Anwenderhandbuch i-views 6.1

Inhalt

1. Knowledge-Builder

1.1. Grundlagen

Mit i-views funktionieren Datenbanken so, wie Menschen denken: einfach, agil, flexibel. Deswegen ist in i-views einiges anders als bei relationalen Datenbanken: Wir arbeiten nicht mit Tabellen und Keys, sondern mit Objekten und Beziehungen zwischen ihnen. Die Modellierung der Daten ist visuell und beispielorientiert, sodass wir sie auch mit den Nutzern aus den Fachabteilungen teilen können.

Wir bauen mit i-views keinen reinen Datenspeicher, sondern intelligente Datennetze, die bereits viel Business-Logik enthalten und mit denen das Verhalten unserer Anwendung schon weitgehend bestimmt werden kann. Dazu nutzen wir Vererbung, Mechanismen zum Schlussfolgern und zur Definition von Sichten sowie eine Vielzahl von Suchverfahren, die i-views bietet.

Unser zentrales Werkzeug dazu ist der Knowledge-Builder, eine der Kernkomponenten von i-views. Mit dem Knowledge-Builder können wir:

- Schemata definieren, aber auch Beispiele aufbauen und vor allem visualisieren
- Importe und Abbildungen einer Datenquelle definieren
- Abfragen formulieren, vernetzte Daten traversieren, Strings verarbeiten und Nähen berechnen
- Rechte, Trigger und Sichten definieren

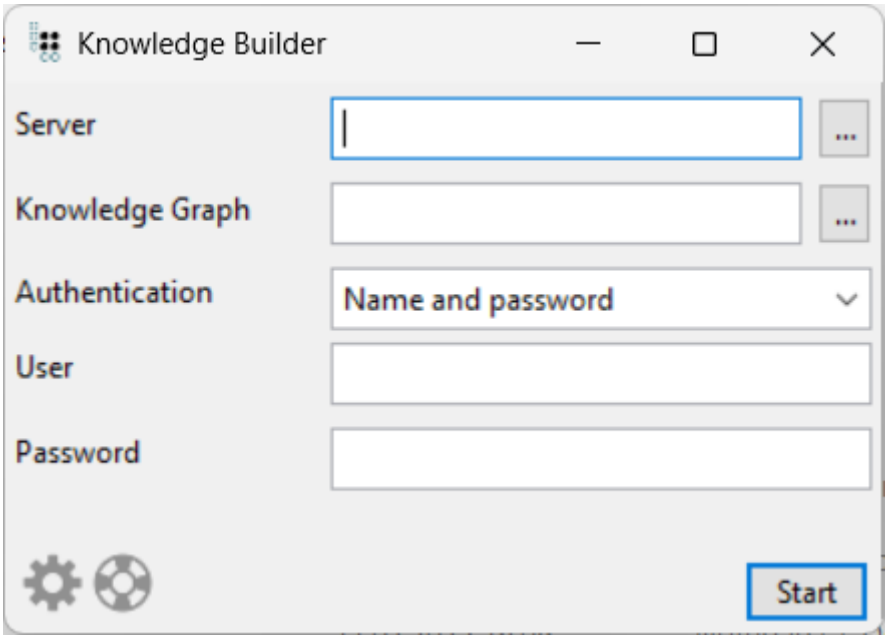
Alle diese Funktionen sind Gegenstand dieser Dokumentation. Als durchgehendes Beispiel dient dabei ein Knowledge Graph rund um Musik, Bands, Songs etc.

1.1.1. Die Knowledge-Builder Anwendung

Der Begriff "kb" ist ein Akronym für den i-views "Knowledge-Builder", mit dessen Hilfe wir den Knowledge Graphen verwalten. Beim Umgang mit dem Knowledge-Builder wird von diversen Fachbegriffen Gebrauch gemacht, um die Orientierung und das gemeinsamen Verständnis zu erleichtern:

- **Backend:** Die Knowledge-Builder (KB) Anwendung selbst.
- **Frontend:** Das Web-Frontend, welches in einem Browsers mithilfe des Viewconfiguration-Mappers (VCM) dargestellt wird
- **Volume:** Das Volume umfasst alle Daten des Knowledge Graphen, auf welchen mithilfe des Knowledge-Builders zugegriffen werden kann.
- **Semantisches Element:** Ein semantisches Element ist entweder ein Typ oder die Instanz eines Typs. Hierzu zählen Objekttypen und deren Objekte, Attributtypen und deren Attributinstanzen sowie Relationstypen und Relationen.

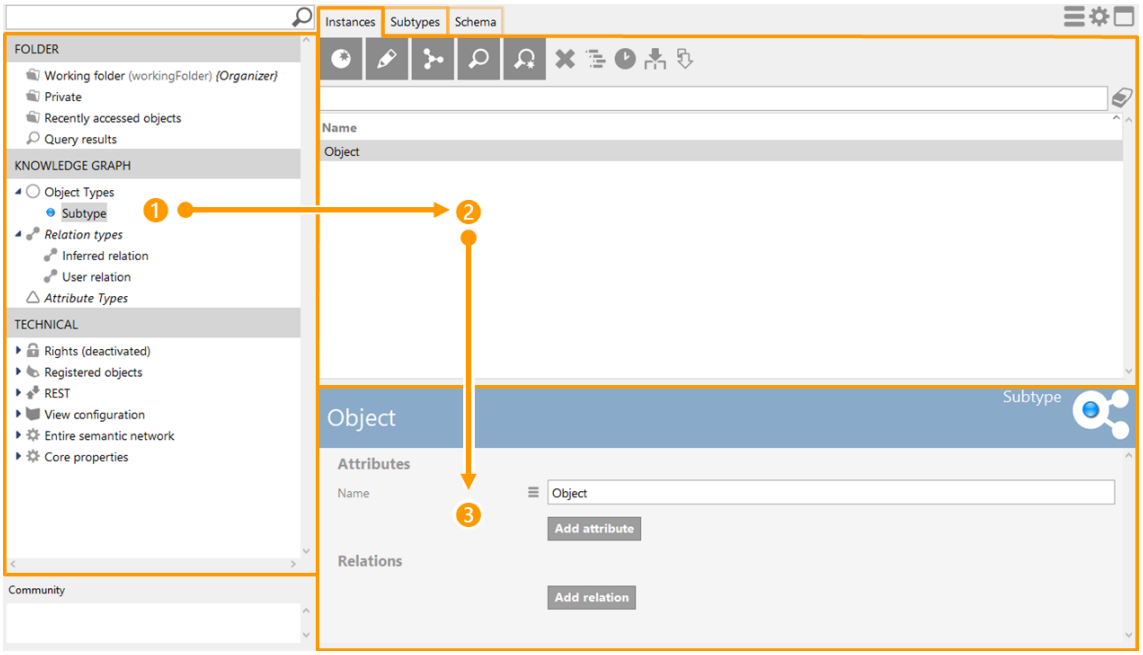
Wenn wir den Knowledge-Builder starten, wird der Login-Dialog angezeigt:



- **Server**: Für den Server gibt es drei Arten des Zugriffs:
  - **Ohne Server** (leeres Feld): Zugriff auf das Volume des Knowledge Graphen über das lokale Dateisystem. In diesem Fall muss sich das Volume in einem "volumes"-Ordner befinden, welcher sich wiederum im selben Verzeichnis befindet wie die Knowledge-Builder Anwendung selbst. Weil hierdurch der Zugriff ohne Mediator erfolgt, kann nur eine Client-Anwendung auf einmal Zugriff auf das Volume erhalten — beispielsweise der Knowledge-Builder *oder* die Bridge für einen Web-Frontend Zugang.
  - **localhost** : Diese Option bietet den Zugriff auf das Volume über einen Mediator, welcher sich in der Regel im selben Verzeichnis befindet wie das Volumes-Verzeichnis und der Knowledge-Builder. Der Mediator ist eine zusätzliche Anwendung, die den simultanen Zugriff auf das Volume durch mehrere Client-Anwendungen vermittelt — beispielsweise der Knowledge-Builder *und* eine Bridge für den Web-Frontend Zugang.
  - **Server-Adresse und Server-Port** : Weil der Knowledge-Builder bevorzugterweise eine unter vielen Client-Anwendungen ist, welche mithilfe eines Mediators den kollaborativen Fernzugriff auf den Knowledge Graphen des zentral per Server bereitgestellten Volumes gestattet, ist diese Art der Verbindung die gebräuchlichste. Server-Adresse und Portnummer werden durch einen Doppelpunkt getrennt angegeben in der Form *serveradresse:portnummer*.
- **Knowledge Graph:** Hier wird der Name des Knowledge Graphs angegeben.
- **Benutzer:** Benutzername für den Backend-Zugang auf das Volume.
- **Passwort:** Passwort für den Backend-Zugang auf das Volume.

HINWEIS	Für das Erstellen eines neuen Volumes wird das Admin-Tool benötigt.

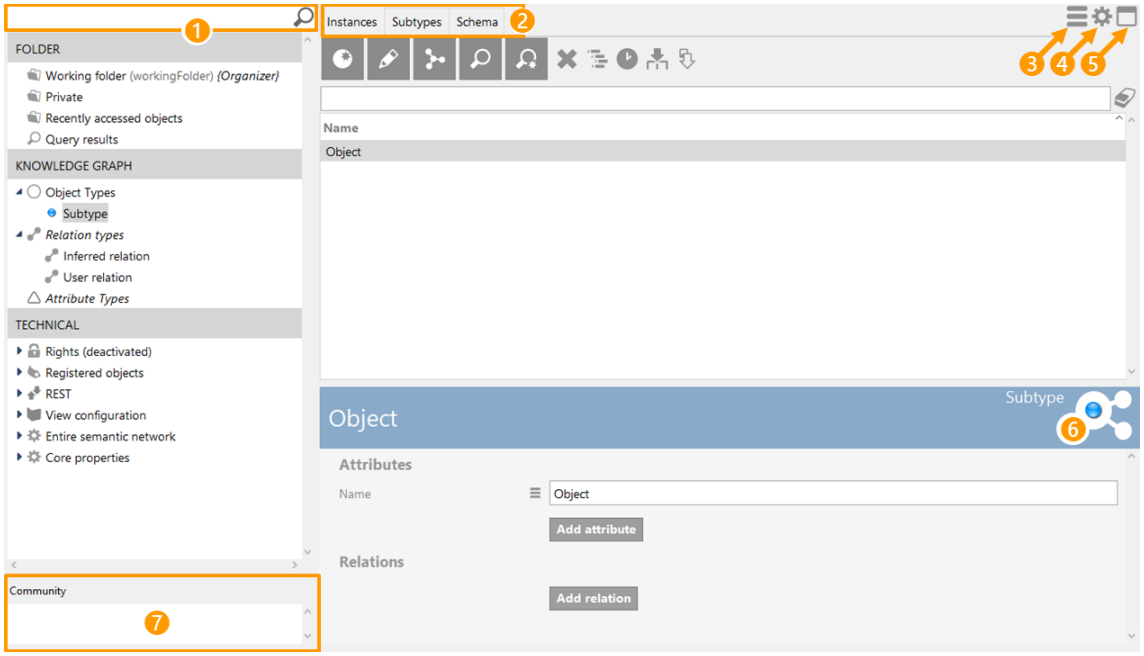
Die Anwenderoberfläche des Knowledge-Builder ist in folgende funktionelle Bereiche aufgeteilt:



- ❶ **Organizer:** Hierarchische Typenansicht auf der linken Seite der Knowledge-Builder Oberfläche.
- ❷ **Listenansicht/Objektliste:** Oberer, rechter Teil des Knowledge-Builders, welcher die Instanzen des jeweiligen Typs anzeigt, der im Organizer ausgewählt wurde. Die Listenansicht im Knowledge-Builder besteht ausschließlich aus Tabellen-Ansichten. Wenn mehrere Listenansichten für einen Typ definiert wurden, dann werden die Tabellen in unterschiedlichen Reitern dargestellt.
- ❸ **Detaileditor/Detailansicht:** Unterer, rechter Teil des Knowledge-Builders, in welchem die Detailansicht zu einer aus der Listenansicht gewählten Instanz angezeigt wird. Die Detailansicht kann aus unterschiedlichen Ansichten ("Views") zusammengestellt werden.

Aufgrund dieses Aufbaus gehen wir zur Bearbeitung von Eigenschaften eines semantischen Elements wir folgt vor: Auswahl des Typs im Organizer ❶, Auswahl des Untertyps oder der Instanz in der Listenansicht ❷ und anschließendes Bearbeiten der Eigenschaften des gewählten Elements im Detaileditor ❸.

Neben den funktionalen Bereichen des Knowledge-Builders stehen weitere Aktionen und Auswahlmöglichkeiten zur Verfügung:



- ❶ **Globale Suche:** Die globale Suche erlaubt Zugriff auf alle Elemente des Knowledge Graphen. Weitere Suchen können durch Drag&Drop von einem Ordner in die Sucheingabzeile hinzugefügt werden.
- ❷ **Listen-Reiter:** Die Listen-Ansichten sind nach Untertypen (Objekttyp, Attributtyp oder Relationstyp) und Instanzen (Objekte, Attribute oder Relationen) getrennt aufgeführt. Mit Version 5.4 ist zusätzlich ein Schema-Reiter mit einem separaten Detaileditor vorhanden, welcher ausschließlich für die Schemadefinition von Eigenschaften oder Eigenschaftstypen des ausgewählten Untertyps zur Verfügung steht.
- ❸ **Globale Aktionen:** Das Hauptmenü des Knowledge-Builders stellt elementunabhängige Aktionen zur Verfügung. Für weitergehende Informationen siehe entpsrechendes Kapitel zu Beginn des i-views Knowledge-Builder Technical Handbook.
- ❹ **Globale Einstellungen:** Die globalen Einstellungen enthalten die benutzerspezifischen Einstellungen und die administrativen Einstellungen, welche nur für Anwender mit Administrator-Status zur Verfügung stehen. Für weitergehende Informationen siehe entpsrechendes Kapitel zu Beginn des i-views Knowledge-Builder Technical Handbook.
- ❺ **Neues Fenster:** Dieser Button ermöglicht es, die angezeigte Listenansicht oder bspw. ein Importmapping in einem separaten Fenster zu öffnen. Dies hat zu Vorteil, dass die Ansicht ungeachtet der Auswahl im Organizer erhalten bleibt.
- ❻ **Kontextmenü:** Das Kontextmenü enthält alle elementspezifischen Aktionen. Bei Klick auf den großen Kreis wird das Kontextmenü geöffnet; ein Klick auf einen der kleineren Kreise öffnet das semantische Element in einem Graph-Editor. Der große Kreis dient darüber hinaus als Anfasser, um das semantische Element per Drag&Drop in einem bereits geöffneten Graph-Editor oder einem Ordner-Element hinzuzufügen.
- ❼ **Community:** Alle momentan im Backend eingeloggten Benutzer sind hier aufgelistet und können für kollaborative Arbeiten per Chat-Funktion kontaktiert werden.

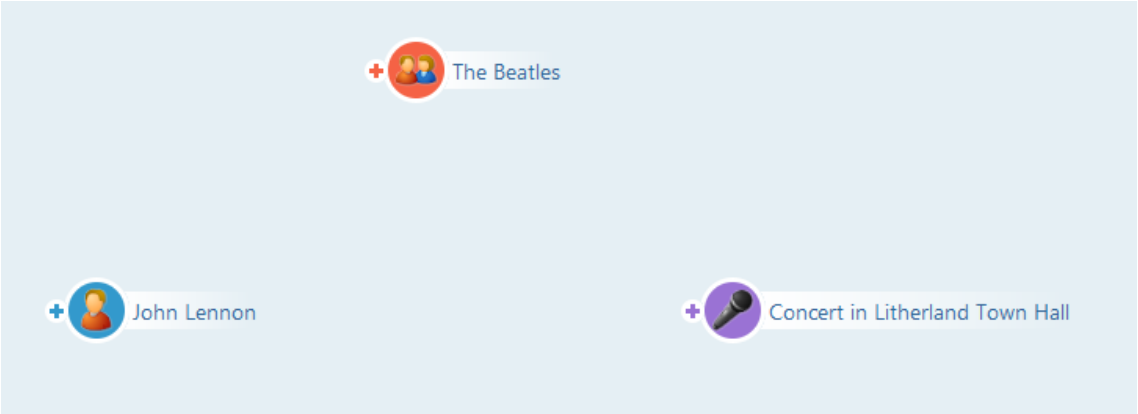
Für weitergehende Informationen siehe folgende Kapitel.

1.1.2. Grundbausteine

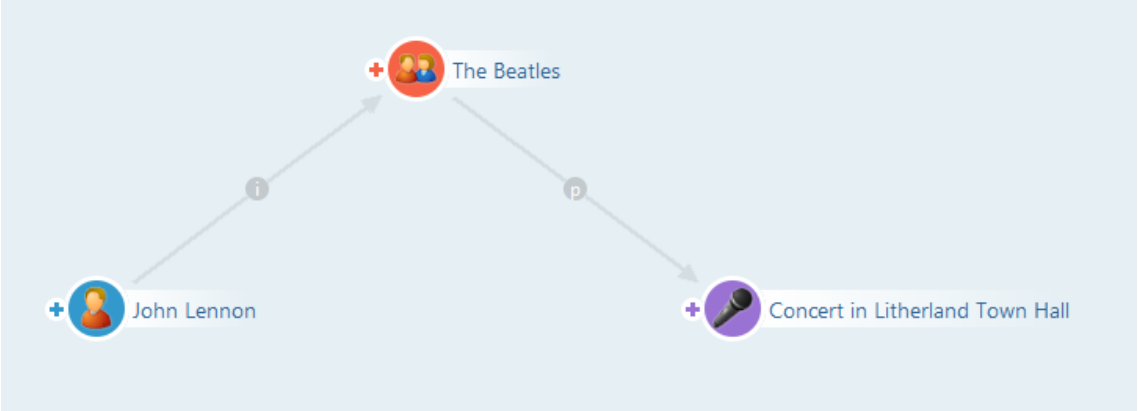
Die Grundbausteine der Modellierung in i-views sind:

- konkrete Objekte
- Beziehungen
- Attribute
- Objekttypen
- Beziehungstypen
- Attributtypen

Konkrete Objekte: Beispiele für konkrete Objekte sind John Lennon, die Beatles, Liverpool, das Konzert in der Litherland Town Hall, die Fußball-WM 1970 in Mexiko, der Schiefe Turm von Pisa etc.:



Beziehungen werden in i-views als Relationen bezeichnet. Diese konkreten Objekte können wir durch Beziehungen miteinander verbinden: "John Lennon ist Mitglied der Beatles", "Die Beatles treten auf bei einem Konzert in der Litherland Town Hall".

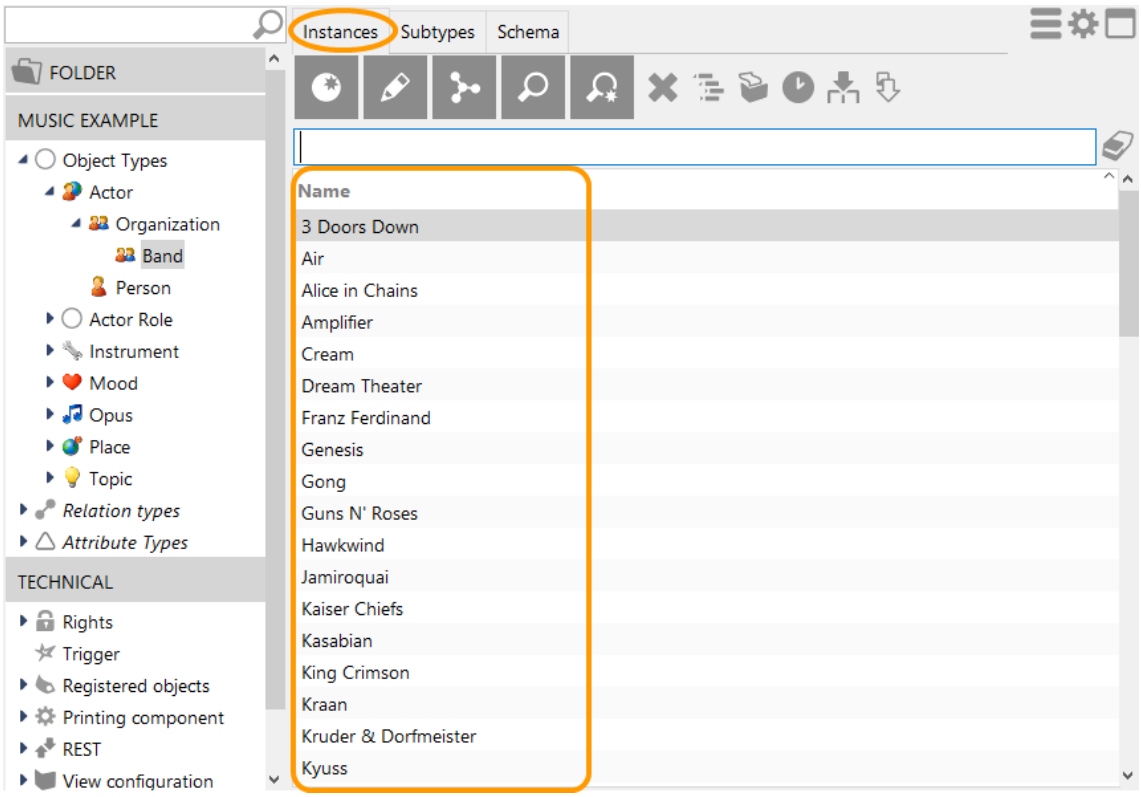


Attribute: Konkrete Objekte können Attribute haben. Konkrete Attribute von John Lennon sind

beispielsweise sein Vorname "John", sein Nachname "Lennon" und sein Geburtsdatum, der 9. Oktober 1940.

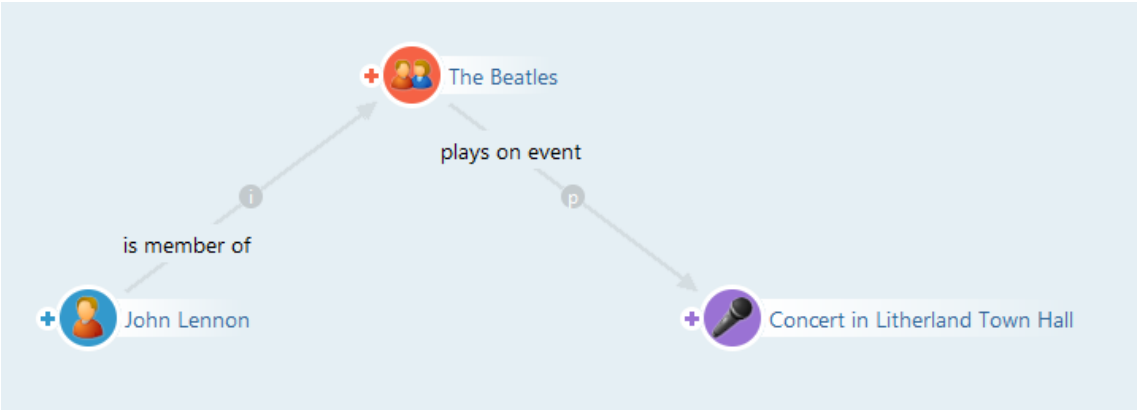
Objekttypen: Nebenbei haben wir hier Objekttypen eingeführt. Konkrete Objekte haben immer einen Objekttyp, z.B. den Typ der Personen, den der Städte, der Veranstaltungen oder der Bands — Objekttypen, die Sie in Ihrem Datenmodell frei definieren können.

Ein konkretes Objekt ist dann einem Objekttyp zugehörig, wenn man eine "ist ein"-Beziehung ziehen kann. Die "ist ein"-Beziehung ist gleichbedeutend mit "hat Typ"-Beziehung; in i-views heißt die Beziehung standardmäßig "ist konkretes Objekt von". Beispiele: John Lennon (konkretes Objekt) "ist eine" Person (Objekttyp), The Beatles (konkretes Objekt) "ist eine" Band (Objekttyp) und Konzert in Litherland Town Hall (konkretes Objekt) "ist ein" Konzert (Objekttyp).



Das Hauptfenster von i-views: Links die Objekttypen, rechts die dazugehörigen konkreten Objekte — Hier sehen wir auch: Die Typen der i-views-Netze stehen in einer Hierarchie. Mehr zur Typenhierarchie erfahren Sie im nächsten Abschnitt Typenhierarchie — Vererbung.

Beziehungstypen (auch Relationstypen genannt): Auch die Beziehungen haben unterschiedliche Typen. Zwischen John Lennon und den Beatles gibt es die Beziehung "ist Mitglied von"; zwischen den Beatles und ihrem Konzert kann die Beziehung "treten auf" heißen — wenn wir etwas mehr verallgemeinern wollen, ist vielleicht "nimmt Teil an" ein sinnvoller Relationstyp.

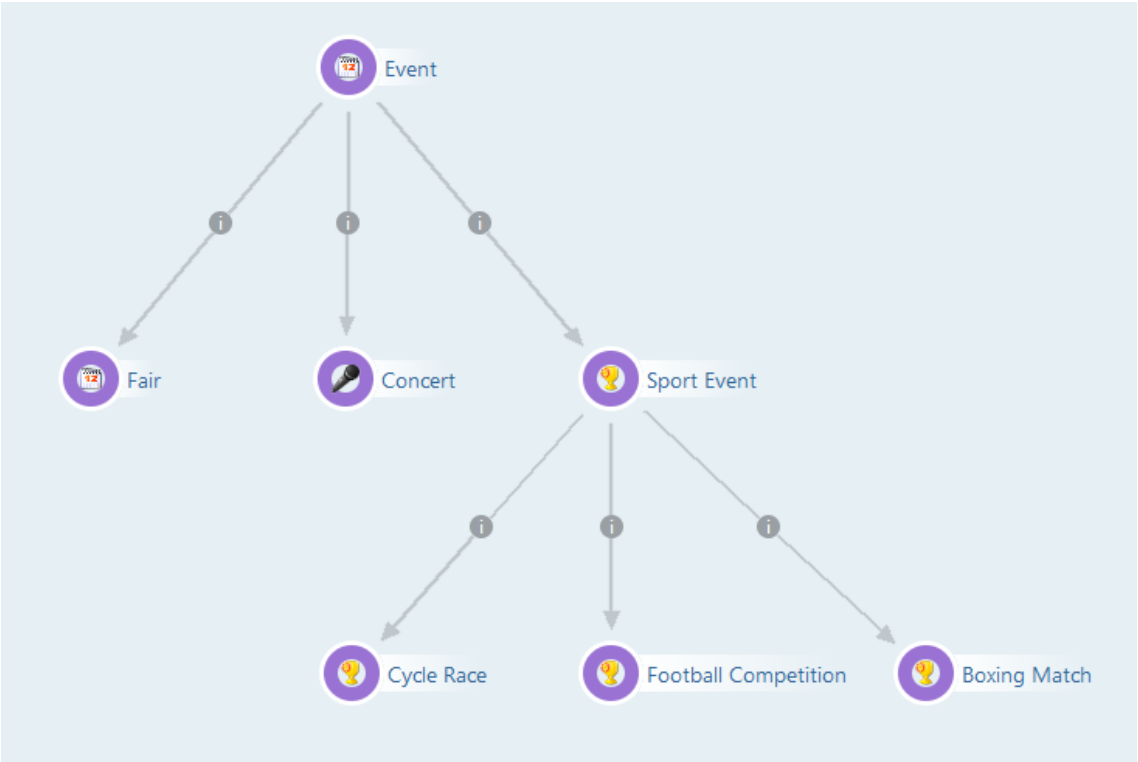


Attributtypen: Neben konkreten Objekten und Beziehungen können auch Attribute Typen haben. Im Fall einer Person können dies der Name oder das Geburtsdatum sein. Konkrete Personen (Objekte des Typs "Person") können dann Namen, Geburtsdatum, Geburts- und Wohnorte, Augenfarbe etc. als Attributtypen haben. Veranstaltungen können einen Ort und eine Zeitspanne haben. Attribute und Relationen werden immer beim Objekt selbst definiert.

1.1.3. Typhierarchie — Vererbung

Objekttypen können wir feiner oder weniger fein einteilen: Wir können die Fußball-WM 1970 mit allen anderen Veranstaltungen (die Buchmesse 2015, das Woodstock-Festival…) in einen Topf werfen, dann haben wir nur einen Typ namens "Veranstaltung" oder wir unterscheiden zwischen Sportveranstaltungen, Messen, Ausstellungen, Musikveranstaltungen etc. Alle diese Typen von Veranstaltungen können wir natürlich auch noch feiner unterteilen: Sportveranstaltungen können z.B. nach Sportarten unterschieden werden (ein Fußballspiel, ein Basketballspiel, ein Fahrradrennen, ein Boxkampf).

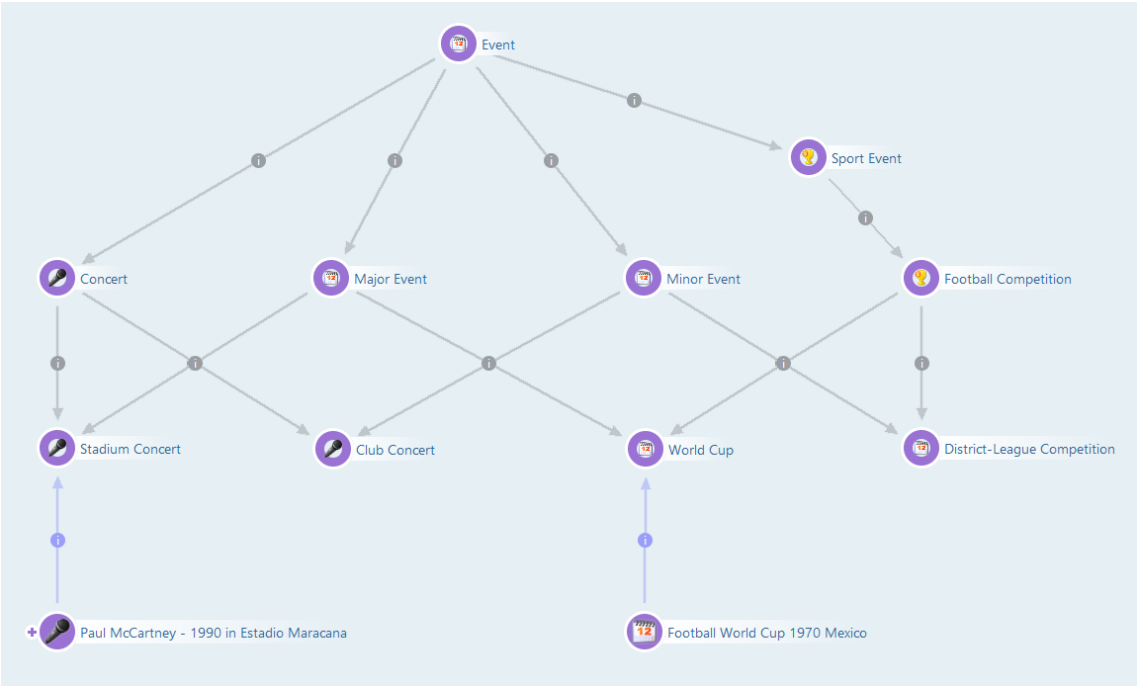
So bekommen wir eine Hierarchie von Ober- und Untertypen:



Die Hierarchie ist transitiv: wenn wir i-views nach allen Veranstaltungen fragen, werden nicht nur alle konkreten Objekte angezeigt, die direkt am Objekttyp "Veranstaltung" hängen, sondern auch alle Sportveranstaltungen und alle Radrennen, Boxkämpfe und Fußballspiele. Da der Typ "Boxkampf" also nicht nur ein Untertyp von "Sportveranstaltung" sondern damit auch ein Untertyp von "Veranstaltung" ist, wird i-views eine direkte Ober-/Untertyp-Relation zwischen Veranstaltung und Boxkampf ablehnen — mit dem Hinweis, dass dieser Zusammenhang bereits bekannt ist.

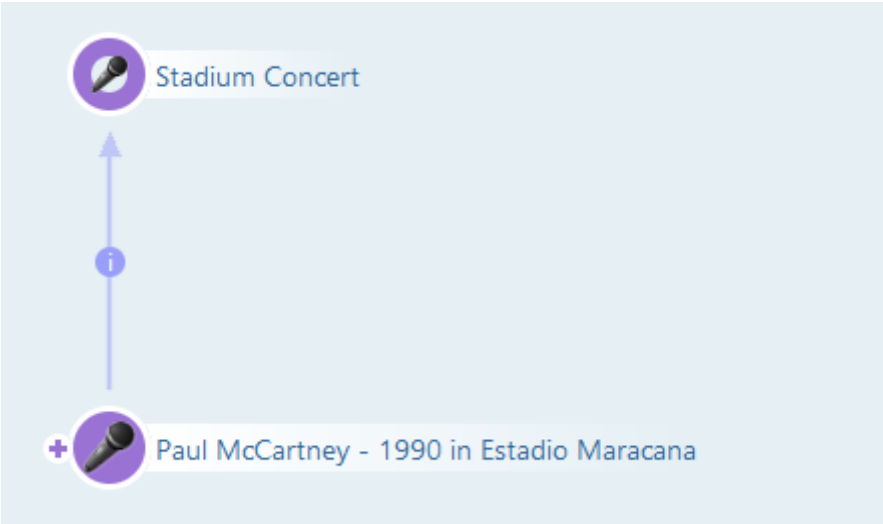
Die hierarchische Struktur muss nicht zwangsläufig die Struktur eines Baumes haben, in der jeder Typ nur genau einen Obertyp haben kann. In einem Knowledge Graph kann ein Objekttyp mehrere Obertypen haben. Ein konkretes Objekt hingegen kann aber nur genau einen Objekttyp haben.

Am Beispiel des konkreten Konzerts von Paul McCartney 1990, das sowohl ein Konzert, als auch eine Großveranstaltung ist, kann man sehen, was das bedeutet. Da das konkrete Konzert nicht zwei Objekttypen haben kann, wird ein neuer Objekttyp benötigt, der die Aspekte Konzert und Großveranstaltung zusammenführt, hier "Stadionkonzert":

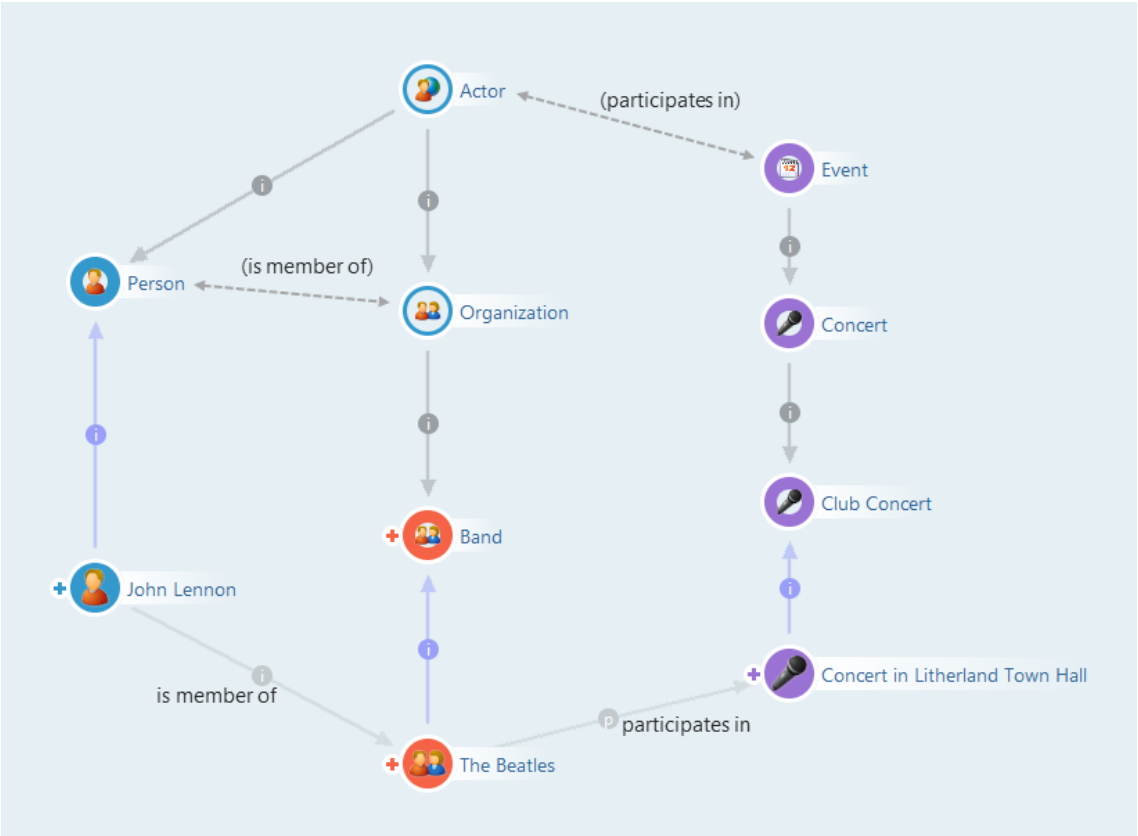


Typenhierarchie mit Mehrfachvererbung

Die Zugehörigkeit von konkreten Objekten zu einem Objekttyp wird in i-views ebenfalls als Relation ausgedrückt und kann als solche abgefragt werden:



Wann unterscheiden wir überhaupt Typen? Typen unterscheiden sich nicht nur ggf. in Icon und Farbe — bei den Objekttypen werden auch die Eigenschaften definiert und nach Typen kann bei Abfragen ganz einfach gefiltert werden. In allen diesen Fragen spielt die Vererbung eine wichtige Rolle: Eigenschaften vererben sich, auch Eigenschaften, welche die Darstellung im Knowledge-Builder beeinflussen wie Icons und Farben vererben sich. Und wenn wir in Abfragen sagen, dass wir Veranstaltungen sehen wollen, dann werden auch alle Objekte der Untertypen als Ergebnis angezeigt.

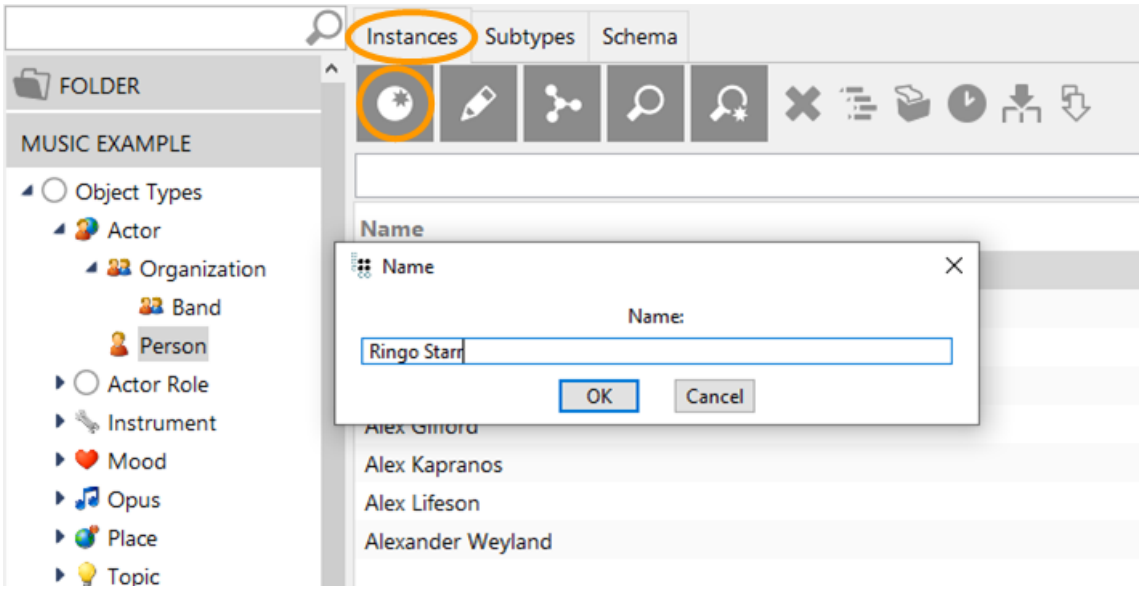


Vererbung macht es möglich, Beziehungstypen (und Attributtypen) weiter oben in der Objekttypen-Hierarchie zu definieren und damit für verschiedene Typen von Objekten (z.B. für Bands und andere Organisationen) zu nutzen.

1.1.4. Objekte anlegen und bearbeiten

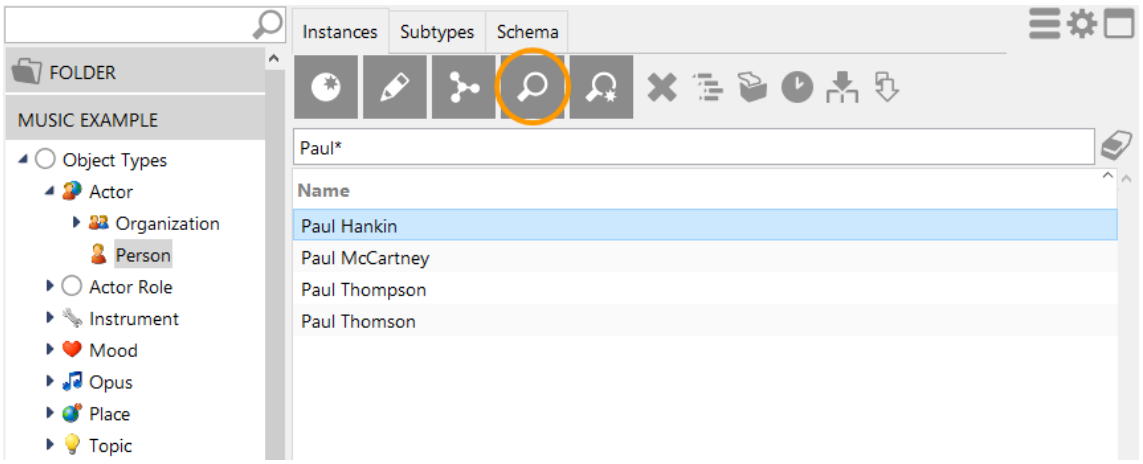
Anlegen von konkreten Objekten

Konkrete Objekte lassen sich im Knowledge-Builder überall dort anlegen, wo Objekttypen zu sehen sind. Ausgehend von den Objekttypen lassen sich über Kontextmenüs die Objekte neu anlegen.



Über die Schaltfläche "Neu" kann ein neues Objekt angelegt werden. Lediglich der Name des Objektes muss zunächst angegeben werden.

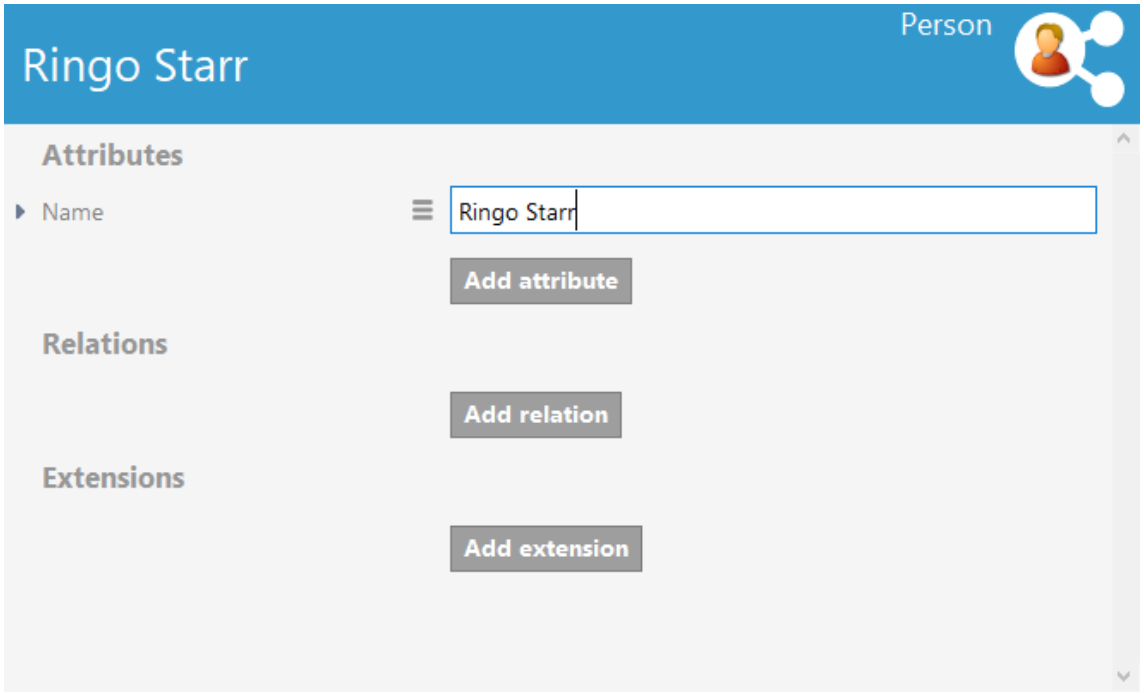
Im Hauptfenster befindet sich unter der Kopfzeile die Liste mit bereits vorhandenen konkreten Objekten. Damit Objekte nicht versehentlich doppelt angelegt werden, lässt sich über das Suchfeld in der Kopfzeile der Name des Objektes suchen. Die Suche unterscheidet per Default nicht zwischen Groß- und Kleinschreibung und der Suchbegriff lässt sich links und rechts abschneiden (durch Platzhalter "*" und "?" ergänzen):



Die Suche nach "Paul" zeigt uns, dass es bereits 4 Personen mit dem Namen "Paul" gibt.*

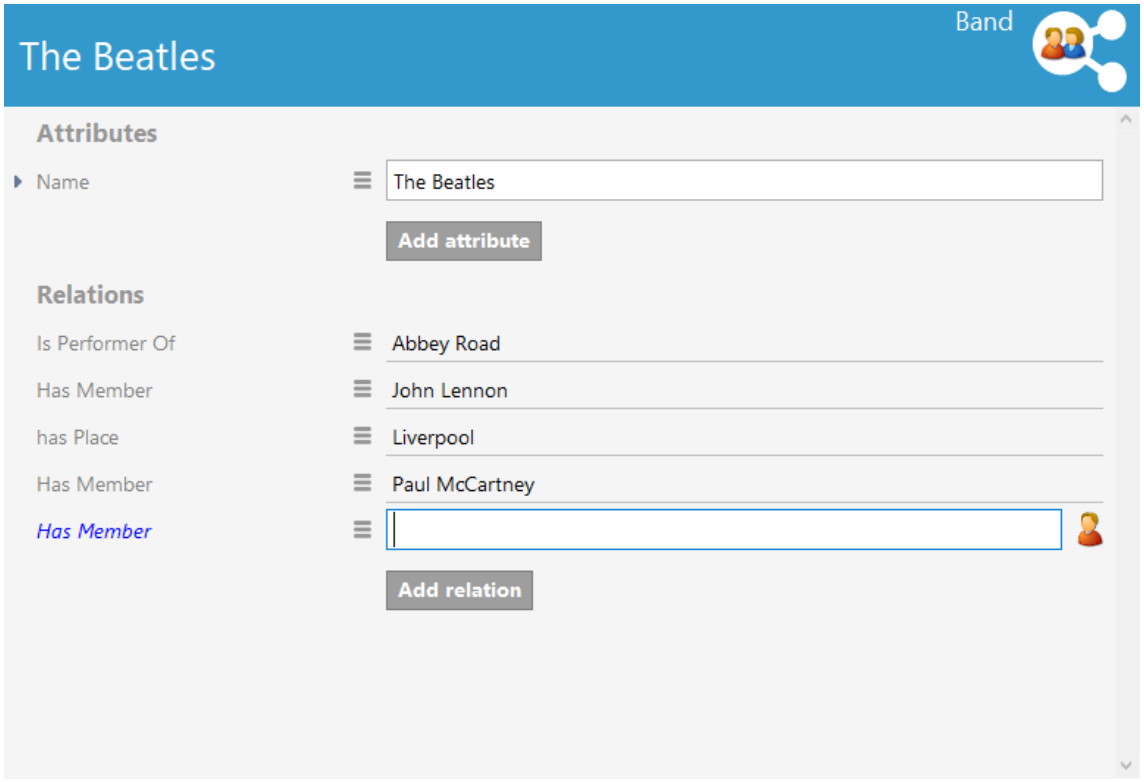
Bearbeiten von Objekten

Nach Eingabe und Bestätigung des Objektnamens können im Editor weitere Details für das angelegte Objekt eingeben werden. Dem Objekt lassen sich über Schaltflächen Attribute, Relationen und Erweiterungen zuweisen.

