildet werden lässt sich dann über den folgenden Ausdruck (Formel 1) beschreiben (es wird dabei von einer Gleichverteilung der Datenpunkte auf die Teilintervalle ausgegangen):

$$\mathrm{k}\left(\frac{\mathrm{a}}{\mathrm{k}} * \frac{\mathrm{b}}{\mathrm{k}}\right) * \mathrm{t}_0 = \frac{\mathrm{a} * \mathrm{b}}{k} * \mathrm{t}_0 \quad (1)$$

Mit k: die Anzahl der Teilintervalle [arb.unit] (k>=1)

a: die Anzahl der aus der Vorfilterung für die erste Filterbedingung sich ergebenden Datenpunkte [arb.unit]

b: die Anzahl der aus der Vorfilterung für die zweite Filterbedingung sich ergebenden Datenpunkte [arb.unit]

t_0: Zeitdauer für die Auswertung der einzelnen Kombination [s]

Dieser Ausdruck geht für „*k* gegen unendlich“ gegen 0. Er verschwindet also für eine ausreichend große Anzahl von Teilintervallen. Man kann also den Zeitaufwand für die Auswertung der Kreuzprodukte (INNER JOINs) für eine große Anzahl von Zeitintervallen vernachlässigen. Zusätzlich zu dem

Zeitaufwand für die Auswertung der INNER JOINs entsteht auch noch ein fester Zeitaufwand für die Hinführung der einzelnen Teilanfrage zu ihrer Auswertung. Eine genaue Diskussion dieses Zeitaufwandes erfolgt in Kapitel 4.

Außerdem entsteht für jede Teilanfrage ein fester Zeitaufwand für die Vorfilterung. Dieser Zeitaufwand wird mit kürzer werdendem Zeitintervall nicht kleiner. Die beiden genannten Zeitaufwände seien im Folgenden im Faktor *d* zusammengefasst. Damit lässt sich der feste zusätzliche Zeitaufwand, der insgesamt bei der Auswertung aller Teilanfragen entsteht, mit dem Ausdruck k*d beschreiben. Fasst man die erhaltenen Ausdrücke zusammen erhält man einen Gesamtzeitaufwand von:

$$t = \frac{a * b}{k} * t_0 + kd \quad (2)$$

mit t: Gesamtzeitaufwand [s]

k: die Anzahl der Teilintervalle [arb. unit] (k>=1)

a: die Anzahl der aus der Vorfilterung für die erste Filterbedingung sich ergebenden Datenpunkte [arb. unit]

b: die Anzahl der aus der Vorfilterung für die zweite Filterbedingung sich ergebenden Datenpunkte [arb. unit]

d: zusätzliche Zeitaufwand für die Vorfilterung und die Hinführung der einzelnen Teilanfrage zu ihrer Auswertung [s]

t_0: Zeitdauer für die Auswertung der einzelnen Kombination [s]

Der mathematische Ausdruck in Formel 2 liefert eine qualitative mathematische Erklärung für das gemessene Verhalten: Ein steigendes *k* führt zu einer Reduktion des ersten Summanden und einem Anstieg des zweiten Summanden. Für kleine bis mittlere k übersteigt die Reduktion des ersten Summanden den Anstieg des zweiten Summanden, aber ab einem bestimmten *k* ist der erste Summand bereits so niedrig, dass eine weitere Erhöhung von *k* lediglich zu einer Reduktion führt die geringer ist als die durch die k-Erhöhung ebenfalls erhaltene Erhöhung des zweiten Summanden.