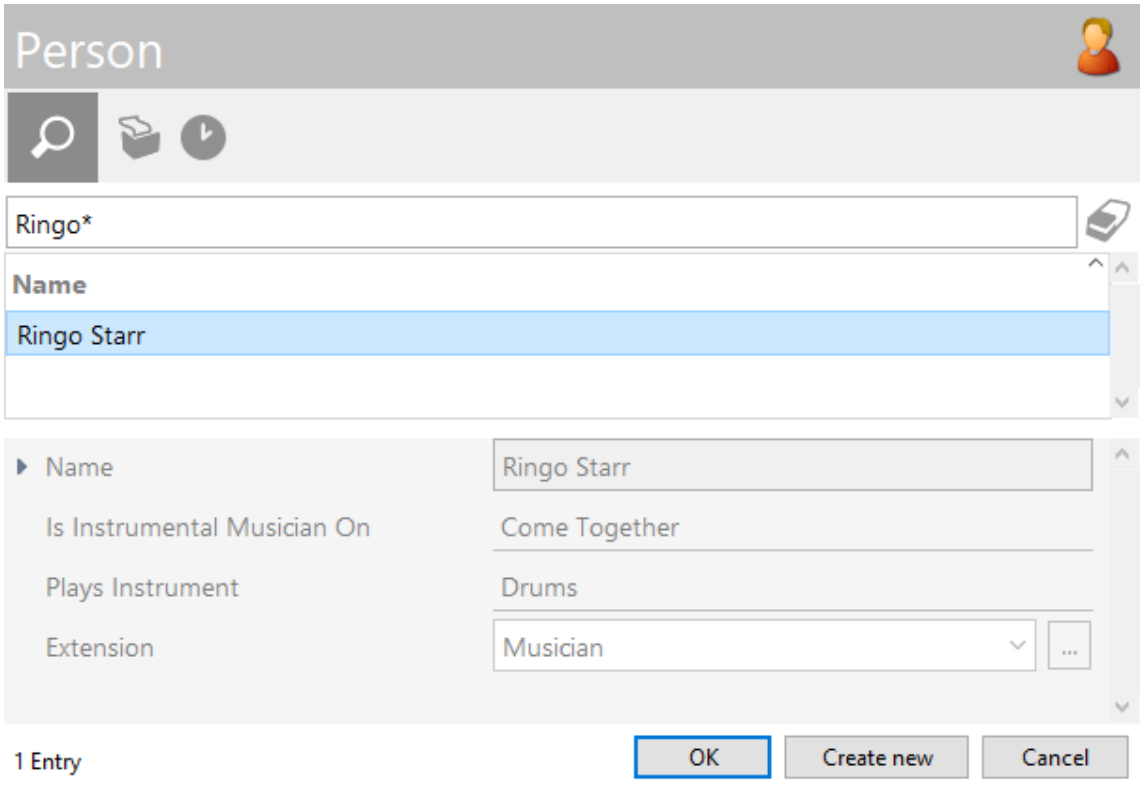


Bei der Bearbeitung eines Objekts können wir neben der Verknüpfung mit einem anderen Objekt gleichzeitig auch das Ziel der Verknüpfung neu erzeugen, sofern es noch nicht existiert.

Beispielsweise sollen Mitglieder einer Musikgruppe vollständig erfasst werden. Über die Relation *hat Mitglied* soll das Zielobjekt Ringo Starr mit dem Objekt "The Beatles" verknüpft werden. Falls noch nicht bekannt ist, ob das Objekt Ringo Starr schon in i-views erfasst ist, kann im Eingabefeld gesucht werden. Gibt man also "Ringo Starr" ein und bestätigt die Eingabe, und es ist noch kein Objekt mit diesem Namen vorhanden, so öffnet sich ein Dialog, der fragt, ob man ein neues Objekt mit diesem Namen anlegen möchte. Sollte es mehrere Ringo Starrs geben, öffnet sich ein Auswahlfenster.



Man hat zudem die Möglichkeit über die Schaltfläche *Relationsziel wählen* aus einer durchsuchbaren Liste mit allen möglichen Relationszielen ein Objekt auszuwählen.



Das Löschen der Relation *hat Mitglied* kann auf zwei Arten vorgenommen werden:

1. Im Kontextmenü unter der Schaltfläche *Weitere Aktionen* mit der Option *Löschen*.
2. Mit dem Cursor über Schaltfläche *Weitere Aktionen* bei gedrückter Strg-Taste.

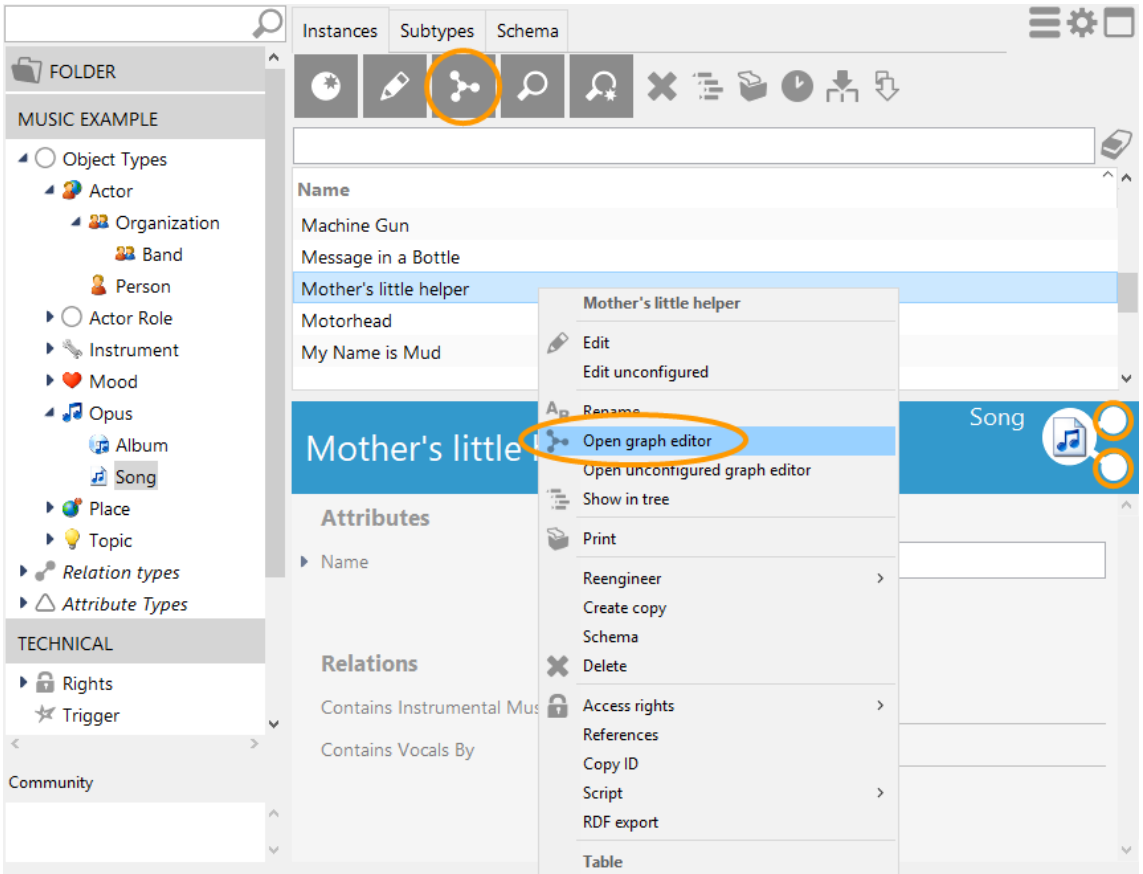
Das Zielobjekt der Relation selbst ist damit jedoch nicht gelöscht. Soll ein Objekt gelöscht werden, so geht das mit der Schaltfläche im Hauptfenster oder über das Kontextmenü direkt auf diesem Objekt.

Objekte können auch über den Graph-Editor angelegt werden. Das Vorgehen hierzu wird im den nachfolgenden Kapitel beschrieben.

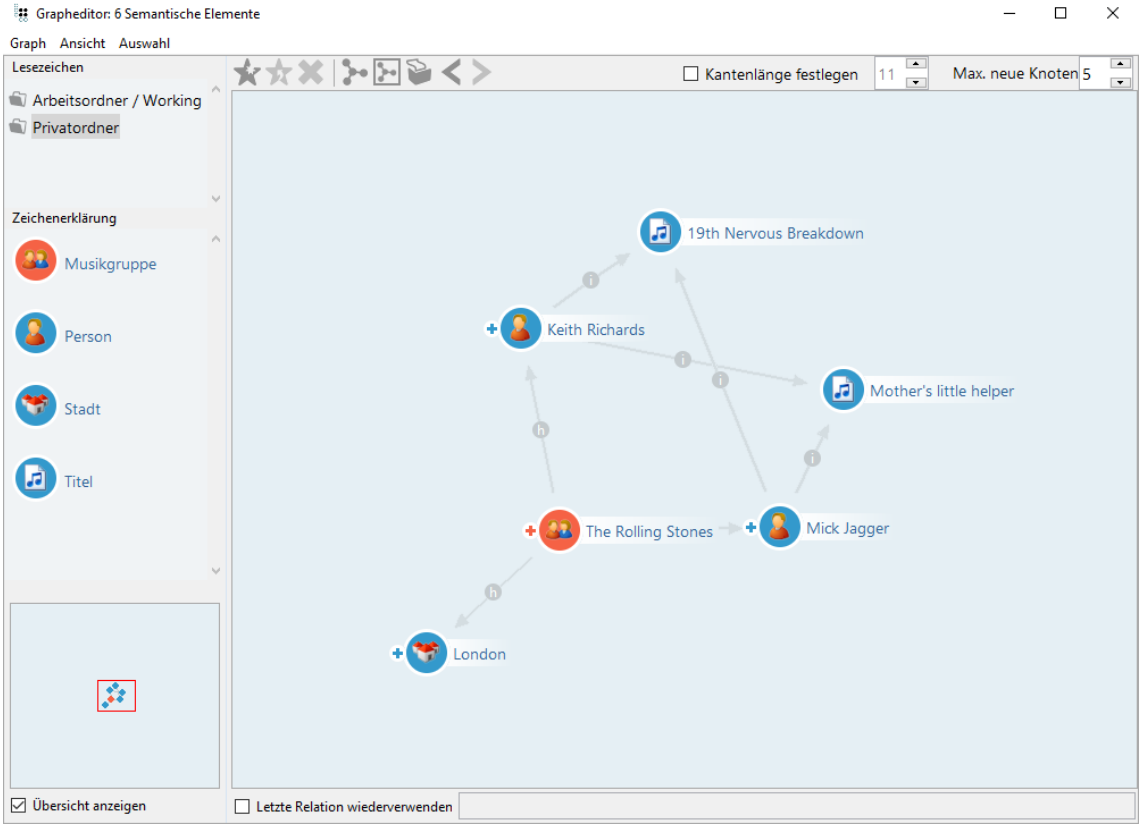
1.1.5. Graph-Editor

1.1.5.1. Einführung Graph-Editor

Mit dem Graph-Editor lassen sich die Inhalte des Knowledge Graphen mit ihren Objekten und Verbindungen graphisch darstellen. Der Graph-Editor lässt sich mit Hilfe des *Graph* Knopfes oder über das Kontextmenü von Objekten öffnen.



Die Objekte werden als Knoten dargestellt, ihre Beziehungen untereinander als Pfeile.



Der Graph zeigt immer nur einen Ausschnitt des Netzes. Objekte können in der Graphansicht ein- und ausgeblendet und es kann durch den Graph navigiert werden.

Durch Scrollen im Graph kann an der Position der Maus gezoomt werden. Der daraus resultierende Ausschnitt kann dann durch Klicken und Ziehen am Hintergrund oder durch das Drücken von Alt + Pfeiltasten bewegt werden.

Unten links in der Ecke befindet sich eine Übersicht über den gesamten Graphen, welche alle Knoten und Verbindungen, sowie den momentan rechts dargestellten Ausschnitt anzeigt. Durch Klicken auf die Übersicht wird der angeklickte Ausschnitt rechts angezeigt und durch Scrollen kann an der Mausposition gezoomt werden. Unter der Übersicht kann man auswählen, ob diese angezeigt werden soll oder nicht.

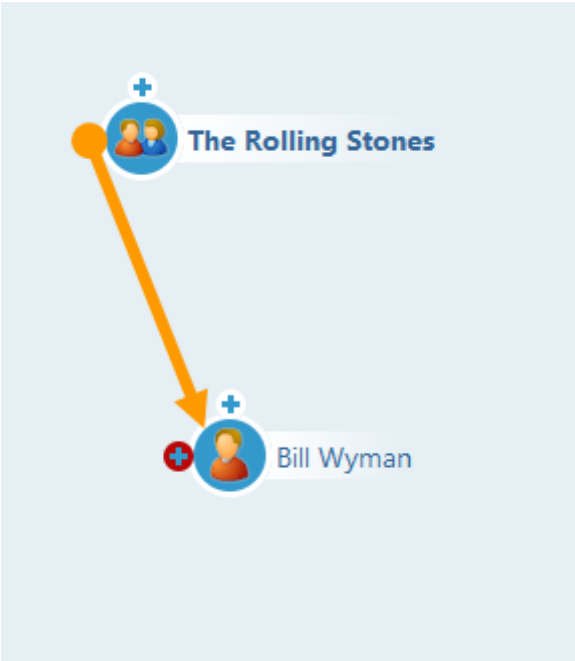
Doch nicht nur zur übersichtlichen Darstellung der Objekte und Relationen eignet sich der Graph. Im Graph-Editor können Objekte und Relationen außerdem editiert werden.

Auf der linken Seite eines Knotens befindet sich ein Anfasser in Form eines Plus-Zeichens für die Interaktion mit dem Objekt. Durch einen Doppelklick auf den Anfasser werden alle Benutzerrelationen des Objekts angezeigt bzw. ausgeblendet.

Das Verknüpfen von Objekten über eine Relation wird im Graph-Editor wie folgt vorgenommen:

1. Den Cursor über dem Anfasser links vom Objekt mit der linken Maustaste positionieren.
2. Cursor in gedrückter Position zu einem anderen Objekt ziehen (Drag&Drop). Falls mehrere

Relationen zur Auswahl stehen, erscheint eine Liste aller möglichen Relationen zur Auswahl. Falls nur eine mögliche Relation zwischen den beiden Objekten existiert, wird diese gezogen und keine Liste angezeigt. Eine existierende Relation kann, sofern es das Schema zulässt, ebenfalls per Drag & Drop umgezogen werden.



Letzte Relation wiederverwenden

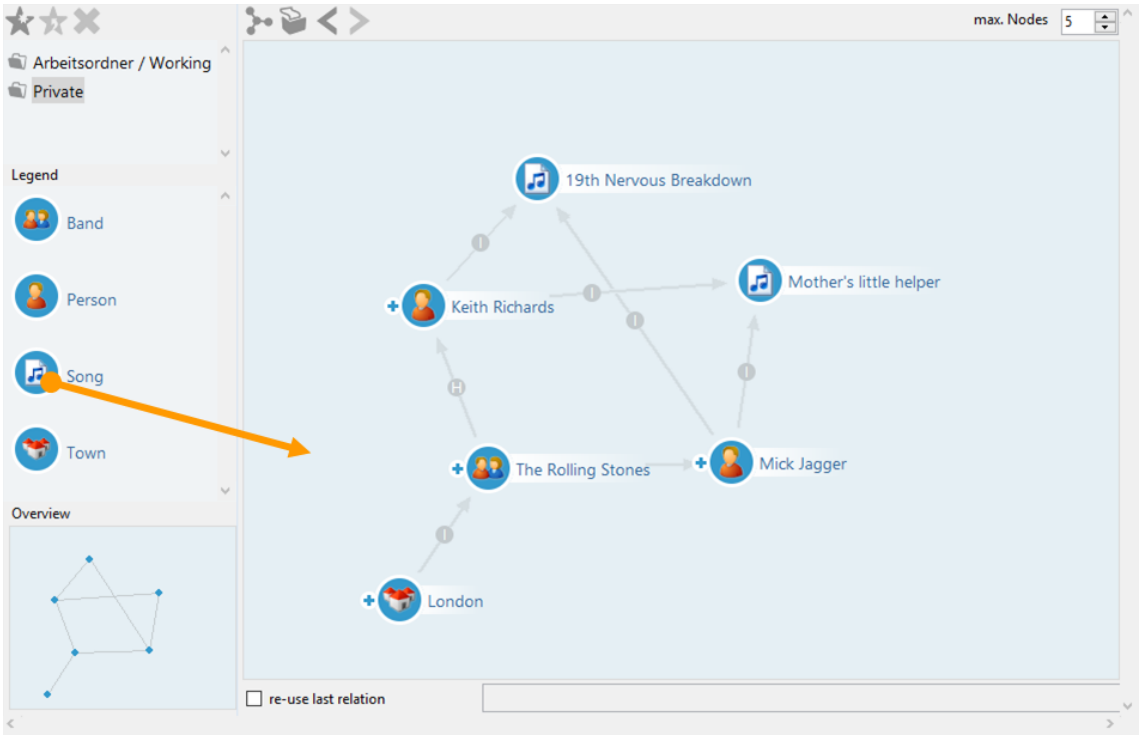
Wenn dieses Häkchen unter der Graphansicht gesetzt ist, wird beim Ziehen einer Relation zwischen zwei Knoten nicht gefragt, welche Relation man erstellen möchte, sondern es wird ungefragt die Relation benutzt, welche zuletzt gezogen wurde. Die verwendete Relation wird auch im rechten Kasten angezeigt.

Um Objekte im Graph-Editor anzuzeigen, gibt es verschiedene Möglichkeiten:

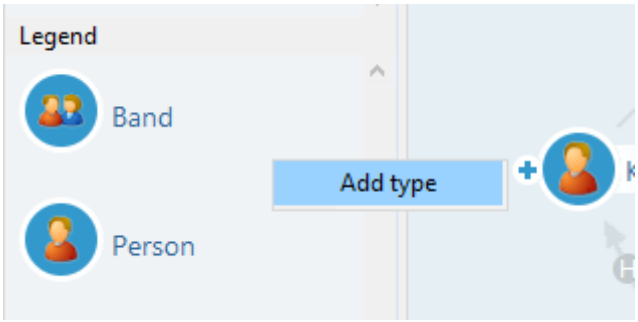
- Objekte können aus der Trefferliste im Hauptfenster von i-views mit Drag&Drop in das Fenster des Graph-Editors hineingezogen werden.
- Wenn der Name des Objekts bekannt ist, kann durch einen Rechtsklick auf eine leere Stelle im Graph-Editor das Kontextmenü geöffnet werden, wo über die Funktion "Objekt anzeigen" der Name des gewünschten Objektes eingegeben werden kann.

HINWEIS	Soll ein Objekt im Graph-Editor ausgeblendet werden, kann es durch Anklicken mit gleichzeitig gedrückter Strg-Taste aus dem Graph-Editor entfernt werden. Damit wird nichts in den Daten geändert: Das Objekt existiert unverändert im Knowledge Graph, es wird nur nicht mehr im aktuellen Graph-Editor-Ausschnitt gezeigt, d.h. es wird ausgeblendet.

Im Graph-Editor können auch neue Objekte angelegt werden. Dazu ziehen wir aus der Legende links im Fenster des Graph-Editors den gewünschten Objekttyp mit Drag&Drop auf die Zeichenfläche:



Sind keine Objekttypen in der Legende zu sehen, können diese mit rechtsklick im Feld der Legende gesucht werden. Anschließen wird der Name des Objektes vergeben.



Es erscheint wieder der Editor, in dem sich die für das Objekt möglichen Relationen, Attribute und Erweiterungen hinzufügen lassen.

1.1.5.2. Operationen auf Objekten im Graph-Editor

Das Kontextmenü der Knoten besteht aus mehreren Teilen:

- Die Überschrift, sprich der Name des Objekts. Diese hat als Untermenü das normale Kontextmenü des Objekts, was man auch findet, wenn man im KB einen Rechtsklick auf das Objekt macht.
- Operationen, die das Objekt selbst beeinflussen. Hier sind einige Operationen aus dem Kontextmenü des Objekts selbst übernommen worden, um schneller auf diese zugreifen zu können. Zusätzlich gibt es hier die Option, zwei Objekte zussammenzufassen.

- Operationen die sich auf den Graph Editor bzw. dessen Knoten beziehen und die Objekte selbst nicht ändern:

Fixieren

Fixierte Knoten werden durch die Layout-Funktion nicht verschoben.

Lösen

Hebt das Fixieren von Knoten wieder auf.

Kürzester Pfad

Blendet Knoten für alle Objekte ein, welche die kürzeste Verbindung zwischen zwei ausgewählten Knoten darstellen. Ist nur ein Knoten ausgewählt, erscheint ein Dialog um einen weiteren Knoten auszuwählen.

Knoten zentrieren

Der ausgewählte Knoten wir din die Mitte des angezeigten Bereichs verschoben.

Anzeigen

Hier befinden sich Operationen, um je nach Objektart unterschiedliche neue Knoten anzuzeigen:

Attribute

Blendet alle Attribute ein, welche an diesem Objekt definiert sind.

Objekte

Blendet alle Objekte dieses Typs ein.

Erweiterungtypen

Blendet alle Erweiterungstypen ein, die für eiesen Typ definiert sind.

Berechnete Relationen

Blendet alle Objekte ausgewählter berechneter Relationen ein.

Eigenschaftstypen

Blendet alle Eigenschaftstypen ein, welche für diesen Typ definiert sind.

Eigenschaftsobjekte

Blendet alle konkreten Attribute diesen Typs ein, welche an Objekten in diesem Graph definiert sind.

Definiert für

Blendet alle Objekttypen ein, für die diese Eigenschaft definiert ist.

Erweiterungen

Blendet alle Erweiterungen ein, welche an diesem Objekt definiert sind.

Ausblenden

Hier befinden sich Operationen, um je nach Objektart unterschiedliche Knoten auszublenden:

Ausgewählte Knoten

Entfernt alle ausgewählten Knoten.

Verbundene Knoten

Entfernt alle Knoten, welche mit diesem Knoten verbunden sind.

Untertypen

Entfernt alle Untertypen dieses Typen.

Objekte

Entfernt alle Objekte dieses Typen.

Eigenschaftsobjekte

Entfernt alle konkreten Attribute dieses Attributtypen.

Darstellung

Hier befinden sich Operationen, um das Aussehen von Knoten zu beeinflussen:

Kleine Icons

Ändert die Größe des Knotens zu klein.

Mittelgroße Icons

Ändert die Größe des Knotens zu mittel.

Große Icons

Ändert die Größe des Knotens zu groß.

Neue Beschriftung

Ändert die Beschriftung des Knotens.

Icon ändern

Ändert das Icon des Knotens zu einem der Icons, welche an dem Objekt, einem Typ oder einem Obertyp definiert ist.

HINWEIS

Es können auch mehrere Objekte ausgewählt werden, indem man die Umschalt-Taste gedrückt hält, während man einen Mausklick macht und die Maus zieht oder dabei direkt auf die gewünschten Knoten klickt. Dadurch können alle ausgewählten Objekte zusammen bewegt werden (z.B. mittels Strg + Pfeiltasten) und es können Operationen auf mehreren Objekten gleichzeitig ausgeführt werden.

1.1.5.3. Ansicht

Das Menü **Ansicht** stellt viele weitere Funktionen für die graphische Darstellung von Objekten und Objekttypen zur Verfügung:

Voreinstellungen

Öffnet die KB Einstellungen für Graph-Editor Einstellungen (auch zugänglich im globalen Einstellungsfenster  unter **Persönlich** > **Graph**).

Hintergrund ändern

Hier kann die Farbe des Hintergrunds geändert oder ein eigenes Bild als Hintergrund eingefügt werden.

Knoten automatisch ausblenden

Blendet automatisch überschüssige Knoten aus, sobald mehr als die gewünschte Anzahl an Knoten sichtbar ist. Die Anzahl kann im Eingabefeld "max. neue Knoten" in der Symbolleiste eingestellt werden.

Knoten automatisch positionieren

Führt für neu eingeblendete Knoten automatisch die Layout-Funktion aus.

Positionierungshilfe

Sorgt dafür, dass Knoten, die im Umkreis von anderen Knoten bewegt werden, auf deren x oder y Koordinate einrasten, wenn sie in deren Nähe kommen.

Beschreibungen fixieren

Mit dieser Option sind die Namen aller Relationen immer sichtbar, nicht nur beim Roll-over mit der Maus. Alternativ kann gezielt im Kontextmenü einer Relation deren Beschreibung fixiert werden.

Interne Namen anzeigen

Blendet an Typknoten die internen Namen der Typen ein.

Ausgeblendete Kanten wieder darstellen

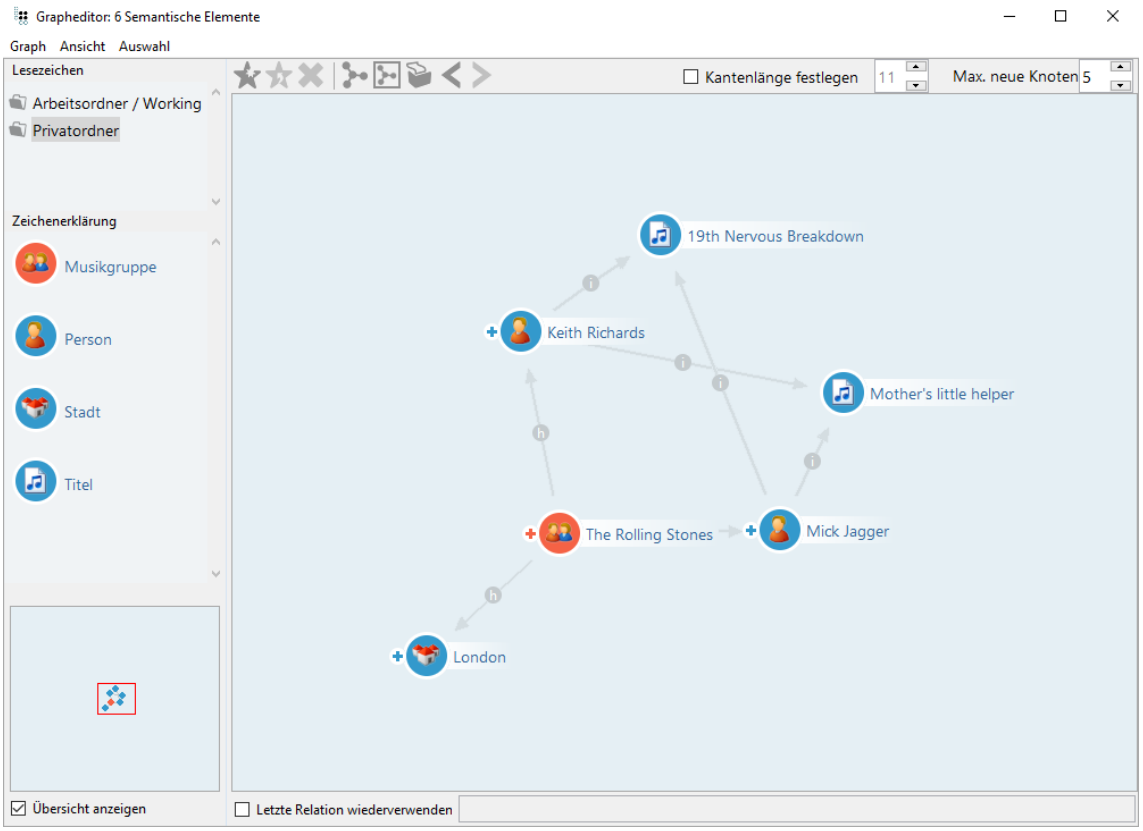
Alle Kanten, die per Kontextmenü ausgeblendet wurden, werden wieder angezeigt.

Kanten immer hervorheben

Alle Kanten werden immer hervorgehoben.

Das Fenster des Graph-Editors und das Hauptfenster des Knowledge-Builders stellen noch weitere Menüpunkte zur Verfügung, die bei der Modellierung eine Hilfestellung bieten können.

Links im Fenster des Graph-Editoren befindet sich die Legende der Objekttypen.



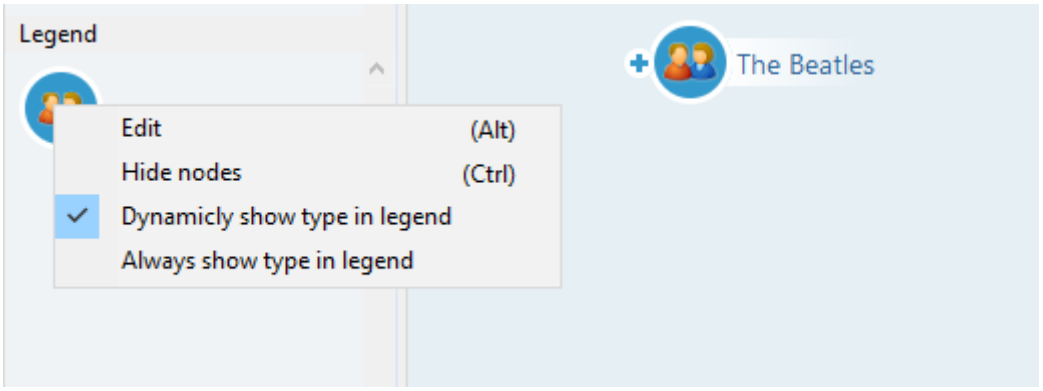
Diese Legende zeigt die Objekttypen zu den konkreten Objekten auf der rechten Seite.

Durch Drag&Drop eines Eintrags aus der Legende in die Zeichenfläche können Sie ein neues konkretes Objekt des entsprechenden Typs erzeugen oder ein bestehendes hinzufügen.

Durch Rechtsklick in der Legende können weiter Typen permanent hinzugefügt werden. Dies ermöglicht das Hinzufügen weiterer Objekte in den Graph per Drag & Drop.

HINWEIS	Elemente aus dem Knowledge Graph können durch Drag & Drop vom Knowledge-Builder zum Graph-Editor hinzugefügt werden.

Über das Kontextmenü auf den Legendeneinträgen können alle konkreten Objekte dieses Typs aus der Darstellung ausgeblendet werden. Hier lassen sich auch Legendeneinträge "festhalten", und neue Objekttypen in die Legende aufnehmen (unabhängig davon, ob konkrete Objekte von diesem Typ in der Darstellung vertreten sind).

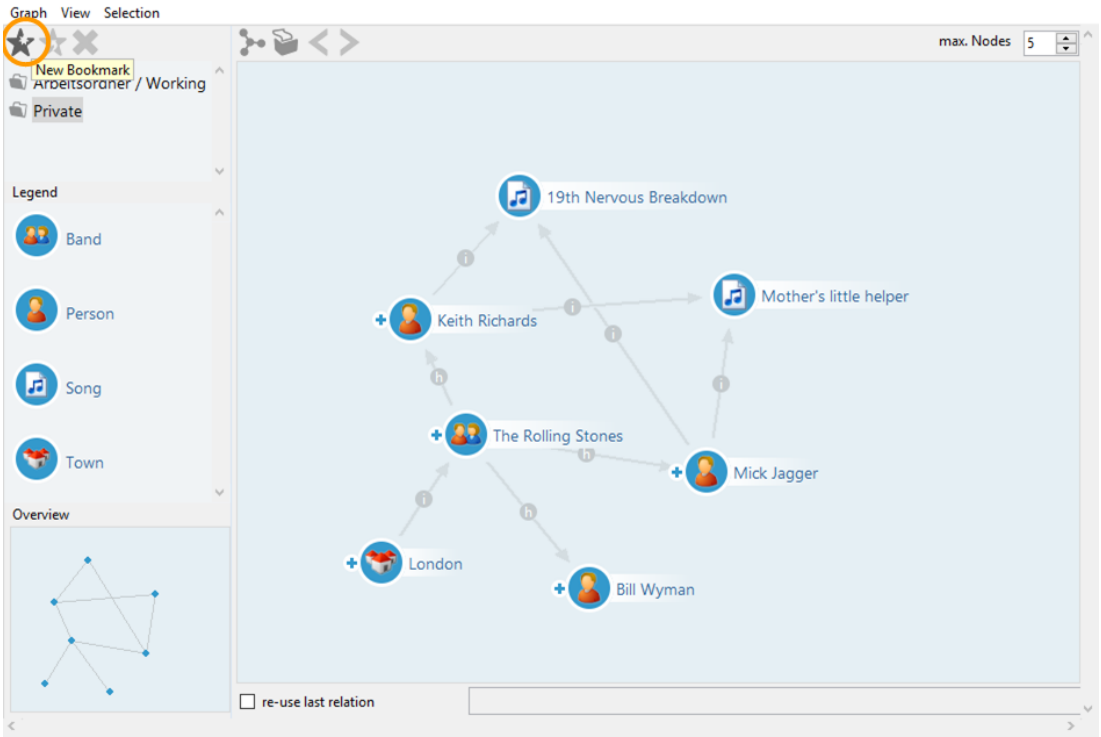


1.1.5.4. Lesezeichen und weitere Bedienelemente

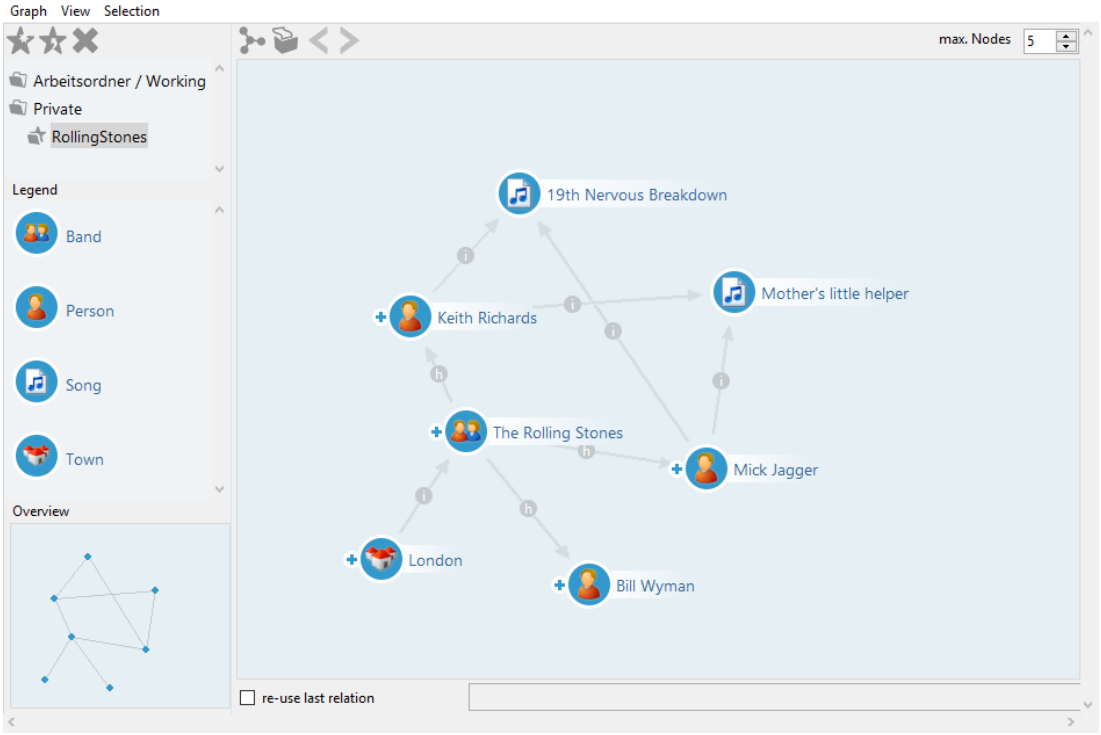
Auf der Toolbar befinden sich einige praktische Knöpfe zur Bedienung des Graph-Editors:

Lesezeichen

Die erstellten Ansichten im Graph-Editor lassen sich als Lesezeichen speichern. Die Objekte werden in der Position gespeichert, wie sie im Graph-Editor platziert wurden.



Wenn ein Lesezeichen angelegt werden soll, muss hierfür zunächst ein Ordner ausgewählt werden. Anschließend kann das Lesezeichen benannt werden.



Lesezeichen sind jedoch keine Datenbackups: Objekte und Beziehungen, die nach dem Speichern eines Lesezeichens gelöscht wurden, sind auch beim Anzeigen des Lesezeichens nicht mehr in der Darstellung vorhanden.

Layout

Mit der Layout-Funktion lassen sich Knoten automatisch in den momentan herangezoomten Ausschnitt positionieren, wenn viele Knoten nicht manuell positioniert werden sollen. Beim Einblenden weiterer Knoten werden diese ebenfalls über die Layout-Funktion automatisch im Graph positioniert. Voraussetzung hierfür ist, dass die Option "Knoten automatisch positionieren" aktiviert ist (siehe vorhergehendes Kapitel).

Ausschnitt auf alle Knoten anpassen

Im Gegensatz zu der Layout-Funktion wird hier der angezeigte Ausschnitt angepasst, um alle Knoten sichtbar zu machen.

Drucken

Die Druck-Funktion öffnet das Dialogfenster zum Ausdrucken, oder zur Generierung einer PDF-Datei des angezeigten Graphs.

Historie

Mit den Schaltflächen "Navigation rückgängig" und "Navigation wiederherstellen" können die Elemente in der Reihenfolge, in der sie eingeblendet wurden, wieder ausgeblendet werden (und vice versa). Außerdem machen die Schaltflächen die neue Anordnung der Knoten des Auto-Layouts rückgängig. Die Schaltflächen befinden sich in der Kopfzeile im Fenster des Graph-Editors, oder im Menü "Graph".

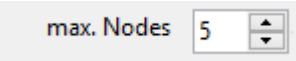
Kantenlänge festlegen

Hiermit kann der Abstand bestimmt werden, in dem neue Knoten hinzugefügt werden. Ist das Häkchen nicht gesetzt, ist der Abstand relativ zur aktuellen Zoom-Stufe.

Max. neue Knoten

Wenn ein Knoten/Objekt viele Nachbarobjekte hat, ist es oft nicht sinnvoll, alle beim Klick auf den Anfasser gleich einzublenden. Hierfür kann an zwei Stellen eine maximale Anzahl einzublendender Knoten definiert werden.

1. Über das globale Einstellungsfester Registerkarte "Persönlich" Graph lässt sich die Anzahl bei max. Knoten festlegen, oder
2. im Fenster des Graph-Editors oben rechts ist diese Aktion ebenfalls aufrufbar.



Wird der Anfasser zur Einblendung der Nachbarobjekte angeklickt und es gibt mehr Nachbarobjekte als das eingestellte Maximum, erscheint anstelle der Objekte eine Auswahlliste.

Im Menü **Graph** stehen weitere Funktionen für den Graph-Editor zur Verfügung:

In Zwischenablage kopieren

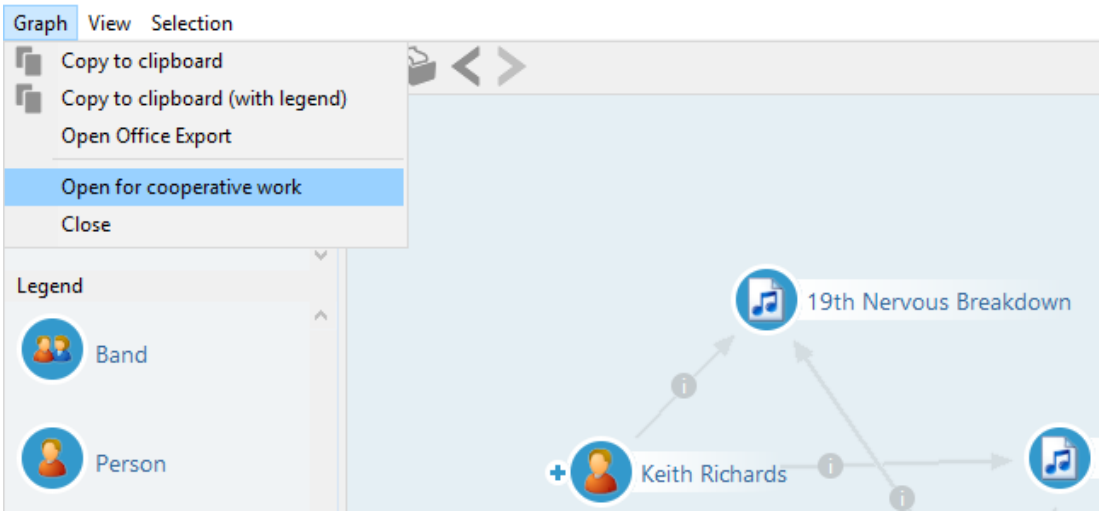
Diese Funktion erzeugt einen Screenshot des aktuellen Graph-Editor-Inhalts. Dieses Bild kann dann beispielsweise in ein Zeichen- oder Bildbearbeitungsprogramm eingefügt werden.

In Zwischenablage kopieren (mit Legende)

Macht dasselbe wie die vorige Option, aber das Bild wird die Legende enthalten.

Für kooperative Arbeit öffnen

Diese Funktion ermöglicht anderen Benutzern, gemeinsam und gleichzeitig am Graph zu arbeiten. Alle Änderungen und Selektionen eines Benutzers am Graph (Layout, Ein-/Ausblenden von Knoten usw.) werden dann synchron bei allen anderen Benutzern angezeigt.



1.1.5.5. Einstellungen

Die Einstellungen zum Graph-Editor sind in den KB-Einstellungen unter **Persönlich** > **Graph** zu finden. Sie können ebenfalls über das Menü **Ansicht** > **Voreinstellungen** aus dem Graph-Editor selbst geöffnet werden.

Einige dieser Einstellungen können auch im Graph-Editor direkt überschrieben werden, gelten dann aber nur für dieses spezielle Graph-Editor-Fenster.

Tooltips mit Details anzeigen

Wenn angekreuzt, wird, wenn der Mauszeiger über einen Knoten gehalten wird, ein Fenster mit den Eigenschften des Objekts angezeigt.

Max. Anzahl Zeilen für Tooltips

Bestimmt nach wie vielen Zeilen das Fenster abgeschnitten wird.

Knoten automatisch ausblenden

Blendet automatisch überschüssige Knoten aus, sobald mehr als die gewünschte Anzahl an Knoten sichtbar ist. Die Anzahl kann im Eingabefeld "max. neue Knoten" in der Symbolleiste eingestellt werden.

Knoten automatisch positionieren

Führt für neu eingeblendete Knoten automatisch die Layout-Funktion aus.

Positionierungshilfe

Sorgt dafür, dass Knoten, die im Umkreis von anderen Knoten bewegt werden, auf deren x oder y Koordinate einrasten, wenn sie in deren Nähe kommen.

Cairo-Bibliothek zur Darstellung verwenden

Kann nur aktiviert werden, wenn die Cairo Bibliothek der KB-Anwendung beigelegt ist. Wenn angekreuzt, wird diese Bibliothek zur Darstellung bestimmter Dinge verwendet.

Standard Zoom

Das Zoomlevel mit dem der Graph-Editor sich öffnet.

Max. neue Knoten

Wenn ein Knoten/Objekt viele Nachbarobjekte hat, ist es oft nicht sinnvoll, alle beim Klick auf den Anfasser gleich einzublenden. Hiermit wird definiert, wie viele neue Knoten auf einmal ohne Nachfrage eingeblendet werden können.

Max. Textlänge

Definiert, nach wie vielen Zeichen die Labels von Knoten abgeschnitten werden.

Knotengröße

Bestimmt mit welcher Größe Knoten einem Graph hinzugefügt werden.

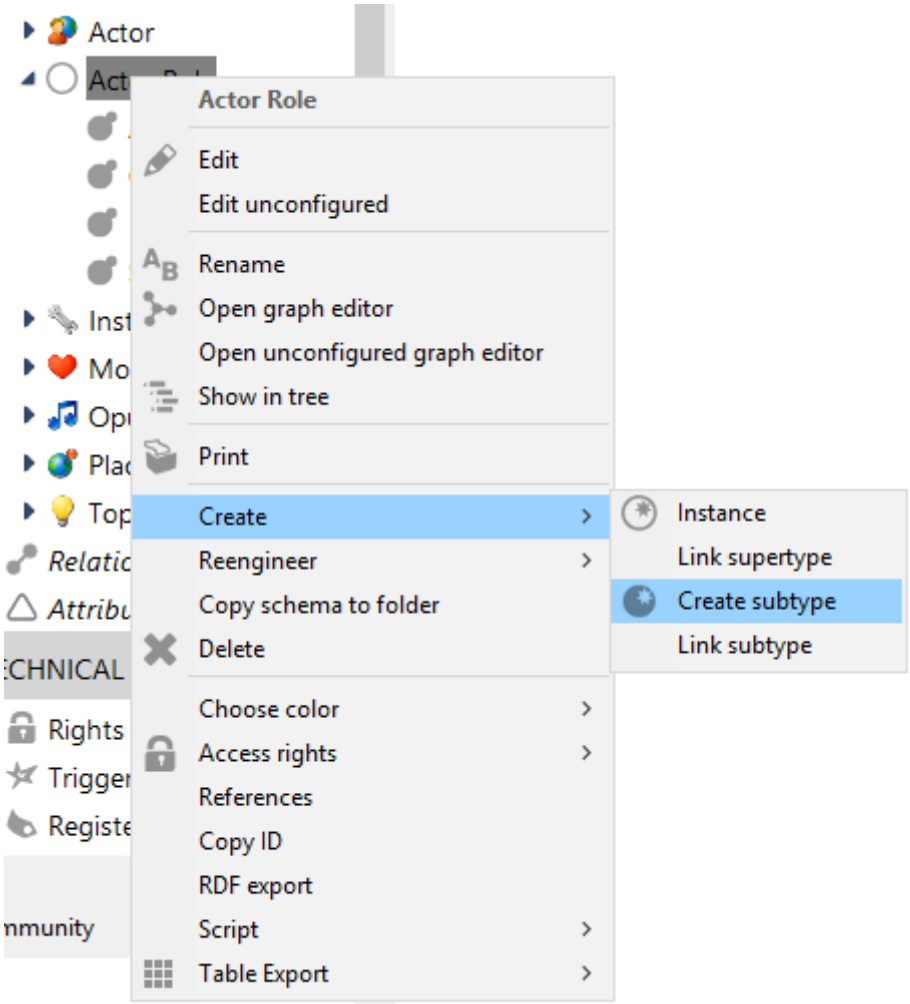
Konfiguration der Legende

Hier können Typen definiert werden, welche immer beim Öffnen eines Graph-Editors in der Legende angezeigt werden.

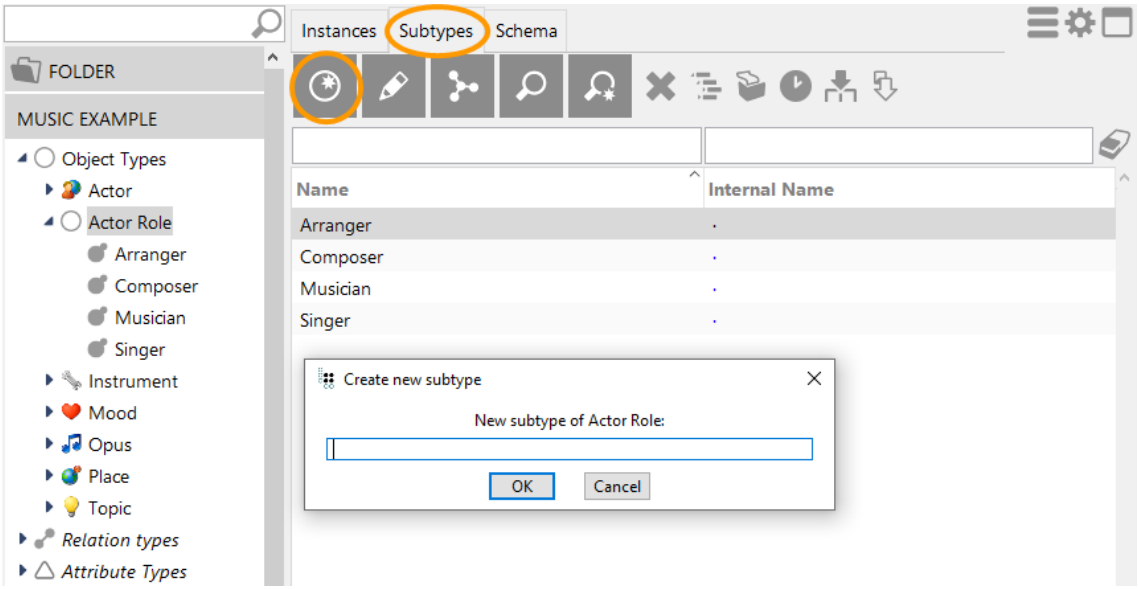
1.2. Schemadefinition / Modell

1.2.1. Typen definieren

Im Kapitel Grundbausteine ist das Prinzip der Typenhierarchie bereits vorgestellt worden. Sollen neue Typen angelegt werden, erfolgt dies immer als Untertyp eines existierenden Typs. Das Anlegen der Untertypen kann entweder über das Kontextmenü Erstellen > Untertyp



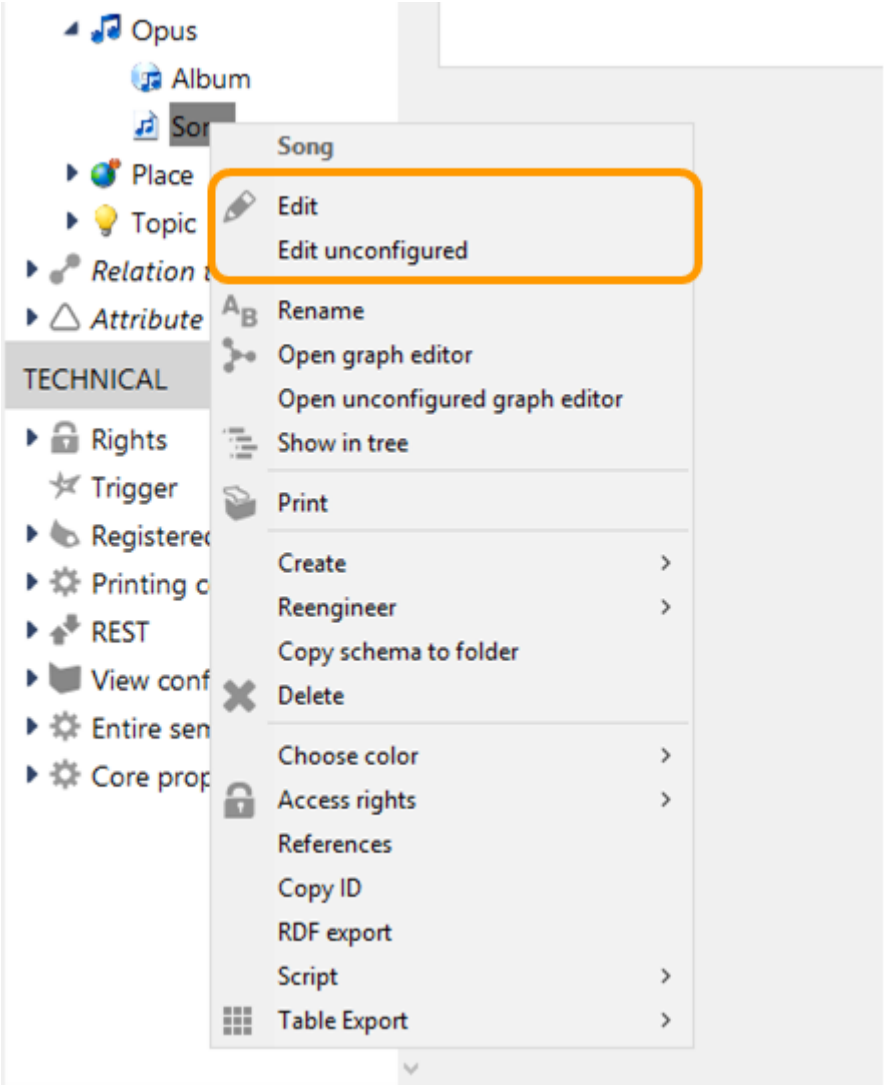
oder im Hauptfenster im Reiter "Untertypen" über das Suchfeld und die Schaltfläche "Neu" vorgenommen werden:



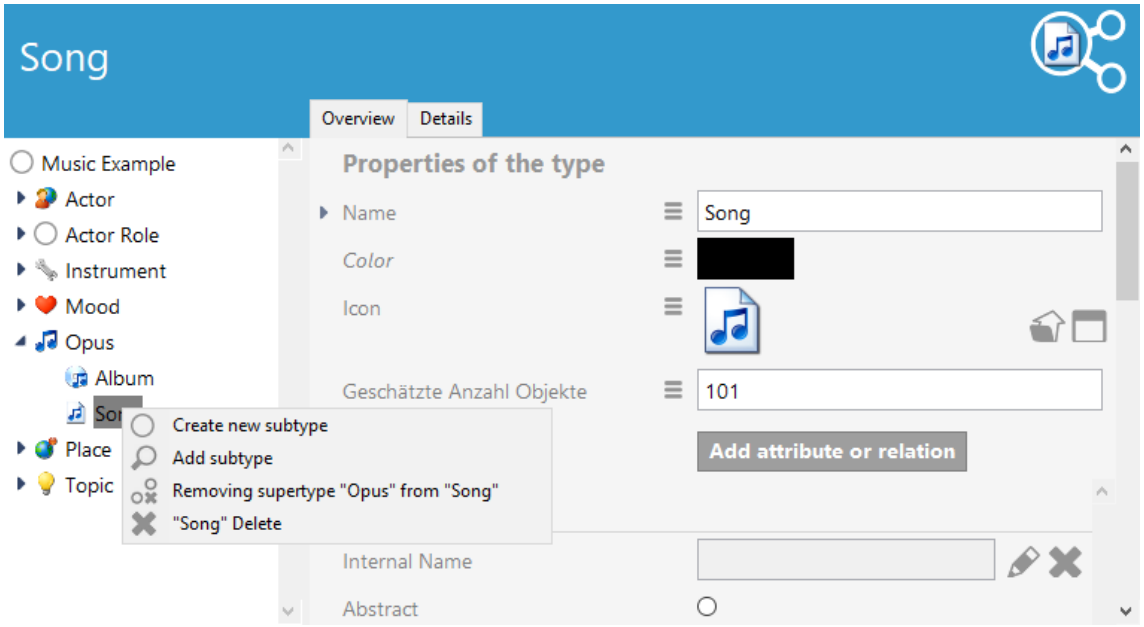
Typenhierarchie ändern

Zum Ändern der Typenhierarchie stehen uns der Baum der Objekttypen im Hauptfenster und der Graph-Editor zur Verfügung.

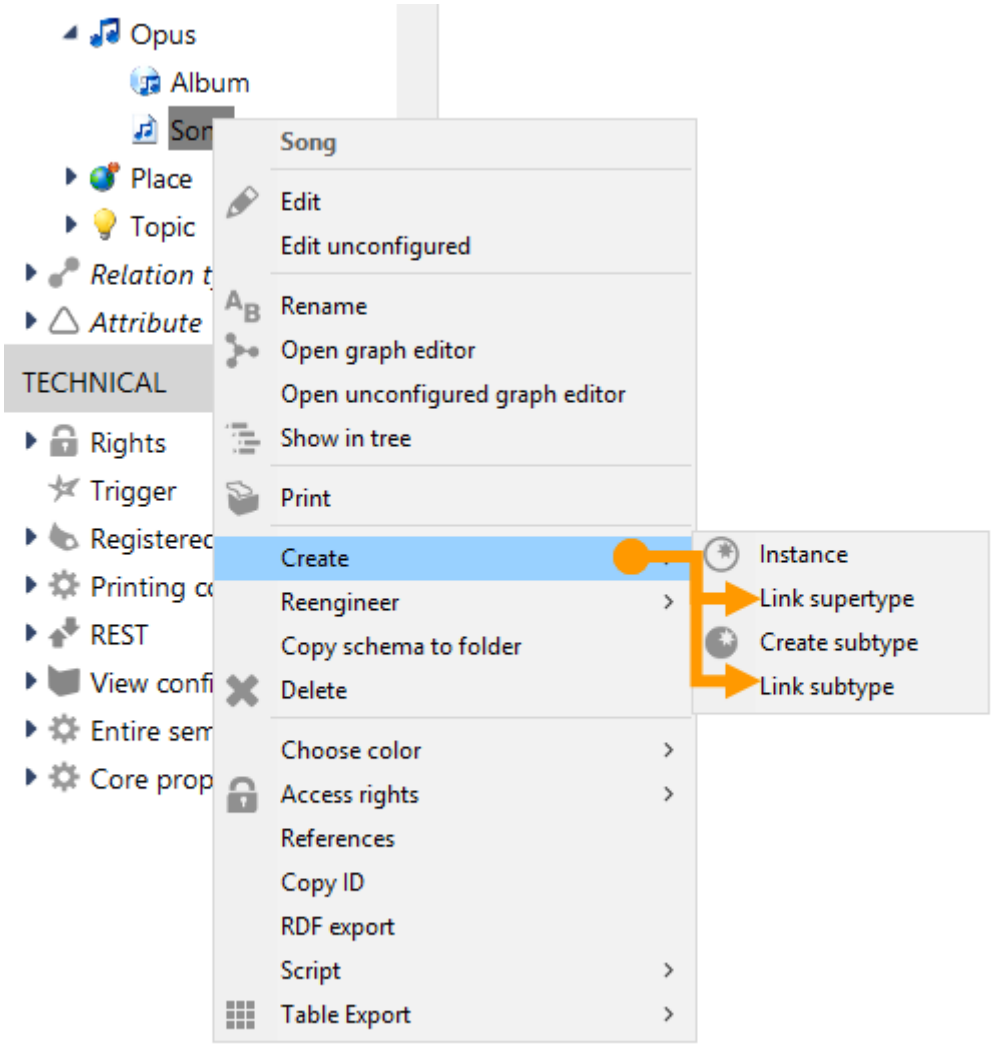
Wir können auch die Zuordnung zu anderen Typen bestimmen. Hierfür öffnen wir den Detail-Editor durch Auswahl der Option "Bearbeiten" oder "Unkonfiguriert bearbeiten" im Kontextmenü:



Im Hierarchie-Baum des Detail-Editors finden wir im Kontextmenü die Option "Entferne Obertyp xy".

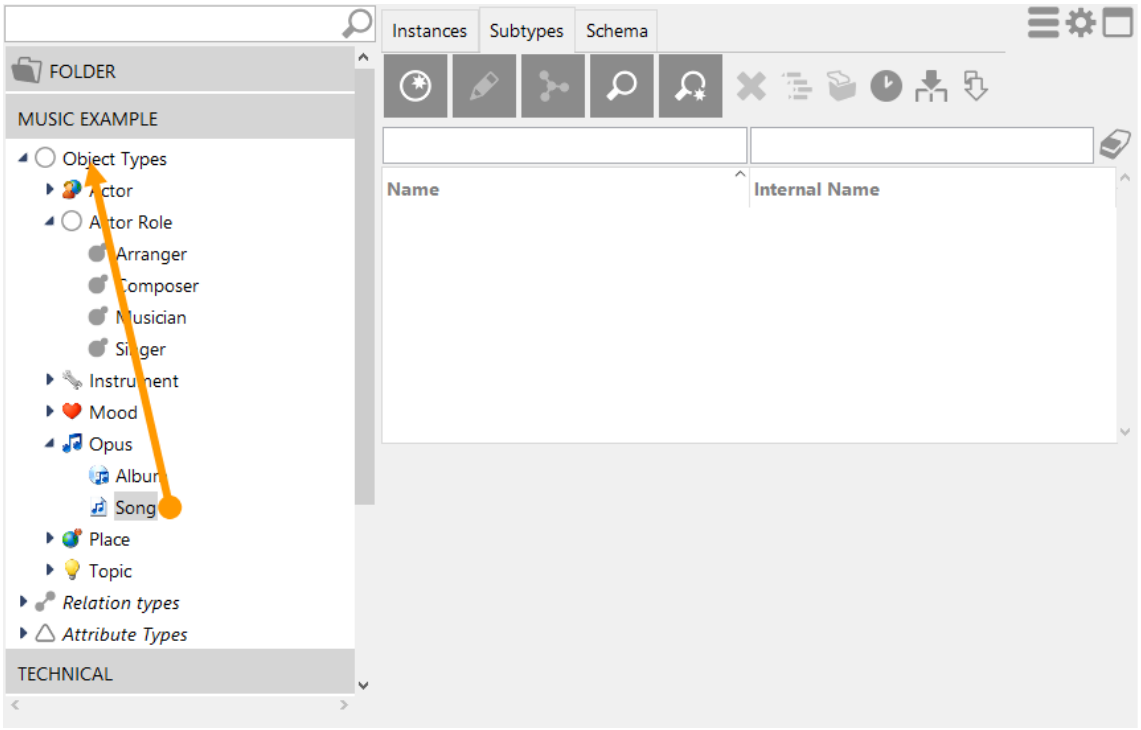


Damit können wir den aktuell selektierten Objekttyp aus seiner Position in der Objekttypen-Hierarchie herauslösen. Im Organizer können wir Typen mit anderen Typen verlinken, um ein multi-hierarchisches Schema zu erzeugen:



Shortcut: Mit Drag&Drop können wir einen Objekttyp in einen anderen Ast der Hierarchie verschieben. Halten wir beim Drag&Drop die **Strg-Taste** gedrückt, wird der Objekttyp nicht verschoben, sondern zusätzlich unter einen weiteren Objekttyp eingeordnet.

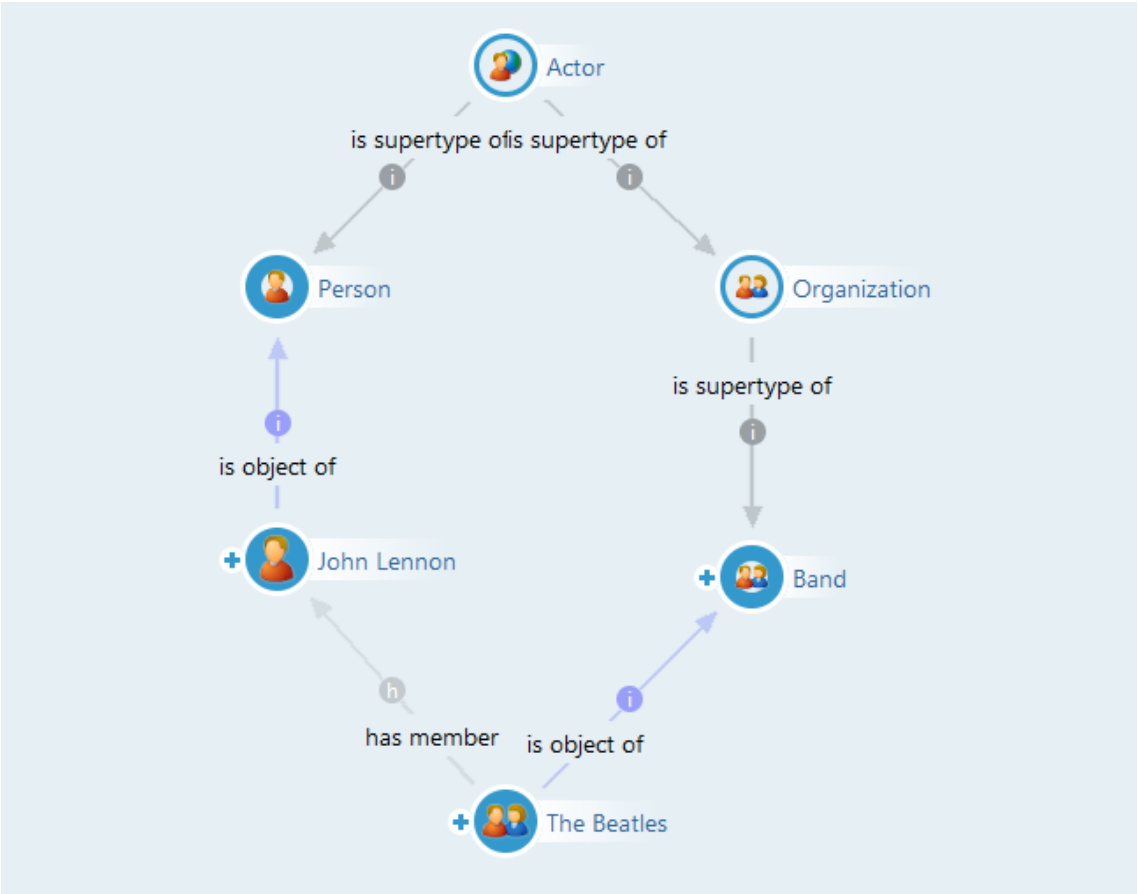
Nach wie vor gilt: Die Hierarchie der Objekttypen erlaubt Mehrfach-Einordnung und -Vererbung.

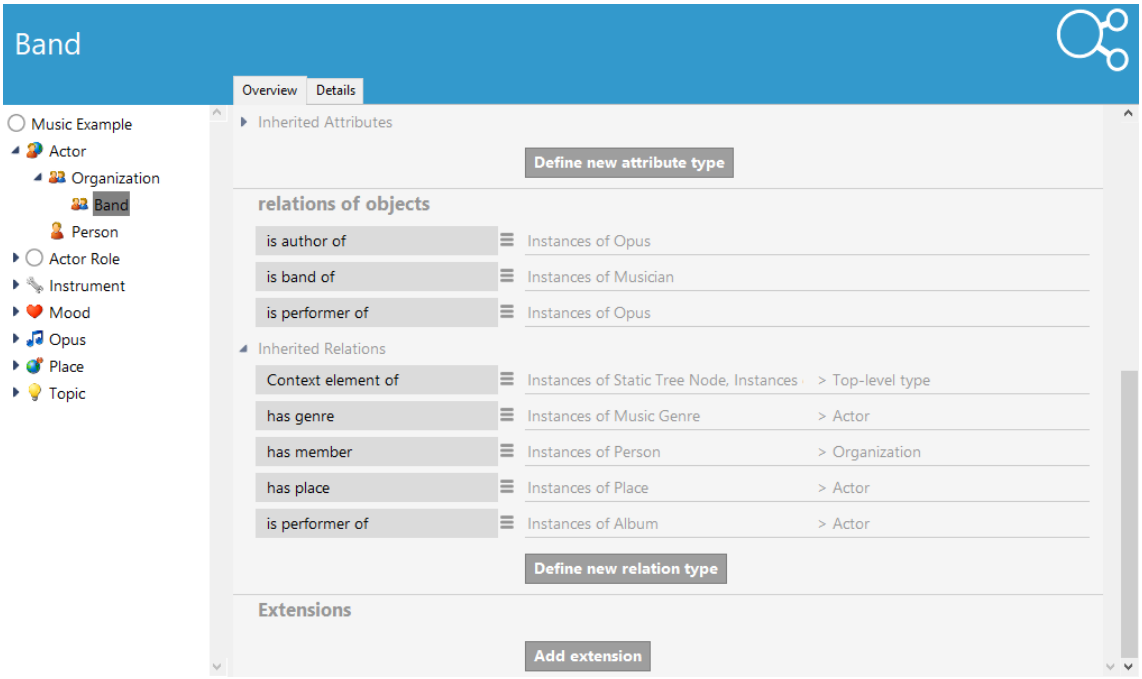


Objekttypen mit Eigenschaften ausstatten

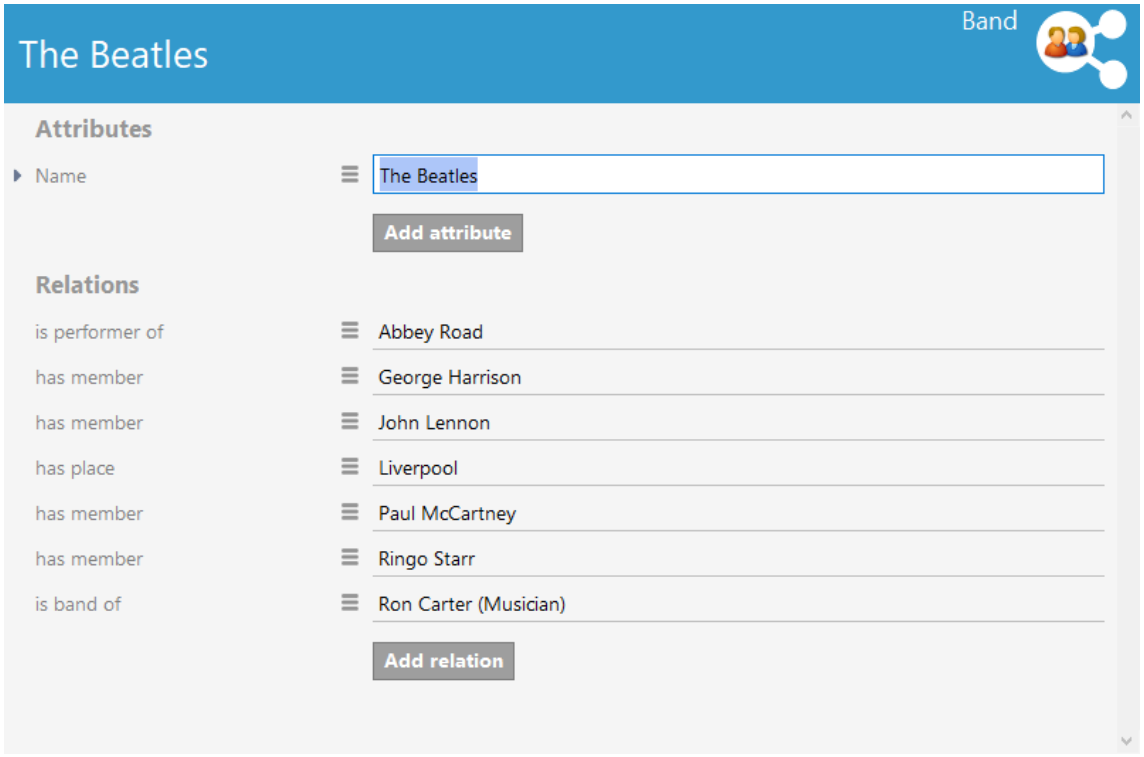
Im einfachsten Fall definieren wir Relationen und Attribute bei einem Objekttyp wie z.B. "Band" oder "Person" und stellen sie damit für die konkreten Objekte dieses Typs zur Verfügung. (Bspw. Gründungsjahr und -ort bei Bands, Geburtsdatum und Geschlecht bei Personen, Veranstaltungsort und -datum bei Veranstaltungen.)

Hat der Objekttyp, bei dem die Eigenschaften definiert sind weitere Untertypen, so greift hier das Prinzip der Vererbung: Eigenschaften stehen nun auch für die konkreten Objekte der Untertypen zur Verfügung. Beispiel: eine Band erbt als Untertyp einer Organisation die Möglichkeit, Personen als Mitglieder zu haben. Als Untertyp von "Person oder Gruppe" erbt die Band auch die Möglichkeit, an Veranstaltungen teilzunehmen:





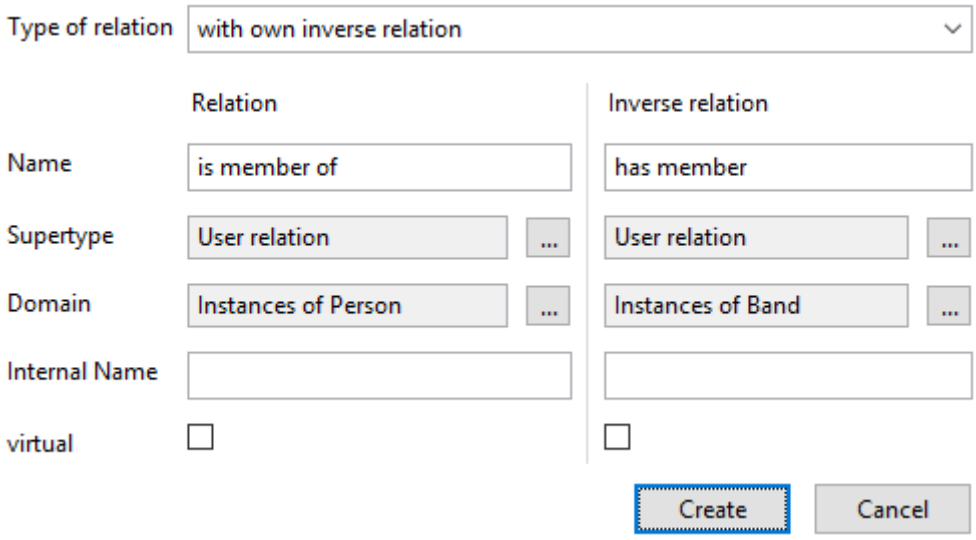
Der Editor für den Objekttyp "Band" mit direkt dort definierten und geerbten Relationen.



Beim konkreten Objekt stehen die geerbten Eigenschaften ohne Weiteres zur Verfügung, hier wird der Unterschied gar nicht bemerkt.

Beziehungen definieren

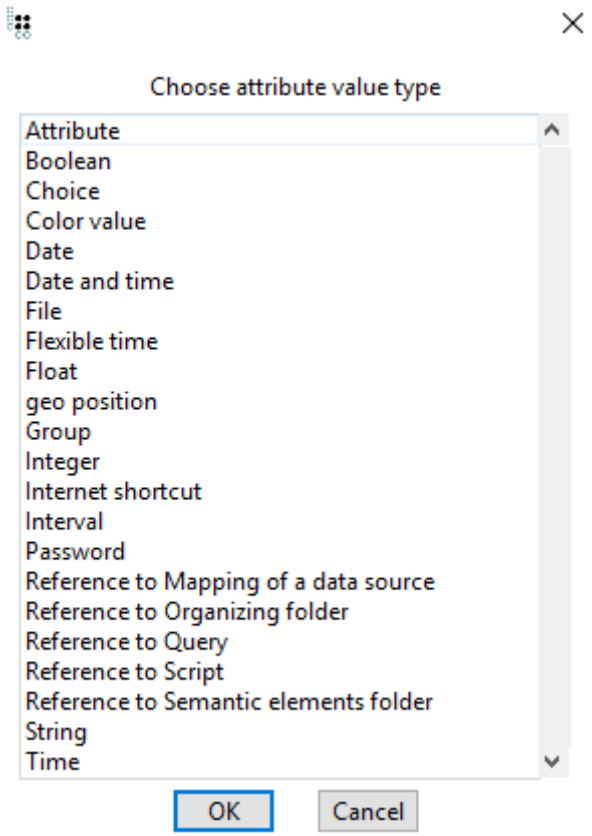
Beim Umgang mit Beziehungen herrscht in i-views folgendes Grundprinzip: Eine Beziehung kann nicht nur einseitig sein. Wenn wir für die konkrete Person "John Lennon" eine Beziehung "ist Mitglied von" zur Musikgruppe Beatles kennen, dann impliziert das wiederum für die Beatles den Sachverhalt "haben Mitglied John Lennon". Diese beiden Richtungen sind nicht zu trennen. Deswegen verlangt i-views beim Anlegen neuer Relationstypen von uns die Typen von Quelle ("Domäne") und Ziel der Relationen ("Zieldomäne") — in unserem Beispiel wären das Person und Band — sowie unterschiedliche Benennungen: "ist Mitglied von" und "hat Mitglied".



Damit ist die Relation definiert und kann jetzt zwischen Objekten per Drag&Drop gezogen werden.

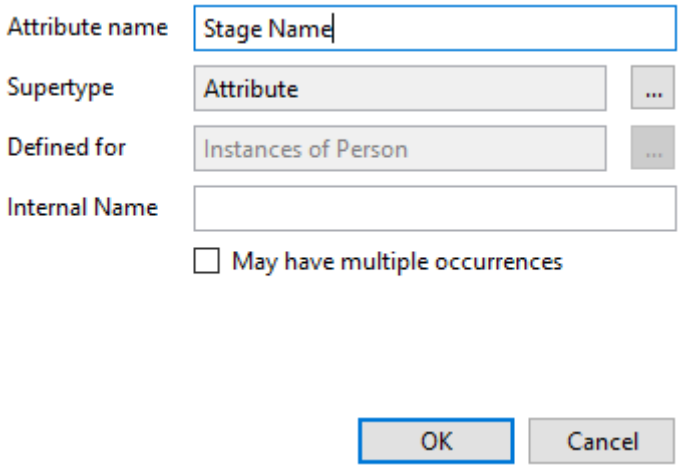
Attribute definieren

Bei der Definition neuer Attributtypen benötigt i-views neben dem Namen den technischen Datentyp.



Die Intension der Nutzung solcher Datentypen ist es, nicht einfach alles als Zeichenkette zu definieren. Technische Datentypen in festgelegter Formatierung bieten später spezielle Möglichkeiten zum abfragen und vergleichen. Beispielweise können Zahlenangaben innerhalb der Strukturabfragen mit größeren oder kleineren Werten verglichen werden, für Geokoordinaten kann eine Umkreissuche definiert werden, u.v.m.

Nach Auswahl des Attributwertetyps kann der Name für den Attributtyp vergeben werden:



1.2.2. Relations- und Attributtypen

Relations- und Attributtypen (kurz Eigenschaftstypen) sind die Typen der konkreten Eigenschaften, die an den Objekten gespeichert sind.

1.2.2.1. Neuen Relationstyp anlegen

Über die Schaltfläche im Objekt-Editor "Relation hinzufügen" startet der Editor zum Anlegen eines neuen Relationstyps:

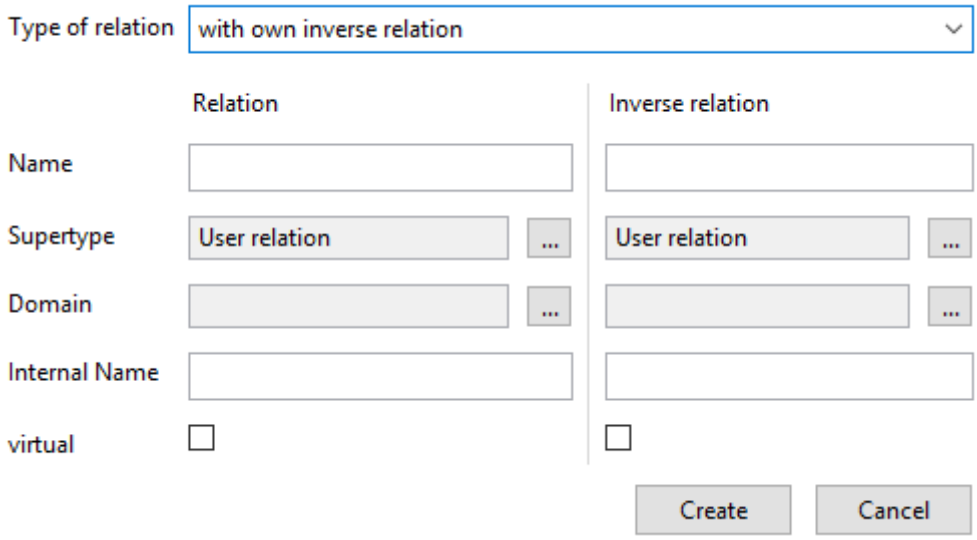


Figure 1. Editor für das Anlegen eines neuen Relationstyps

Relationstyp

"mit eigener Rückrelation" ist die Standardeinstellung, be der für jede Relationshälfte eine eigene Bezeichnung verwendet wird. "Symmetrisch" bestimmt, dass die Relation nur zwischen Instanzen jeweils einer Domäne verwendet werden kann. Beide Relationshälften erhalten in diesem Fall dieselbe Bezeichnung.

Name

Benennungen für Relationstypen lassen sich in i-views frei vergeben, sollten aber nach der Prämisse eines verständlichen Datenmodells gewählt werden. Die folgende Konvention kann dazu hilfreich sein: Der Namen der Beziehung wird so formuliert, dass die Struktur [Name des Quellobjekts] [Relationsname] [Name des Zielobjekts] einen verständlichen Satz ergibt:

```
[John Lennon] [ist Mitglied von] [The Beatles]
```

Weiterhin ist es hilfreich, wenn die Gegenrichtung (inverse Beziehung) die Wortwahl der Hauptrichtung aufgreift: "hat Mitglied / ist Mitglied von".

Obertyp

Spezifiziert den Relationsobertypen in der Relationstyp-Hierarchie.

Genau wie die Objekttypen können auch die Relations- und Attributtypen in einer Hierarchie stehen. Die Hierarchie der Relationstypen ist ein einfaches, aber mächtiges Instrument, um Komplexität im Griff zu behalten.

Beispiel: Für Abfragen kann über eine Relation "hat Autor" differenziert werden, wer den Text des Songs, und wer die Musik geschrieben hat. Gleichzeitig haben wir auch Abfragen, bei denen keine Ausdifferenzierung gewünscht ist und alle Beteiligten zugeliefert werden sollen.

Ohne die Relationshierarchie müssten wir jetzt alle diese Suchen komplizierter machen, indem wir beide Relationen abfragen. Stattdessen können wir die Relationen "schreibt Text" und "schreibt Musik" als Untertypen von "schreibt Song" (oder: ist Autor von) definieren, damit können wir immer noch auf der Ebene "schreibt Song" abfragen. i-views fragt automatisch die Unterrelationen mit ab.

Der Untertyp impliziert somit den Obertyp. Dieses Prinzip greift bei Relationen genauso wie bei Attributen und Objekten.

Domäne und Zieldomäne

Hier wird bestimmt bei welchen Typen von Objekten die Relation angelegt werden soll: Ein Objekttyp bildet die Quelle der Relation ("Domäne"), ein weiterer Objekttyp das Ziel ("Zieldomäne"). Der Ziel-Objekttyp bildet wiederum den Definitionsbereich der inversen Relation. Beim Anlegen kann der Einfachheit halber an dieser Stelle nur ein Objekttyp eingetragen werden. Im Relationstyp-Editor (siehe unten) lassen sich im Nachhinein noch weitere Objekttypen definieren.

Interner Name

Wird dafür verwendet, um Relationen per Skript identifizieren und referenzieren zu können.

Virtuell

Wenn einseitige Relationen benötigt werden, können wir hier bestimmen, welche der Relationshälften einseitig ist und welche der Relationshälften virtuell ist. Die virtuelle Relationshälfte ist nicht persistent, sondern wird nur zur Auflistung im Knowledge-Builder gerendert oder kann in Abfragen verwendet werden. Für mehr Informationen siehe "Einseitige Relationen".

1.2.2.2. Neuen Attributtyp anlegen

Über die Schaltfläche im Objekt-Editor "Neues Attribut definieren" startet der Editor zum Anlegen eines neuen Attributtyps:

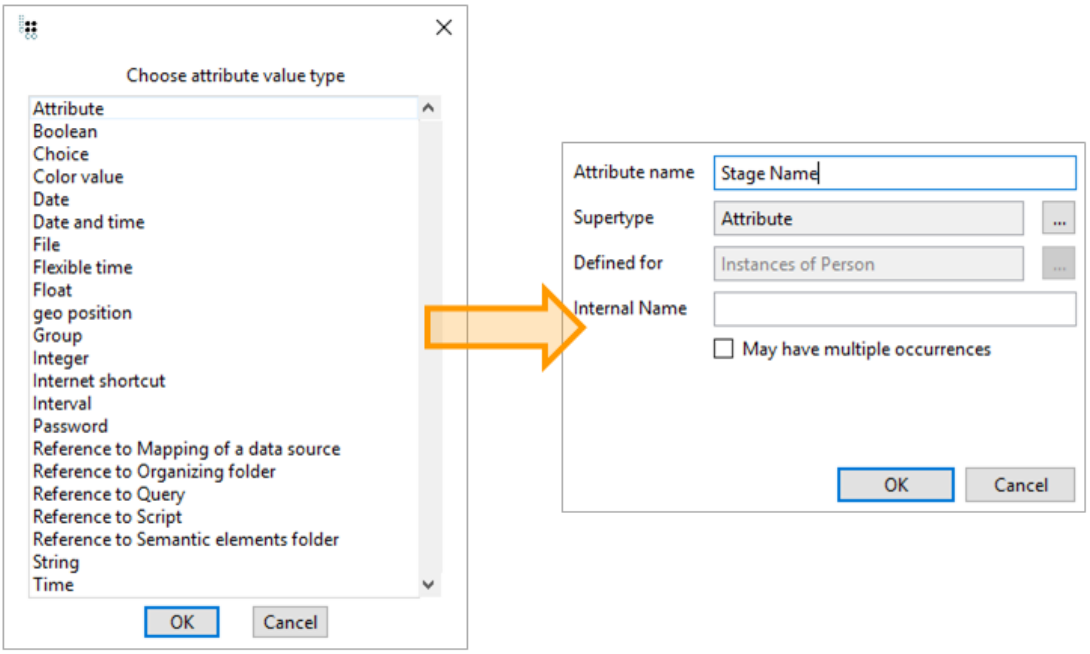


Figure 2. Zweistufiger Dialog zum Anlegen eines neuen Attributtyps

Im Fenster links wird das Format des Attributtyps definiert (Datum, Gleitkommazahl, Zeichenkette usw.)

Folgende technische Datentypen stehen zur Verfügung:

Datentyp	Wie sehen die Werte aus?	Beispiel (Musik-Datenbank)
Attribut	abstraktes Attribut, ohne Attribut-Wert	
Auswahl	frei definierbare Auswahlliste	Bauart eines Musik-Instruments (Hollowbody, Fretless usw.)
Boolesch	"ja" oder "nein"	Musikgruppe noch aktiv?
Datei	externe Datei eines beliebigen Formats, die als "Blob" in den Knowledge Graphen importiert wird	wav-Datei eines Musiktitels
Datum	Datumsangabe tt.mm.jjjj (in der deutschen Spracheinstellung)	Veröffentlichungsdatum eines Tonträgers
Datum und Uhrzeit	Datums- und Uhrzeitangabe tt.mm.jjjj hh:mm:ss	Beginn einer Veranstaltung, bspw. Konzert
Farbwert	Farbauswahl aus Farbpalette	
Flexible Zeit	Monat, Monat + Tag, Jahres-zahl, Uhrzeit, Timestamp	Ungefähres Eintrittsdatum eines Mitglieds in eine Musikgruppe

Datentyp	Wie sehen die Werte aus?	Beispiel (Musik-Datenbank)
Fließkommazahl	Zahlenwert mit beliebiger Anzahl von Nachkommastellen	Preis einer Eintrittskarte zu einer Veranstaltung
Ganzzahl	Zahlenwert ohne Nachkommastellen	Laufzeit eines Musiktitels in Sekunden
Geographische Position	Geographische Koordinaten im WGS84-Format	Ort einer Veranstaltung
Geometrie	Geometrische oder geographische Form im (E)WKT-Format	Polygone, Linien, Punkte, z.B. Form eines Gebäudes auf einer Landkarte
Gruppe	ohne Attributs-Wert, dient als Träger zu gruppierender Meta-Attribute	
Internet-Verknüpfung	URL	Webseite einer Musikgruppe
Intervall	Datumsintervall: Intervalle von Zahlen, Zeichenkette, Zeit- oder Datumswert	Zeitraum zwischen Produktion eines Albums und seiner Veröffentlichung
Passwort	Ein gehashter Wert (SHA256), der zum Validieren des Passworts verwendet wird	
Verweis auf […]	Verweis auf Teile der Konfiguration des Netzes: Suchen, Abbildung einer Datenquelle, Skripte und Ordner. Wird beispiels- weise in der REST- Konfiguration benutzt.	
Zeichenkette	beliebige Folge alphanumerischer Zeichen	Rezensionstext zu einem Tonträger
Zeit	Zeitangabe hh:mm:ss	Startzeit einer Veranstaltung

Nach Auswählen und Bestätigen des Attributtyps kann dieser im Folgedialog mit Attributnamen weiter spezifiziert werden.

Obertyp

Hier wird festgelegt, an welcher Hierarchiestufe der Attributtyp stehen soll.

Interner Name

Wird dafür verwendet, um Attribute per Skript identifizieren und referenzieren zu können.

Definiert für

Hier wird bestimmt bei welchen Typen von Objekten die Attribute angelegt werden können. Beim Anlegen kann der Einfachheit halber an dieser Stelle nur eine Domäne eingetragen werden. Im Attributtyp-Editor (siehe Details bearbeiten) lassen sich im Nachhinein noch weitere Domänen definieren.

Kann mehrfach vorkommen

Attribute können je nach Attributtyp einfach oder mehrfach vorkommen: eine Person hat nur ein Geburtsdatum, kann aber bspw. mehrere akademische Titel zur gleichen Zeit haben (bspw. Doktor, Professor und Honorarkonsul).

1.2.2.2.1. Attributtypen im Detail

Diese Sektion beschreibt Implementierung-Limitationen einzelner Attributtypen sowie solche, die über zusätzliche Konfigurationsoptionen näher bestimmt werden.

Auswahl

Auswahlattribute haben als Wert genau eine Auswahl aus einer Liste von vordefinierten Optionen.

Einträge

In dieser Liste werden die möglichen Auswahloptionen definiert. Übersetzungen und Reihenfolge der Optionen können später über den Schemaeditor spezifiziert werden, siehe Details bearbeiten.

Datei

Speicherung

Dateien können entweder direkt in der Datenbank oder in einem externen Blobservice gespeichert werden.

Datum und Uhrzeit, Zeit

Es ist zu beachten, dass die Speicherung Sekunden-genau ist.

Flexible Zeit

Flexible Zeit ist überall dort nützlich, wo genaue Werte nicht bekannt sind oder keine Rolle spielen. Auch historische Datumsangaben wie "200 v.Chr." sind möglich. Über die Angabe der Formate kann im Schema festgelegt werden, welche Zeitangaben bei der Benutzer-Eingabe akzeptiert werden. Es ist zu beachten, dass die Speicherung bestenfalls Minuten-genau ist. Formate, denen eine Jahresangabe fehlt, sind wiederkehrende Zeitpunkte. Suchen bzw. Vergleiche mit solchen Formaten sind semantisch fragwürdig, weshalb von der Nutzung abzuraten ist. Zukünftige Versionen von i-views werden wiederkehrende Zeitpunkte nicht mehr unterstützen.

Geometrie

Geometrieattribute speichern geometrische Formen (z.B. Punkte oder Linien in einem kartesischen Koordinatensystem) und geographische Formen (z.B. Flächen auf einer Landkarte) im *Well known text (WKT)*-Format, welches auch von gängigen Geoinformationssystemen (GIS) und anderen Datenbanksystemen (z.B. Elasticsearch) unterstützt wird.

Punkt

POINT (10 10)

Linienzug

LINESTRING (10 10, 20 20, 30 40)

Fläche

POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))

Fläche mit Loch

POLYGON ((0 0, 0 20, 20 20, 20 0, 0 0),(5 5, 5 15, 15 15, 15 5, 5 5))

Für geographische Formen ist zusätzlich das geographische Bezugssystem in Form eines EPSG-Identifikators hinterlegt. Das weltweit geläufigste Referenzsystem WGS84 entspricht z.B. dem EPSG-Identifikator 4326. Je nach Referenzsystem müssen die Positionsangaben in Grad oder Metern hinterlegt werden.

HINWEIS In WKT werden Positionen in Grad immer in der Reihenfolge Länge, Breite angegeben, während im Alltag die Reihenfolge Breite, Länge gebräuchlich ist.

Geographische Position 49.87283° N, 8.6512° E

SRID=4326;POINT (8.6512, 49.87283)

Rechteck um München im UTM-System

SRID=25832;POLYGON ((679578 5325187, 702805 5325961, 702165 5344040, 679011 5343266, 679578 5325187))

Bei der Eingabe akzeptiert Geometrieattribute auch das GeoJSON-Format, sowie für geographische Attribute gängige Darstellungsformen wie Länge/Breite, MGRS und UTM. Der Raumbezug wird in diesen Fällen automatisch ermittelt.

Format	Eingabe	Ausgabe
Lat/Long	N49° 52' 18" E8° 39' 0.96"	SRID=4326;POINT(8.6502666667, 49.8716666667)

Format	Eingabe	Ausgabe
UTM	4Q FJ 1 6	SRID=32604;POINT(610000, 2360000)
GeoJSON	`{ "type": "LineString", "coordinates": [[8.62, 49.86], [8.64, 49.87]] }`	SRID=4326;LINESTRING (8.62 49.86, 8.64 49.87)

Beim Anlegen eines neuen Geometrie-Attributtypen stehen folgende zusätzliche Konfigurationsoptionen zur Verfügung:

Hat Messwert

Spezifiziert, ob die Formen über einen zusätzliche dritten Koordinatenbestandteil M verfügen, welcher einen beliebigen Messwert repräsentiert.

```
POINT M (10 10 42)
```

Hat Z-Koordinaten

Spezifiziert, ob die Formen über einen zusätzliche dritten Koordinatenbestandteil verfügen, welcher die Höhe repräsentiert. Für geographische Daten ist dies die Höhe über Normal Null in Metern.

```
POINT Z (10 10 300)
```

Z und M sind auch kombinierbar. Punkte bestehen dann aus vier Koordinatenbestandteilen, wovon der dritte den Z- und der vierte M-Wert ausdrückt.

```
POINT ZM (10 10 300 42)
```

Hat Raumbezugssystem

Legt fest, ob es sich um geographische Daten handelt.

Auf Referenzsystem einschränken

Hier kann eine Komma-separierte Liste von EPSG-Identifikatoren angegeben werden. Es werden dann nur Werte akzeptiert, denen einer der erlaubten Identifikatoren zugeordnet ist.

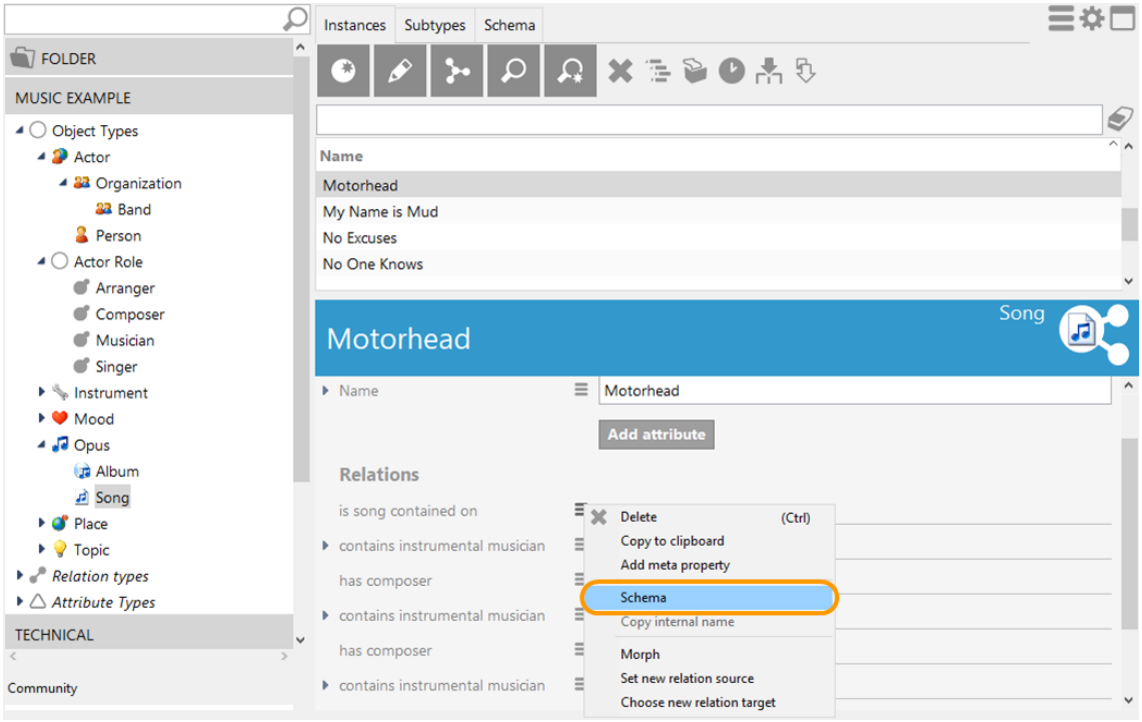
Auf Formen einschränken

Hier wird definiert, welche Formen im Attribut gespeichert werden können. Wenn keine Form spezifiziert ist, sind alle Formen erlaubt.

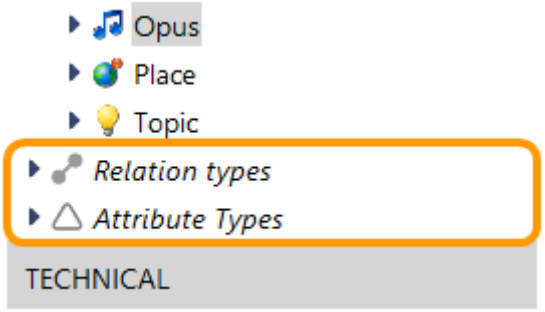
1.2.2.3. Details bearbeiten

Die Dialoge zum Anlegen von neuen Attribut- und Relationstypen sind reduzierte Ansichten der Attribut- und Relationstyp-Editoren. Zur Bearbeitung von Details von Relationen und Attributen müssen Editoren mit erweitertem Funktionsumfang gestartet werden.