Auf der Grundlage der vorangegangenen Überlegungen wird für die Auswahl der Variante folgendes Vorgehen empfohlen: Bei der Definition einer einzigen Filterbedingung pro Zeile sollte keine Aufteilung der SQL-Gesamtanfrage in Teilanfragen durchgeführt werden und damit Variante 1 verwendet werden. Beim Vorhandensein von Zeilen mit zwei oder mehr Filterbedingungen sollte eine Aufteilung der SQL-Gesamtanfrage in Teilanfragen durchgeführt werden und damit Variante 2 verwendet werden. Insbesondere für Zeilen mit mehr als zwei Filterbedingungen erhält man leicht Ausführungszeiten in einem nicht akzeptablen Bereich. Variante 2 lässt in diesen Fällen noch die

Chance durch eine vorteilhafte Wahl der Anzahl der Teilintervalle die Ausführungszeit auf einen akzeptablen Wert zu reduzieren. Allerdings hängt die Dauer des von der einzelnen Teilanfrage zu behandelnden Zeitintervalls dabei von den in der Datenbank vorhandenen Daten und den gewählten Filterbedingungen ab. Je nach den in der Datenbank vorhandenen Messwerten und den vom Anwender definierten Filterbedingungen ergibt die Vorfilterung (Phase 1 der Auswertung der Datenbankanfrage, siehe Unterabschnitt „Entwurf der Datenbankabfrage“) mehr oder weniger viele Datenpunkte und damit eine mehr oder weniger hohe Problemgröße für die anschließende Kreuzproduktbildung (Phase 2). Die Messungen ergaben für zwei Filterbedingungen die geringste Ausführungsdauer (16,08 Sekunden) für eine Teilintervall-Dauer von 60 Minuten. Dieses Ergebnis kann nicht verallgemeinert werden. Bei der Wahl von zwei anderen Filterbedingungen liegt die optimale Teilintervall-Dauer nicht unbedingt bei 60 Minuten. Das Ergebnis deutet lediglich an, dass bei gegebenen Filterbedingungen bei der Verwendung von Variante 2 tatsächlich Geschwindigkeitsvorteile gegenüber Variante 1 möglich sind, diese aber stark von der Anzahl und der Dauer der Teilintervalle abhängt.

Es wurde auch versucht Messungen mit 3 Filterbedingungen pro Zeile durchzuführen, allerdings mussten diese Messungen aufgrund der hohen Ausführungsdauer abgebrochen werden. In diesen Fällen kann man versuchen die Filterbedingungen so anzupassen, dass sie von weniger Datenpunkten erfüllt werden. Ein anderer Ansatz wäre eine Anpassung der Datenbankanfrage, sodass diese nicht alle aus der Vorfilterung erhaltenen Datenpunkte an die Kreuzproduktbildung übergibt, sondern nur einen Teil der Daten (z.B. jeden dritten Datenpunkt).

Schließlich können die beiden im Kapitel 1.3 gestellten ingenieurtechnischen Fragestellungen wie folgt beantwortet werden: Die in der ersten Frage ausgedrückte Anforderung an die Benutzerfreundlichkeit lässt sich durch den Verzicht auf einen Teil des mathematisch-logischen Formalismus erreichen. Eine Benutzeroberfläche zur Definition von Filterbedingungen über logische Formeln sollte eine komplizierte Klammerung bei den logischen Formeln vermeiden. Zur Verbesserung der Übersichtlichkeit kann auf die Möglichkeit Klammern explizit zu setzen ganz verzichtet werden. Stattdessen können Klammern implizit gesetzt werden.

Die zweite Frage kann über die Analyse und den Vergleich der beiden entwickelten Abfragevarianten beantwortet werden. Falls durch den Anwender höchstens eine einzige Filterbedingung pro Zeile definiert wird ist Variante 1 einzusetzen, falls der Anwender in mindestens einer Zeile zwei oder mehr Filterbedingungen definiert hat ist Variante 2 einzusetzen. Dabei ist zu berücksichtigen, dass keine Empfehlung für die zu verwendende Dauer der Teilintervalle gegeben werden kann, da die optimale Dauer von der Wahl der Filterbedingungen und den in der Datenbank vorhandenen Messwerten abhängt.

5 Schlussfolgerung und Ausblick

In dieser Arbeit wurde eine funktionale Erweiterung für OCLIDA beschrieben, einer Schnittstelle für den webbasierten Zugriff auf Patienten-Monitoring-Daten. Die Erweiterung umfasst Funktionalität zur Definition und Auswertung von Bedingungen für den gefilterten Abruf der Monitoring-Daten. Die Definition der Filterbedingungen erfolgt durch eine grafische Benutzeroberfläche, die für eine Bedienung durch medizinisches Fachpersonal entworfen wurde. Für die Auswertung der definierten Filterbedingungen werden Datenbankabfragen in der Abfragesprache SQL generiert. In der Arbeit wurden zwei mögliche Varianten der Datenbankabfragen hinsichtlich ihrer Ausführungs-geschwindigkeit analysiert und verglichen.