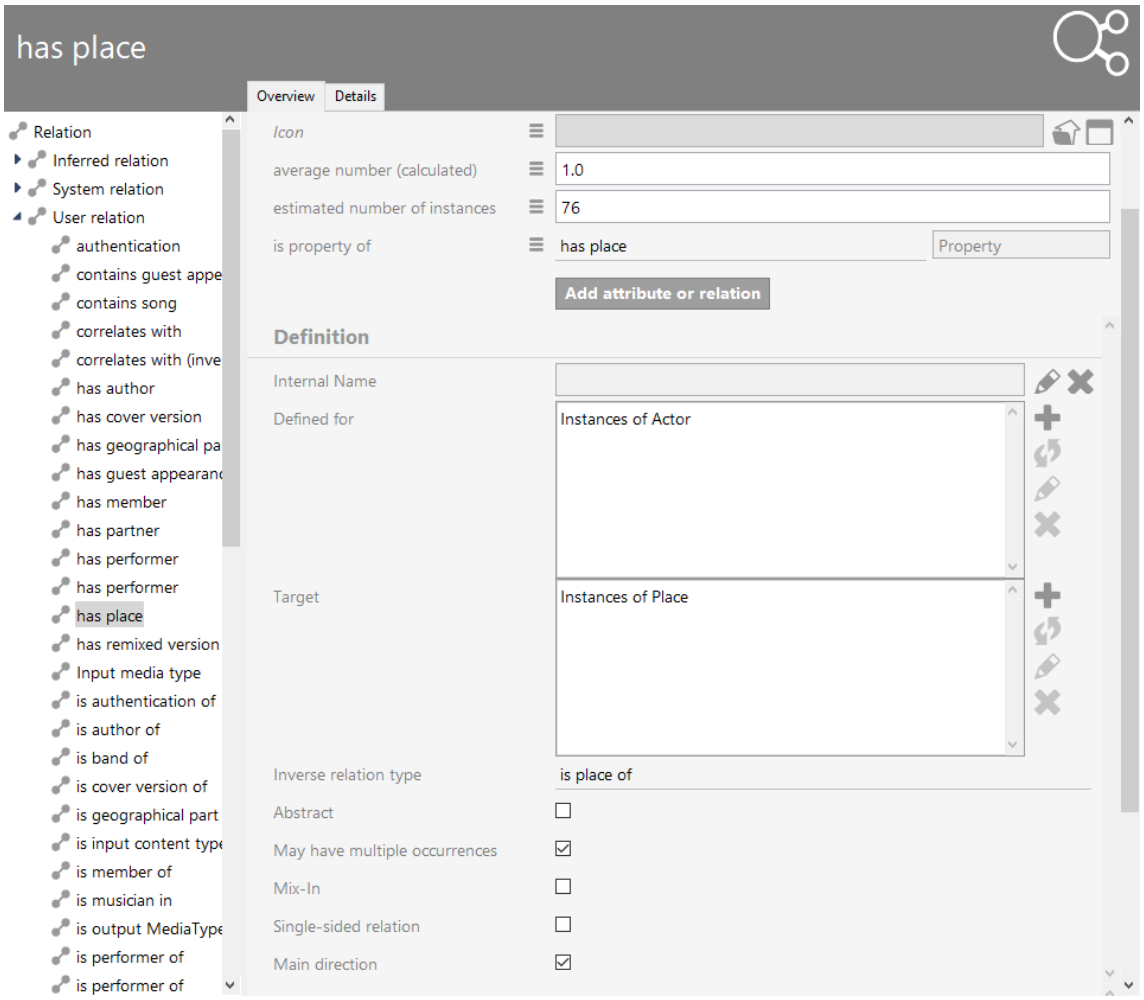
Zu diesen beiden Editoren gelangt man über die Auflistung der Relationen und Attribute im Reiter "Schema" des Objekt-Editors:



Alternativ kann über den Hierarchie-Baum links im Hauptfenster zugegriffen werden. Unter den Objekttypen befinden sich die Hierarchien für Relations- und Attributtypen. Die Editoren werden mit Rechtsklick auf die zu bearbeitende Relation oder Attribut im Kontextmenü mit "Bearbeiten" gestartet.



Im Folgenden schauen wir uns die Details der Definition von Eigenschaften am Beispiel des Relationstyp-Editors an (die Attributtypdefinition ist eine Untermenge davon):



Definiert für

Hier können wir nachträglich ändern, bei welchen Objekttypen die Relation angelegt werden kann. Relationen können zwischen mehreren Objekten definiert werden und damit mehrere Quellen und Ziele haben. So können wir es z.B. im Schema erlauben, dass sowohl Personen als auch Bands Autoren eines Songs sein können oder einem Ort zugeordnet werden — auch wenn sie keinen gemeinsamen Obertyp haben.

Mit der Schaltfläche "Hinzufügen" können wir weitere Objekttypen hinzunehmen. Mit "Entfernen" können wir dem selektierten Objekttyp und allen seinen Objekten die Möglichkeit entziehen diese Relation einzugehen. "Ändern" ermöglicht das Austauschen eines Objekttyps. Bereits existierende Relationen werden dann vom System gelöscht. Falls es zu löschende Relationen gibt, erscheint vor der Durchführung der Änderung ein Bestätigungsdialog.

Ziel

Hier lässt sich nachträglich ändern, zu welchen Typen von Objekten die Relation gezogen werden kann. Um den Ziel-Objekttyp zu ändern, muss zum inversen Relationstyp gewechselt werden: Die Schaltfläche zum Wechseln trägt die Benennung des inversen Relationstyps. Nach

anklicken der Schaltfläche erscheint die inverse Relation im Editor und kann auf dieselbe Weise wie die vorherige Relation bearbeitet werden.

Abstrakt

Wenn wir eine Relation definieren wollen, die nur zur Gruppierung dient aber selber keine konkreten Eigenschaften ausprägen soll, dann definieren wir sie als "abstrakt"

Beispiel: Wenn die Relation "schreibt Song" als abstrakt definiert wird, bedeutet dies: wenn wir Songs und ihre Relation zu Künstlern oder Bands anlegen, können wir nur spezifische Angaben machen (wer hat den Text geschrieben, wer die Musik). Die unspezifische Relation "schreibt Song" kann nicht in den konkreten Daten angelegt werden, sondern nur für Abfragen verwendet werden.

Kann mehrfach vorkommen

Ein Merkmal von Relationen ist es, ob sie mehrfach vorkommen können. Beispiel: die Relation "hat Geburtsort" kann für jede Person nur einmal auftreten, während bspw. die Relation "ist Mitglied von" mehrfach für eine Person auftreten kann. Somit lassen sich logische Sachverhalte präzise modellieren. Beispielsweise können Musiker als Person nur einen Geburtsort haben, aber (auch zeitgleich) Mitglied in mehreren Musikgruppen sein. Ob die Relation mehrfach vorkommen kann, wird für beide Richtungen der Relation unabhängig angegeben: Eine Person ist nur an einem Ort geboren, der Ort kann aber wiederum Geburtsort von mehreren Personen sein.

Die Option kann nur ausgeschaltet werden wenn die Relation im tatsächlichen Datenbestand nicht mehrfach vorkommt. Bei mehrfachen Vorkommen kann das System nicht automatisch entscheiden, welche der Relationen entfernt werden soll.

Mix-In

Mix-In-Eigenschaften werden im Kapitel Erweiterungen erklärt.

Einseitige Relation

Wenn die Relationsrichtung viele Ausgangsobjekte zu einem "Hot Spot"-Objekt zusammenführt, dient eine Einseitige Relation zur Verbesserung der Zugriffsperformance. Weitere Informationen hierzu sind im Unterkapitel zu "Neuen Relationstyp anlegen" zu finden.

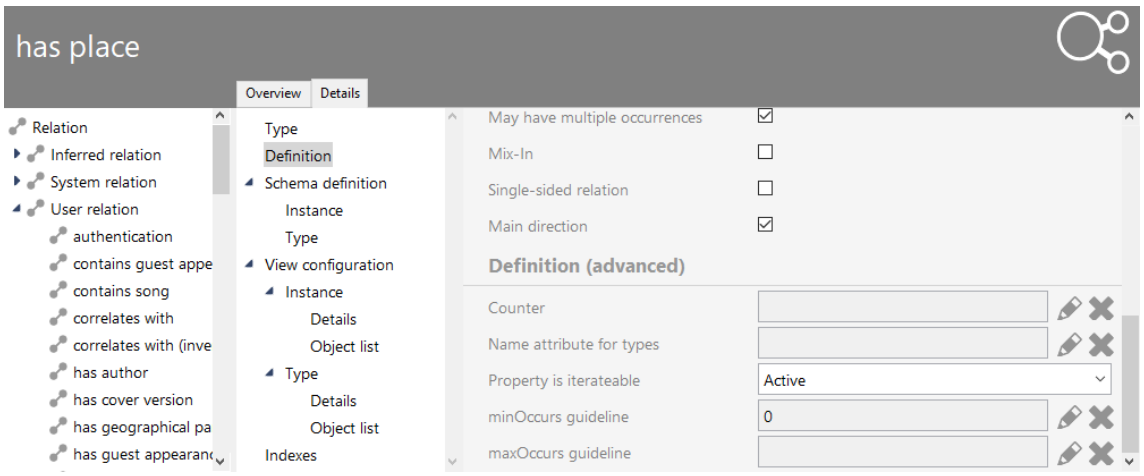
Hauptrichtung

Zu jeder Beziehung gehört die Gegenrichtung. Im Kern sind beide Richtungen gleichwertig, aber es gibt zwei Stellen, wo es hilfreich ist eine Hauptrichtung zu bestimmen:

- Im Graph-Editor: Hier stellen sich die Relationen, was Pfeilrichtung und Beschriftung angeht, immer in Hauptrichtung dar — unabhängig davon in welche Richtung sie angelegt wurden.
- Bei einseitigen Relationen (ohne Rückrelation)

Weitere Einstellungsmöglichkeiten für Relationen und Attribute finden sich im Reiter "Details" im Unterpunkt "Definition". Die Einstellungsmöglichkeiten unter Definition werden oft gebraucht und sind darum auch bereits im Übersichtsreiter vorhanden. Unter "Definition (erweitert)" finden sich

hingegen Einstellungsmöglichkeiten, die nicht so oft gebraucht werden.



Zähler

Wenn eine Zahl im Zähler eingegeben wird, ist das die Zahl, mit der Objekte dieses Typs weiter hochgezählt werden. Über die JavaScript-Funktionen getCounter(), increaseCounter() und setCounter() kann auf den Zähler zugegriffen werden.

Namensattribut für Objekte

Typischerweise wird in vielen Ansichten in i-views ein Objekt über seinen Namen repräsentiert (z.B. in Objektlisten, Hierarchien, im Graph-Editor, der Relationszielsuche, etc.). Statt des Namen kann man hier ein beliebiges anderes Attribut der Objekte verwenden, mit denen es repräsentiert werden kann. Prominentes Beispiel bei Produkten: Die Artikelnummer.

HINWEIS	Nur an Objekttypen einstellbar, nicht an Relations- oder Attributtypen

Namensattribut für Typen

Hiermit kann auch für Typen ein alternatives Attribut zur repräsentativen Anzeige gewählt werden.

Eigenschaft ist iterierbar

Auswahlmöglichkeiten: Aktiv / Nur schreibend / inaktiv. Default: Aktiv.

Manchmal kommt es vor, dass die Pflege des Index zum Iterieren von Eigenschaften die Performance stark verlangsamt. Typischerweise ist dies bei Meta-Eigenschaften wie "geändert von" oder "geändert am" der Fall, die nicht unbedingt immer berücksichtigt werden müssen. In solchen Fällen wird empfohlen diese Eigenschaften auf nicht iterierbar zu schalten, indem hier die Auswahlmöglichkeit "inaktiv" verwendet wird. "Nur schreiben" dient dazu, zunächst nur den lesenden Zugriff zu verbieten, aber den schreibenden noch zu erlauben. Auf diese Weise kann getestet werden, ob ungewollte Seiteneffekte auftreten.

Richtwert für minimales Auftreten

Dieser Richtwert gibt an, wie oft die Eigenschaft mindestens an einem Objekt vorkommen soll. Wenn die angegebene Zahl unterschritten wird, dann wird die Eigenschaft im User-Interface

zwar rot dargestellt, das Objekt kann aber dennoch existieren. Ein Import ignoriert den Richtwert.

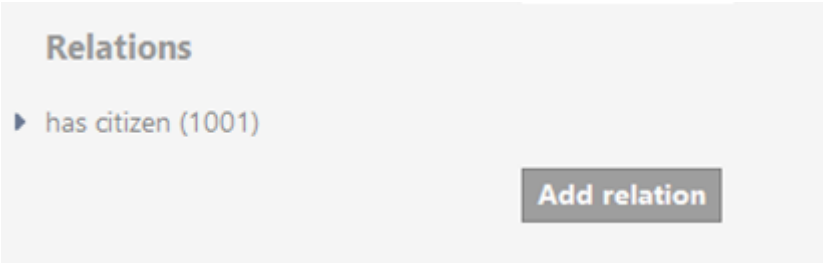
Richtwert für maximales Auftreten

Dieser Richtwert gibt an, wie oft die Eigenschaft maximal an einem Objekt vorkommen soll. Wird die angegebene Zahl erreicht, können keine weiteren Eigenschaften angelegt werden. Ein Import ignoriert den Richtwert.

1.2.2.4. Einseitige Relationen

1.2.2.4.1. Anwendungsgebiet für einseitige Relationen — Grundlagen

Wenn zum Zweck des Imports oder der Anzeige in der View-Configuration ein Objekt aus dem Datenbestand aufgerufen werden muss, dann werden (auch im Falle unzureichender Indexierung) alle Eigenschaften des Objekts in den Speicher geladen.



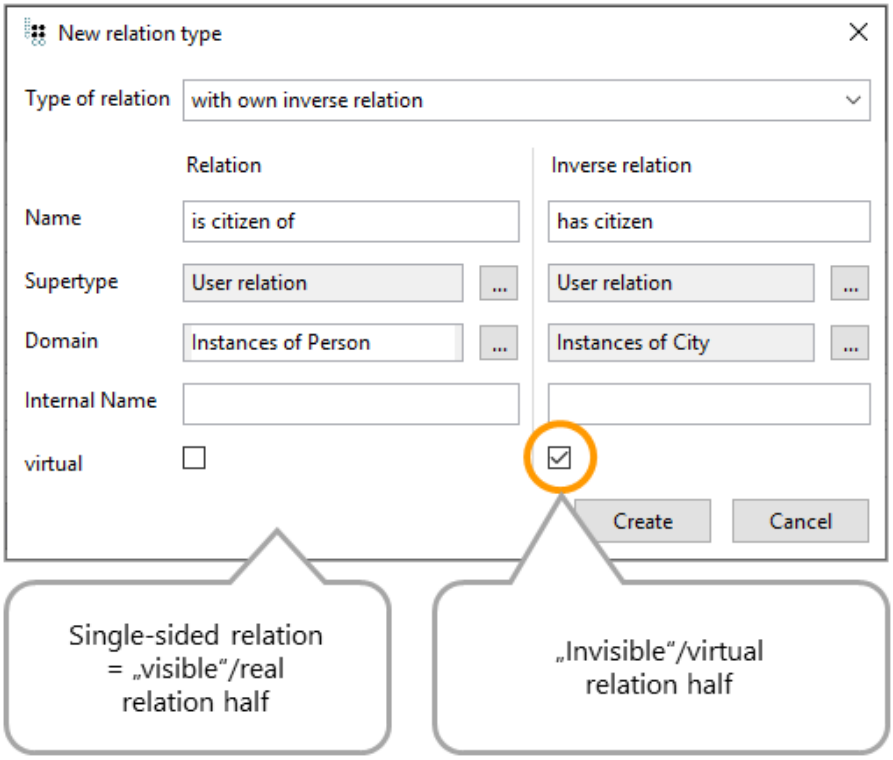
Insbesondere bei *Katalog-Objekten* können unnötigerweise längere Ladezeiten auftreten, weil dadurch Eigenschaften mitgeladen werden, die für die Anzeige gar nicht benötigt werden. Mit einem Katalog-Objekt ist dabei ein Objekt gemeint, das als zentrale Referenz für viele andere Objekte dient.

In einem Knowledge Graph gibt es Objekte des Typs "Stadt", die per Relation mit den Einwohnern verknüpft ist. Wenn nun eine Detailansicht der Stadt geladen werden soll, die lediglich die Anzahl der Einwohner anzeigen soll, nicht aber alle Einwohner mit ihren Namen, Adressen und Hobbies, dann ist die Verwendung einseitiger Relationen sinnvoll.

In diesem Fall verläuft die einseitige Relation von den einzelnen Satelliten-Objekten hin zum Katalog-Objekt. Die Folge hiervon ist, dass nur bei den Einwohnern die Relation "ist Einwohner von" sichtbar ist, bei der Stadt aber keine Beziehung zu den einzelnen Einwohnern sichtbar ist. Trotzdem ist als Gegenrichtung die *"virtuelle"* Relationshälfte "hat Einwohner" für Strukturabfragen zur Auswertung verwendbar und im Schema auffindbar.

1.2.2.4.2. Das Definieren einer einseitigen Relation

Zum Definieren einer einseitigen Relation müssen wir im Dialog angeben, welche der Relationshälfte (Hin- oder Rückrichtung) virtuell, also unsichtbar bleiben soll. Hierzu wählen wir die Checkbox "virtuell" an. Die andere Relationshälfte wird somit automatisch zur reellen Relationshälfte, die die einseitige Relation zwischen Start- und Zieldomäne bildet.

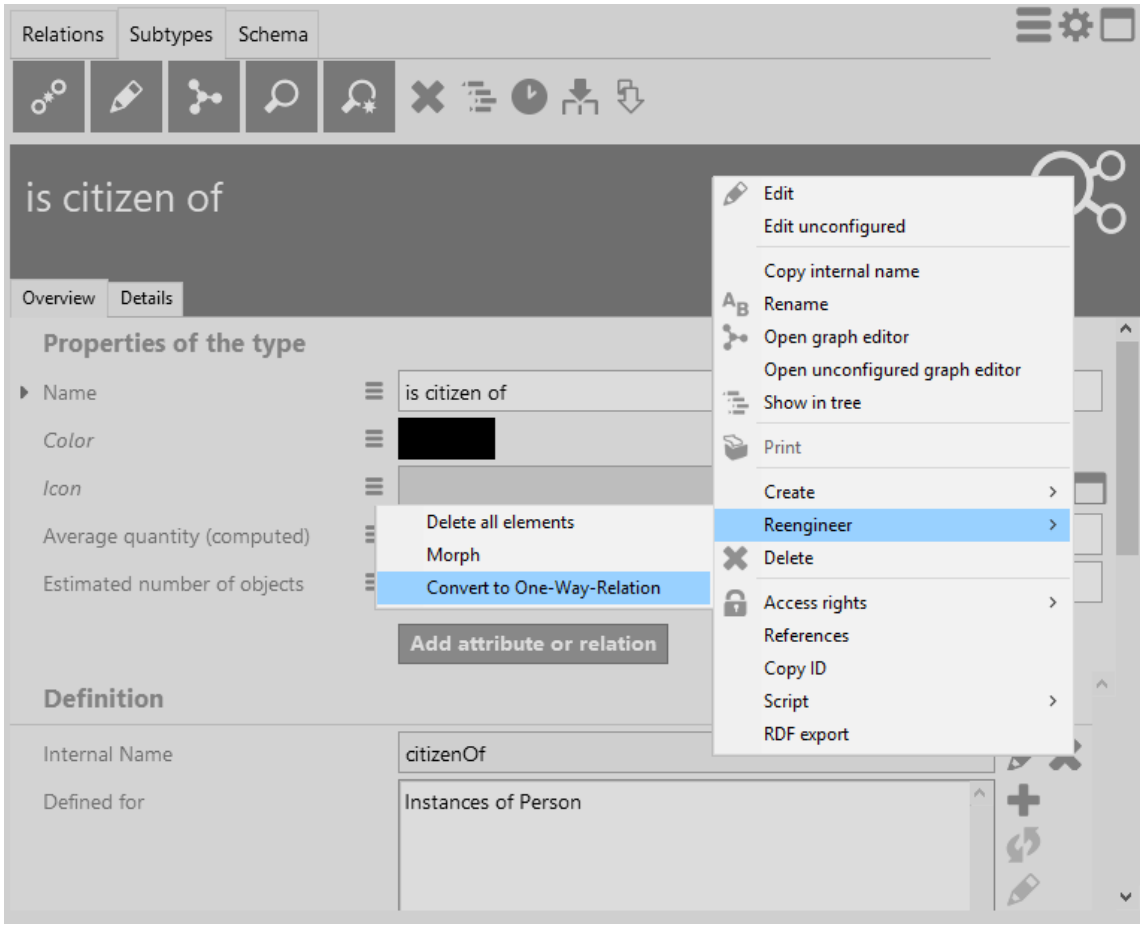


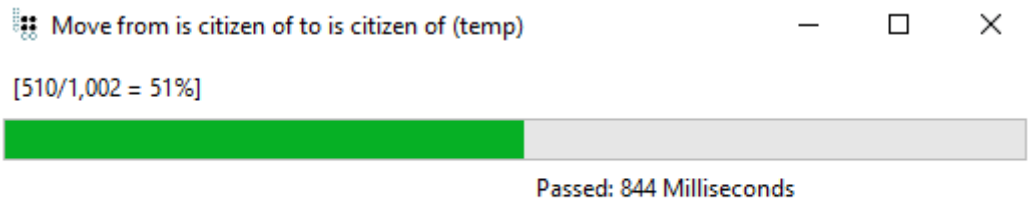
1.2.2.4.3. Nachträgliches Deklarieren einer beidseitigen Relation als einseitige Relation

Wenn ein bislang beidseitiger Relationstyp in einen einseitigen Relationstyp umgewandelt wird, dann werden die die Instanzen der virtuellen Relationstyphälfte automatisch gelöscht. Dieser Vorgang kann bei erneuter Umstellung rückgängig gemacht werden; in diesem Fall werden die jeweiligen Relationshälften erneut angelegt.

Die Umstellung auf einseitige Relationen wirkt sich folgendermaßen aus: Für ein Katalog-Objekt werden die virtuellen Relationshälften mit ihren Relationszielen nicht mehr angezeigt, werden aber dennoch als Instanz im Knowledge-Graph repräsentiert und können für Strukturabfragen verwendet werden.

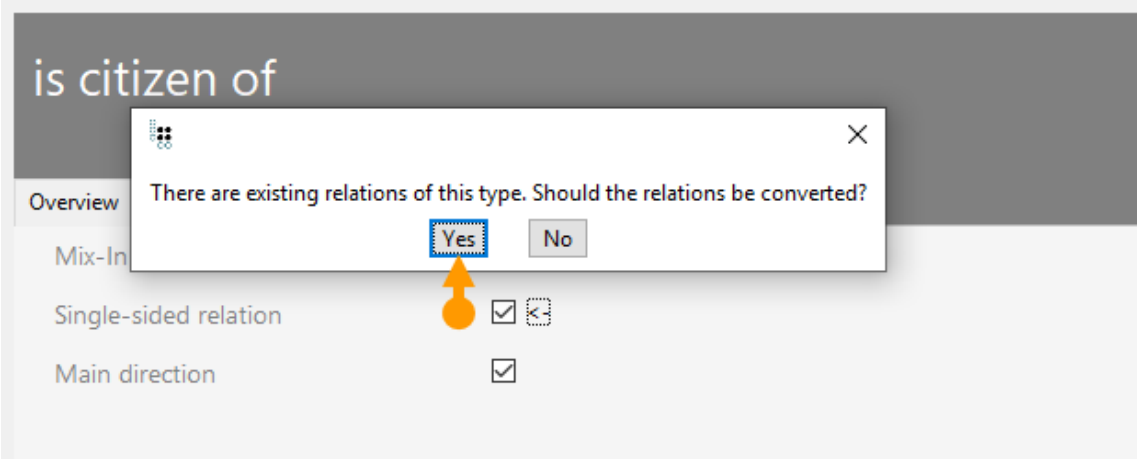
Im Idealfall wird für den Import großer Objektmengen zu einem Katalog-Objekt die reelle, einseitige Relationstyphälfte verwendet. Dies kann zu einer Beschleunigung des Importes führen.





Als Resultat zeigt die Checkbox "Einseitige Relation" an, dass die Relationshälfte als einseitige Relation verwendet wird.

HINWEIS	Bis zur Version 5.3 dient die Checkbox des booleschen Attributs "Einseitige Relation" nur zu Anzeige-Zwecken. Seit i-views 5.4 kann eine Überarbeitung der Relationsart kann über das Kontextmenü oder durch Klick auf die Checkbox vorgenommen werden.

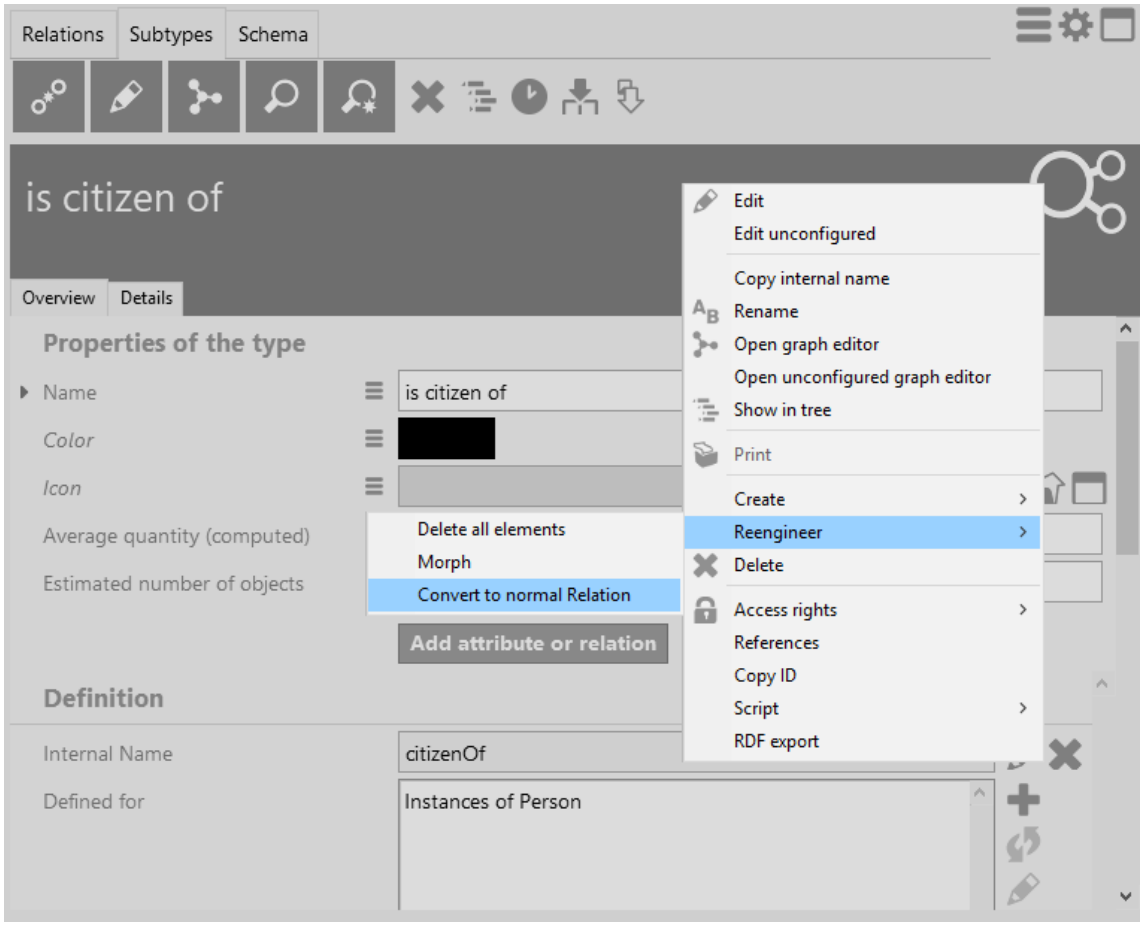


HINWEIS

Nach einer Umstellung auf einseitige Relationen kann die Performance der Abfragen für die **virtuelle** Relationsrichtung verbessert werden, wenn der virtuellen Relationshälfte ein Index vergeben wird.

1.2.2.4.4. Nachträgliches Umändern einer einseitigen Relation hin zur beidseitigen Relation

Wenn wir im Nachhinein feststellen, dass ein Relationstyp eigentlich beidseitig hätte deklariert werden müssen, so kann dies problemlos im Nachhinein geändert werden. Hierzu klicken wir beim Relationstyp auf das Kontextmenü und wählen Überarbeiten > In normale Relation umwandeln oder deaktivieren die Checkbox "Einseitige Relation".



Der Knowledge-Builder ändert anschließend alle bestehenden virtuellen und einseitigen Relationen in beidseitige Relationen um.

1.2.2.4.5. Nachträgliches Umändern der Orientierung der einseitigen Relation

Die nachträgliche Umänderung der Orientierung der einseitigen Relation erfolgt analog über das Kontextmenü per Befehl "Überarbeiten" oder über die Checkbox. Hierzu wechseln wir zum Detaileditor der Relationsrichtung, die von virtuell auf einseitig umgestellt werden soll und wählen im Kontextmenü Überarbeiten > In einseitige Relation umwandeln oder wählen die Checkbox "Einseitige Relation".

1.2.3. Modelländerungen

In i-views lassen sich am Modell zur Laufzeit Änderungen durchführen:

- neue Typen einführen
- die Typenhierarchie beliebig ändern (ohne dafür Tabellen anlegen und uns um Primär- und Fremdschlüssel Gedanken zu machen).

Systemseitig wird für Konsistenz gesorgt. Beim Anlegen von Objekten und Eigenschaften wird die Gegenrichtung einer Relation automatisch mitgezogen. Attributwerte werden darauf geprüft, ob sie zum definierten technischen Datentyp passen (in ein Datumsfeld können wir beispielsweise keine

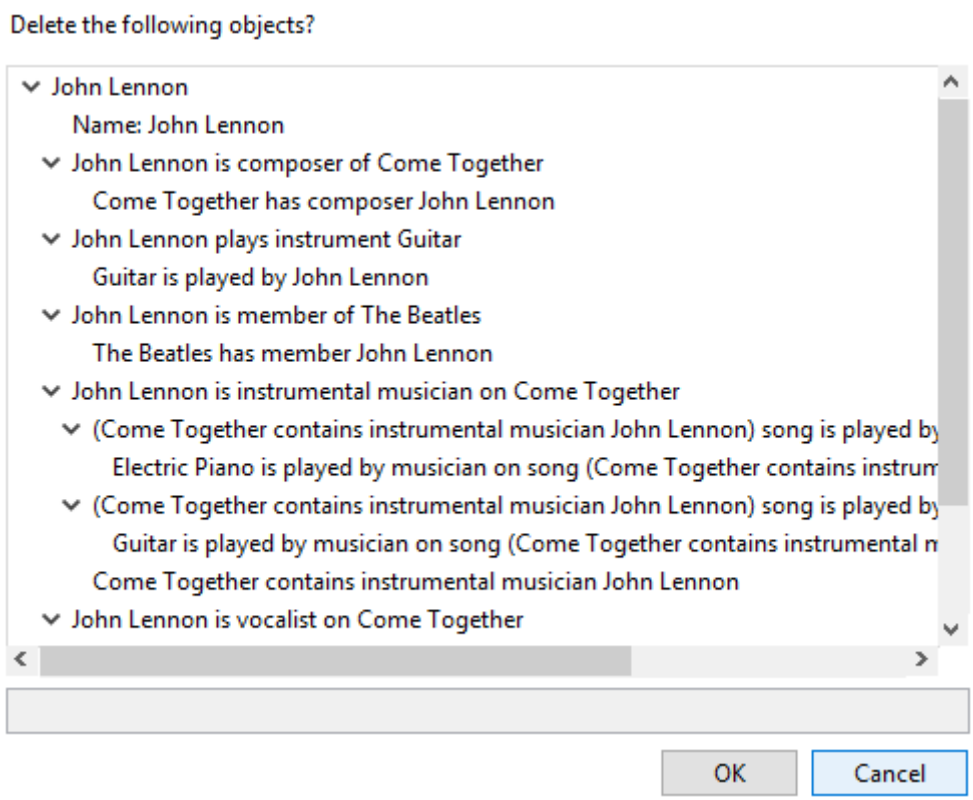
beliebige Zeichenkette eintragen).

Konsistenz ist auch beim Löschen wichtig: Abhängige Elemente müssen immer mit gelöscht werden, sodass keine Reste von Daten gelöschter Elemente in der semantischen Graphdatenbank zurückbleiben.

- Wenn also ein Objekt gelöscht wird, werden auch alle seine Eigenschaften mit gelöscht. Wenn wir z.B. das Objekt "John Lennon" löschen, dann löschen wir damit sein Geburtsdatum, seinen Biographie Text, den wir als Freitext-Attribut bei jeder Person haben können etc. Es wird ebenfalls auch seine Relation "ist Mitglied bei" zu den Beatles und "ist liiert mit" zu Yoko Ono. Die Objekte "The Beatles" und "Yoko Ono" werden nicht gelöscht, sie verlieren lediglich ihre Verbindung zu John Lennon.
- Beim Löschen einer Relation wird automatisch die Gegenrichtung mit gelöscht.

Da i-views immer dafür sorgt, dass die Objekte und Eigenschaften dem Schema entsprechen, ist das Löschen eines Objekttyps ggf. eine Operation mit weitreichenden Konsequenzen: Wenn ein Objekttyp gelöscht wird, werden alle seine konkreten Objekte gelöscht — analog bei Beziehungs- und Attributtypen. **

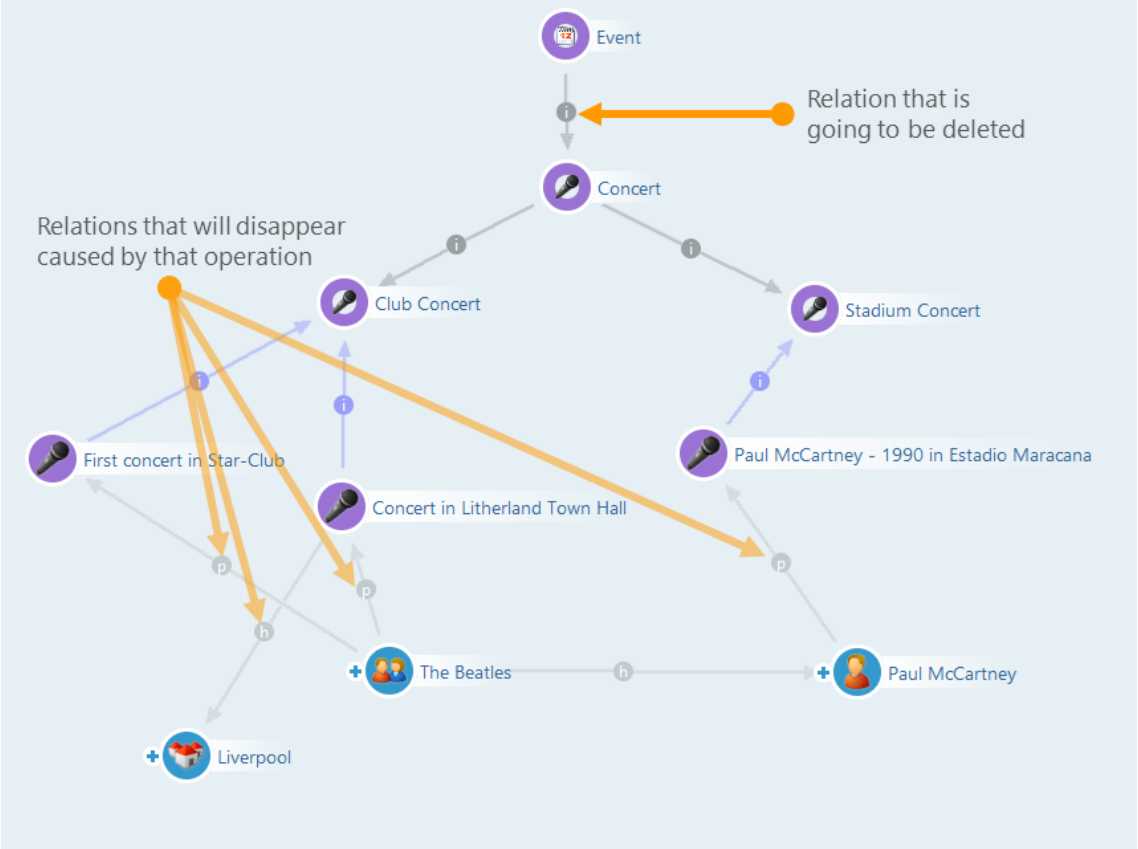
Dabei informiert i-views immer über die Konsequenzen einer Operation. Wenn ein Objekt gelöscht werden soll, listet i-views alle Eigenschaften, die damit wegfallen im Bestätigungsdialog der Löschoperation auf:



i-views kontrolliert, wo durch die Änderung Objekte, Beziehungen oder Attribute verloren gehen. Der

Benutzer wird auf die Konsequenzen der Löschung hingewiesen.

Nicht nur das Löschen, auch das Umwandeln und Ändern der Typenhierarchie kann Konsequenzen nach sich ziehen. Etwa wenn Objekte Eigenschaften haben die bei Typwechsel oder bei Wechsel in der Vererbung nicht mehr dem Schema entsprechen.



Nehmen wir an, wir löschen die Relation "ist Obertyp von" zwischen "Veranstaltung" und "Konzert" und lösen damit den Objekttyp "Konzert" und alle seine Untertypen aus der Vererbungshierarchie von Event heraus, um ihn z.B. unter "Werk" zu hängen. Hier macht uns i-views darauf aufmerksam, dass die "hat Teilnehmer" Relationen der konkreten Konzerte wegfallen würden. Diese Relation ist nämlich bei "Veranstaltung" definiert und würde damit für die Konzerte nicht mehr gelten.

Es gibt Möglichkeiten zu verhindern, das durch Modelländerungen zu erhaltende Relationen wegfallen. Soll bspw. ein Objekttyp innerhalb der Typenhierarchie umziehen, muss zuerst das Schema der betroffenen Relation vorher angepasst werden.

Beispielsweise soll "Konzert" in der Hierarchie nicht mehr unter "Veranstaltung", sondern unter "Werk" verortet werden. Hierzu wird der Relation "hat Teilnehmer" eine zweite Quelle zugewiesen: das kann entweder der Objekttyp Konzert selbst sein, oder die neue Position "Werk". Die Relation geht somit nicht verloren.

Die Typenhierarchie wird von i-views besonders berücksichtigt. Wenn wir aus der Mitte der Hierarchie einen Typ heraus löschen oder eine Ober-/Untertyp-Relation entfernen, dann schließt i-views die entstehende Lücke und hängt die Typen, denen der Obertypen abhanden gekommen ist,

wieder in die Typenhierarchie ein — und zwar so dass die seine Eigenschaften möglichst weitgehend bewahrt bleiben.

Bei Änderungen am Schema ist Grundsätzlich zu bedenken, dass eine Wiederherstellung eines vorherigen Zustandes nur durch Einspielen eines Backups möglich ist. Analog zu relationalen Datenbanken existiert keine "Rückgängig"-Funktion.

1.2.3.1. Spezielle Funktionen

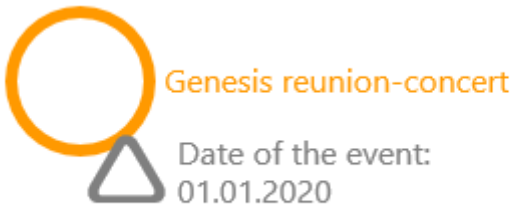
1.2.3.1.1. Typ wechseln

Bereits in der semantischen Graphdatenbank vorhandene Objekte lassen sich zu Objekten eines anderen Typs verschieben. Bspw. ist der Objekttyp "Veranstaltung" ausdifferenziert in "Sportveranstaltung" und "Konzert". Sind bereits Objekte des Typs Sportveranstaltung oder Konzert in der semantischen Graphdatenbank vorhanden, können diese in der Liste im Hauptfenster selektiert und ganz einfach mit Drag&Drop unter einen neuen, passenderen Objekttyp verschoben werden.

Alternativ dazu finden wir im Kontextmenü den Punkt "Überarbeiten".

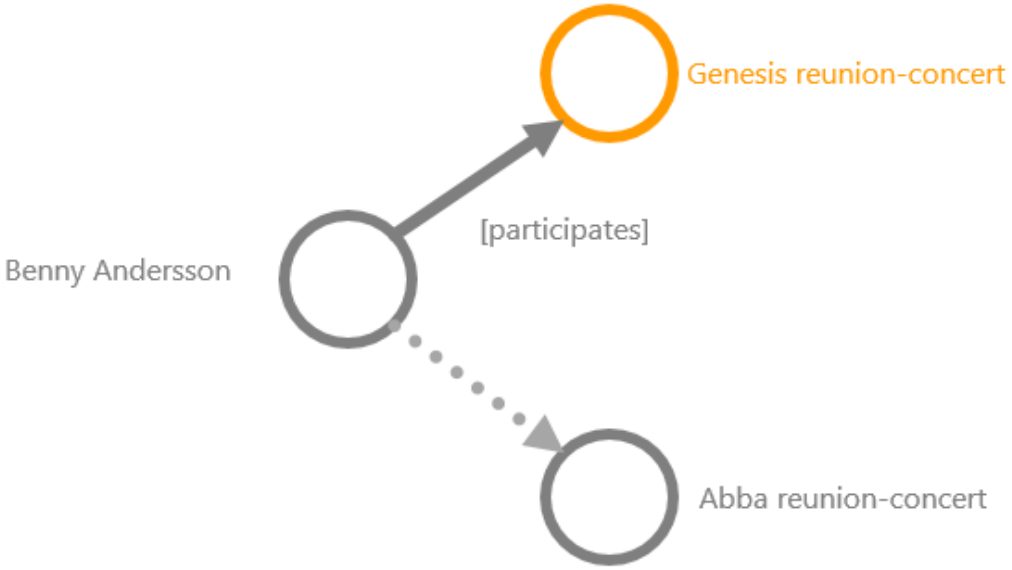
1.2.3.1.2. Typ auswählen

Mit dieser Operation können wir eine Eigenschaft einem neuen Objekt zuweisen.



1.2.3.1.3. Relationsziel neu wählen

Bei Relationen gilt das nicht nur für die Quelle, sondern auch für das Relationsziel.

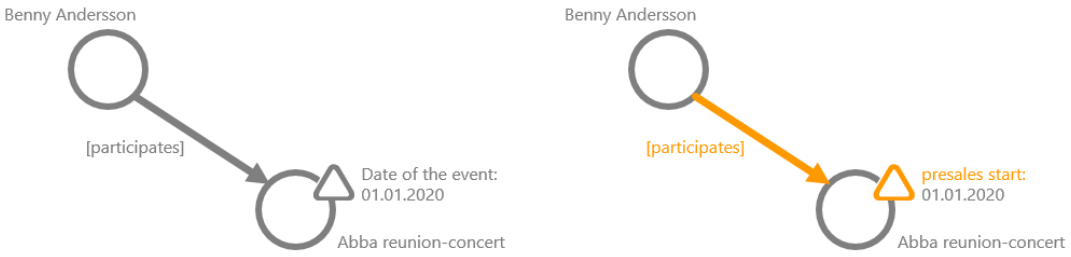


1.2.3.1.4. Untertypen in konkrete Objekte umwandeln (und umgekehrt)

Die Grenze zwischen Objekttypen und konkreten Objekten ist in vielen Fällen offensichtlich, aber nicht immer. Statt nur einen Objekttyp namens "Musikrichtung" einzurichten, wie in unserem Beispielprojekt, hätten wir auch eine ganze Typenhierarchie von Musikrichtungen aufbauen können (Wir haben uns in diesem Netz dagegen entschieden, weil die Musikrichtungen so unterschiedliche Dinge wie Bands, Alben und Songs klassifizieren, und sie daher keine guten Typen abgeben). Es kann aber passieren, dass wir uns mitten in der Modellierung um entscheiden. Aus diesem Grund gibt es die Möglichkeit, Untertypen in konkrete Objekte umzuwandeln und konkrete Objekte in Untertypen. Eventuell bestehende Relationen gehen dabei verloren, falls sie nicht zum neuen Schema passen.

1.2.3.1.5. Umwandeln

Diese Funktion wandelt Eigenschaften eines Typs zu einem anderen Typ um. Für Relationen bleiben Quelle und Ziel bestehen, nur der Relationstyp wird umgewandelt. Für Attribute bleibt die Quelle erhalten, es ist aber in einigen Fällen möglich, den Wertetyp zu wechseln (z.B. ein Ganzzahl-Attribut in ein Fließkomma-Attribut umzuwandeln).



Bei Umwandlung der einzelnen Relation sind wir in der Regel schneller, wenn wir diese löschen und durch eine andere ersetzen. Jetzt kann es aber vorkommen, dass an den Eigenschaften Meta-Eigenschaften hängen, die wir nicht verlieren möchten. Zum Andern stehen die Umwandeln-Operationen auch für alle Eigenschaften eines Typs bzw. eine Auswahl davon zur Verfügung. Voraussetzung ist natürlich, dass der neue Relations- oder Attributtyp für die Quell- und Zielobjekte auch definiert ist.

1.2.3.1.6. Schemaüberarbeitung für Geometrie-Attribute

Für Geometrie-Attributtypen stehen im *Überarbeiten*-Untermenü folgende zusätzliche Funktionen zur Verfügung:

Raumbezug hinzufügen

Wandelt ein geometrisches in ein geographisches Attribut um. Dabei muss der Identifikator eines Raumbezugssystems angegeben werden. Die Werte werden ohne Umrechnung übernommen, aus einem Attribut mit Wert `POINT(9 50)` wird also z.B. `SRID=4326;POINT(9 50)` (50°N 9°E).

Raumbezug entfernen

Wandelt ein geographisches in ein geometrisches Attribut um. Die Werte werden ohne Umrechnung übernommen, egal welchen Raumbezug sie vorher hatten. Aus einem Attribut mit Wert `SRID=4326;POINT(9 50)` wird also z.B. `POINT(9 50)`, aus `SRID=3857;POINT (1001875 6446275)` wird `POINT (1001875 6446275)`.

Messwert hinzufügen/entfernen

Fügt einen M-Koordinatenbestandteil hinzu oder entfernt existierende Messwerte. Entfernte Werte können nicht wiederhergestellt werden! Beim Hinzufügen kann ein Standardwert angegeben werden, den alle existierenden Attribute bekommen.

Z-Koordinate hinzufügen/entfernen

Fügt einen Z-Koordinatenbestandteil hinzu oder entfernt existierende Z-Koordinaten. Entfernte Werte können nicht wiederhergestellt werden! Beim Hinzufügen kann ein Standardwert angegeben werden, den alle existierenden Attribute bekommen.

Raumbezug umwandeln

Hiermit lassen sich vorhandene geographische Attribute in ein neues Raumbezugssystem umrechnen.

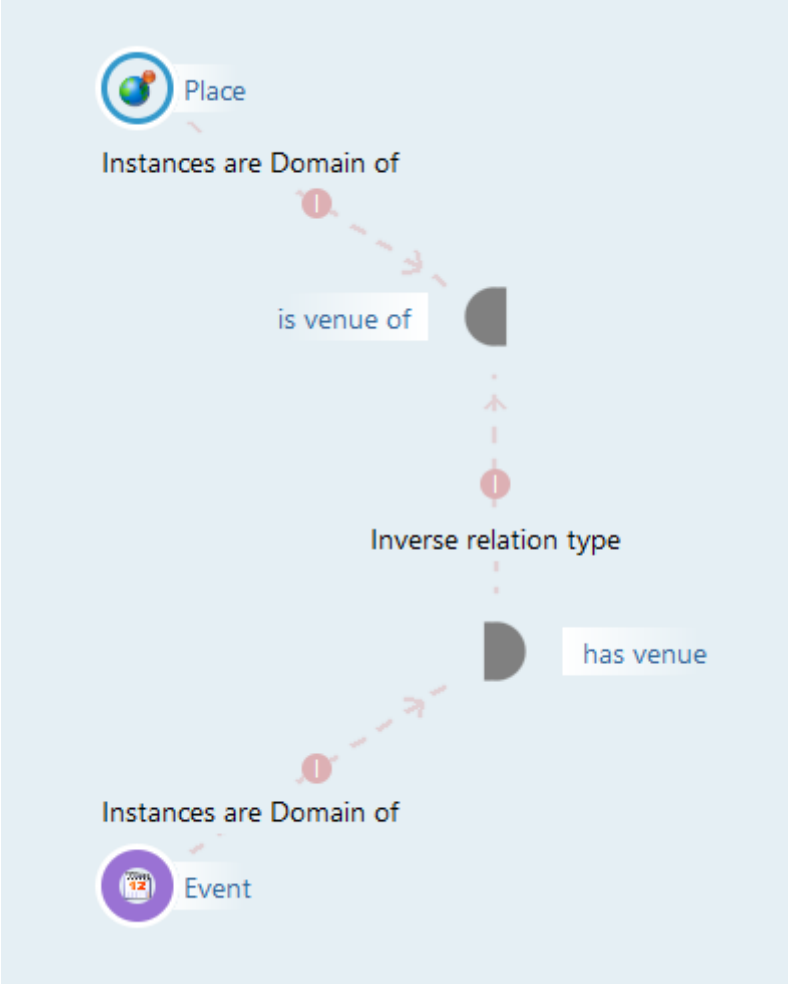
HINWEIS	Für diese Funktion wird die externe Softwarebibliothek *proj* benötigt. Weitere Informationen können dem technischen Handbuch entnommen werden.

1.2.4. Darstellung von Schema im Graph-Editor

Bisher hatten wir es im Graph-Editor hauptsächlich die Vernetzung konkreter Objekte zu tun. Sich solche konkreten Beispiele vor Augen zu führen, mit anderen zu diskutieren und sie ggf. zu

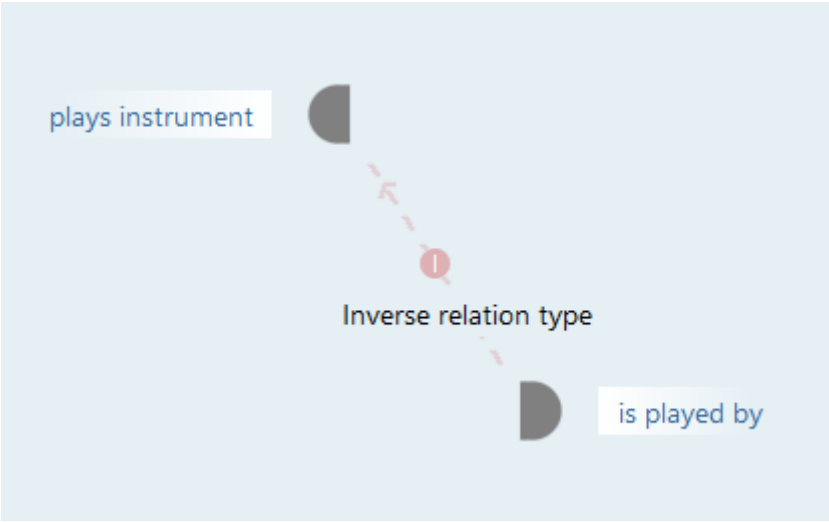
bearbeiten ist auch die Hauptfunktion des Graph-Editors. Wir können aber mit dem Graph-Editor auch das Schema des Knowledge Graphen direkt darstellen, beispielsweise die Typenhierarchie eines Netzes.

Typen von Objekten werden dabei als farbig hinterlegte Knoten, Typen von Relationen werden als gestrichelte Linie dargestellt:

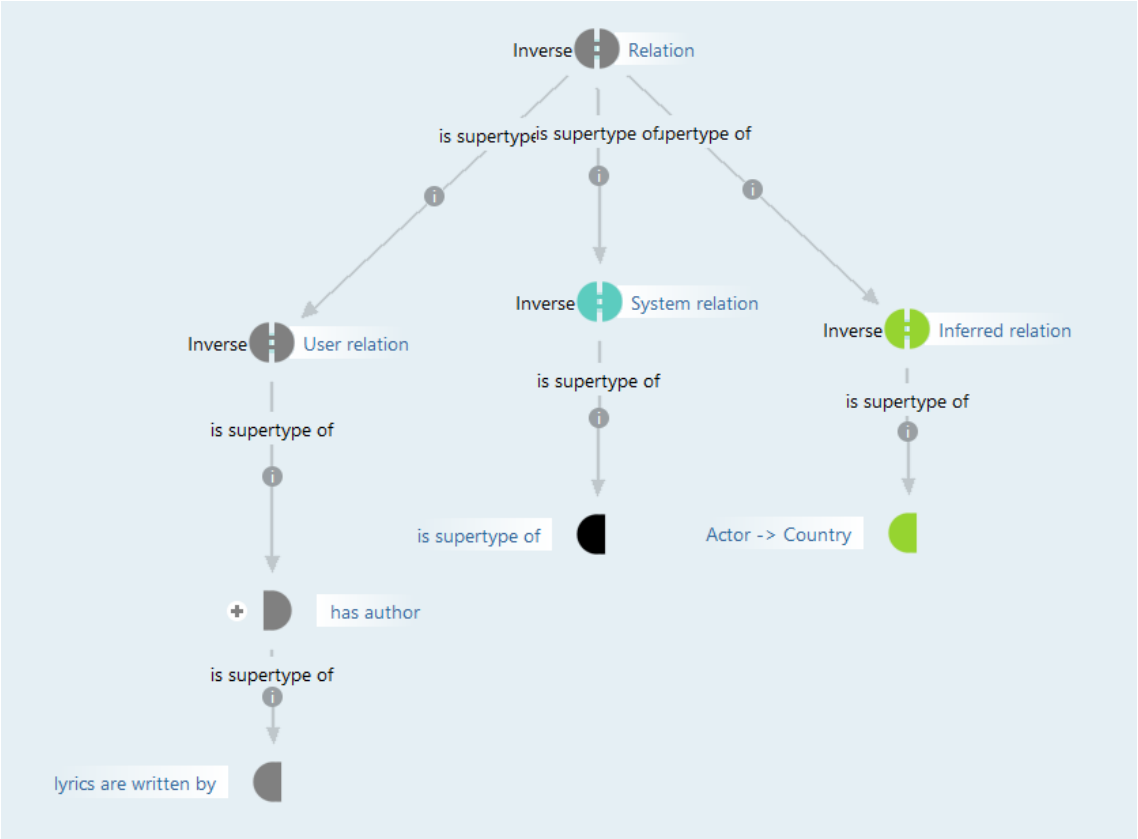


Relationsbegriffe im Graph-Editor

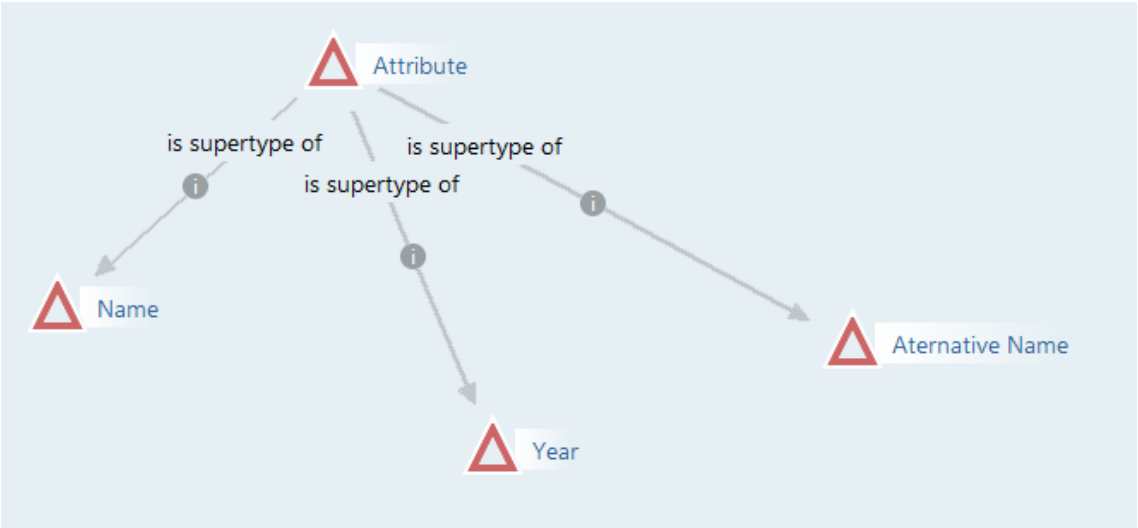
Wenn bisher von Relationen im Graph-Editor die Rede war, so waren damit Relationsobjekte zwischen bestimmten Objekten des Knowledge Graphen gemeint. Aber auch die allgemeinen Relationstypen (also die Schemata der Relationen) lassen sich im Graph-Editor darstellen. Eine Relation wird im Graph-Editor durch zwei Halbkreise dargestellt, die die beiden Richtungen (Hauptrichtung und inverse Richtung) repräsentieren. Zwischen diesen beiden Knoten besteht also die Beziehung "Inverser Relationstyp":



Die Darstellung eines Relationsbegriffs und der Hierarchie im Graph-Editor kann analog zum Objekteditor mit allen Ober- und Untertypen dargestellt werden:



Auch Attributtypen lassen sich im Graph-Editor anzeigen — Sie werden durch dreieckige Knoten repräsentiert.



Analog zur Objekttyp-Hierarchie kann auch die Hierarchie der Relationen und Attribute im Graph-Editor durch Löschen und Ziehen der Obertyp-Relation verändert werden.

1.2.5. Metamodellierung und fortgeschrittene Konstrukte

1.2.5.1. Dynamische Typisierung

Objekte haben bei ihrer Erstellung genau einen Typ, den sie über ihre Lebensdauer nicht wechseln. So ist beispielsweise das Objekt "George" Zeit seines Lebens vom Typ "Person". Dabei bestimmt der Typ, welche Eigenschaften ein Objekt haben kann. "George" kann als "Person" beispielsweise die Eigenschaft "Nachname" haben.

Möchte man einem Objekt dynamisch weiteren Typen zuordnen, dann kann man das mittels sogenannter "Ergänzungstypen". Die Menge der Eigenschaften, die ein Objekt haben kann, bestimmt sich dann zusätzlich über die zugeordneten Ergänzungstypen. Ist "George" beispielsweise auch vom Ergänzungstyp "Produzent", dann kann er eine Relation "produziert" zu einem Objekt vom Typ "Album" eingehen.

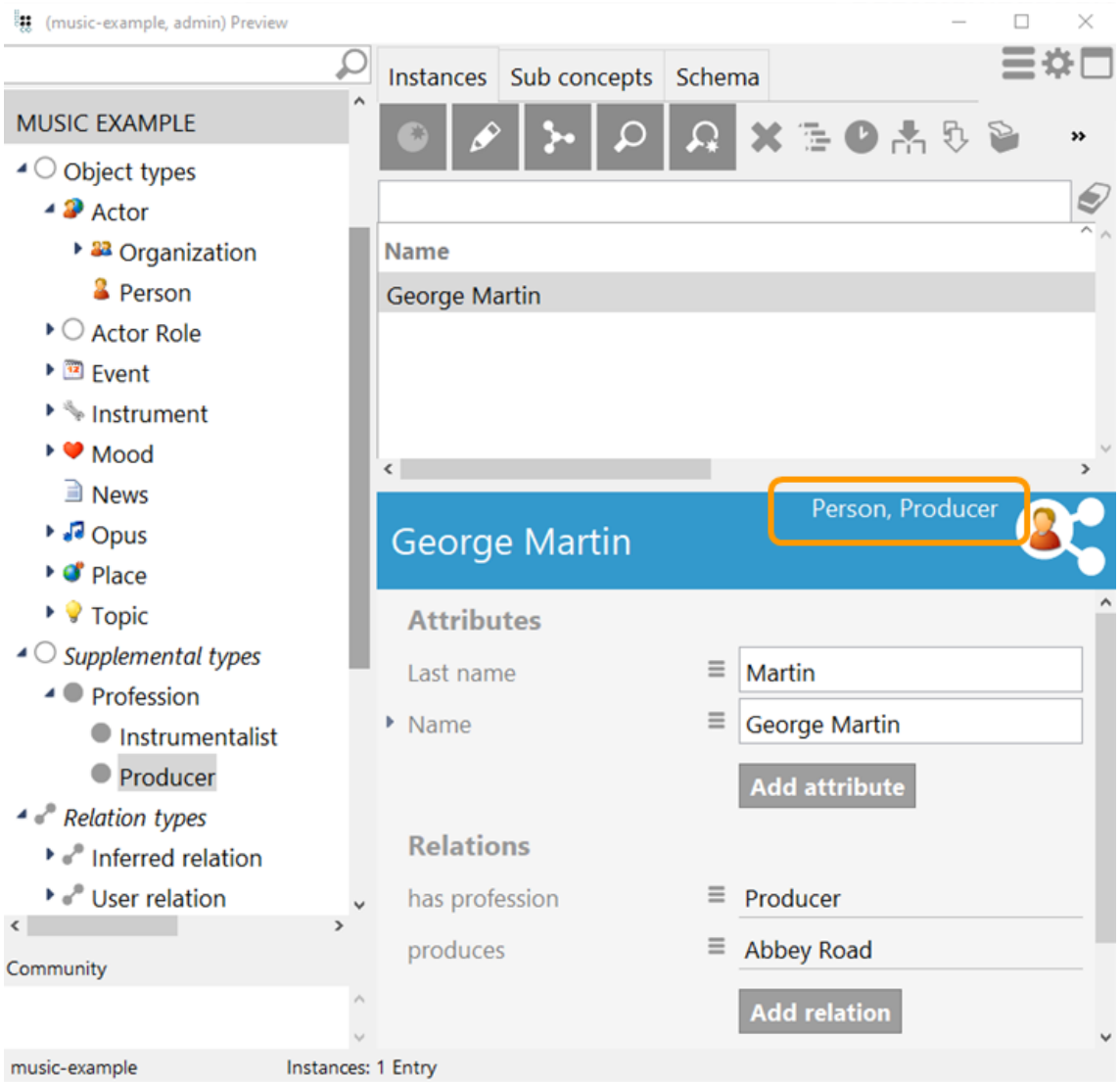


Figure 3. Sind einem Objekt Ergänzungstypen zugeordnet, so wird das im Titel der Objektansicht angezeigt. Das Objekt wird außerdem zusätzlich in den Objektlisten seiner Ergänzungstypen gelistet.

Folgende Schritte sind zur Einrichtung von Ergänzungtsypen notwendig:

1. Erstellung der gewünschten Ergänzungstypen als Subtypen von "Ergänzungstyp"
2. Erstellung einer Zuordnungsrelation, die es erlaubt, einem Objekt eines bestimmten Typs die gewünschten Ergänzungstypen zuweisen zu können. Dabei wird Zuordnungsfunktionalität auf alle Subtypen des gewählten Ergänzungstyps vererbt.

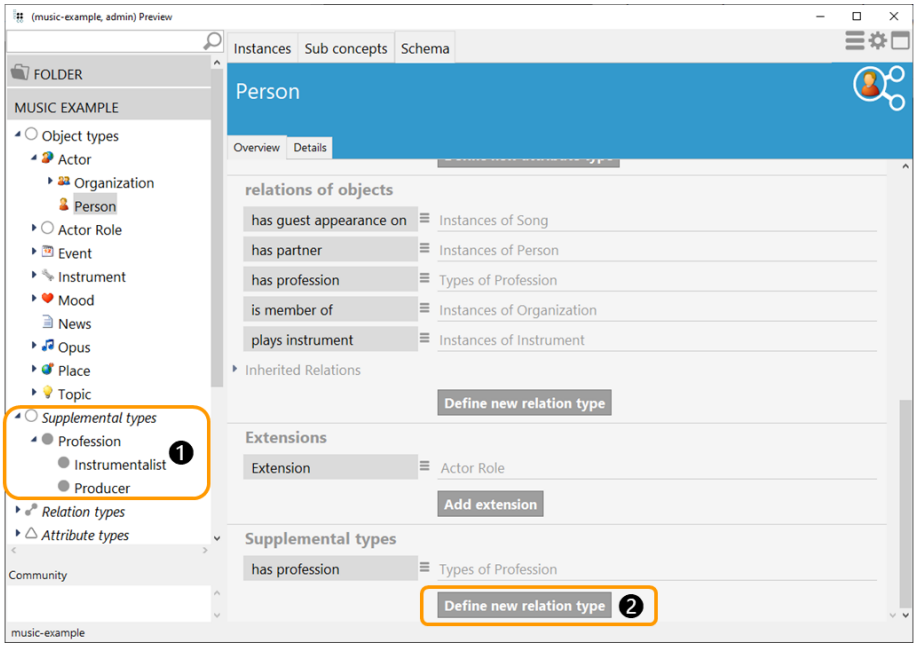
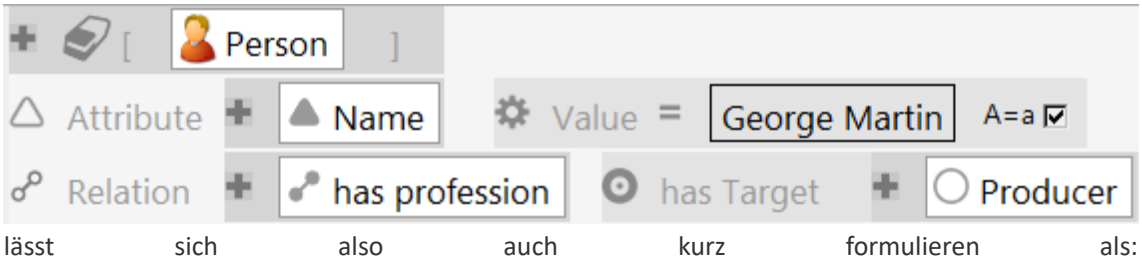


Figure 4. Schritte zur Definition von Ergänzungstypen.

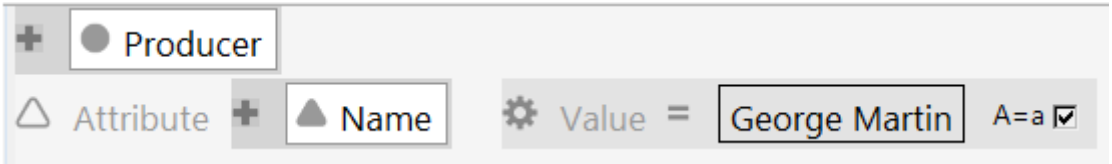
Die Zuordnungsrelation (im Beispiel hier "has profession") ist abgesehen von ihrer Funktion, einen Ergänzungstyp zuzuordnen, eine gewöhnliche Relation, die auf die übliche Weise angezeigt oder erstellt werden kann wie jede andere Relation auch. Dies gilt insbesondere für die Darstellung in konfigurierten Ansichten oder in Datenabbildungen (Import und Export).

In Strukturabfragen können Ergänzungstypen genau wie andere Typen zur Einschränkung der Treffermenge verwendet werden.

Die Abfrage



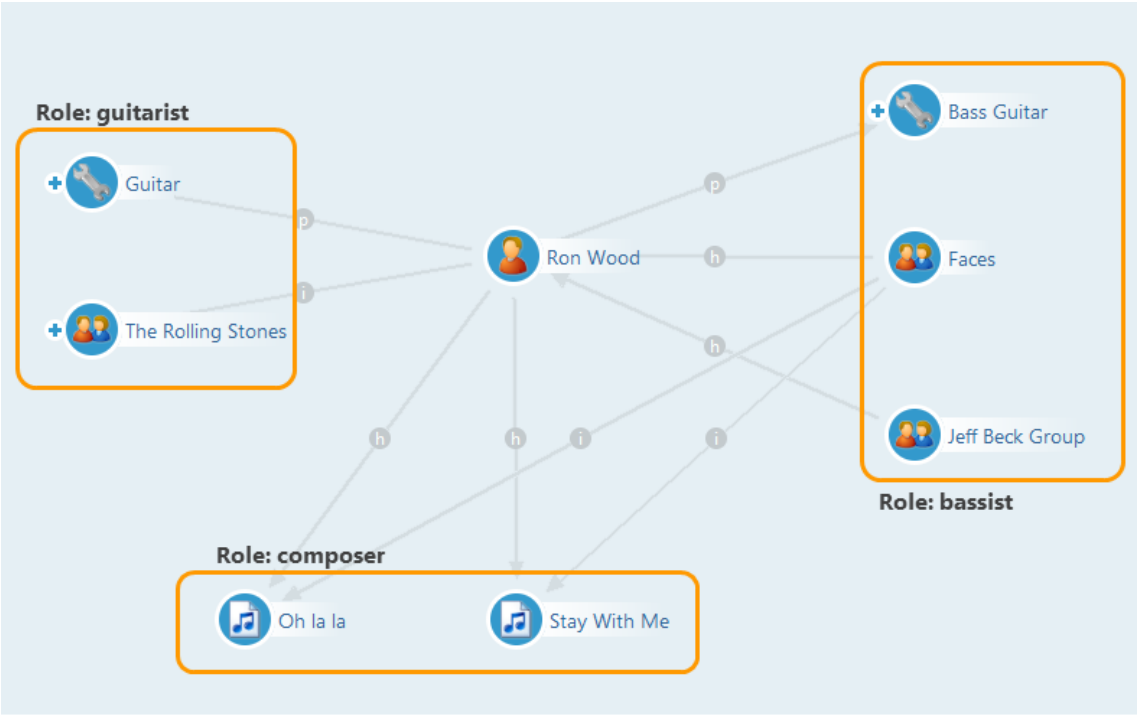
lässt sich also auch kurz formulieren als:



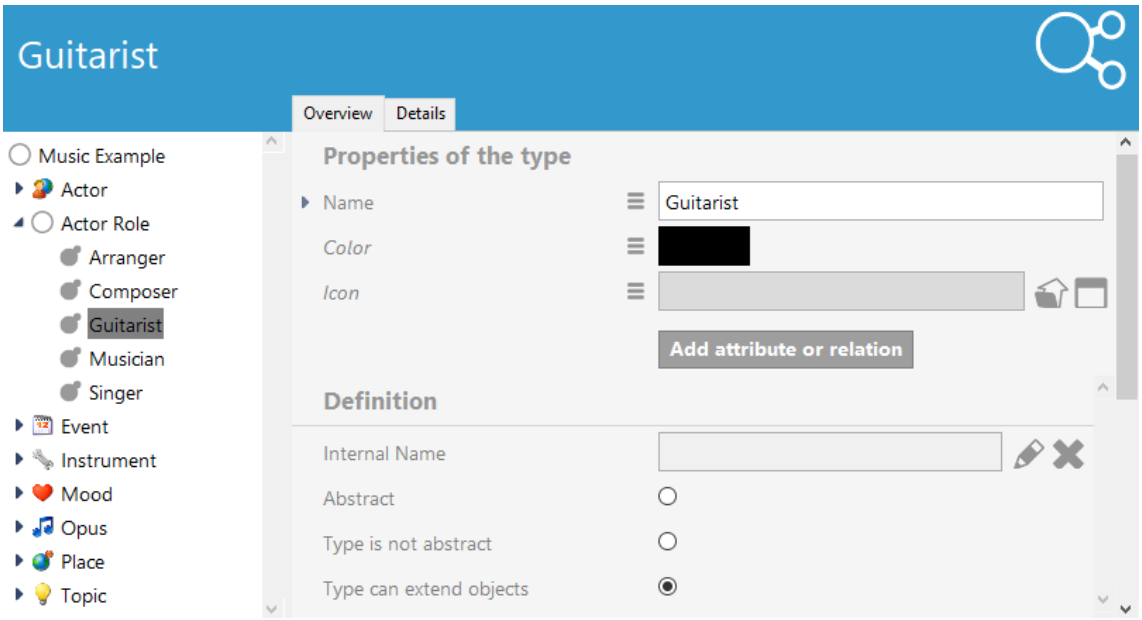
1.2.5.2. Erweiterungen

i-views bietet als weiteres Modellierungsmittel die Möglichkeit, Objekte zu erweitern.

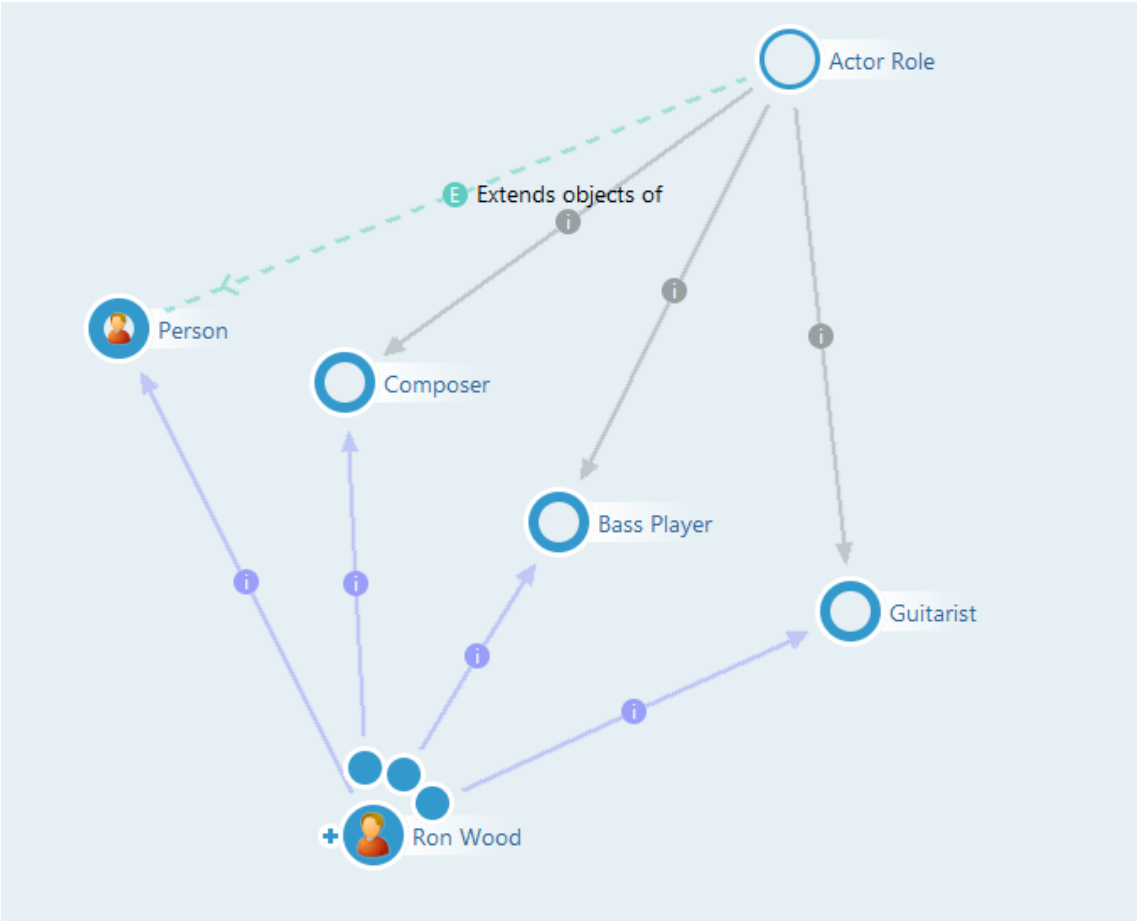
Beispielweise tritt eine Person in der Rolle als Gitarrist in einer Band auf, spielt aber in einer anderen Band ein wiederum anderes Instrument. Zusätzlich übt die Person noch die Rolle des Komponisten aus.



Die verschiedenen Rollen einer Person können über eine spezielle Form eines Objekttyps abgebildet werden. Dieser kann keine Objekte enthalten, jedoch Objekte eines anderen Objekttyps (hier zum Beispiel "Person") erweitern. Hierzu wird bspw. der Objekttyp "Rolle" in den Knowledge-Graphen eingeführt und die verschiedenen Rollen werden für Personen als Untertyp angelegt: Gitarrist, Komponist, Sänger, Bassist usw. Damit diese "Rollen-Objekttypen" Objekte erweitern können, wird diese Funktion im Objekttyp-Editor definiert, indem "Typ kann Objekte erweitern" ausgewählt wird:

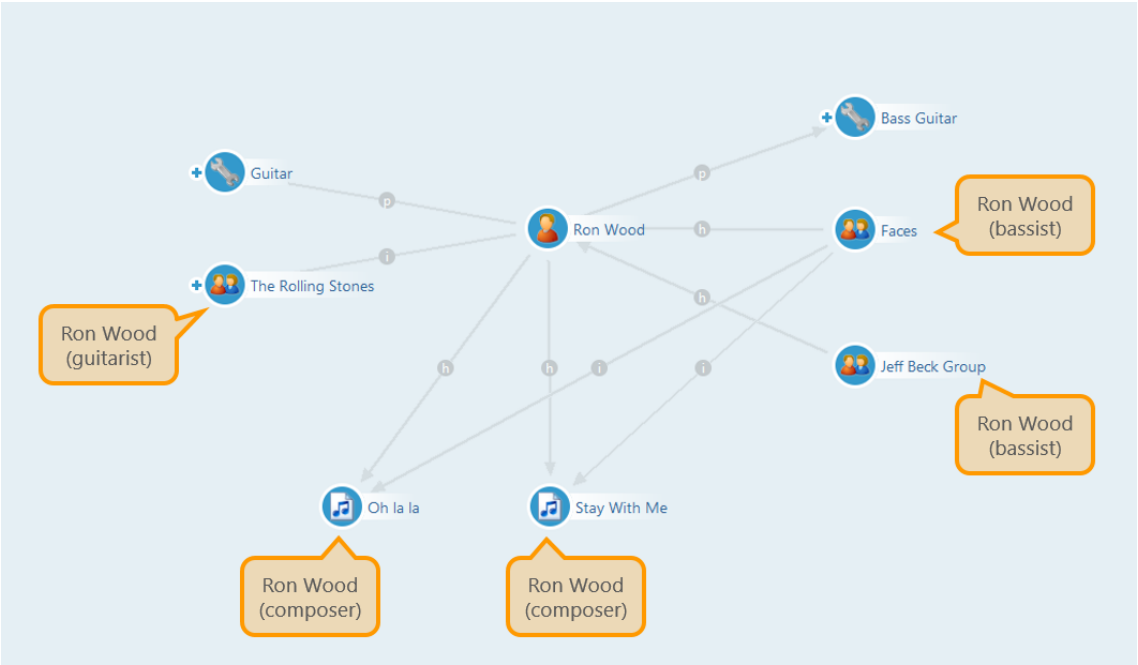


Im Graph-Editor werden Erweiterungen durch eine blaue gestrichelte Linie dargestellt:



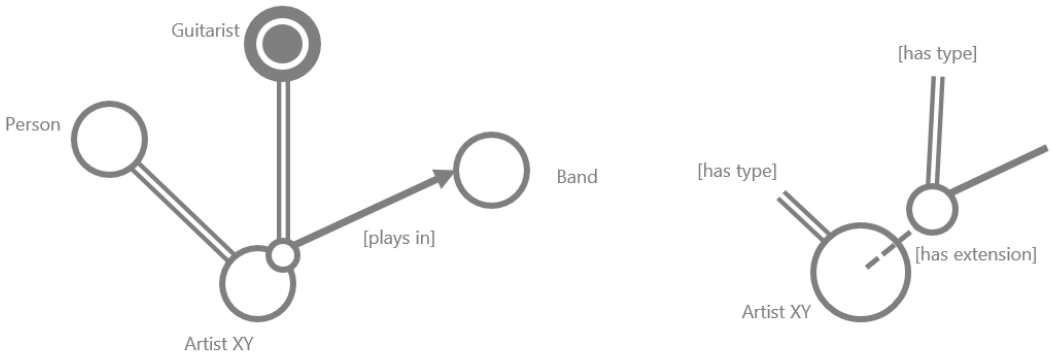
Mit dieser Erweiterung haben wir mehrere Dinge gleichzeitig erreicht:

- Wir haben Subobjekte für die Personen gebildet (können wir uns auch als Abschnitte vorstellen oder — bei Personen — als Rollen). Diese Subobjekte können einzeln betrachtet und abgefragt werden. Sie sind unselbstständig, wenn die Person gelöscht wird, ist auch die Erweiterung "Gitarrist" mitsamt den Relationen zu den Bands oder Titeln weg.
- Wir haben einen mehrstelligen Sachverhalt ausgedrückt. Über separate Relationen zwischen Person, Instrument, Titel/Band können wir das nicht ausdrücken — hier würde die Zuordnung nicht mehr gelingen.



Dazu muss die Relation "spielt in Band" bei der Erweiterung "Gitarrist" definiert sein. Dieser Effekt, dass Personen über die Erweiterung zusätzliches Schema erben, kann auch unabhängig von mehrstelligen Sachverhalten hilfreich sein.

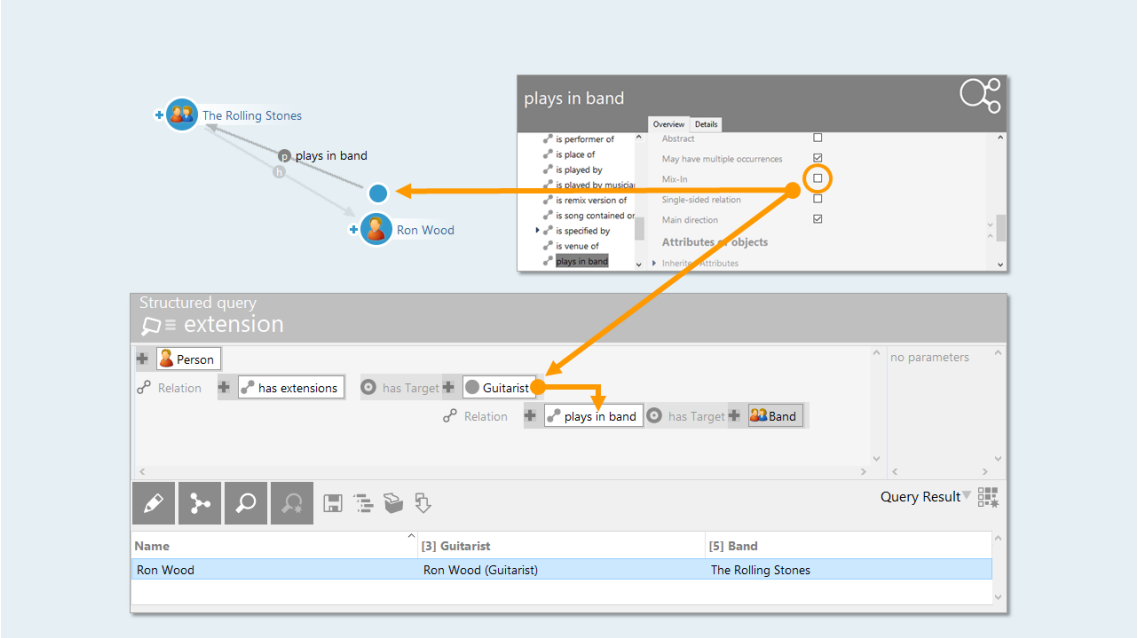
Technisch gesehen ist die Erweiterung ein unselbstständiges Objekt, das durch die Systemrelation "hat Erweiterung" oder invers "erweitert Individuum" mit dem Kernindividuum verbunden ist. Sein Typ (Systemrelation "hat Typ") bildet den Erweiterungstyp.



Bei der Definition einer neuen Erweiterung sind spielen zwei Objekttypen eine Rolle: in unserem Beispiel wollen wir Personen eine Erweiterung geben, das müssen wir ihrem Typ "Person" mitteilen. Die Erweiterung selbst hat auch wieder einen Objekttyp (meist sogar eine ganze Menge von Objekttypen); in unserem Fall "Gitarrist". Beim Typ "Gitarrist" (und bei allen anderen mit denen wir Personen erweitern wollen) werden seine konkreten Objekte unselbstständig sein.

Beim Abfragen von Erweiterungen in der Struktursuche müssen wir die einzelnen Relationen traversieren: Von der konkreten Person über die Relation "hat Erweiterung" über das Erweiterungsobjekt "Gitarrist". Von dort aus kann über die Relation "spielt in Band" zur Band

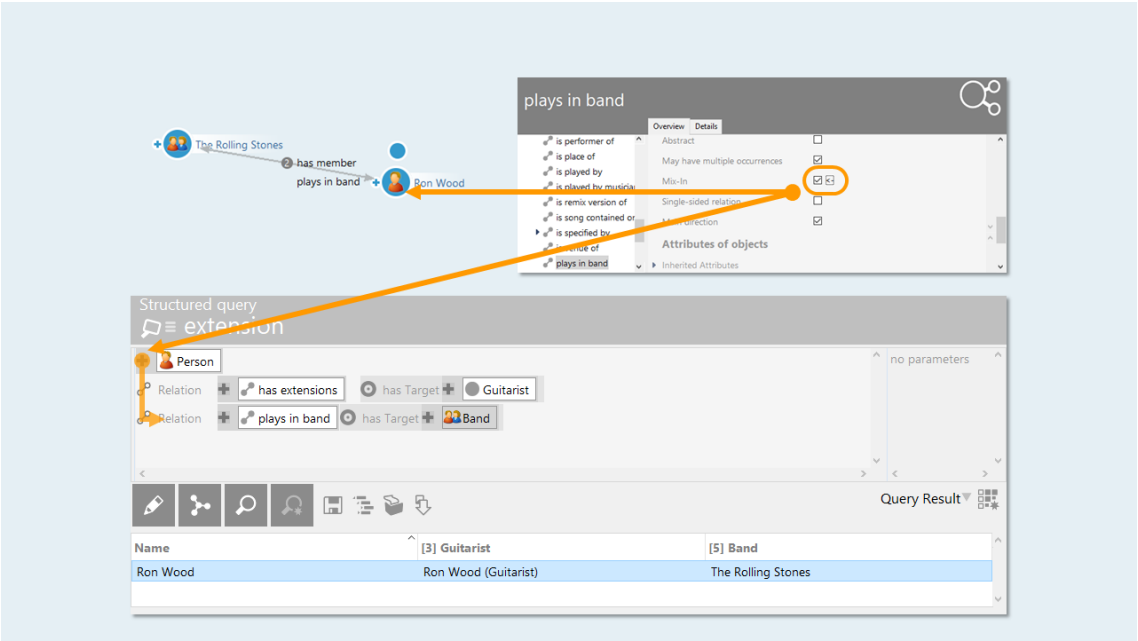
traversiert werden.



Mix-In

Die Essenz dieses Beispiels mit der Rolle "Gitarrist" ist, dass die Relation "spielt in Band" an der Erweiterung, jedoch nicht mit der Person verknüpft ist. Somit ist auch bei mehreren Instrumenten und mehreren Bands einen konsistente Zuordnung möglich.

Wenn die Option Mix-In angewählt ist, dann wird die Relation dagegen beim Kernobjekt (Person) selbst angelegt. Grund dafür ist, das Erweiterungen gelegentlich nicht benutzt werden, um komplexere Sachverhalte auszudrücken, sondern um ein Objekt polyhierarchisch an verschiedene Typen zuzuordnen. Dieses Objekt erbt auf diese Weise Relationen und Attribute mehrerer Typen.



Wenn wir bspw. eine umfangreiche Typen-Hierarchie von Veranstaltungen aufbauen, mit der Unterteilung in große und kleine Veranstaltungen, Freiluft- und Hallenveranstaltungen, Sport- und Kulturveranstaltungen, können wir entweder alle Kombinationen ausprägen (großes Freiluftkonzert, kleines Hallenfußballturnier etc.) oder legen die verschiedenen Veranstaltungstypen als mögliche Erweiterungen der Objekte vom Typ "Veranstaltung" an. Dann können wir eine Veranstaltung über ihre Erweiterungen als Fußballturnier und gleichzeitig als Freiluft-Veranstaltung sowie als Großveranstaltung einordnen. Über die Erweiterung "Fußballturnier" wird dann vielleicht die Relation "teilnehmende Mannschaft" geerbt, über die Erweiterung "Freiluft-Veranstaltung" bspw. noch die Eigenschaft "Flutlicht vorhanden". Wenn wir diese Eigenschaften auf Mix-In gesetzt haben, dann können sie bei den Veranstaltungen wie direkte Eigenschaften abgefragt werden.

Wenn eine Mix-In Erweiterung gelöscht wird, dann verhält sie sich wie eine "normale" Erweiterung: Es muss mindestens eine Erweiterung vorhanden sein, die die Mix-In-Eigenschaft mit sich bringt. Wenn die letzte dieser Erweiterungen gelöscht wird, dann wird auch die Relation oder das Attribut beim Kernobjekt gelöscht.

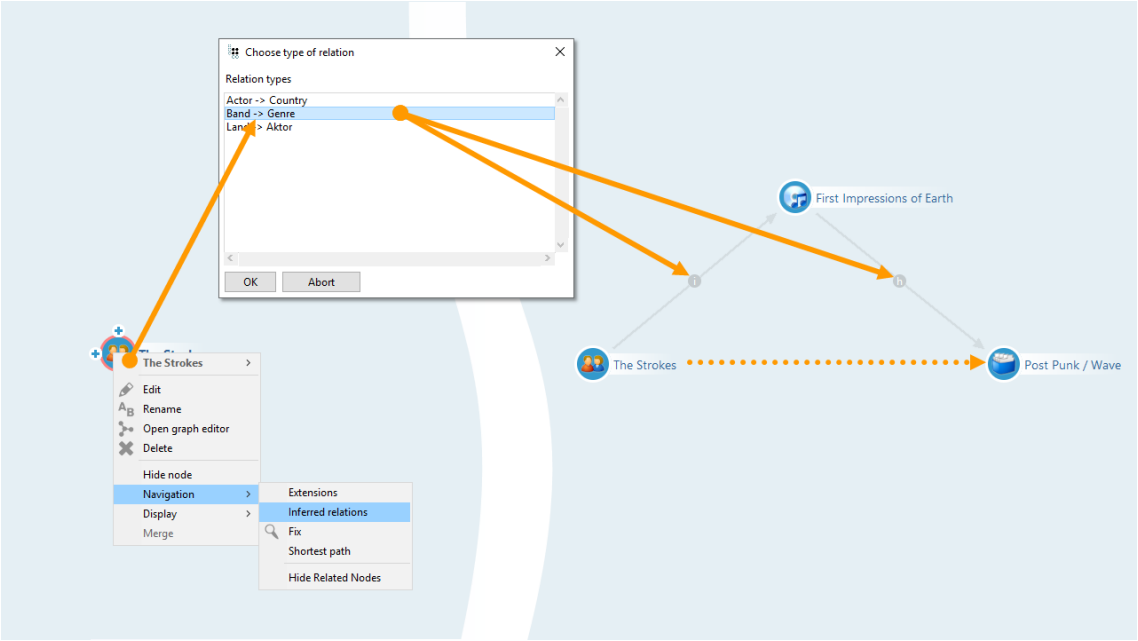
Erweiterungen mit Mix-Ins werden ab Version 6.0 von dem einfacheren sowie flexibleren Mechanismus der Ergänzungstypen abgelöst. Mix-Ins lassen sich zwar weiter verwenden, es empfiehlt sich für die dynamische Typisierung aber die Verwendung der Ergänzungstypen. Über das "Überarbeiten"-Menü lässt sich mittels "Transferiere Erweiterungstypen mit Mix-In-Eigenschaften in Ergänzungstypen" ein Erweiterungstyp in einen Ergänzungstyp umwandeln. Dabei bleibt die ursprüngliche Modellierung (redundant) zur Kontrolle erhalten, kann aber dann verlustfrei gelöscht werden.

1.2.5.3. Berechnete Relationen

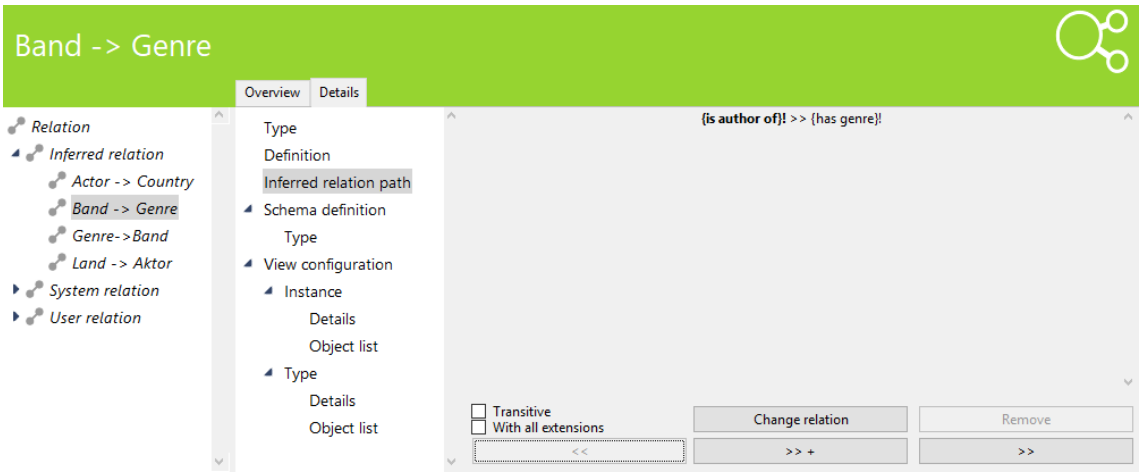
Eine spezielle Form der Relation ist die berechnete Relation. Dahinter verbirgt sich die Möglichkeit, mehrere bereits vorhandene Relationen, durch eine geeignete Relation abzukürzen.

Auf diese Weise kann das System in gewissem Rahmen von einem Objekt A der semantischen Graph-Datenbank, das über mehrere Knoten mit einem anderen Objekt B verbunden ist, einen direkten Schluss von A auf B ziehen. Bei der Anzeige eines semantischen Elements im Graphen und in Strukturabfragen können somit alle Elemente der berechneten Relation in einem Schritt ermittelt und eingeblendet werden.

Beispielsweise veröffentlicht eine Musikgruppe einen Tonträger in einem bestimmten Musikgenre, ergo kann dieses Musikgenre ebenfalls der Musikgruppe zugewiesen werden:



Im Formular-Editor wird der Berechnungspfad über die Relationen "ist Autor von" und "hat Genre" definiert.



Einstellungsmöglichkeiten bei der Definition des Berechnungspfades:

- "Transitiv": Die Relation darf beliebig oft auftreten (ein bis unendlich).

- "mit allen Erweiterungen": Die Erweiterungen werden berücksichtigt. (Die Einstellung wird an der Relation ausgewählt, die an der Erweiterung definiert ist. Möchte man einen Berechnungspfad definieren, in dem man von der Erweiterung zurück zum Kern der Erweiterung laufen möchte, so muss die Relation "erweitert Objekt" explizit im Berechnungspfad angegeben werden.)

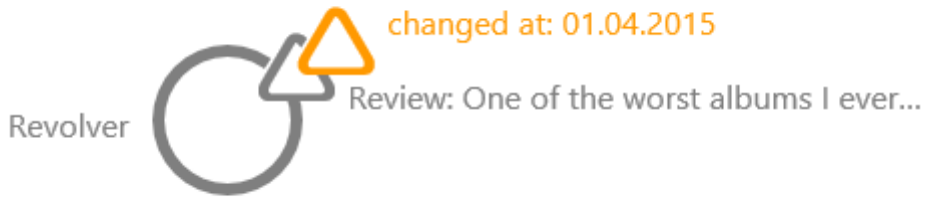
In den Abfragen kann die berechnete Relation benutzt werden wie jede andere Relation auch.

Anmerkung: In der aktuellen Version von i-views wird wegen der besseren Übersicht bei Strukturabfragen empfohlen, mehrere Knoten und Kanten über Suchbausteine abzufragen.

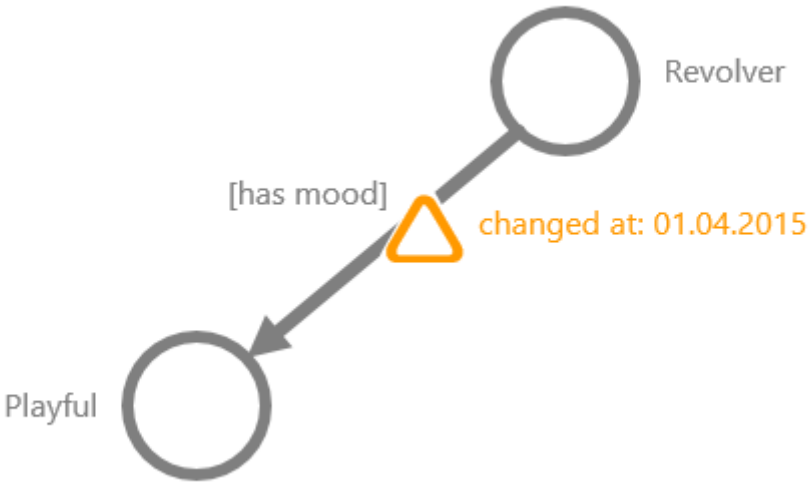
1.2.5.4. Meta-Eigenschaften

Bisher wurden Eigenschaften von geringer Komplexität bei Objekttypen für Objekte definiert. Beispielsweise können Anwender über eine Webanwendung in der hier als Beispiel behandelten Musik-Datenbank Inhalte hinzufügen oder editieren. Es soll aber festhalten werden, von wem welche Information zu welchem Zeitpunkt geändert wurde. Dazu werden Attribute und Beziehungen wiederum für Attribute und Beziehungen, in allen Kombinationen benötigt.

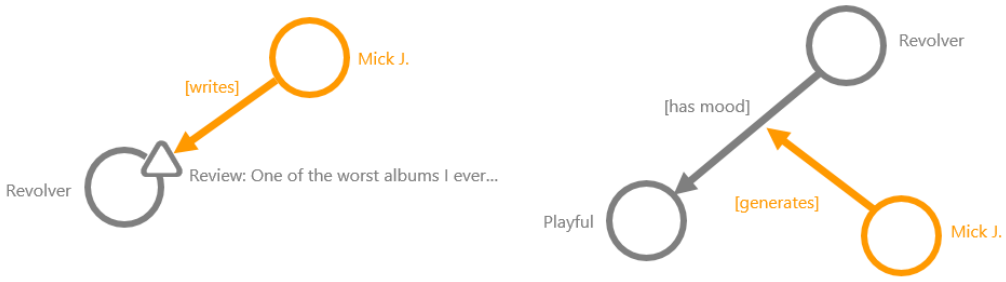
Attribute auf Attributen: Beispielsweise sind für Musik-Alben in der Musik-Datenbank Besprechungen als Textattribute hinterlegt. Soll festhalten wollen, wann die Besprechung hinzugefügt wurde oder wann das letzte Mal geändert, können wir ein Datumsattribut definieren, das den Besprechungs-Attributen angegliedert wird:



Attribute auf Relationen: Dieses Datumsattribut kann sich auch auf einer Relation zwischen Alben und persönlichen Stimmungen, etwa "Moods" befinden, wenn den Nutzern die Möglichkeit zum Tagging gegeben wurde:



Relationen lassen sich auf Attribute und auf Relationen anwenden. Bspw. sollen Nutzer protokolliert werden, die zu bestimmten Zeitpunkten Attribute (etwa eine Besprechung eines Albums), oder eine Relation zwischen einem Album und einem Mood erzeugt oder geändert haben:



Diese Beispiele wird mit den Bearbeitungsinformationen bilden eine klar abgegrenzte Meta-Ebene. Eigenschaften auf Eigenschaften sind aber auch für komplexe "Primärinformationen" verwendbar:

Soll bspw. die Zuordnung von Bands oder Titeln zu den Genres gewichtet werden, kann ein Wert als "Gewicht" als Attribut an die Relation vergeben werden.