Abschließend soll ein Ausblick auf mögliche Weiterentwicklungen gegeben werden. Zur Reduktion der Ausführungsdauern könnte man die Datenbankanfrage so überarbeiten, dass diese bei zwei oder mehr Filterbedingungen nicht alle aus der Vorfilterung erhaltenen Datenpunkte an die anschließende Kreuzproduktbildung übergibt, sondern nur einen Teil der Daten (z.B. jeden dritten Datenpunkt).
Weiterhin sollte die technische Möglichkeit geschaffen werden die Eigenschaften der generierten Datenbankabfrage(n) an die Eigenschaften der vom Nutzer definierten Filterbedingungen anzupassen.
Schließlich sollte in den Algorithmus zur Generierung der Datenbankabfragen eine Optimierung für Zeilen eingebaut werden, in denen der Benutzer zwei Filterbedingungen für ein- und denselben Parameter definiert hat, in denen er also prüfen will, ob die Patienten-Monitoring-Daten für diesen Parameter in einem bestimmten Wertebereich liegen. Die empfohlene Optimierung besteht darin für beide Filterbedingungen zum einen nur eine einzige Unterabfrage zur Vorfilterung zu definieren. Diese könnte prüfen, ob die gefundenen Parameterwerte in dem definierten Wertebereich liegen. Zum anderen kann dann in der SQL-Abfrage die Überprüfung der gleichzeitigen Erfüllung der beiden Filterbedingungen entfallen, da beide Filterbedingungen ein- und denselben Parameter betreffen.

6 Literaturverzeichnis

[1]. Haisong Wang. Zugriffsorganisation für klinisch erfasste Monitoring-Daten in einem Data Warehouse mit strukturiertem Datenexport zur Bildsynchronisation in der Neurochirurgie. [Diplomarbeit]. 23. Ausgust 2019.

[2]. MediClin Reha-Zentrum Spreewald. Bluthochdruck ein Ratgeber für Betroffen. [pdf]. Januar 2021. https://www.mediclin.de/fileadmin/02_Dokumente_Share_verzeichnis/01_Klinikuebergreifende_Dokumente/Gruene_Reihe/Bluthochdruck.pdf [Zugriff am 15. Mai 2021].

[3]. Selma Aydemir. Monitoring (Intensivüberwachung). 12. September 2015. https://www.medpertise.de/monitoring-intensivueberwachung/ [Zugriff am 15. Mai 2021].

[4]. Koninklijke Philips. Connect care for early intervention. November 2015. [pdf] https://www.documents.philips.com/assets/20170523/4584e98c784e4f8ea2b8a77c014853fa.pdf [Zugriff am 15. Mai 2021].

[5]. KNIME AG, Zurich, Switzerland. KNIME Quickstart Guide. 07. September 2020. [pdf] https://docs.knime.com/202012/analytics_platform_quickstart_guide/analytics_platform_quickstart_guide.pdf [Zugriff am 15. Mai 2021]

Anhang

Requirements for Customizable Filter Generator

Graphical User Interface

The graphical user interface (GUI) is to be designed to run with-in a web browser as served by the KNIME Server. The GUI is supposed to allow most flexible drag and drop generation of customized filters (Figure 1).