Ein Attribut einer Relation kann aber auch der Betrag einer Überweisung, oder die Dauer einer Teilnahme oder Mitgliedschaft sein.

Mit Relationen auf Relationen lassen sich ebenso "mehrstellige Sachverhalte" ausdrücken. Bspw. die Tatsache, dass eine Band bei einem Festival auftritt (das ist eine Relation) und sich dabei einen Gastmusiker dazu holt. Der spielt nicht immer bei der Band und hat somit keine direkte Relation zu ihr. Ebenfalls kann er auch nicht pauschal dem Festival zugewiesen werden, sondern wird der Auftrittsrelation zugewiesen.

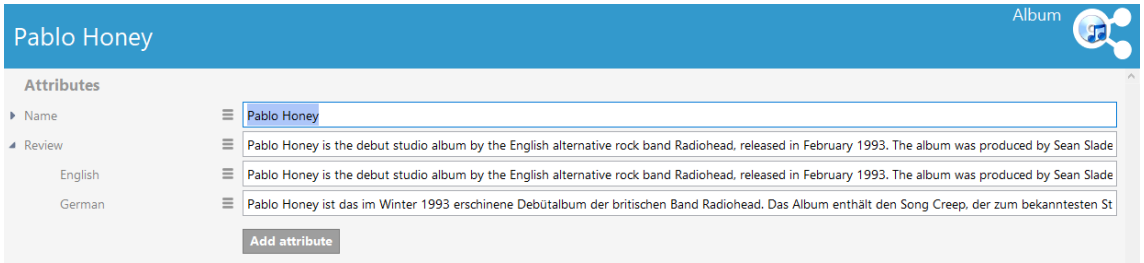
Das Modellieren von Meta-Eigenschaften lässt sich natürlich auch dadurch realisieren, dass

zusätzliche Objekte eingeführt werden. Im letzten Beispiel ließe sich die Tatsache, dass die Band bei einem Festival auftritt, ebenfalls als Objekt vom Typ "Auftritt" modellieren. Ein wesentlicher Unterschied besteht darin, dass im Metamodell die Primärinformationen einfach von der Meta-Ebene getrennt werden können: Der Graph-Editor zeigt die Meta-Informationen erst auf Anforderungen, und in Abfragen und in der Definition von Sichten kann die Meta-Information einfach weggelassen werden. Der zweite Unterschied liegt im Löschverhalten: Objekte sind eigenständig lebensfähig. Eigenschaften, auch Meta-Eigenschaften wiederum nicht; wenn Primärobjekte und ihre Eigenschaften gelöscht werden, werden die Meta-Eigenschaften ebenfalls gelöscht.

Übrigens: Eigenschaften können nicht nur für konkrete Objekte, sondern auch für die Typen selbst definiert werden. Ein typisches Beispiel dafür ist eine ausführliche schriftliche Definition bei einem Objekttyp, bspw. "was verstehen wir unter einer Firma?" Deswegen werden wir beim Anlegen neuer Eigenschaften immer wieder gefragt ob wir sie für konkrete Objekte oder Untertypen anlegen wollen.

1.2.5.5. Mehrsprachigkeit

Die Attribute "Zeichenkette", "Datei-Attribut" und "Auswahl" können mehrsprachig angelegt werden. Beim Zeichenketten-Attribut und bei Dateien können dann mehrere Zeichketten für ein Attribut eingegeben werden:

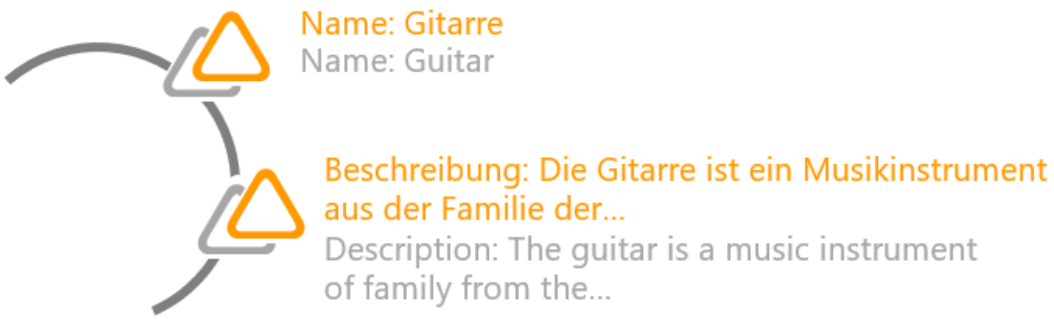


Bei Datei-Attributen können analog mehrere Bilder (z.B. mit anderssprachigen Beschriftungen) hochgeladen werden. Bei Auswahl-Attributen werden alle Auswahlmöglichkeiten in der Attributdefinition hinterlegt; hier ist es egal, in welcher Sprache dann die Auswahl für das konkrete Objekt getroffen wird.

Alle anderen Attribute stellen sich in allen Sprachen gleich dar, wie z.B. Boolesche Attribute, Ganzzahlen oder URLs.

Sofern die Darstellung in anderen Sprachen abweicht, passen Attribute ihre Darstellung je nach Sprache automatisch an: Etwa werden Datumsangaben nach europäischer Schreibweise Tag|Monat|Jahr im US-amerikanische Format Monat|Tag|Jahr dargestellt.

In i-views werden für anderssprachige Werte nicht einfach separate Attribute angelegt, sondern es bleibt bei einen Attribut mit Sprachvarianten als separate Layer. Es muss bei Entwicklung einer Anwendung nicht um das Management der verschiedenen Sprachen gekümmert werden, sondern nur die gewünschte Sprache bei der jeweiligen Anfrage:



In i-views lassen sich bevorzugte Ersatzsprachen definieren: sollte ein Attributwert, z.B. ein Beschreibungstext in der angefragten Sprache nicht vorliegen, kann der fehlende Text, wenn er in anderen Sprachen vorliegt angezeigt werden. Die Reihenfolge der Ersatzsprachen lässt sich ebenfalls festlegen.

1.2.5.5.1. Sprachobjekte in i-views

Generell werden Sprachen in i-views durch Objekte vom Typ Sprache repräsentiert. Die Liste der im Graphen verwendeten Sprachen findet sich im *Technik*-Panel.

Einstellungen, die sich auf Sprachen beziehen, wie eventuelle Präferenzen oder Restriktionen, werden durch Relationen auf die jeweiligen Sprachobjekte abgebildet. Das Entfernen des Sprachobjekts führt dementsprechend zur Auflösung der Relation und damit der Aufhebung der verbundenen Einstellungen.

Die Sprachenobjekte können, wie andere Objekte, beliebig manuell angelegt werden, werden bei Bedarf aber auch automatisch von i-views erzeugt. Manuelle Änderungen an den Attributen sollten vermieden werden.

1.2.5.5.2. Sprachgruppen und Einschränkungen

Einzelne Sprachobjekte lassen sich kombinieren und in sogenannte Sprachgruppen zusammenfassen. Diese eignen sich zum Beispiel zur Modellierung von Einschränkungen wie der Begrenzung der im Graphen verwendeten Sprachen auf eine Teilmenge.

Solche Einschränkungen auf Sprachen/Sprachgruppen lassen sich an mehreren Stellen der Anwendung vornehmen.

So lässt sich zum Beispiel die View-Konfiguration nutzen, um Vorlieben und Einschränkungen für einzelne Attribute bis hin zu applikationsweiten Regeln für ganze Anwendungen zu konfigurieren.

Im Schema lässt sich die Menge möglicher Übersetzungen für einzelne Attribute individuell festlegen, die Objekte zu den hier referenzierten Sprachen werden automatisch angelegt.

1.2.5.5.3. Kontextabhängige Sprachen

Neben einfachen Sprachen und Sprachgruppen bietet i-views drei kontextabhängige Sprachen für

zusätzliche Möglichkeiten zur Modellierung:

- **Bevorzugte Sprache**: Diese Sprache symbolisiert die aktuell eingestellte Systemsprache der Anwendung.
- **Beliebige Sprache**: Eignet sich zur Modellierung von "irgendeiner" Sprache. Ist beispielsweise die Anzeigesprache eines Attributs eingeschränkt auf "Beliebige Sprache", so wird für das Objekt immer die (nach alphabetischer Reihenfolge der ISO 639-2 Kürzel) erste verfügbare Sprache angezeigt.
- **Modellierungssprache**: Diese Sprache hat Priorität über alle anderen Sprachen. Wann immer ein Attribut einen übersetzten Wert in dieser Sprache hat, wird grundsätzlich dieser zur Anzeige ausgewählt. Ausgewählt wird sie aus der Liste der regulären Sprachen unter dem Kürzel `vol`.

1.2.5.5.4. Übersetzungsauswahl bei der Sprachensuche

Die am Ende angezeigten/ausgewählten Sprachen leiten sich aus der Schnittmenge der nach allen Restriktionen übrig gebliebenen Sprachen her. Sollte in bestimmten Fällen die geforderte Sprache aufgrund von Einschränkungen oder fehlender Definition nicht verfügbar sein, wählt das System den "bestmöglichen" Ersatz für die gesuchte Sprache. Diese Wahl folgt immer der folgenden Priorisierungsreihenfolge:

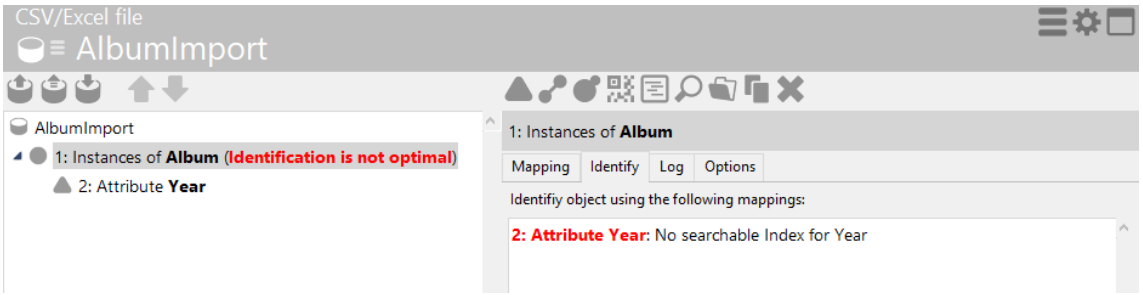
1. **Modellierungssprache:** Falls für das Attribut eine Übersetzung in der Modellierungssprache vorliegt, wird grundsätzlich diese angezeigt. Das gilt auch wenn die ursprünglich gesuchte Sprache vorfügbar ist.
2. **Direkte Entsprechung:** Liegt für die gesuchte Sprache eine Übersetzung vor, dann ist keine weitere Auswahl nötig.
3. **Primärsprache:** Falls es sich bei der Gesuchten um eine lokalisierte regionale Sprachvariante handelt, z.B. "Englisch (USA)", wird zuerst gesucht, ob die entsprechende nicht lokalisierte Primärsprache verfügbar ist, also "Englisch".
4. **Regionale Variante:** Sollte dies nicht der Fall sein, wird überprüft, ob eine andere regionale Variante derselben Sprache verfügbar ist, z.B. "Englisch (GB)" für "Englisch (USA)".
5. **Bevorzugte Sprache:** Für den Fall, dass auch keine andere Variante gefunden wurde, muss eine andere Sprache gewählt werden. Die erste Wahl fällt hier immer auf die "Bevorzugte Sprache", also die Systemsprache, in der der Knowledge-Builder ausgeführt wird.
6. **Liste der Ersatzsprachen:** Sollte auch für die bevorzugte Spraceh keine Übersetzung vorliegen, wird die einstellbare, geordnete Liste der Ersatzsprachen der Reihe nach abgesucht. Diese finden Sie unter *Einstellungen* > *System* > *Sprachen*. Für jedes Element dieser Liste wird ebenfalls nach Primärsprache und alternativen lokalisierten Varianten gesucht, bevor zum Nächsten übergegangen wird.
7. **Beliebige vorliegende:** Sollte bis zu diesem Punkt immer noch keine verfügbare Sprache gefunden sein, wählt das System eine der vorliegenden aus. Gewählt wird die nach alphabetischer Sortierung des ISO 639-2 erste vorliegende Sprache.

1.2.6. Indexierung

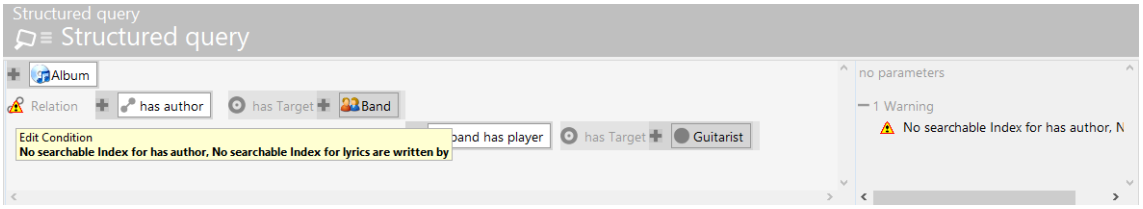
Die Indexierung bildet einen Bestandteil der internen Datenverwaltung von Datenbanken. Richtig eingesetzt, kann das Setzen von Indizes eine deutliche Verbesserung der Performance hervorrufen.

Hintergrund: Grundsätzlich werden in i-views alle Wissensnetz-Elemente (Typen oder Objekte) zusammen mit ihren Eigenschaften (Attribute oder Relationshälften) in einem Cluster abgespeichert. Für gewisse Transaktionen oder Anwendungszwecke kann es allerdings von Vorteil sein, nur einen Teil der Informationen zu laden. Anstatt bei Anfragen die kompletten Elemente bzw. Cluster für das Auslesen weniger Eigenschaften laden zu müssen, wird durch einen entsprechenden Index auf die ausschließlich benötigten Eigenschaften verwiesen. Im übertragenen Sinne sind Indizes zugleich Wegweiser und Abkürzung zu benötigten Teilinformationen.

Der Bedarf für das Indexieren wird dabei in Strukturabfragen oder beim Importmapping durch diverse Hinweise deutlich: Wird ein Objekt im Importmapping anders als erwartet nicht durch den Primärnamen, sondern durch ein anderes Attribut identifiziert, so erscheint der Hinweis: "Kein nutzbarer Index für […]".



Importmapping mit Rückmeldung zu fehlender Indexierung



Strukturabfrage mit Rückmeldung zu fehlender Indexierung

Benötigt wird die Indexierung u.a. für:

- Suchen
- Importe mit identifizierenden Eigenschaften

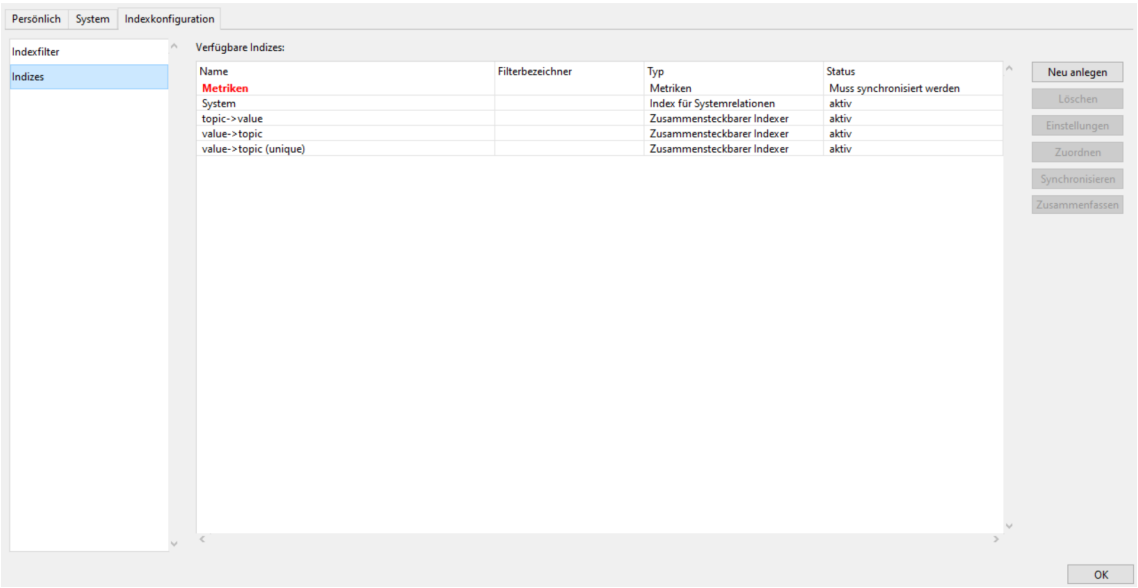
Je nach Verwendungsabsicht müssen für bestimmte Attribute oder Relationen die passenden Indizes gewählt werden.

Das **Definieren** der Indizes wird in den Einstellungen des Knowledge-Builders, das **Zuordnen** der Indizes kann entweder in den Einstellungen des KB oder im Detaileditor eines Typs erfolgen (Details > Indexierung > Index zuordnen).

1.2.6.1. Verfügbare Indizes verwalten und anwenden

Verfügbarer Indizes (Einstellungen > Indexkonfiguration)

Alle im Knowledge-Builder angelegten Indizes können in den Einstellungen zentral verwaltet werden.



Rubrik "Indizes"

Mit Hilfe dieser Einstellungsmöglichkeit lassen sich die Indexstrukturen verwalten. Unter "Verfügbare Indizes" werden alle verfügbaren Indextypen aufgeführt. Jeder Indextyp kann für bestimmte Arten von Attributen oder Relationen verwendet werden.

Falls ein Index grau dargestellt wird, ist der Index zurzeit deaktiviert, ist er rot hervorgehoben, so ist der Index momentan nicht synchron.

Auf der rechten Seite befinden sich Schaltflächen zum Erzeugen, Löschen, Konfigurieren, Zuordnen und Synchronisieren.

Index	Verwendung
Lucene-Volltextindex (JNI)	Volltextsuche
Metriken	Verbesserung der Performance in Strukturabfragen durch Berückschtigung von Elementmengen
System	Systemrelationen (vordefiniert, kann nicht verändert werden) Wird verwendet für Relationen "erweitert Objekt"/"hat Erweiterung"/"ist Obertyp von"/"ist Untertyp von"
topic → value	Für die Auflistung von Attributwerten/Relationszielen in Objektlisten

Index	Verwendung
topic → value (domain segmented)	Für die Auflistung von Attributwerten/Relationszielen in Objektlisten
value → topic und topic → value	Für Einwegrelationen; bewirkt Speedup für gewichtete inverse Einwegrelationen
value → topic	Attributwerte für ein Objekt
value → topic (unique)	Attributwerte, die nur einmal pro Attributtyp für ein Objekt vorkommen dürfen

Rubrik "Index für Relationen"/ "Index für Attributwerte "

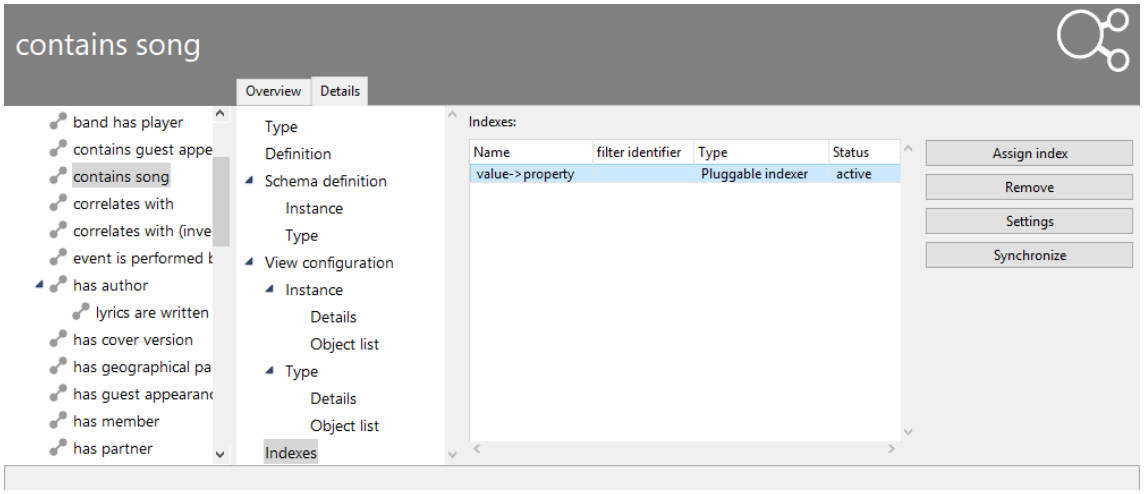
Indizes können unter verschiedenen Aspekten eingeteilt werden. Zunächst kann man zwischen vorwärts und rückwärts gerichteten Indizes unterscheiden. Bei den rückwärts gerichteten Indizes kann es ggf. sinnvoll sein, vom Ziel/Wert auf die Eigenschaft zu verweisen, um Metabedingungen an der Eigenschaft zu lösen. Schließlich kann ein Index optional eine Segmentierung je Typ des Quellobjektes vornehmen, um Strukturabfragen, die auf Objekte untergeordneter Typen eingeschränkt sind, effizienter zu lösen.

Anwendungsspezifisch kann es auch vorkommen, dass manche Eigenschaften keinen Index benötigen. (Diese können dann mit der Markierung "Ignorieren" versehen werden. Sie werden bei diesem Optimierungsschritt nicht weiter betrachtet.)

- Relationen können anstelle eines vorwärts gerichteten Index einen rückwärts gerichteten Index an der Inversen verwenden — und umgekehrt.
- Attribute können auch mit modifizierten/normalisierten Werten indexiert werden (z.B. Volltext mit Wortgrundformen). Nach diesen kann dann über einen entsprechenden Operator gesucht werden.

Anwendbare Indizes (Detailkonfiguration)

Die für einen Relations- oder Attributtyp anwendbaren Indizes können über die Detailkonfiguration zugeordnet werden.



Das Zuweisen von Indizes in der Typen-Detailkonfiguration

Attributtypen	Relationstypen
topic → value	topic → value
topic → value (domain segmented)	topic → value (domain segmented)
value → property	value → property
value → topic	value → topic
value → topic (unique)	

1.2.6.2. Neuen Index erstellen

Ein neuer Index wird in den Einstellungen des Knowledge-Builders angelegt unter: Einstellungen > Indexkonfiguration > Indizes > Neu anlegen

Zu Beginn ist folgende Auswahl verfügbar:

Index	Verwendung
Zusammensteckbarer Index	Kombinierte Verwendung von Verteiler- und Index-Bausteinen für angepasste Indexierung; spezifische Konfiguration durch Anwendung von Indexfilter möglich
Lucene-Volltextindex (JNI)	Volltext-Suche

Im Folgenden wird die Konfiguration der zusammensteckbaren Indexer beschrieben, da diese am flexibelsten einsetzbar sind und nahezu jeden Einsatzbereich abdecken.

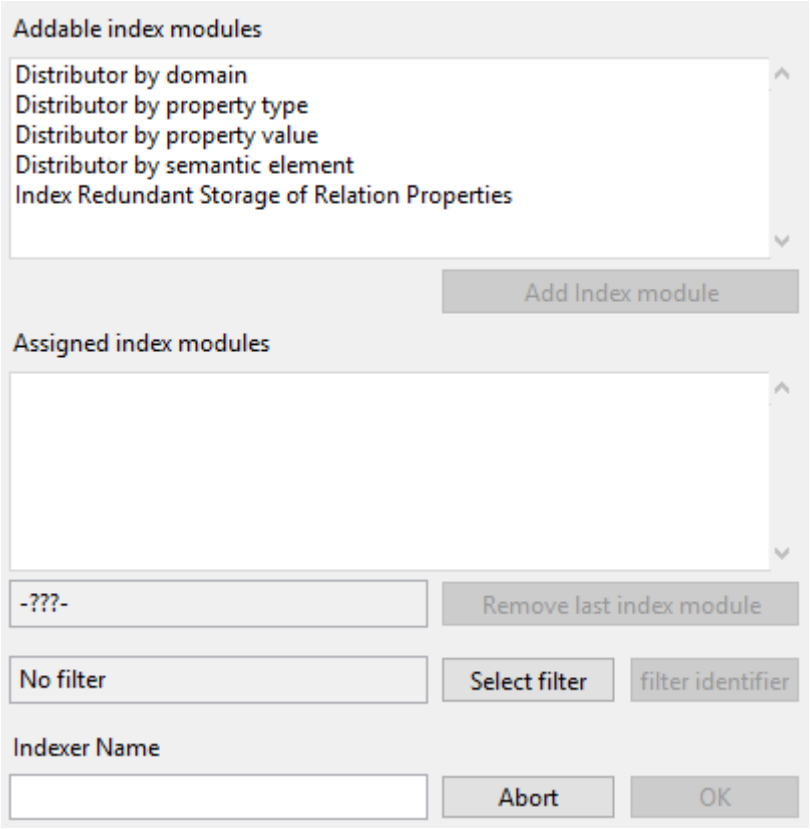
Hinzufügbare Indexbausteine

Zusammensteckbare Indizes erlauben es dem Administrator, einen Indexer aus vorgefertigten Bausteinen zusammenzustellen, um ein zugehöriges Verhalten des Indexers zu erreichen.

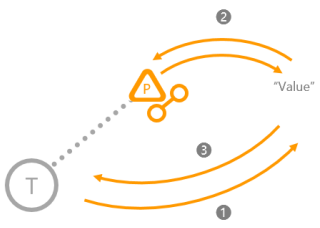
Ein zusammensteckbarer Indexer besteht aus Verteilerstufen, die durch eine Index-Stufe abgeschlossen werden, welche die Datenspeicherung regelt. Dabei kann ein Indexer sowohl Attribute als auch Relationen indexieren.

Wenn dem Indexer ein optionaler Indexfilter zugewiesen wird, so lässt sich das Indexerverhalten noch weiter beeinflussen, es können dann nur noch passende Eigenschaften-Typen dem Indexer zugeordnet werden.

Da Eigenschaften Attribute und Relationen umfassen, wird im Folgenden ein Attributswert bzw. Relationsziel als Wert der Eigenschaft bezeichnet.



Zusammensteckbarer Index



T = Topic = Objekt/Element/Instanz P = Property = Attribut/Relation "V" = Value = Attributwert/Relationsziel

Verteiler/Index	Verwendung

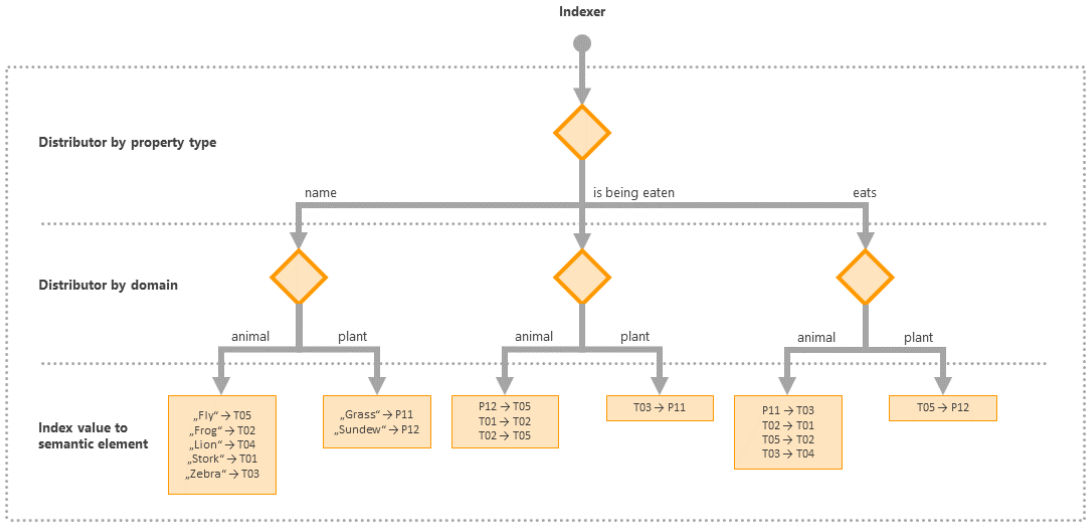
Zusammensteckbarer Index	
Verteiler je Definitionsbereich (danach sind alle anderen Verteiler auswählbar)	Für die Suche nach Teilmenge von Objekttypen, die gemeinsam eine Eigenschaft verwenden
Verteiler je Eigenschaftstyp (Index danach auswählbar:)	Unterscheidung, ob Attribut oder Relation
Index Eigenschaft auf Wert/Ziel	Attribut → Attributwert, Relation → Zielobjekt/Zieltyp Für das Auffinden von Relationszielen in Strukturabfragen mit Einschränkung über Metaeigenschaft
❶ Index Objekt auf Wert/Ziel = topic → value = topic → value (domain segmented)	Objekt → Attribut, Objekt → Zielobjekt von Relation Für die Auflistung von Attributwerten/Relationszielen in Objektlisten
❷ Index Wert/Ziel auf Eigenschaft = value → property	Attributwert → Attribut Metarelationsziel → Attribut Relationsziel → Relation Metaattribut(wert) → Relation Für Einwegrelationen; bewirkt Speedup für gewichtete inverse Einwegrelationen
Index Wert/Ziel auf Eigenschaft (Eindeutigkeitsprüfung)	Attributwert → Attribut Für die Suche nach Metaeigenschaften
❸ Index Wert/Ziel auf Objekt = value → topic	Attributwert → Attribut Relationsziel → Relation Für die Unterstützung von Strukturabfragen nach Objekten mit gegebenen Werten/Zielen an Attributen/Relationen
❸ Index Wert/Ziel auf Objekt (Eindeutigkeitsprüfung) = value → topic (unique)	Attributwert → Objekt (Bsp.: Email-Adresse)
Verteiler je Eigenschaftswert	Zusammen mit "Index Eigenschaft": Für eine kompakte Speicherung von sehr vielen gleichen Werten/Zielen; gleiches Verhalten wie bei "Index Wert/Ziel auf Eigenschaft"
Verteiler je Objekt	Für Einwegrückrelationen
Index redundante Speicherung für Relationseigenschaften	(Schließt sich ggs. mit der Verwendung zusammensteckbarer Indizes aus) Schnellere Anzeige von Metaeigenschaften an Relationen bei Verwendung symmetrischer Relationseigenschaften

Filter	
Filter-Typ	**Verwendung**
Breitengrad	Für Indexierung eines Attributtyps vom Wertetyp "Geographische Position"
Längengrad	Für Indexierung eines Attributtyps vom Wertetyp "Geographische Position"

Filter	
Intervall-Startwert	Für Indexierung eines Attributtyps vom Wertetyp "Intervall"
Intervall-Stopwert	Für Indexierung eines Attributtyps vom Wertetyp "Intervall"
Zeichenketten-Filterung	.
Zeichenketten-Zerlegung	Für die Zerlegung eingegebener Zeichenketten in einzelne Worte

1.2.6.3. Details zu Indexerbausteinen

Es wird unterschieden zwischen den aufschlüsselnden Indexerbausteinen und den indexierenden Indexerbausteinen. Ein aufschlüsselnder Indexerbaustein partitioniert den Index nach unterschiedlichen Aspekten. Dahinter folgt entweder eine weitere Aufschlüsselung oder ein indexierender Indexerbaustein, welcher die Indexeinträge speichert.



Die Abbildung zeigt exemplarisch, wie ein zusammensteckbarer Indexer aus drei Bausteinen (ohne Wertefilter) die Indexeinträge gruppiert. Dieser Index kann nun effizient beantworten

- Welche Tiere beginnen mit S
- Welche Pflanzen fressen andere Lebewesen
- Welche Tiere fressen Zebra (T03)
- u.s.w.

Fragestellungen wie zum Beispiel

- Welche Lebewesen beginnen mit S
- Welche Lebewesen fressen Fliegen (T05)

können ebenfalls beantwortet werden, hierzu wäre bereits eine Indexer-Konfiguration ohne Verteiler je Definitionsbereich ausreichend (und ist je nach Datenlage effizienter).

1.2.6.3.1. Verteiler

- **Verteiler je Eigenschaftstyp**

Der wichtigste Baustein, ohne den die meisten indexierenden Bausteine nicht eingefügt werden können. Er sollte in der Regel an erster Stelle kommen und partitioniert die Einträge nach deren Eigenschaftsbegriff.

- **Verteiler je Definitionsbereich**

Ermöglicht eine Partitionierung nach den jeweiligen Begriffen der eigenschaftstragenden Objekte. Der Baustein kann nur bei Eigenschaften an Individuen sinnvoll verbaut werden. Kann eine Eigenschaft an mehreren Objekttypen vorkommen und wird in einer Suche nur eine Teilmenge dieser Objekttypen gesucht, so beschleunigt dieser Baustein die Suche durch entsprechende Indexzugriffe.

- **Verteiler je Objekt**

Bei der Indexierung zum Zusammenfassen der Relationsziele am Quellobjekt kann dieser Baustein verwendet werden. Wie der vorherige Baustein dient er dem Abbilden älterer Indexer und ist sein i-views 3.1 nur bei Einwegrückrelationen sinnvoll.

- **Verteiler je Eigenschaftswert**

Dient zum Partitionieren nach Relationsziel bzw. nach Attributwert. Indexiert werden kann dann nur noch die Eigenschaft (siehe Index Eigenschaft).

1.2.6.3.2. Indizes

- **Index Wert/Ziel auf Objekt** Mit diesem Indexbaustein werden Attributwert auf Objekt bzw. Relationsziel auf Relationsquelle im Index hinterlegt. Diese Indexierungsart ist dann sinnvoll, wenn Expertensuchen nach Objekten mit gegebenen Werten an den indexierten Attributen (bzw. mit gegebenen Zielen an den indexierten Relationen) unterstützt werden sollen.
- **Index Objekt auf Wert/Ziel**

Dieser Indexbaustein indexiert genau umgekehrt wie der "Index Wert/Ziel auf Objekt" und kann bei Attributen genutzt werden, um für Objektlisten die Spaltenwerte der indexierten Attribute zu ermitteln. Bei Relationen kann er genauso verwendet werden wie der "Index Wert/Ziel auf Objekt", wenn entweder die inverse Relation indexiert ist oder das Quellobjekt durch die Suche bereits stärker eingeschränkt ist als das Zielobjekt. Möchte man Expertensuchen mit der indexierten Relation in beide Richtungen (Quelle-Ziel und Ziel-Quelle) unterstützen, so kann die Relation entweder mit diesem und dem "Index Wert/Ziel auf Objekt" indexiert werden oder die Relation und ihre inverse Relation werden beide mit einer der beiden Index-Arten indexiert. Hierbei kann es eine Rolle spielen, ob der Indexbaustein mit einem "Verteiler je Definitionsbereich" kombiniert ist, denn durch die Verwendung dieses Verteiler-Bausteines für einen Index auf der inversen Relation kann eine Partitionierung mittels der Zieldomain erreicht werden.

- **Index Wert/Ziel auf Eigenschaft**

Mit diesem Indexbaustein werden Wert auf Attribut bzw. Ziel auf Relation im Index hinterlegt. Diese Indexierungsart ist dann sinnvoll, wenn an den indexierten Attributen und Relationen auch Suchen nach weitere Metaeigenschaften unterstützt werden sollen. Damit dieser Index in einer Suche auch für die Objekte der Eigenschaft (analog dem "Index Wert/Ziel auf Objekt") benutzt werden kann, muss die jeweilige Eigenschaft im zugehörigen Begriffseditor bei "Eigenschaft ist iterierbar" auf "Aktiv" bleiben.

- **Index Eigenschaft auf Wert/Ziel** Dieser Indexbaustein unterstützt Expertensuchen, bei denen Ziele der Relationen gesucht sind. Dafür muss die stärkste Einschränkung über Metaeigenschaften der Relation erfolgen. Einfache Quelle-Ziel Bedingungen werden jedoch nicht unterstützt.
- **Index Eigenschaft** Zusammen mit dem Verteiler je Eigenschaftswert kann dasselbe Verhalten wie bei einem Index Wert/Ziel auf Eigenschaft erreicht werden. Bei sehr vielen gleichen Werten bzw. Zielen kann so eine kompaktere Speicherung erreicht werden, andernfalls bietet diese Kombination keine Vorteile.
- **Index Eigenschaftswert** Dieser Index speichert nur die Attributwerte bzw. die Relationsziele. Eine Verwendung ist dann sinnvoll, wenn ein "Verteiler je Objekt" vorgeschaltet ist und wenige Objekte viele Werte/Ziele haben.
- **Index redundante Speicherung für Relationseigenschaften** Dieser Baustein kann nur alleine verwendet werden und dient der schnelleren Anzeige von Metaeigenschaften an Relationen, wenn symmetrische Relationseigenschaften verwendet werden. Auf technischer Ebene wird keine Indexstruktur angelegt, der Indexer kann aber über dieselben Konfigurations- und Programmschnittstellen angesprochen werden.

1.2.6.3.3. Eindeutigkeitsprüfung

Die Bausteine Index Wert/Ziel auf Objekt und Index Wert/Ziel auf Eigenschaft können um eine Eindeutigkeitsprüfung ergänzt werden. Üblicherweise werden die auf diese Weise ergänzte Bausteine zur Konsistenzprüfung eindeutiger Kennzeichner verwendet. Sie stehen in der Auswahlliste der hinzufügbaren Indexbausteine zur Auswahl (z.B. Index Wert/Ziel auf Objekt (Eindeutigkeitsprüfung)).

Wenn ein neuer Wert geschrieben werden soll und einen gleichen Wert im Index vorfindet, so kann dieser neue Wert nicht übernommen werden. Werte werden dann als gleich erkannt, wenn sie auch von allen Verteilern des Indexes gleich gruppiert werden. Möchte man zum Beispiel eine Eindeutigkeitsprüfung nur je Domain (zum Beispiel erlaubt dies die Koexistenz von "modern" als Individuum von Verb und als Individuum von Adjektiv), so muss ein Verteiler je Definitionsbereich im Index enthalten sein.

Wenn auch ein Wertefilter konfiguriert wird, so wird die Eindeutigkeitsprüfung auf den gefilterten Werten durchgeführt. So kann zum Beispiel "arm" und "Arm" als gleich erkannt werden. Anmerkung: ein Wertefilter, der Zeichenketten zerlegt (für Volltext) kann zwar mit der Eindeutigkeitsprüfung kombiniert werden, dies erscheint in der Regel nicht sinnvoll, da bereits eine Teilzeichenkette nach der Zerlegung zu einem Duplikat führen kann, zum Beispiel "Das Haus" und "Haus und Hof".

Der Index Wert/Ziel auf Objekt kann bei mehrfach vorkommenden Eigenschaften keine doppelten Werte dieser Eigenschaften an einem Objekt als Duplikate erkennen. Es könnten also zwei gleichartige Attribute mit gleichem Wert am selben Objekt existieren, nicht jedoch an verschiedenen Objekten. Will man dies nicht erlauben, so muss am Attributbegriff das mehrfache Vorkommen deaktiviert werden oder stattdessen ein Index Wert/Ziel auf Eigenschaft zur Eindeutigkeitsprüfung verwendet werden.

1.2.6.4. Details zu Wertefilter

1.2.6.4.1. Wertzerlegung

Für Geokoordinaten und Intervall-Attribute kann kein atomarer Attributwert indexiert werden. Stattdessen wird mit Längengrad und Breitengrad bzw Intervall-Startwert und Intervall-Stopwert nur eine Komponente des Wertes indexiert. Für eine komplette Indexierung muss ein entsprechender Indexer für die jeweils andere Komponente des Wertes konfiguriert werden.

1.2.6.4.2. Zeichenketten-Manipulationen

Für Zeichenketten können im Admintool Volltext-Filter konfiguriert werden. An diesen kann konfiguriert werden, welche Manipulation an den Zeichenketten vorgenommen werden und wie die Zeichenketten in einzelne Worte zerlegt werden sollen. In Expertensuchen werden dann zusätzliche Operatoren angeboten, die mit der jeweiligen Bezeichnung des Filters ergänzt sind, damit gezielt mittels dieses Filters gesucht werden kann.

Mittels einer "Zeichenketten-Filterung" können die Zeichenketten in manipulierter Form indexiert werden, bei einer Suche werden dann alle Attributwerte als Treffer interpretiert, die durch den Filter auf die gleiche Zeichenkette abgebildet werden wie die Sucheingabe. Mittels einer "Zeichenketten-Zerlegung" können aus einem Text mehrere (manipulierte) Teilzeichenketten (Tokens) indexiert werden. Der zugehörige Index ermöglicht dann Expertensuchen, die mittels der Operatoren "Enthält Worte" und "Enthält Phrase" innerhalb der Zeichenketten suchen.

1.2.6.5. Metriken

An allen Eigenschaftstypen kann ein Attribut "Durchschnittliche Anzahl (berechnet)" angelegt werden. Der Wert des Attributes gibt an, wie viele Ausprägungen der zugehörigen Eigenschaft ein Objekt aus der Eigenschaftsdomäne durchschnittlich hat.

Mit dieser Information können Strukturabfragen besser entscheiden, wie sie ihre Ergebnismenge ermitteln. Zusätzlich kann ein Attribut "Durchschnittliche Anzahl (manuell)" angelegt werden, dessen Wert diesen Wert überschreibt. (Das ist dann sinnvoll, wenn die Domäne abstrakt ist, aber die Eigenschaft in Abfragen nur bei tatsächlichen Vorkommen angewendet werden soll.)

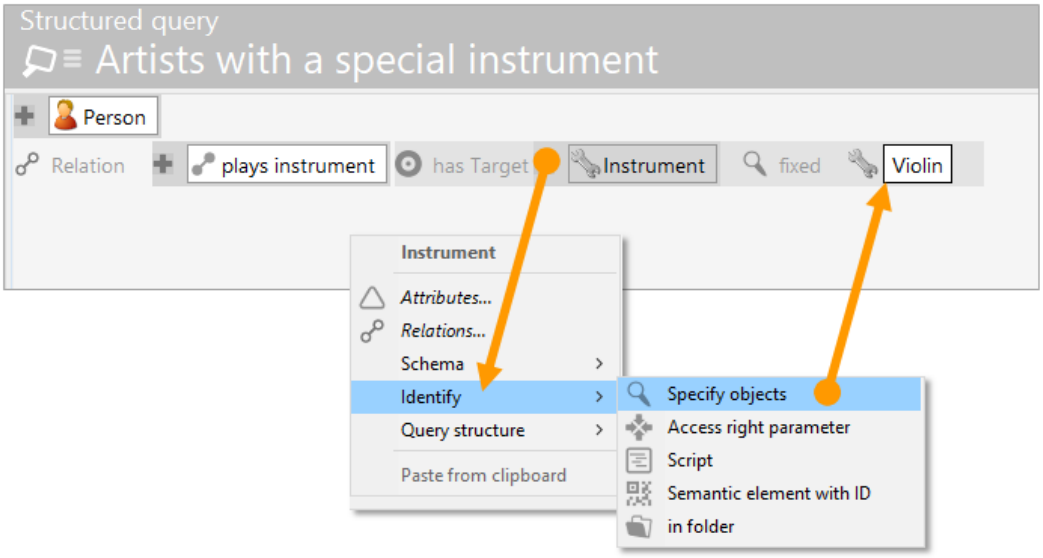
1.3. Suchen / Abfragen

Bei Abfragen des Knowledge Graphen unterscheiden wir zwischen verschiedenen Teilaufgaben: Gelegentlich möchten wir eine Eingabe des Nutzers über ein Suchfeld (Zeichenkette) verarbeiten. Meistens möchten wir bestimmte Pfade im Knowledge Graph herausfiltern, manchmal wollen wir dabei Gewichte vergeben. In i-views stehen dazu verschiedene Arten von Suchen zur Verfügung:

- Strukturabfragen
- Direkte Abfragen (Einfache Suche, Volltextsuche, Trigramm-Suche, Suche mit regulären Ausdrücken, Suche mit parametrisierter Trefferqualität)
- Such-Pipelines

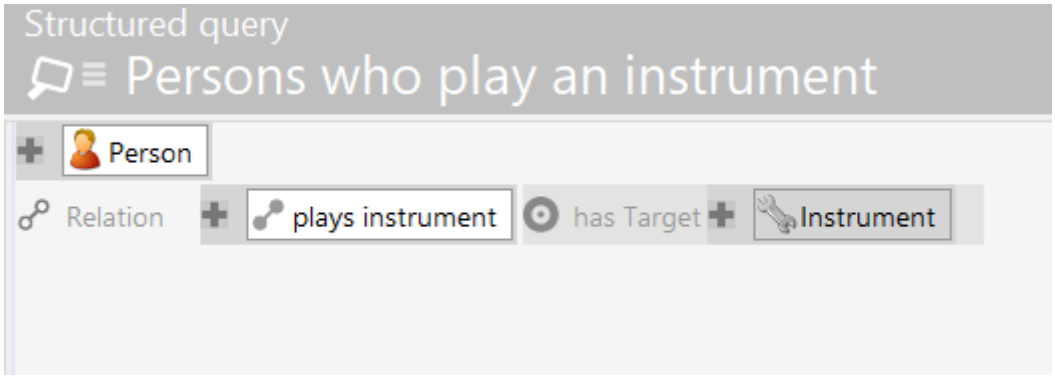
1.3.1. Strukturabfragen

Mit Strukturabfragen können Objekte zusammengesucht werden, die bestimmte Bedingungen erfüllten. Ein einfaches Beispiel für eine Strukturabfrage ist folgende: Es sollen alle Personen gefunden werden, die ein bestimmtes Instrument beherrschen.

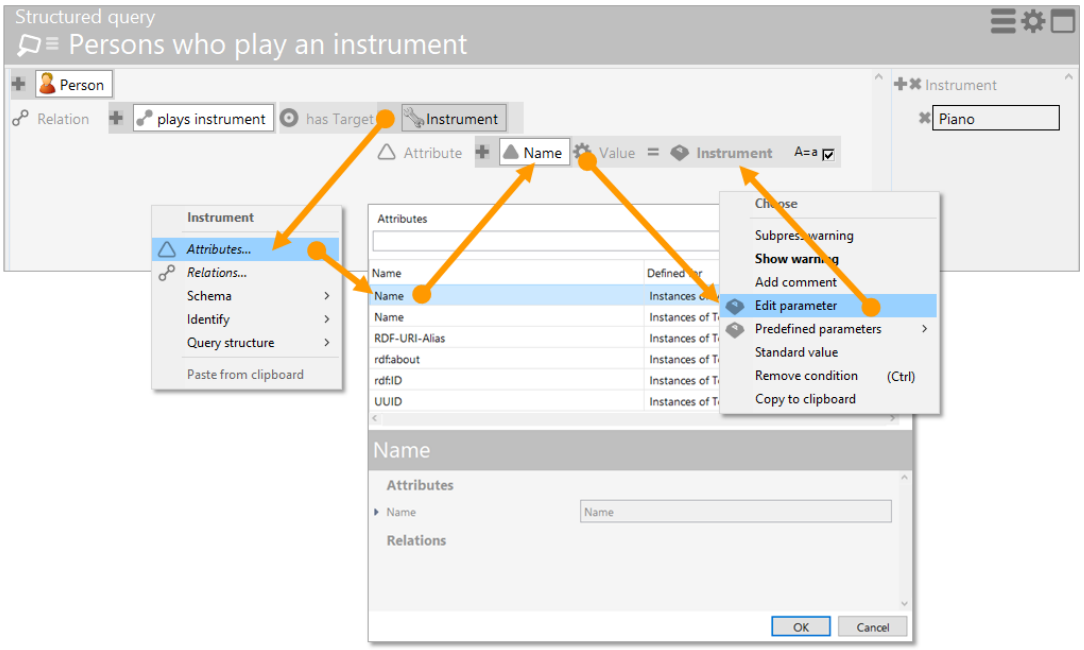


Als erstes kommt die Typbedingung: Es werden Objekte des Typs Person gesucht. Die zweite Bedingung: die Personen müssen ein Instrument beherrschen. Dritte Bedingung: dieses Instrument muss die Violine sein.

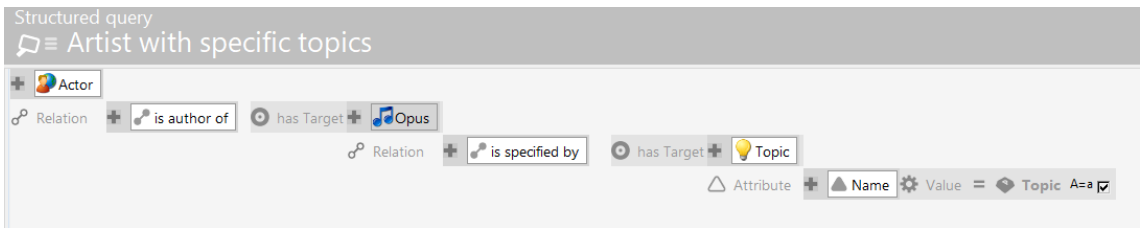
In der Strukturabfrage bilden die Relation "spielt Instrument" und ihr Ziel "Violine" zwei verschiedene Suchbedingungen. Wenn die zweite Bedingung, dass das Instrument eine Violine sein muss, weggelassen wird, finden sich in der Treffermenge Personen, welche mindestens ein (beliebiges) Instrument spielen:



Oft sollen Bedingungen (hier das Instrument) nicht schon vorher festlegt, oder aber vollständig zugelassen werden. Je nach Situation lässt sich in der Anwendung ein Instrument als Parameter mitgeben:



Die Bedingungen können dabei beliebig komplex werden und das Netz beliebig weit traversieren:

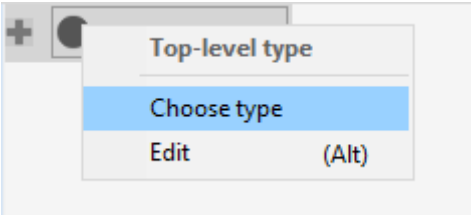


Geringfügig komplexeres Beispiel: Personen oder Bands, die in ihren Songs (genauer gesagt in mindestens einem) ein bestimmtes Thema behandeln. Hier wird der Suche nicht der Name, sondern die ID des Themas als Parameter mitgegeben — typisch für Suchen, die z.B. über einen REST-Service oder per Skript aufgerufen werden.

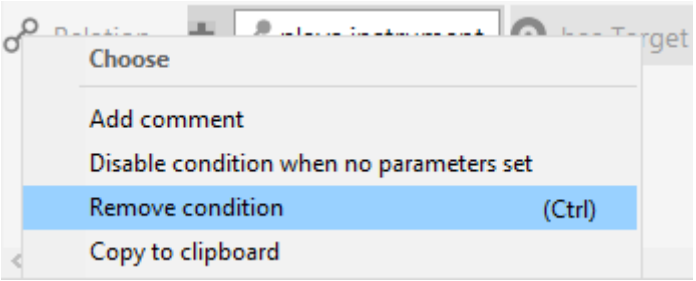
Die Typhierarchien werden in den Strukturabfragen automatisch mit berücksichtigt: Die Typbedingung "Werk" in der Suche oben schließt seine Untertypen "Album" und "Song" mit ein. Auch die Relationshierarchie wird berücksichtigt: Wenn es unterhalb von "ist Autor von" noch eine Differenzierung gibt (z.B. "schreibt Text" oder "schreibt Musik") werden beide Unterrelationen mit in die Suche einbezogen. Das gleiche gilt für die Attributtyp-Hierarchie.

1.3.1.1. Interaktion

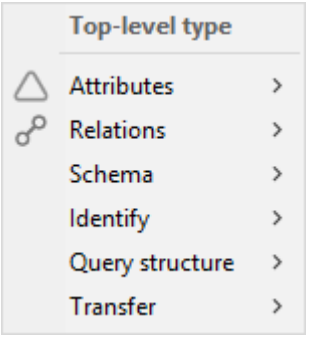
Wird eine neue Strukturabfrage angelegt, ist standardmäßig der Wurzeltyp des Knowledge Graph als Wurzelbedingung eingetragen. Um die Abfrage weiter einzuschränken, kann der Name einfach überschrieben, oder über Klick auf das Icon über *Typ ändern* ausgewählt werden.



Die Schaltfläche ✚ fügt weitere Bedingungen zur Strukturabfrage hinzu. Das Löschen von Bedingungen geschieht jeweils am Anfang jeder Zeile, wo die Art der Bedingung aufgeführt ist (Relation, Attribut, Ziel etc.). Alternativ kann die Bedingung durch Strg + Klick entfernt werden.



Beim Klick auf die Schaltfläche ✚ erscheint folgendes Menu, das je nach Kontext leicht abweichen kann.



Eine vollständige Erläuterung aller Bedingungen und Optionen der Strukturabfragen befinden sich in den nächsten Kapiteln.

1.3.1.2. Verwendung Strukturabfragen

Ein Hauptzweck der Strukturabfragen ist, in Anwendungen zu einem bestimmten Kontext Informationen zu liefern. Die Strukturabfrage aus dem letzten Abschnitt kann z.B. dem Endnutzer in einem Musikportal zu einem Thema wie Liebe, Drogen, Gewalt usw. eine Liste aller Künstler oder Bands generieren, die das Thema in ihren Liedern behandeln.

Dazu wird die Strukturabfrage zum Beispiel über ihren *Registrierungsschlüssel* in einen *REST-Service* eingebaut. Das Thema, für das sich der Nutzer gerade interessiert, geben wir der Abfrage mit seiner ID als Parameter mit.

> Ein Nutzer sucht durch Eingabe eines Suchstrings nach einem Thema. Es liegt also keine ID vor, sondern nur einen String anhand dessen das Thema identifiziert werden soll. Dabei soll aber im Suchergebnis gleich angezeigt werden, welche Bands Songs zu dem Thema geschrieben haben. Zu diesem Zweck kann eine Strukturabfrage als eine Komponente in eine *Such-Pipeline* eingebaut werden, hinter die Abfrage, die den Suchstring verarbeitet.

Strukturabfragen sind unter anderem deshalb ein zentrales Werkzeug innerhalb von i-views, weil auch die Bedingungen für *Rechte und Trigger* mit Strukturabfragen formuliert werden: Angenommen, es wird in einem Musikportal nur Künstlern und Bands erlaubt, Kommentare zu hinterlassen. Im Rechtesystem lässt sich entsprechend formulieren, dass nur Künstler und Bands, die zu einem bestimmten Thema mindestens einen Song geschrieben haben, zu diesem Thema Kommentare hinterlassen dürfen. Strukturabfragen können auch in Exporten benutzt werden, um zu bestimmen, welche Objekte exportiert werden soll.

Alle diese Verwendungen haben eines gemeinsam: wir sind nur an qualitativen, keinen gewichteten Aussagen interessiert. Das ist die Domäne der Strukturabfragen gegenüber den Such-Pipelines.

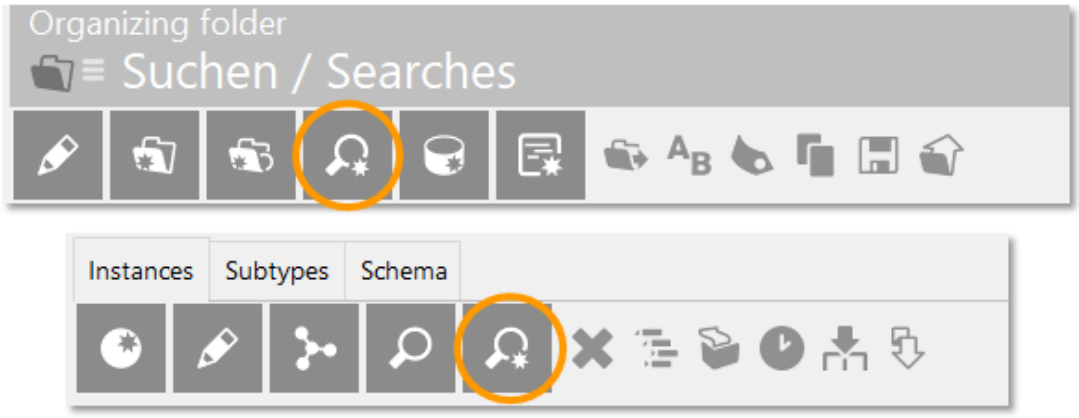
Nicht zuletzt sind Strukturabfragen auch Hilfsmittel für uns Knowledge-Engineers. Mit ihnen können wir uns einen Überblick über das Netz verschaffen und *Reports* sowie *To-do-Listen* zusammenstellen. Beispielfragen, die mithilfe von Strukturabfargen beantwortet werden können, sind:

- Zu welchen Themen gibt es wie viele Künstler/Bands?
- Müssen bestimmte Themen ausgebaut werden weil sich zu viele Relationen ansammelt haben, oder sollten umgekehrt spärlich besetzte Themen zusammenlegt oder geschlossen werden?

Für diese Verwendung ist es sinnvoll, die Strukturabfragen in *Ordnern* organisieren zu können.

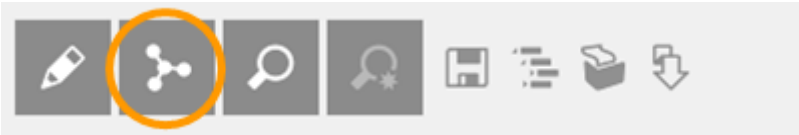
1.3.1.2.1. Ausführen

Durch einen Klick auf die Schaltfläche *Suchen* über dem Fenster für die Trefferliste wird die Strukturabfrage ausgeführt.



Das Suchergebnis kann dann weiter bearbeitet (z.B. mithilfe des Speichersymbols in einen neuen Ordner kopiert) werden.

Den Weg, den die Strukturabfrage genommen hat, kann zur Rückverfolgung der Suchergebnisse im Graph-Editor betrachtet werden. Damit die entsprechende Schaltfläche angeklickt werden kann, müssen die zu betrachtenden Suchergebnisse zunächst selektiert werden.



Eine Strukturabfrage kann kopiert werden, etwa um verschiedene Varianten zu erstellen. Ebenfalls besteht die Möglichkeit sie auch unabhängig vom Netz im XML-Format zu speichern. Die Strukturabfrage ließe sich somit in ein anderes Netz importieren. Das beschränkt sich aber auf Versionen desselben Netzes, z.B. auf Sicherheitskopien, da die Strukturabfrage Objekttypen, Relations- und Attributtypen über ihre internen IDs referenziert.

1.3.1.3. Aufbau von Strukturabfragen

Innerhalb von Strukturabfragen lassen sich auch indirekte Bedingungen formulieren: Durch die Struktur des Knowledge Graphen kann beliebig zwischen den Elementen traversiert werden. Es lassen sich Künstler oder Bands heraussuchen, die Songs zu bestimmten Themen geschrieben haben, welche wir jedoch nicht konkret mit Titel benennen können.

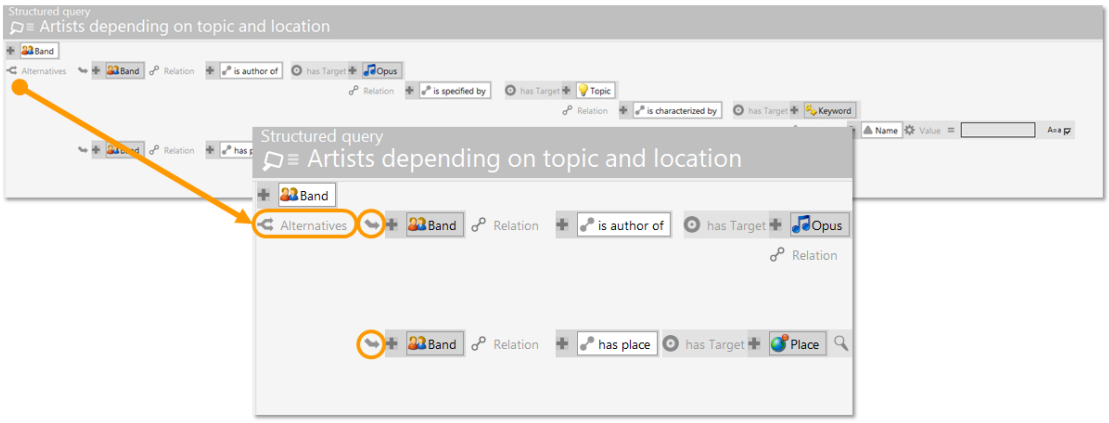
1.3.1.3.1. Mehrere Bedingungen

Bedingungsketten können beliebig tief sein, es lassen sich aber auch mehrere parallele Bedingungen formulieren. An einem beliebigen Bedingungselement werden zusätzliche Bedingungen als weiterer Ast angefügt:

Mehrere Bedingungen: Englische Bands mit Songs über ein bestimmtes Thema

1.3.1.3.2. Alternative Bedingungen

Im oben genannten Beispiel werden nur Künstler oder Bands gefunden, die sowohl Songs zu einem festgelegten Thema geschaffen haben, als auch aus England stammen. Wenn wir stattdessen alle Künstler und Bands finden wollen, auf die eine der beiden Bedingungen zutrifft, werden sie als *Alternative* formuliert. Durch anklicken des Symbols der Bedingung in Form der Relation "ist Autor von" kann dort im Menü eine Alternative gewählt werden:



Alternative Bedingungen: Die Band muss entweder englisch sein oder Songs zu einem bestimmten Thema haben

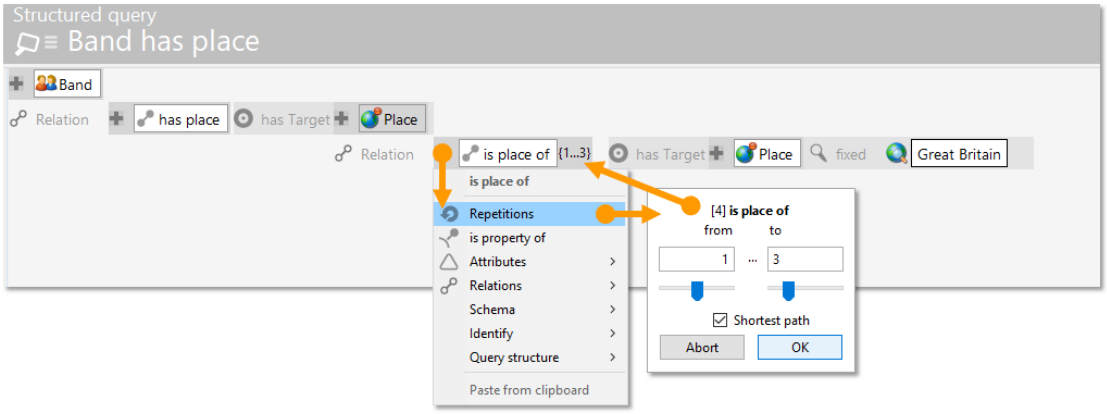
Sind weitere Bedingungen außerhalb der Alternativ-Klammer vorhanden, befinden sich Objekte in der Treffermenge die *mindestens eine* der Alternativen, sowie *alle weiteren* Bedingungen erfüllen.

1.3.1.3.3. Transitivität / Wiederholungen

Angenommen, im Netz sind die Bands entweder Städten oder Ländern zugeordnet. Von diesen wiederum ist bekannt, welche Städte in welchen Ländern liegen. Um diesen Sachverhalt in der Suche zu erfassen, lässt sich die Bedingungskette einfach erweitern: wir können etwa nach Bands suchen, die einer Stadt zugeordnet sind, die wiederum in England liegt. Auf diese Weise werden jedoch die Bands nicht gefunden, welche direkt England zugeordnet sind. Um das zu vermeiden können wir bei der Relation "liegt in" angeben, dass sie optional ist, also nicht vorliegen muss.

Gleichzeitig können wir mit der Funktion *Wiederholungen* auch Hierarchien berücksichtigen, die mehrere Ebenen tief sind. Beispielsweise ist von der Band ZZ Top bekannt, dass sie aus der Stadt Houston stammt. Diese Stadt liegt wiederum in Texas. Um die Band auch als Ergebnis zu erhalten,

wenn nach Bands aus den USA gefragt wird, können wir bei der Relation "liegt in" angeben, dass bis zu n Wiederholungen der Relation nachgegangen werden soll:



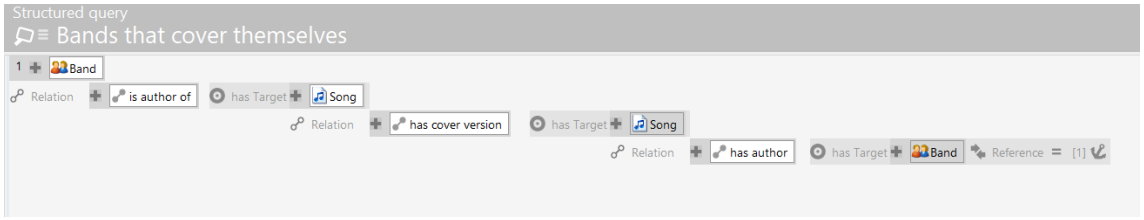
1.3.1.3.4. Negativ-Bedingungen

Bedingungen lassen sich negieren. Beispielsweise sollen Punk-Bands gesucht werden, die jedoch nicht aus Großbritannien kommen. Dazu wird die Negativ-Bedingung als eine sogenannte Hilfssuche aufgebaut:

Die Hilfssuche liefert Bands auf Großbritannien. Von der Hauptsuche aus kann eine Referenz hergestellt und dabei angegeben werden, dass die Suchergebnisse den Kriterien der Hilfssuche gerade nicht entsprechen dürfen. Damit ziehen wir die Ergebnisse der Hilfssuche von denen der Hauptsuche ab und erhalten nur Bands, die nicht aus Großbritannien kommen.

1.3.1.3.5. Referenzen

Durch Referenzen ist es möglich, sich innerhalb einer Strukturabfrage auf andere Bedingungen derselben Abfrage zu beziehen:



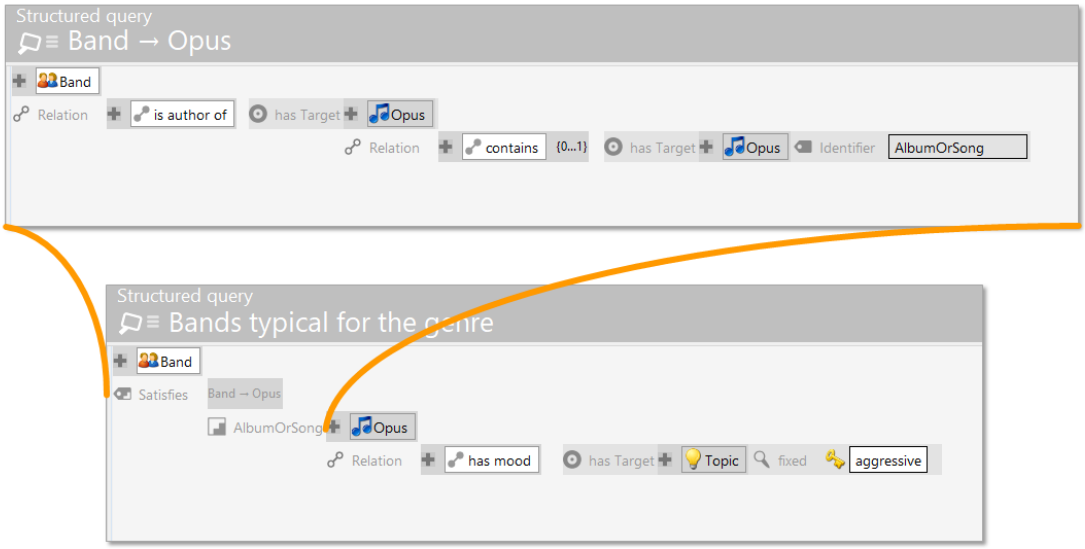
Hier referenziert die letzte Bedingung (Nr. 7) die allererste, d.h. die Band, die die Coverversion schreibt, muss auch Autor des Originals sein. Ohne Referenz würde sich die Suche folgendermaßen lesen: Bands, die Songs geschrieben haben, die andere Songs covern, die von (beliebigen) Bands geschrieben wurden.

1.3.1.3.6. Weitere Optionen im Aufbau der Strukturabfragen

Suchbausteine

Andere Strukturabfragen, aber auch andere Suchen beliebiger Art, lassen sich als Baustein in Strukturabfragen einbinden. Dadurch besteht die Möglichkeit, sich wiederholende Teilabfragen in eigene Bausteine auszulagern und so z.B. bei Modelländerung das Verhalten an einer zentralen Stelle anzupassen. Ein Baustein kann bei jeder Bedingungszeile eingebaut werden.

Beispiel aus unserem Musiknetz: Zu allen Werken einer Band gehören entweder direkt zugeordnet Songs, in einem Album enthaltene Songs oder die Alben selbst. Diesen Zusammenhang brauchen wir als Teilabfrage häufiger, u.a. in einer Strukturabfrage, die Bands zu einem bestimmten *Mood* zurückliefert. Wir beginnen diese Abfrage mit einer Typbedingungen — wir suchen Bands — und binden als Bedingung für diese Bands den vorher definierten Baustein ein:



Die Objekte, die die als Baustein eingebundene Strukturabfrage zurückliefert, müssen vom Typ her zu der Bedingung, bei der sie eingebunden ist, passen.

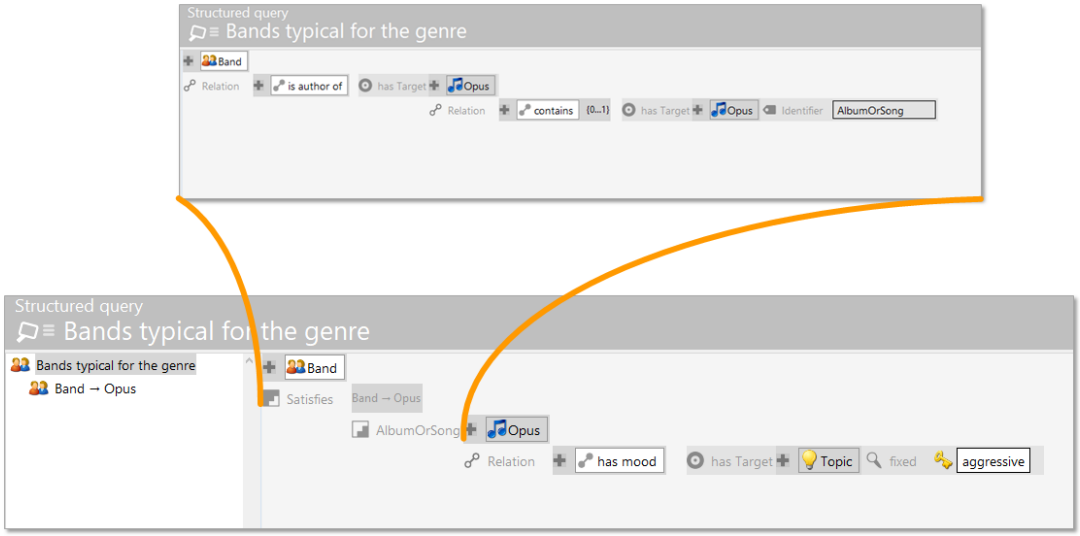
Mit Hilfe der *Bezeichner*-Funktion kann die Abfrage noch mit zusätzlichen Bedingungen

weitergeführt werden.

In unserem Fall werden die Alben und Songs, von denen die Baustein-Abfrage ausgeht, von der aufrufenden Abfrage bestimmt: nämlich Alben und Songs mit dem Mood "Agressive". Das einbinden des Suchbausteins in eine Strukturabfrage wird über das Menü *Abfragestruktur* vorgenommen. Unter *Strukturabfrage-Baustein (registriert)* liegt eine Auswahlliste mit allen registrieren Bausteinen. Der Vorteil hierin liegt in der Wiederverwendbarkeit des Strukturabfrage-Bausteins für weitere Strukturabfragen.

WARNUNG	Ein Deregistrieren des Strukturabfrage-Bausteins führt zur Löschung des Strukturabfrage-Bausteins.

Abgesehen von den registrierten Strukturabfrage-Bausteinen gibt es die Möglichkeit, Strukturabfrage-Bausteine als lokale Makros anzulegen. In diesem Fall ist eine Wiederverwendung innerhalb derselben Strukturabfrage möglich.



Einfache Suche

Mit der Suchbedingung *Suche* kann das Ergebnis einer einfachen Suche oder einer Such-Pipeline als Input für eine Strukturabfrage dienen. Das Eingabefeld beinhaltet die Sucheingabe an die einfache Suche. Durch weitere Bedingungen können z.B. einfache Suchen weiter gefiltert werden.

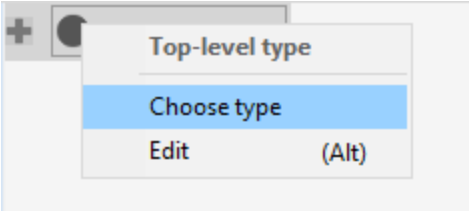
Kardinalitätsbedingung

Eine Suche nach Attributen bzw. Relationen kann mit einem Kardinalitäts-Operator (durch # gekennzeichnet) versehen werden. Möglich sind Kardinalität größer-gleich, kleiner-gleich und gleich. Der normale Gleichheits-Operator der Relations- oder Attributsbedingung entspricht der Kardinalität größer-gleich 1.

1.3.1.4. Bedingungen im Detail

1.3.1.4.1. Typbedingung

An der Wurzel wird in der Strukturabfrage definiert, welche Objekte als Ergebnisse erscheinen sollen. Dazu wird das Typ-Icon der ersten Bedingung angeklickt und im Menü *Typ wählen* die Eingabemaske gestartet, in die der Typname eingeben werden kann.

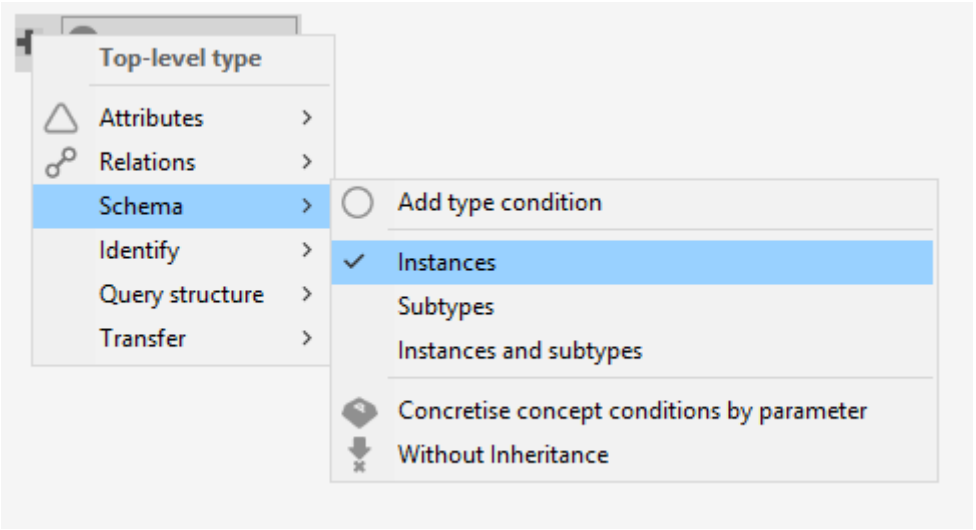


Alternativ kann hinter dem Typ-Icon der Text mit dem Typnamen überschrieben werden.

Im zweiten Schritt wird die Relationsbedingung hinzugefügt. Bspw. wird nach dem Herkunftsort einer Band gesucht und "hat Ort" als Relationsbedingung gesetzt. Es wird automatisch der Zieltyp der Relation eingefügt, der aber ebenfalls geändert werden kann (wenn z.B. die "hat Ort" Relation für Länder, Städte, Regionen gilt, wir aber in unserer Abfrage nur Städte zurückbekommen möchten).

Mehrere hintereinander definierte Typbedingungen werden in der Abfrage als "oder"-Bedingung interpretiert. Folgendermaßen suchen wir z.B. Werke oder Veranstaltungen zu einem bestimmten Musikstil:

Für eine Typbedingung stehen noch weitere Funktionen zur Verfügung. Im Kontextmenü, welches über die Schaltfläche ➕ erreichbar ist, gibt es dazu den Punkt *Schema*:



Wir können statt konkreter Objekte nur Objekttypen suchen oder beides gleichzeitig, indem im

Menü *Schema* der Haken bei *Untertypen* bzw. *Objekte und Untertypen* gesetzt wird:

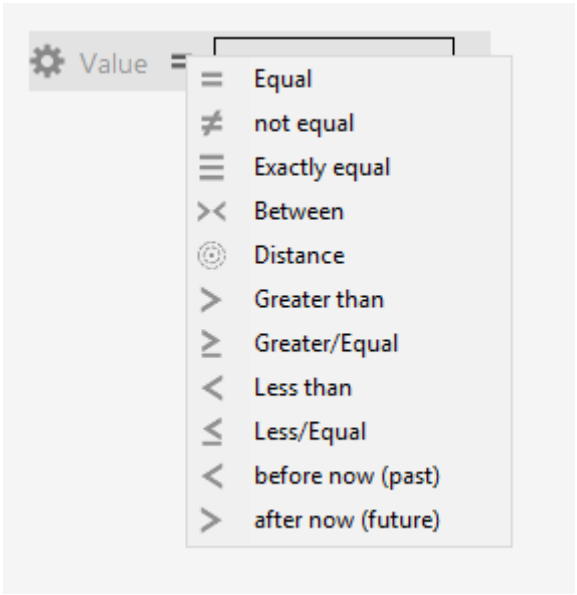
So sieht die Bedingung aus, wenn sowohl konkrete Werke als auch Untertypen von Werk (Alben, Songs) gesucht werden.

Ohne Vererbung

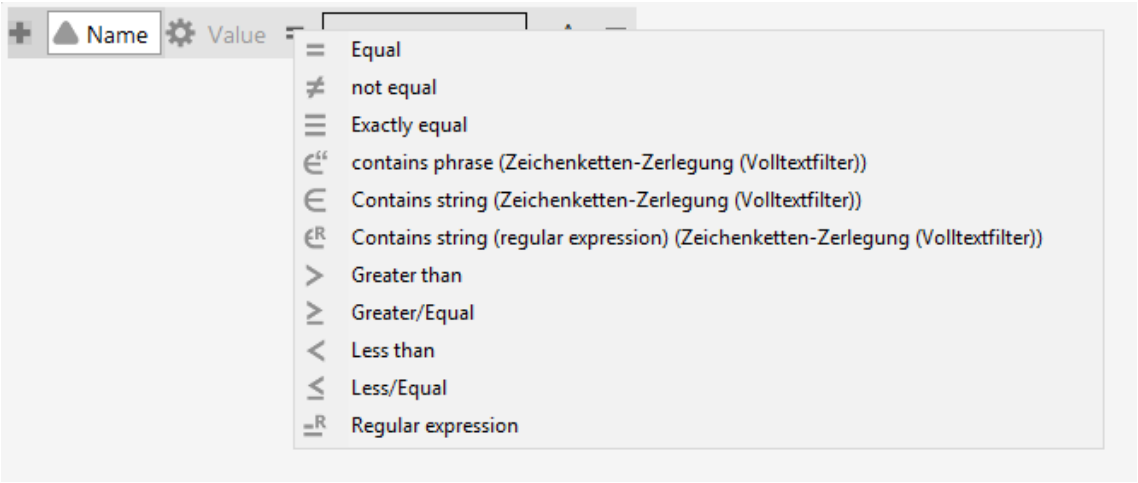
Normalerweise wirkt bei allen Typbedingungen der Strukturabfragen automatisch die Vererbung. Wird nach Veranstaltungen gesucht, bei denen Bands einer bestimmten Musikrichtung spielen, dann werden automatisch alle Untertypen von Veranstaltung mit in die Suche einbezogen und entsprechend Hallenkonzerte, Clubkonzerte, Festivals etc. mit zurückgeliefert. In der überwiegenden Mehrzahl der Fälle ist das genauso gewünscht. Für die Ausnahmen gibt es die Möglichkeit, die Vererbung auszuschalten und die Suche auf direkte Objekte vom Typ Veranstaltung einzuschränken, d.h. die Objekte der Untertypen auszuschließen.

1.3.1.4.2. Operatoren zum Vergleich von Attributwerten

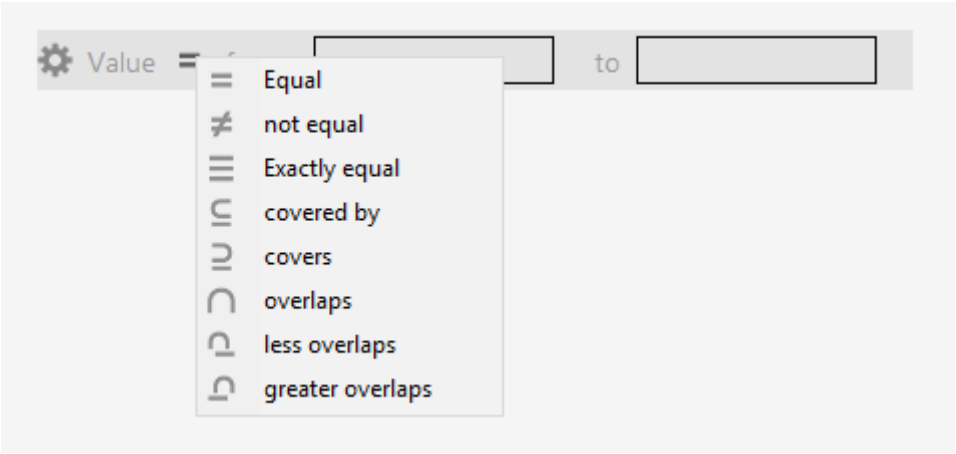
Für Attribute können Wertbedingungen formuliert werden. Etwa sollen Bands, die in der Zeit nach dem Jahre 2005 gegründet wurden oder Songs, die mehr oder weniger 3 Minuten lang sind oder Songs, die im Titel das Wort "Planet" enthalten gesucht werden. Hierzu werden Vergleichsoperatoren benötigt. Welche Vergleichsoperationen uns i-views anbietet, ist abhängig vom Datentyp des Attributs:



Vergleichsoperationen für Datumsangaben und Quantitäten



Vergleichsoperationen für Zeichenketten



Vergleichsoperatoren für Intervalle

Übersicht zu verfügbaren Vergleichsoperatoren

Operator	Attributwerttypen	Beschreibung	Beispiel
Gleich		Findet Werte, die gleich dem Suchwert sind.Bei Zeichenketten werden die Wildcards "*" (beliebige Teilzeichenketten) und "?" (ein beliebiges zeichen) unterstützt.	Suchwert "Stern*" findet "Stern" und "Sternchen"
Exakt gleich	Zeichenkette	Findet Zeichenketten, die gleich dem Suchwert sind, ohne Wildcards anzuwenden	Suchwert "Stern*" findet "Stern*", aber nicht "Stern" oder "Sternchen"

Operator	Attributwerttypen	Beschreibung	Beispiel
Enthält Phrase	Zeichenkette mit Volltextindex	Findet Zeichenketten, die den Suchwert als Teilsatz enthalten.	Suchwert "Bauer Georg" findet "Bauer Georg Grün", aber nicht "Georg Bauer"
Enthält Zeichenkette (Zeichenkettenzerlegung)	Zeichenkette mit Volltextindex	Findet Zeichenketten, die alle Worte des Suchwerts in beliebiger Reihenfolge enthalten	Suchwert "Bauer Georg" findet "Bauer Georg Grün" und "Georg Bauer", aber nicht "Georg Grau"
Enthält Zeichenkette (Regulärer Ausdruck)	Zeichenkette mit Volltextindex	Findet Zeichenketten, bei denen mindestens ein Wort dem als regulären Ausdruck interpretierten Suchwert entspricht.	Suchwert "Ba[yi]e?r" findet "Silke Bayer" und "Emil Bair", aber nicht "Bauer"
Regulärer Ausdruck	Zeichenkette	Findet Zeichenketten, die dem als regulären Ausdruck interpretierten Suchwert entsprechen.	"\d+\s\w+" findet "64293 Darmstadt"
Ungleich		Findet Werte, die nicht gleich dem Suchwert sind.Bei Zeichenketten werden die Wildcards "*" (beliebige Teilzeichenketten) und "?" (ein beliebiges zeichen) unterstützt.	
Grösser als	Alle Attribute mit sortierbaren Werten	Findet Werte, die größer als der Suchwert sind.	
Grösser/Gleich	Alle Attribute mit sortierbaren Werten	Findet Werte, die größer oder gleich dem Suchwert sind.	
Kleiner als	Alle Attribute mit sortierbaren Werten	Findet Werte, die kleiner als der Suchwert sind.	
Kleiner/Gleich	Alle Attribute mit sortierbaren Werten	Findet Werte, die kleiner oder gleich dem Suchwert sind.	

Operator	Attributwerttypen	Beschreibung	Beispiel
Gleich (Geo)	Geo	Findet geographische Positionen, die gleich dem Suchwert	
Gleich jetzt (Gegenwart)	Datum	Findet Datumsangaben, die dem aktuellen Datum entsprechen. Bei Datum+Uhrzeit muss auch die Uhrzeit übereinstimmen.	
Vor jetzt (Vergangenheit)	Datum	Findet Datumsangaben, die in der Vergangenheit liegen.	
Nach jetzt (Zukunft)	Datum	Findet Datumsangaben, die in der Zukunft liegen.	
Abstand	Datum, Geo, Zahl	Findet Werte, deren Abstand zum Suchwert maximal dem Abstandswert entspricht (Datum: Anzahl Tage, Geo: Entfernung in Metern). Als Parameterwert wird der Abstand durch eine Tilde getrennt, also z.B. "1.10.2019 ~ 30" - d.h. am 1.10.2019 plus/minus 30 Tage.	Suchwert "1.10.2019" mit Abstand 30 findet "15.10.2019", aber nicht "1.11.2019"
Zwischen	Alle Attribute mit sortierbaren Werten	Findet Werte, die zwischen dem unteren und oberen Suchwert liegen. Bei Parameterwerten ist ein Bindestrich erforderlich, also z.B. "10.1.2005 — 20.1.2005".	"1.3.1990 — 31.3.1990" findet "1.3.1990" und "4.3.1990", aber nicht "2.4.1990".
Enthält	Intervall	Findet Intervalle, die den Suchwert komplett einschließen	Suchwert "1 — 5" findet "1 — 3", aber nicht "3 — 6"

Operator	Attributwerttypen	Beschreibung	Beispiel
Ist enthalten in	Intervall	Findet Intervalle, die komplett im Suchwert enthalten sind	"2 — 4" findet "1 — 6", aber nicht "1 — 3"
Überschneidet	Intervall	Findet Intervalle, die ein gemeinsames, nicht-leeres Teilintervall mit dem Suchwert haben.	"2 — 4" findet "1 — 3" und "3 — 6", aber nicht "4 — 5"
Überschneidet von oben	Intervall	Findet Intervalle, die ein gemeinsames, nicht-leeres Teilintervall mit dem Suchwert haben, das die untere Grenze des Intervalls beinhaltet.	"2 — 4" findet "3 — 6", aber nicht "1 — 3"
Überschneidet von unten	Intervall	Findet Intervalle, die ein gemeinsames Teilintervall mit dem Suchwert haben, das die obere Grenze des Intervalls beinhaltet.	"2 — 4" findet "1 — 3", aber nicht "3 — 6"

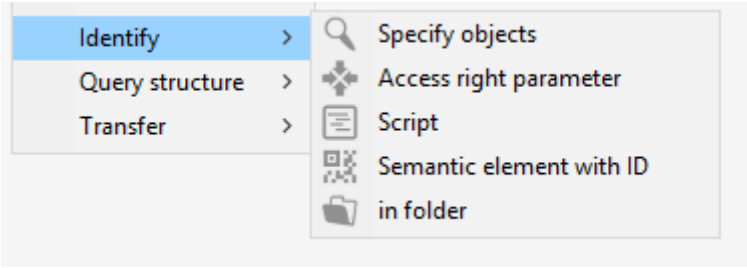
Vergleichswert ergibt sich aus Skript

Attributwertbedingungen lassen sich in Teilsuchen entfernen und durch eine Skript-Attributwertbedingung ersetzen. Die Ergebnisse des Skripts werden dann als Vergleichswert für die Attributwertbedingung benutzt, bspw. falls die Vergleichsoperationen für eine spezifische Abfrage nicht ausreichen.

1.3.1.4.3. Objekte identifizieren

Die Strukturabfrage sieht mehrere Möglichkeiten vor, Objekte in der semantischen Graphdatenbank zu identifizieren. In den bisherigen Beispielen haben der Einfachheit halber oft Objekte festgelegt. Eine solche manuelle Festlegung kann in der Praxis hilfreich sein um Strukturabfragen zu testen oder um für eine Parametereingabe einen (austauschbaren) Default festzulegen.

Hier haben wir bereits die Kombination mit dem Namensattribut kennen gelernt, das kann natürlich ein beliebiges Attribut sein. Im Menüpunkt *Identifizieren* finden wir noch einige andere Möglichkeiten, Startpunkte für die Strukturabfrage zu bestimmen:



1.3.1.4.4. Zugriffsrechtparameter

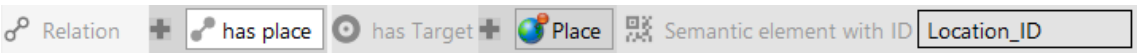
Das Ergebnis der Abfrage vom kann vom Anwendungskontext abhängig gemacht werden. Vor allem in Verbindung mit der Konfiguration von Rechten und Triggern trifft dies zu, wenn im allgemeinen nur *Benutzer* verwendbar ist.

1.3.1.4.5. Skript

Die an dieser Stelle einzugebenden Objekte werden mit dem Ergebnis eines Skriptes bestimmt.

1.3.1.4.6. Knowledge Graph Element mit ID

Man kann ein Objekt auch über seine interne ID festlegen. Diese Bedingung wird in der Regel nur in Verbindung mit Parametern und der Benutzung über die REST-Schnittstelle benutzt.



1.3.1.4.7. Ordner

Mit der Suchbedingung *liegt in Ordner* kann der Inhalt einer Sammlung semantischer Objekte als Input in eine Strukturabfrage rein gegeben werden. Mit dem Auswahlsymbol kann ein Ordner innerhalb der Arbeitsordnerhierarchie ausgewählt werden. Die Objekte der Sammlung werden bezüglich aller weiteren Bedingungen (inkl. Begriffsbedingungen) gefiltert.

1.3.1.4.8. Parameter-Bedingungen

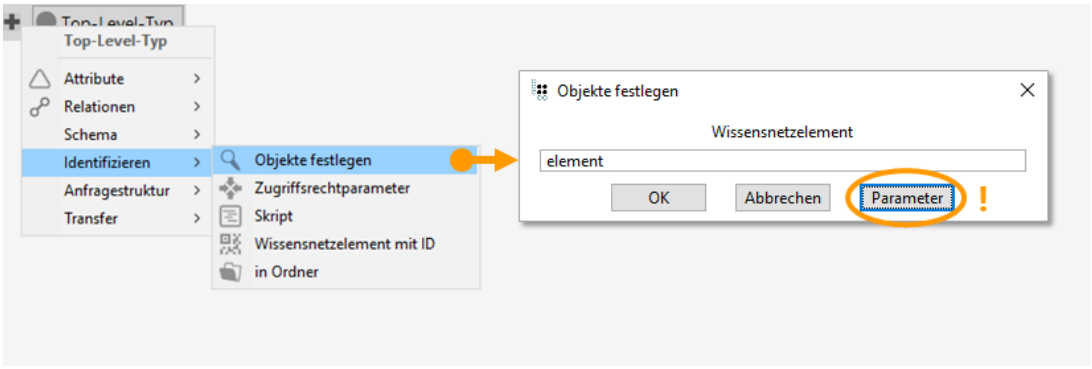
Parameter

Die Verwendung von Parametern in Strukturabfragen erlaubt die Übergabe von Werten im JavaScript-Code an die Abfrage. Parameter können frei vergeben werden, wobei eine Abfrage wie folgt aufgerufen wird:

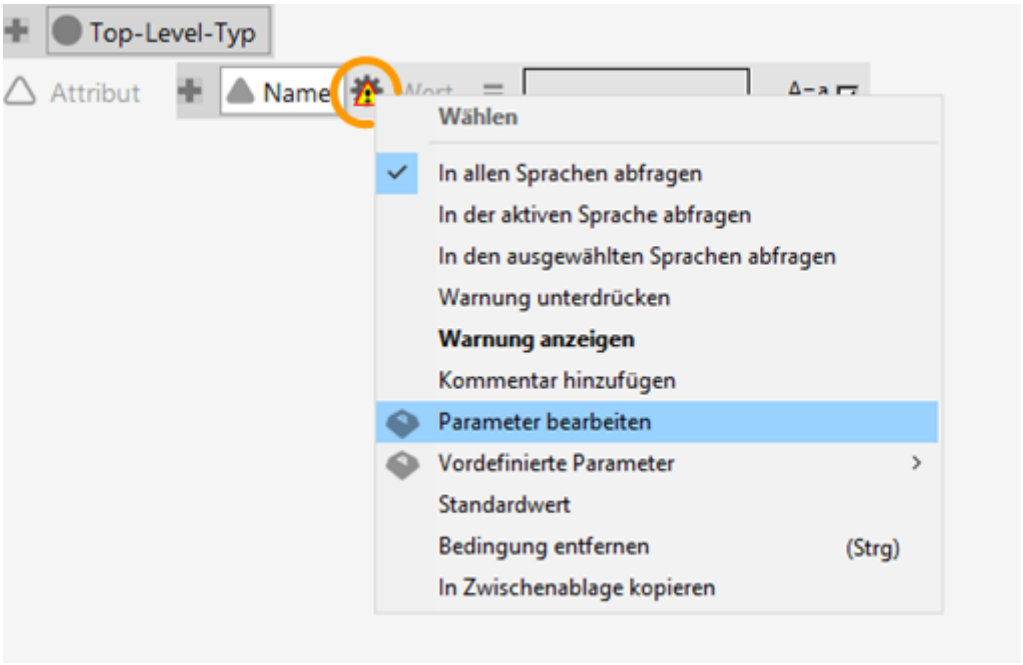
```
$k.Registry.query("<registryKeyOfQuery>").findElements({
  "<parameterNameInQuery>": "<input>"
})
```

Parameter können wie folgt konfiguriert werden:

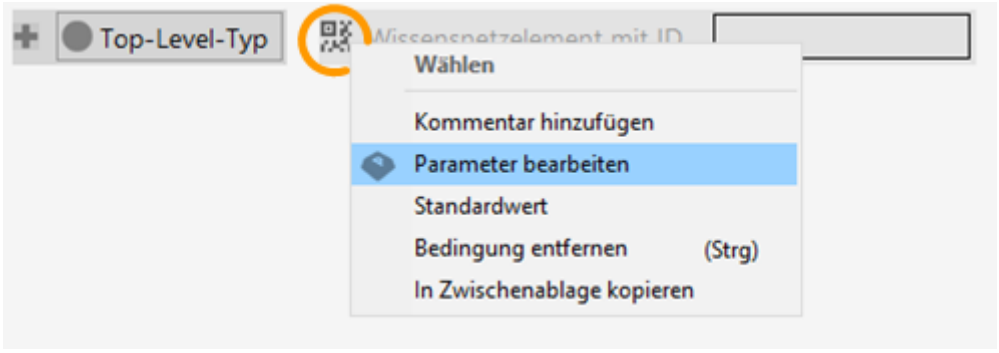
- Als semantisches Element



- Als Attributwert



- In Form einer Element-Id



Das Testen von Strukturabfragen mit Parametern ist auf zwei Arten möglich:

- Verwendung der Testumgebung der Strukturabfrage

- Aufruf der Strukturabfrage mittels Skript (durch Ausführen oder Debuggen des Skriptes)

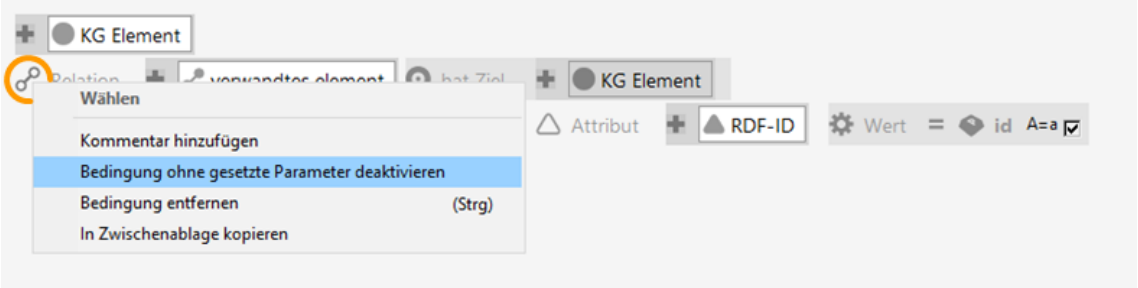
Generell können parameter folgenden Bedingungen unterliegen:

- Parameter ist gesetzt
- Parameter ist nicht gesetzt
- Parameter ist deaktiviert
- (Parameter erhält leere Zeichenkette)

Optionale parameter

Die Strukturabfrage bietet die Möglichkeit, optionale Parameter zu verwenden. An bestimmten Stellen wie Relationen oder Alternativzweigen steht im Kontextmenü ein Eintrag zur Verfügung:

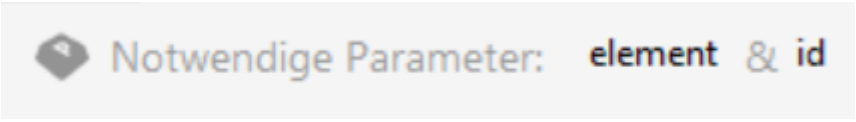
Bis 5.3: *Bedingung ohne gesetzten Parameter deaktivieren*



Ab 5.4: *Notwendiger Parameter*

Wenn diese Option gesetzt ist und zutrifft (Parameter ist tatsächlich per Eingabe deaktiviert), dann wird die Bedingung von dieser Stelle bis zum Ende der Abfrage-Verzweigung bei der Ermittlung der Suchergebnisse nicht berücksichtigt.