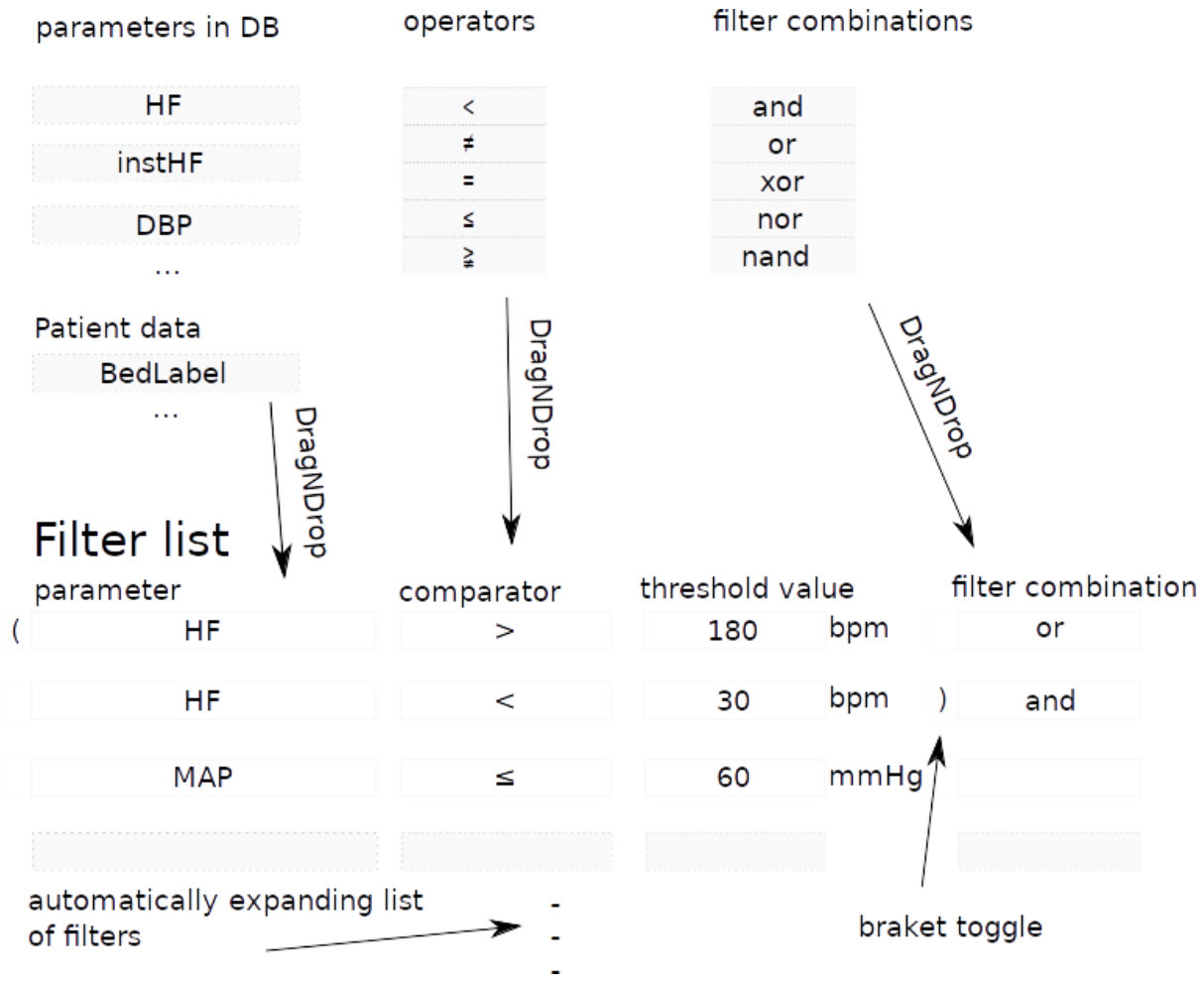


Figure 1: Example filter for all tachycardic or bradycardic and hypotensive episodes.

Notes

- The unit behind each threshold value shall be not editable but shall be set automatically according to the specified parameter.

- The threshold values shall be filled by the user.
- ORs and ANDs shall be considered as + and · operators and thus the later takes preferences above the earlier in the (algebraic) order of operation.
- To allow for a preference of OR on the above call, extend the GUI for open and closed bracket selection fields.

Data base queries

Since the data queried in the filter as described above will be distributed over multiple tables, an appropriate method of data base query shall be used minimizing both the time and the memory consumption of the query:

- Give performance values for a set of self-defined combined queries involving all available tables in the data base.
- Set-up an all-in-one SQL query (combined query solution)
- Compare this combined query solution with a standard sequential query (one query per table).