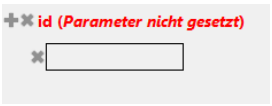
Wenn sich mehrere Parameter in einem Abfragezweig befinden, dann gilt für die Parameter die UND-Logik:

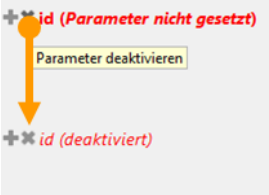


Wenn alle notwendigen Paremeter deaktiviert sind, wird der nachfolgende Abfragezweig bei der Abfrage ausgelassen, andernfalls werden die gesetzten Parameter berücksichtigt.

HINWEIS	Wenn der Parameter nicht gesetzt ist, wirft die Testumgebung trotz des optionalen oder notwendigen Parameters einen Fehler. In diesem Fall muss für ein Ausführen der Abfrage der Parameter **deaktiviert** werden, um die Bedingung des ungesetzten Parameters zu testen.

Wichtige Regeln zum Setzen eines Parameters

Parameter-Bedingung	Zustand in Strukturabfrage	Zustand in JavaScript	Ergebnis
Parameter ist gesetzt	Parameterwert ist eingegeben	Variable für Übergabe an parameter ist definiert	Parameter-Bedingung wird im Suchergebnis berücksichtigt
Parameter ist **nicht gesetzt**	Parameterwert wurde nicht eingegeben (direktes Ausführen der Abfrage) 	Es wird kein Parameter übergeben Verwendung des Methodenaufrufs `findElements()` oder `findHits()` ohne Argumente, oder Setzen des Parameters auf `undefined`.	Fehler: "Parameter xy fehlt"

Parameter-Bedingung	Zustand in Strukturabfrage	Zustand in JavaScript	Ergebnis
Parameter ist **deaktiviert**	Klick auf x neben dem Parameter 	Setzen des Parameters auf `null`.	• Mit herkömmlichem Parameter: Verhalten so, als ob die Parameter-Bedingung in der Strukturabfrage nicht gesetzt worden wäre • Mit optionalem oder notwendigem Parameter: Der Abfragezweig von der optionalen Bedingung bis zum Ende des der Verzweigung wird komplett ausgelassen
Parameter enthält leere Zeichenkette	Eingabe von `''` oder `""`, mit Verwerfen der Dialog-Auswahl	Variable für Parameter enthält leere Zeichenkette `''` oder `""`	Abfragezweig wird ausgelassen; wenn keine Alternativen existieren, liefert u. U. die gesamte Abfrage kein Ergebnis

WARNUNG

Risiko von Suchergebnissen mit falschen Treffern ("false positives")

Für eine Vorhersagbarkeit und Verlässlichkeit von Suchergebnissen in Skripten ist zu vermeiden, dass Parameterwerte unbeabsichtigt `null` sind, da hierbei systemseitig keine Fehlermeldungen ausgegeben werden. Daher sind im JavaScript-Code Kontrollstrukturen zu verwenden, die unerwartete Bedingungen abfangen.

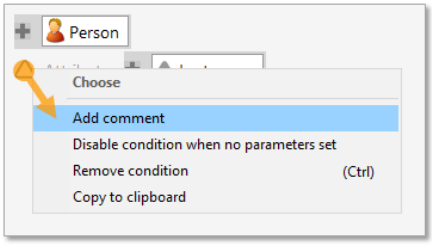
Wenn der optionale Parameter einer Strukturabfrage mithilfe eines Skripts in den Views einer Suche oder in einer Suchergebnis-Ansicht übergeben wird, dann muss auch die Wertart des zu übergebenden Parameters auf *optional* gesetzt werden. Wenn die Wertart auf *obligatorisch* gesetzt ist, dann liefert die Strukturabfrage kein Suchergebnis, wenn das zugrundeliegende Skript den Parameterwert `null` übergibt (mit der Absicht der Deaktivierung des optionalen Parameters).

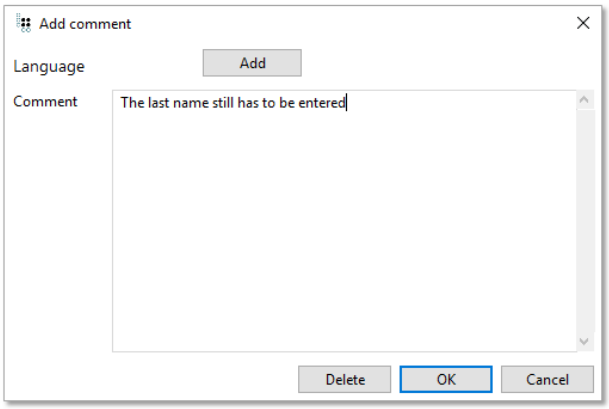
1.3.1.5. Kommentare in Strukturabfragen

1.3.1.5.1. Hinzufügen von Kommentaren

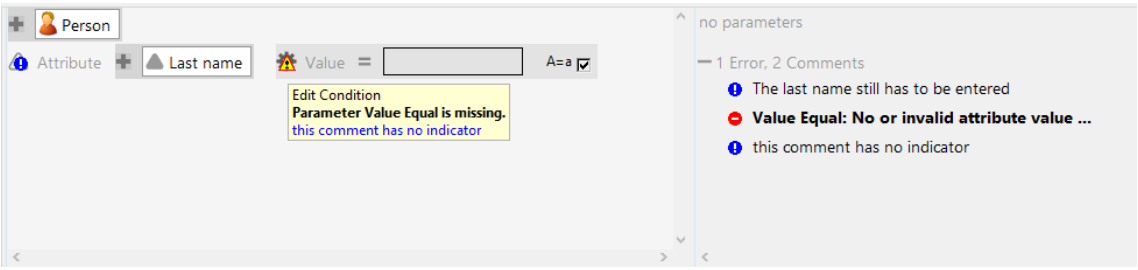
Jede Bedingung kann einen Kommentar erhalten: Kontextmenü *Kommentar hinzufügen*. Ein existierender Kommentar erzeugt an der Bedingung einen blauen Indikator, der bei Mouseover den Text anzeigt.

Mit dem Menüeintrag *Kommentar bearbeiten* kann dieser dann angepasst oder entfernt werden:



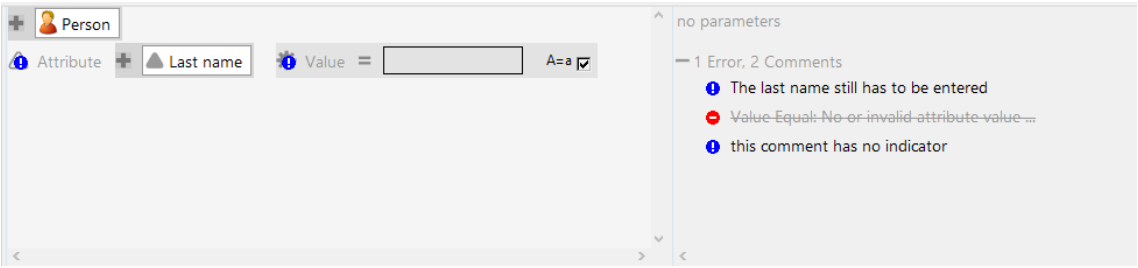


Der Indikator für Kommentare wird nicht angezeigt, wenn an der Bedingung eine Warnung oder Fehlermeldung existiert. In dem Fall sieht man nur den gelben Warn- oder roten Fehler-Indikator. Im Mouseover wird der Kommentar weiterhin angezeigt. Zudem werden alle Fehler, Warnungen und Kommentare der Strukturabfrage zusätzlich in der Reigenfolge des Auftretens (von oben nach unten, von links nach rechts) unterhalb der Parameter-Editoren gelistet.



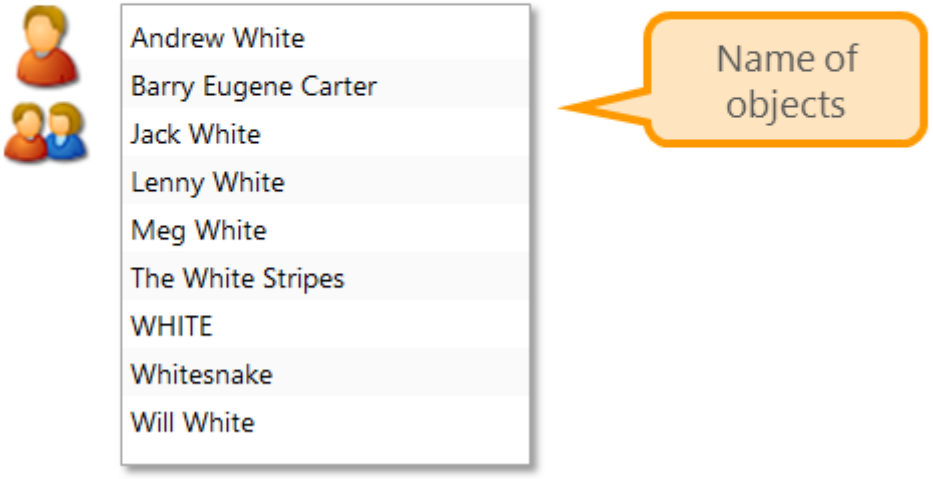
Warnungen und Fehler können in der Indikatoransicht unterdrückt werden, wenn man diese an dieser Stelle ignorieren möchte (Das ist natürlich nicht empfohlen). Dazu kann man das Indikator-Symbol in der Listenansicht anklicken oder die Funktion *Warnung unterdrücken* im Kontextmenü

der Bedingung benutzen. Genauso können sie wieder sichtbar gemacht werden, oder über die Kontext-Funktion *Alle Warnungen anzeigen* des Wurzel-Finders.



1.3.2. Einfache Suche / Volltextsuche

Die Verarbeitung der Suchanfragen von Nutzern kann ohne oder mit Interaktion (z.B. mit Type-Ahead-Vorschlägen) vorgenommen werden. Ausgangspunkt ist in jedem Fall die eingegebene Zeichenkette. In der Konfiguration der einfachen Suchen können wir nun festlegen, bei welchen Objekten wir in welchen Attributen nach der Nutzereingabe suchen und wie weit wir dabei von der eingegebenen Zeichenkette abweichen. Dazu ein Beispiel:



Wie müssen wir die Suche gestalten, um auf die Eingabe "white" die Objekte unten zurückzuliefern? In allen Fällen müssen wir bei der Abfrage konfiguriert haben, dass wir Personen und Bands als Ergebnisse haben wollen. Wie steht es aber mit den Abweichungen von der Nutzereingabe?

1. *Wann ist die (völlig unbekannte) chinesische Experimentalband namens "WHITE" ein Treffer?*

Wenn wir angeben, dass Klein-/Großschreibung egal ist

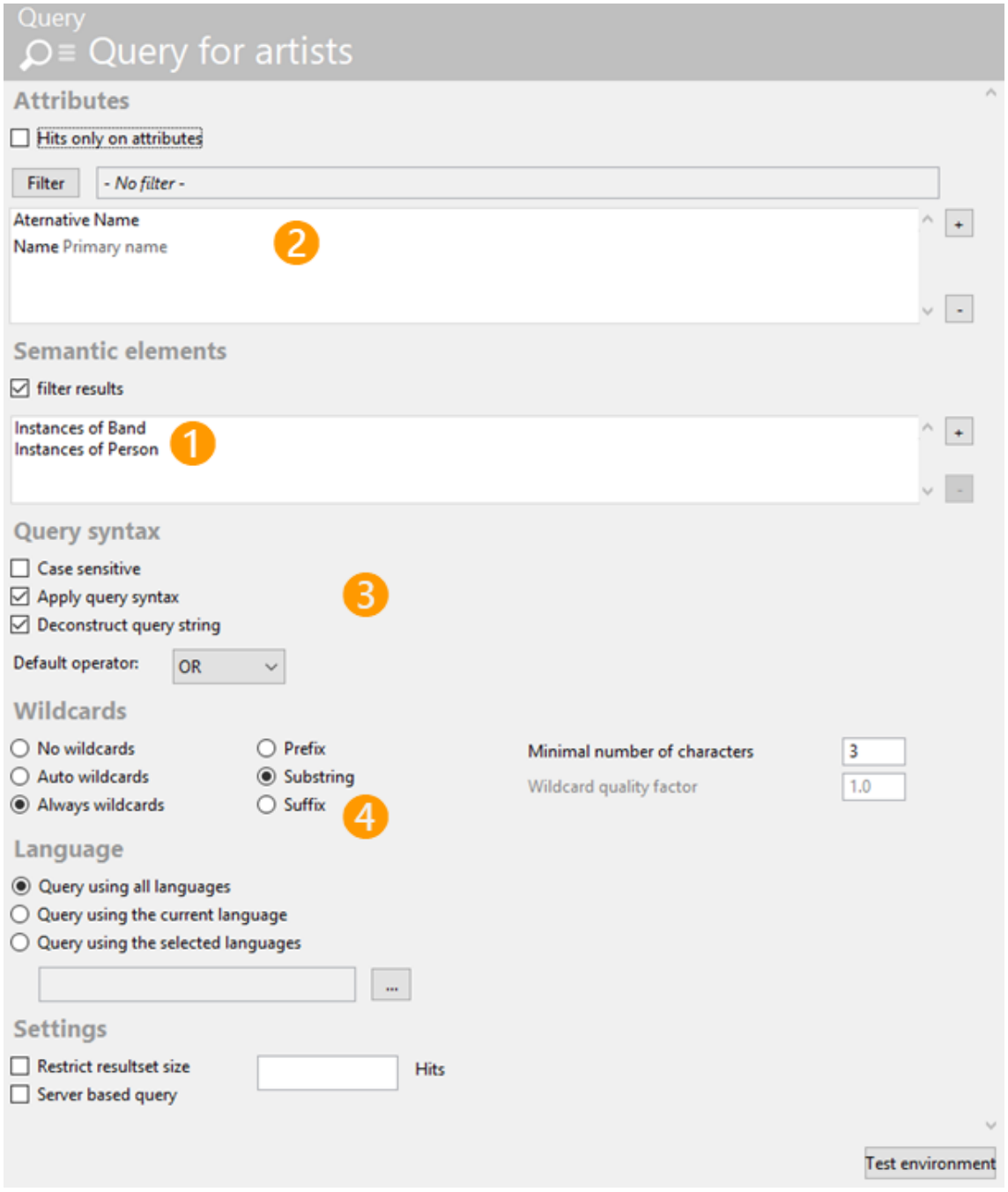
2. *Wann bekommen wir "Whitesnake" als Treffer geliefert?*

   Wenn wir die Eingabe als Teilstring verstehen und eine Wildcard anhängen

3. *Wann "Barry Eugene Carter"?*

   Wenn wir nicht nur den Objektnamen durchsuchen, sondern andere Attribute mit einbeziehen — sein Künstlername ist nämlich "Barry White"

Diese Optionen finden sich folgendermaßen in der Konfiguration der Suche wieder:

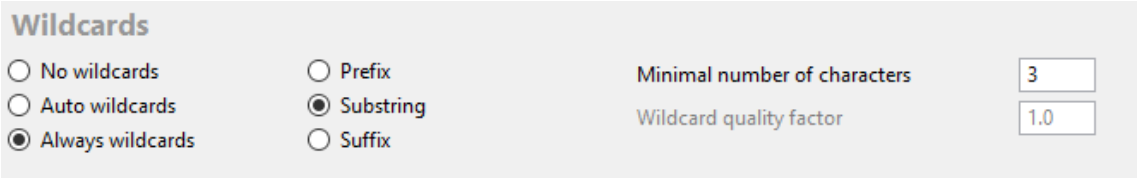


Konfiguration der einfachen Suchen mit (1) Angabe, welche Objekttypen durchsucht werden, (2) in welchen Attributen gesucht wird, (3) Groß- und Kleinschreibung und (4) Platzhaltern

1.3.2.1. Einfache Suche — Einstellungen im Detail

1.3.2.1.1. Platzhalter / Wildcard

Oft ist die Eingabe unvollständig oder wir wollen die Eingabe in längeren Attributfeldern wiederfinden. Dazu können wir in der einfachen Suche Platzhalter benutzten. Folgende Einstellungen für Platzhalter finden sich in den einfachen Suchen:



- *Präfix* (Platzhalter hinten) findet bei der Eingabe "White" die [White Lies]
- *Teilzeichenfolge* (Platzhalter hinten und vorne) findet [The White Stripes]
- *Suffix* (Platzhalter vorne) findet [Jack White]

WARNUNG	Platzhalter vorne ist langsam.

Die Option *Immer Platzhalter* funktioniert so als hätten wir tatsächlich einen Stern vorne oder/und hinten angehängt. Hinter *Automatische Platzhalter* steckt eine Eskalationsstrategie: Bei automatischen Platzhaltern wird erst mit der exakten Benutzereingabe gesucht. Falls diese keine Ergebnisse liefert, so wird mit Platzhalter gesucht, je nachdem welche Platzhalter eingestellt sind. Bei der Option *Präfix oder Teilzeichenfolge* gibt es noch einmal eine Reihenfolge: hier wird zuerst nach einem Präfix gesucht (durch Anhängen einer Wildcard) und, falls immer noch nichts gefunden wurde, nach einer Teilzeichenfolge (durch Voranstellen und Anhängen einer Wildcard).

Falls der Suche das Hinzufügen von Platzhaltern erlaubt ist, kann über das Feld *Minimale Anzahl Buchstaben* angegeben werden, wie viele Buchstaben die Sucheingabe mindestens aufweisen muss, damit die Platzhalter tatsächlich angefügt werden. Bei Eingabe von 0 ist diese Bedingung deaktiviert. Das ist vor allem wichtig, wenn wir eine Type-Ahead-Suche bauen.

Mit dem *Gewichtungsfaktor für Wildcards* lässt sich die Trefferqualität dahingehend anpassen, dass die Anwendung von Platzhaltern niedrigere Qualitäten ergibt. So können wir, wenn wir den Treffern ein Ranking mitgeben wollen, die Unsicherheit, die in den Platzhaltern steckt mit einem niedrigeren Ranking ausdrücken.

Falls die Option *Keine Platzhalter* gewählt ist, so wird die Sucheingabe nicht verändert. Die einzelnen Platzhalter-Einstellungen stehen dann nicht zur Verfügung.

Der Benutzer kann natürlich selbst Platzhalter in der Sucheingabe verwenden, die dann bei der Suche berücksichtigt werden.

1.3.2.1.2. Suchsyntax anwenden

Wenn kein Haken bei der Option *Suchsyntax anwenden* gesetzt ist, so wird eine einfachere Form der Analyse der Sucheingabe verwendet, in der die steuernden Zeichenketten | (Oder-Bedingung), & (Und-Bedingung) und ! (Negierung) keine Wirkung mehr haben. Um dennoch festlegen zu können, wie die Treffer für die einzelnen Tokens zusammengefasst werden sollen, lässt sich noch der *Default-Operator* auf *AND* oder *OR* schalten. Für alle Verknüpfungsoperatoren gilt, dass sie sich nicht auf Werte eines einzelnen Attributes beziehen, sondern auf die Ergebnisobjekte (je nach dem, ob *Treffer nur auf Attributen* gesetzt ist). Ein Treffer für "Online System" liefert somit semantische Objekte, die sowohl für "Online" als auch für "System" ein passendes Attribut besitzen (welches nicht notwendigerweise dasselbe ist).

1.3.2.1.3. Filterung

Einfache Suchen, Volltextsuchen und auch einige der Spezialsuchen können nach Objekttypen gefiltert werden. Im einleitenden Beispiel haben wir so dafür gesorgt, dass nur Personen und Bands als Suchergebnis zurückgeliefert werden. Attribute, die nicht zu einer eventuellen Filterung passen, werden im Suchkonfigurationsdialog fett und rot dargestellt. Das könnte in unserem Fall beispielsweise ein Attribut "Rezension" sein, dass nur für Alben definiert ist.

1.3.2.1.4. Übersetzte Attribute

Bei übersetzten Attributen können wir entweder eine Übersetzung wählen, oder die Sprache dynamisch bestimmen lassen. Mehrsprachige Attribute suchen dann in der aktiven Sprache oder in allen Sprachen, je nachdem, ob die Option *In allen Sprachen suchen* gesetzt ist.

1.3.2.1.5. Ergebnismenge

Eine maximale Ergebnismenge kann durch die Eingabe der maximalen Anzahl im Feld *Ergebnisse* bestimmt werden. Durch die Checkbox *Ergebnismenge begrenzen* kann dieser Mechanismus aktiviert oder deaktiviert werden.

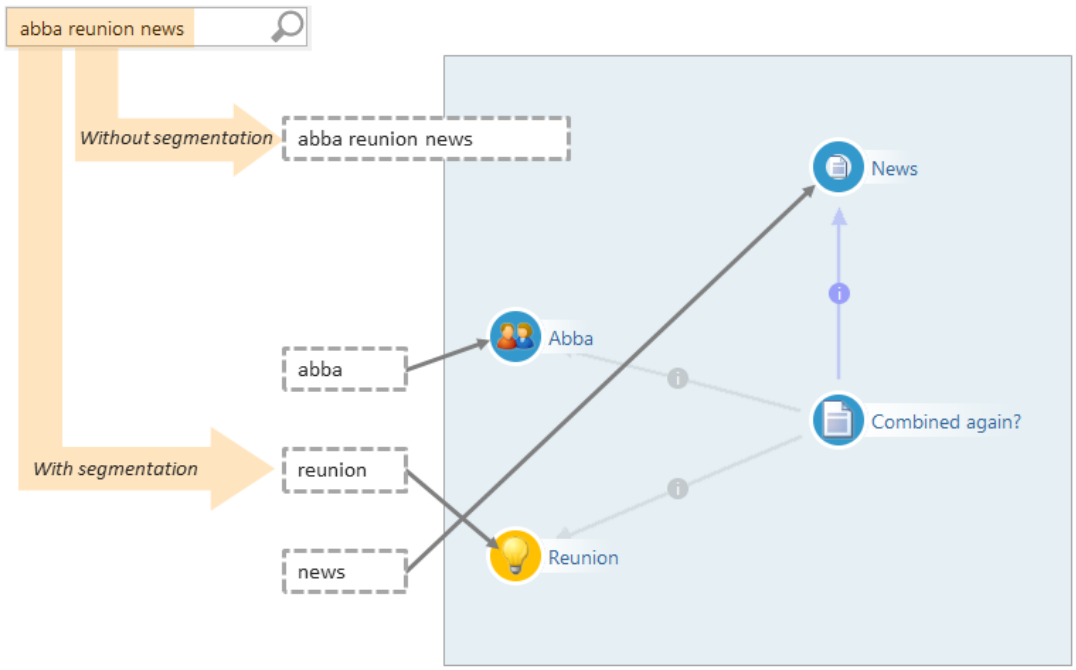
WARNUNG	Wird die Anzahl überschritten, werden keine Ergebnisse angezeigt!

1.3.2.1.6. Serverbasierte Suche

Es kann generell jede Suche auch als serverbasierte Suche ausgeführt werden. Voraussetzung ist, dass ein dazugehöriger Jobclient läuft. Diese Option kann eingesetzt werden, wenn abzusehen ist, dass sehr viele User Suchanfragen stellen werden. Durch die Auslagerung bestimmter Suchen auf externe Server wird der i-views-Server entlastet.

1.3.2.2. Mehrwort-Eingaben

Bei unseren Beispielen zu Abfragen haben die Nutzer bisher immer nur einen Suchbegriff eingegeben. Was aber, wenn der Nutzer z.B. "Abba Reunion News" eingibt und damit vielleicht einen News-Artikel finden möchte, der mit den Themen "Abba" und "Reunion" verschlagwortet ist? Diese Eingabe müssen wir zerlegen, auf den gesamten String wird keines unserer Objekte matchen oder zumindest nicht der gesuchte Artikel:



Unsere bisherigen Beispiele greifen aber nicht nur wegen Mehrwort-Sucheingaben zu kurz. Wir haben auch oft Suchsituationen, in denen es nicht sinnvoll ist, die Namen oder anderen Zeichenketten aus dem Netz, gegen die wir die Eingabe vergleichen, als Blöcke zu betrachten, z.B. weil wir eine Eingabe in einem längeren Text wiederfinden möchten. Hier sind die Wildcards irgendwann kein adäquates Mittel mehr: Wenn wir auch auf der Seite der Objekte und der durchsuchten Textattribute zerlegen möchten, dann wird besser die Volltextsuche verwendet.

1.3.2.3. Volltextsuche und Indexierung

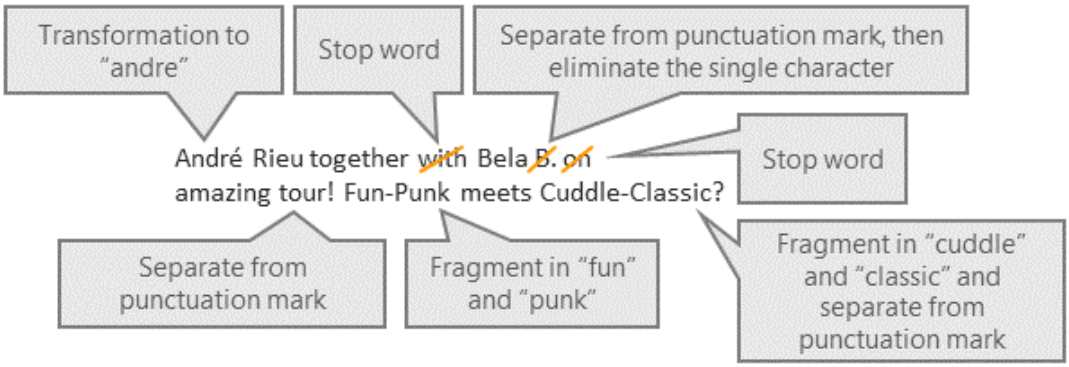
Wenn wir längere Texte, z.B. Beschreibungsattribute, wortweise betrachten bzw. durchsuchen wollen, empfiehlt sich ein Volltextindex. Wie sieht so etwas aus?

Term	Occurrence
aaliyah	Doc#155, Pos. 548644 / Doc#459, Pos. 934875 / Doc#935, Pos. 26526
abba	Doc#132, Pos. 43095 / Doc#459, Pos. 46795 / Doc#935, Pos. 534955 / Doc#353, Pos. 367773 / Doc#711, Pos. 92634
abbey	Doc#464, Pos. 95367 / Doc#2543, Pos. 65258 / Doc#634, Pos. 35241
abbreviation	Doc#436, Pos. 54362
abbreviator	Doc#463, Pos. 234652
abnormity	Doc#253, Pos. 4652
abo	Doc#234, Pos. 32243 / Doc#332, Pos. 23414

Der Volltextindex verzeichnet alle in einem Bestand an Texten vorkommenden Terme/Worte, so dass i-views schnell und einfach nachschlagen kann in welchen Texten (und an welcher Stelle im Text) ein bestimmtes Wort vorkommt.

"Texte" sind dabei in i-views i.d.R. nicht separate Dokumente, sondern die Zeichenketten-Attribute, die durchsucht werden sollen. Ihre Volltextindexierung ist Voraussetzung dafür, dass diese Attribute in der Suchkonfiguration angeboten werden.

Auch bei der Volltextindexierung geht es um die Abweichungen zwischen der genauen Zeichenfolge im Text und dem, was in den Index eingetragen und dementsprechend wiedergefunden wird. Beispiel: eine Nachricht aus der deutschen Musikszene:



In diesem Beispiel finden wir einen kleinen Teil der Filter- und Wortabgrenzungs-Operationen wieder, die typischerweise beim Aufbau eines Volltextindex zum Einsatz kommen:

1.3.2.3.1. Wortabgrenzung / Tokenizing

Im Deutschen werden Satzzeichen wie z.B. das Ausrufezeichen direkt ohne Leerzeichen an das letzte Wort des Satzes gesetzt. In den Volltextindex wollen wir aber den Eintrag {Tour}, nicht den Eintrag {Tour!} aufnehmen — nach letzterem wird wohl kaum jemand suchen. Dazu müssen wir beim Aufbau des Volltextindex angeben können, dass bestimmte Zeichen nicht zum Wort gehören. Nicht immer ist die Entscheidung so einfach: Bei einer Zeichenfolge wie "Kuschel-Klassik", die in einem Text vorkommt, müssen wir uns entscheiden ob wir diese als einen Eintrag in den Volltextindex aufnehmen wollen oder als {Kuschel} und {Klassik}. Im ersten Fall wird unsere Nachricht nur dann gefunden, wenn genau nach "Kuschel-Klassik" oder z.B. "*uschel-K*" gesucht wird, im zweiten Fall auch bei allen "Klassik"-Suchen.

Was wir trotz des Vorkommens von Satzzeichen wahrscheinlich zusammenhalten, d.h. vom Tokenizing ausnehmen wollen, sind Abkürzungen: wenn AC/DC nur a.d.D. nach Deutschland kommen (auf der Durchreise), dann ist es wahrscheinlich besser, die Abkürzung mit im Index zu haben statt der einzelnen Buchstaben.

1.3.2.3.2. Filtern

Durch Filteroperationen können wir sowohl Worte bei ihrer Aufnahme in den Volltextindex abwandeln als auch ihre Aufnahme komplett unterdrücken. Für *Stoppworte* können wir eine Liste pflegen. Auch einzelne Buchstaben (Bela B.) wollen wir wahrscheinlich nicht so im Index stehen haben — die Verwechselungsgefahr ist zu groß. Mit anderen Filtern können wir Worte auf ihre Grundformen zurückführen oder Ersetzungslisten für einzelne Zeichen definieren (um z.B. Akzente

zu eliminieren). Andere Filter wiederum bereinigen den Text um XML-Tags.

All das können wir im Knowledge-Builder in den Globalen Einstellungen unter *Indexkonfiguration* > *Indizes* einstellen. Diese Konfigurationen können wir dann den Zeichenkettenattributen zuweisen. Die Indexkonfiguration ist so organisiert, dass eine Filterung vor der Wortabgrenzung und eine Filterung nach der Wortabgrenzung stattfinden kann.

Bei der Volltextsuche greift die Wildcard-Automatik der anderen Abfragen nicht, aber der Nutzer kann natürlich seine Eingabe mit Wildcards versehen.

1.3.3. Such-Pipeline

Such-Pipelines ermöglichen es, einzelne Komponenten zu komplexen Abfragen zu kombinieren. Die einzelnen Bausteine führen Operationen aus dabei z.B.:

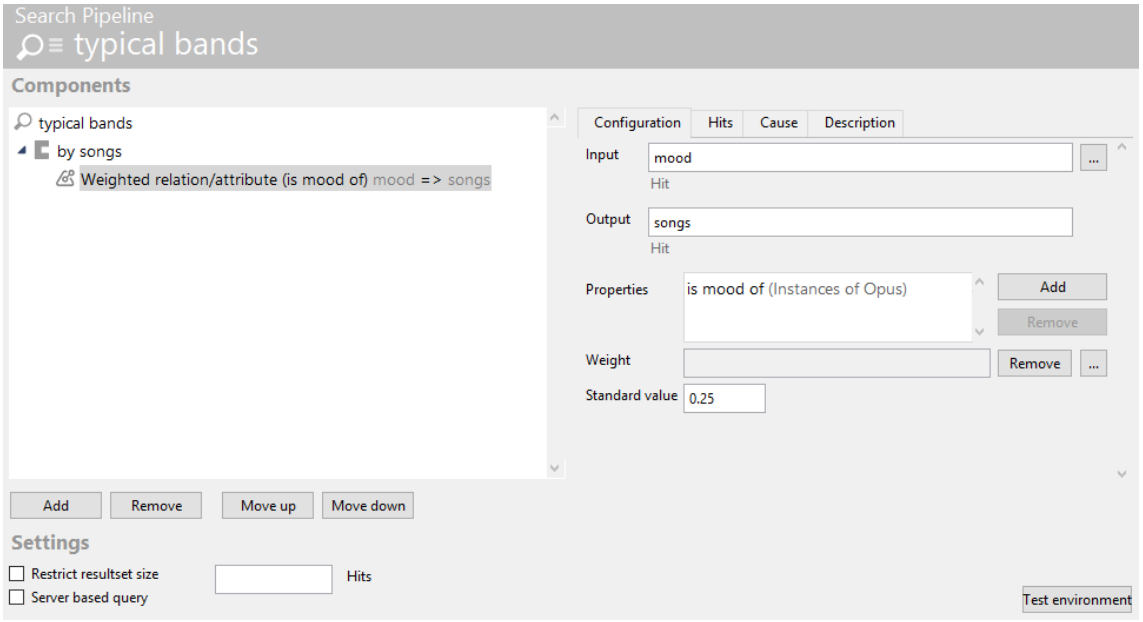
- Das Netz traversieren und dabei Gewichtung bestimmen
- Strukturabfragen und einfache Abfragen ausführen
- Treffermengen zusammenfassen

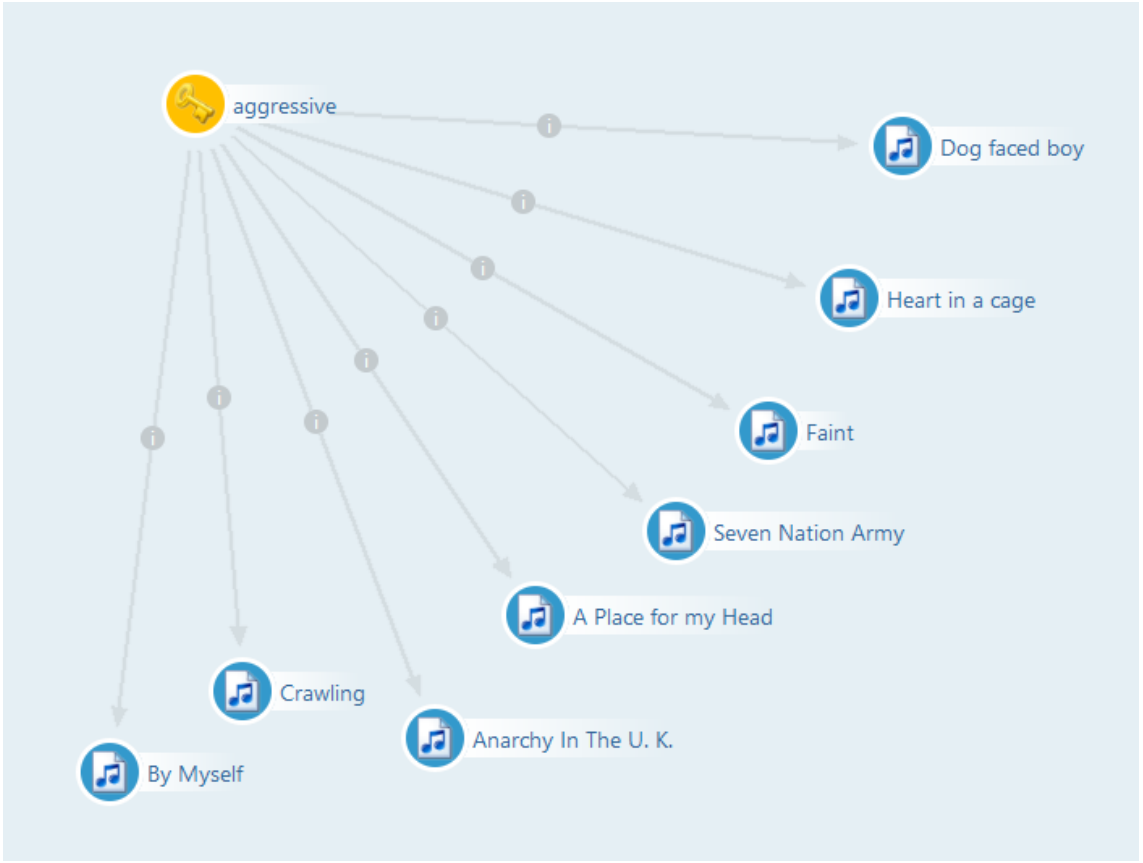
Jeder Abfrageschritt erzeugt eine Ergebnismenge (i.d.R. eine Menge von Objekten). Diese Ergebnismenge kann wiederum als Eingabe für nachfolgende Komponenten in der Pipeline verwendet werden.

1.3.3.1. Beispiel

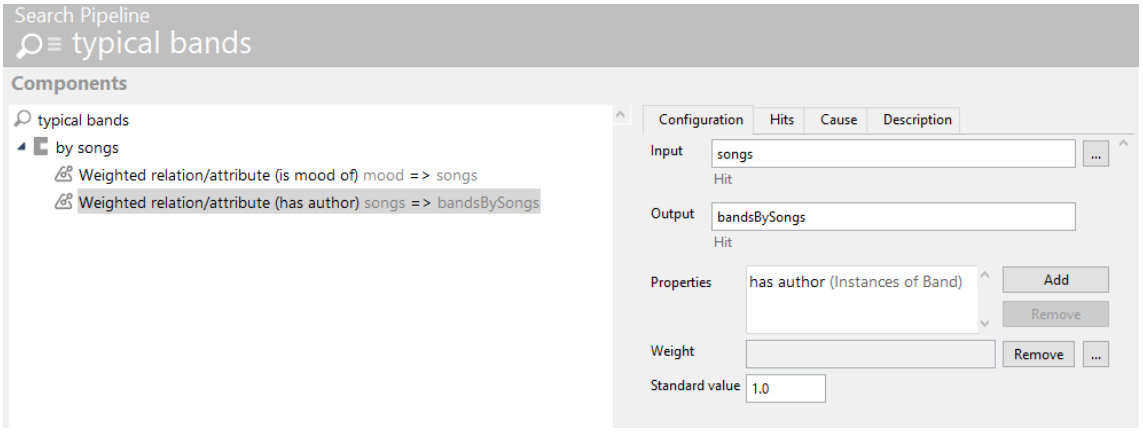
Nehmen wir an, Songs und Künstler unseres Musiknetzes sind mit Tags für ihre typische Stimmung charakterisiert. Ausgehend von einer bestimmten Stimmung wollen wir nun die Bands herausfinden, welche diese Stimmung am besten vertreten.

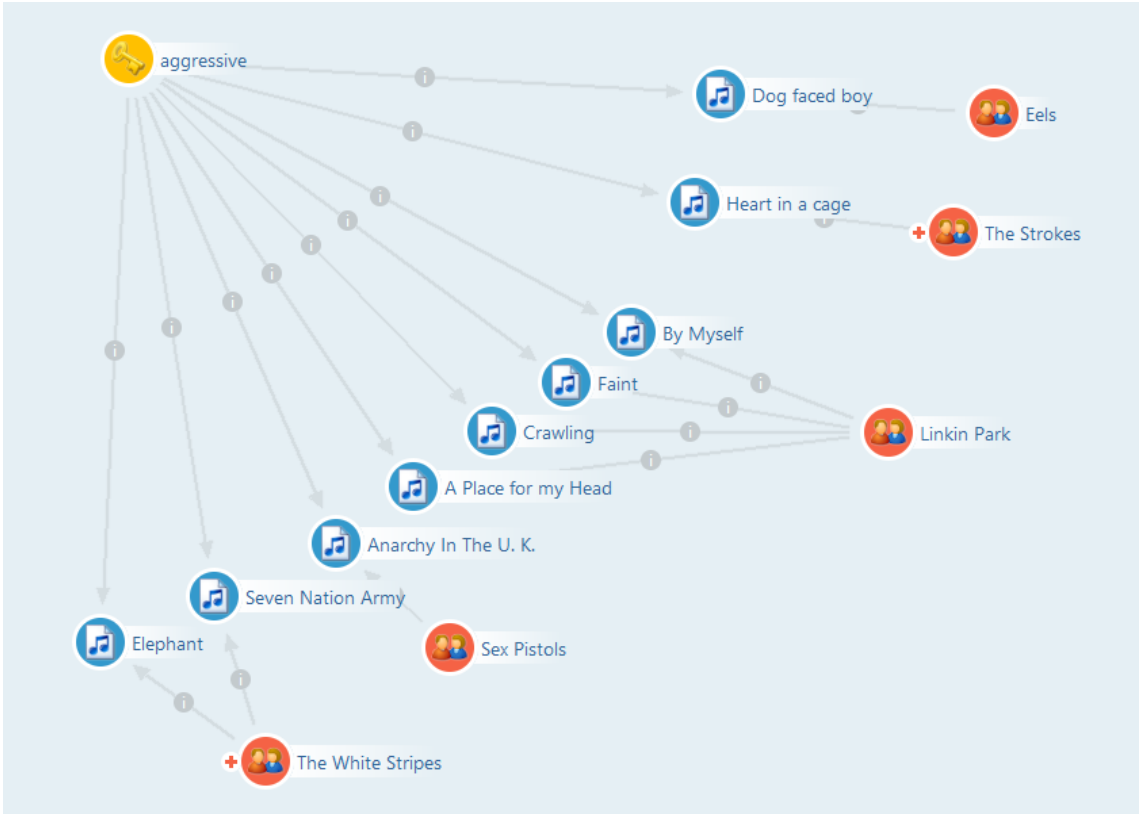
Schritt 0 unserer Such-Pipeline nimmt eine Ausgangsstimmung als Parameter entgegen und belegt damit die Variable namens "mood", in unserem Beispiel nehmen wir die Stimmung "Aggressiv" als Input und schauen uns jetzt an, wie wir mit Hilfe der Suchpipelines für diese Stimmung typische Bands ermitteln: Schritt 1 geht von der Ausgangsstimmung über die Relation *ist Stimmung von* zu den Songs, denen diese Stimmung zugeordnet ist:



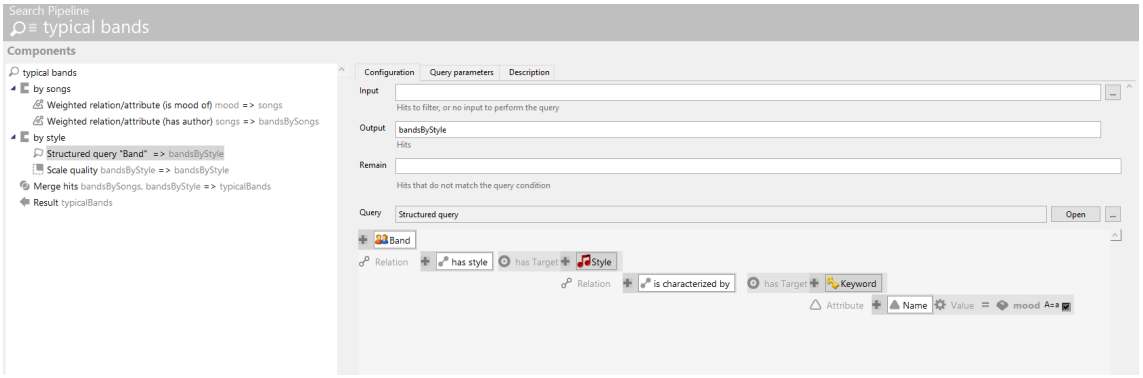


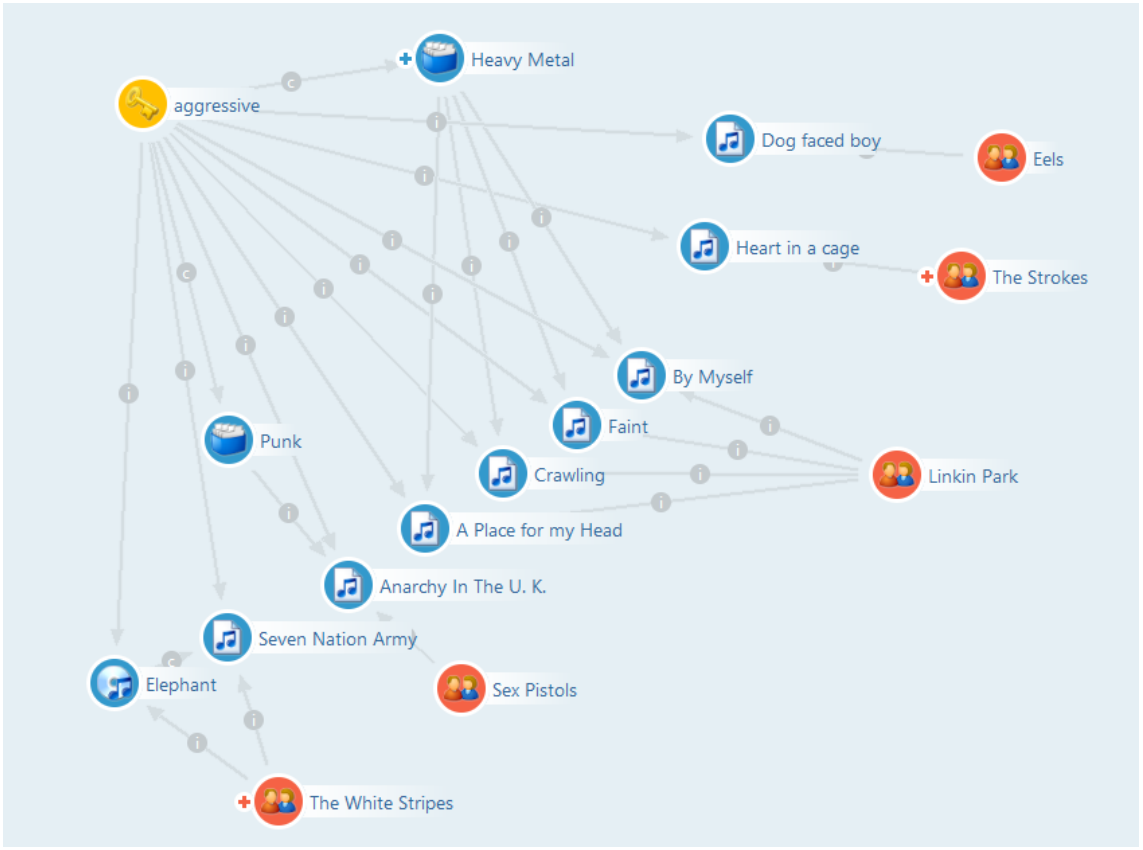
Im zweiten Schritt gehen wir von der Menge an ermittelten Songs zu den entsprechenden Bands über die Relation *hat Autor*:



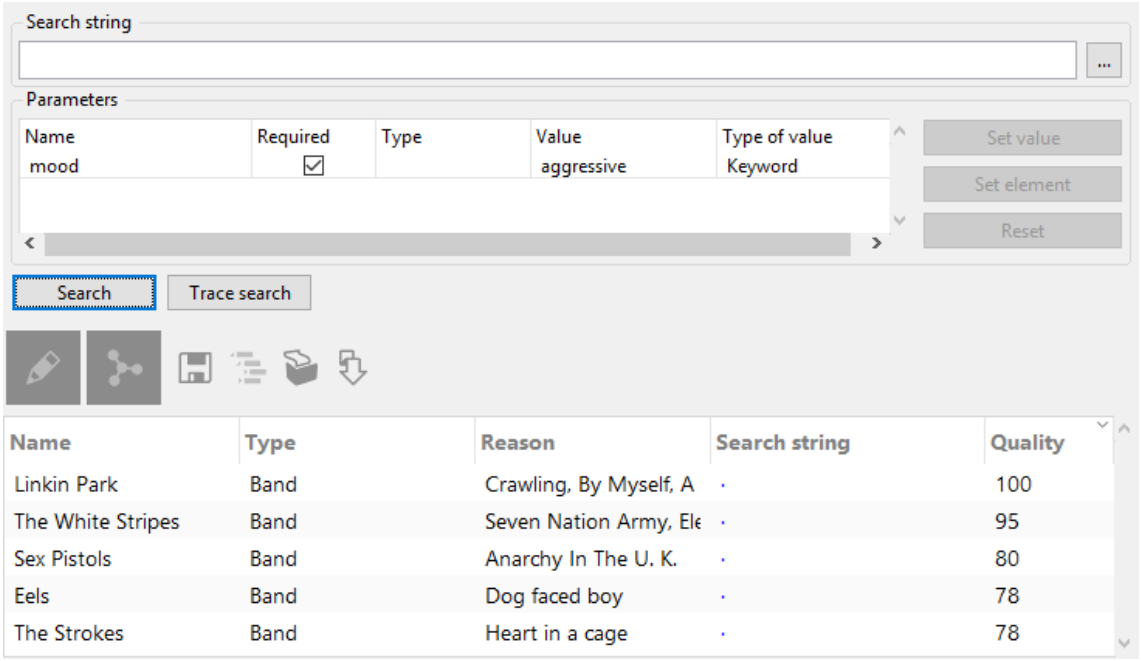


Jetzt möchten wir noch einen zweiten Weg verfolgen: Vom Ausgangspunkt "mood" "Aggressiv" zu Musikrichtungen, die durch Aggressivität charakterisiert werden. Von dieser Menge an relevanten Musikrichtungen ausgehend soll es wieder zu Bands gehen, die zu dieser Musikrichtung zählen. Wir könnten jetzt wieder wie oben Relationen aneinanderhängen, wir können dazu aber auch eine Strukturabfrage in die Suchpipeline einbauen:





Diesem zweiten Weg über die Musikrichtungen geben wir ein etwas niedrigeres Gewicht und fassen zum Schluss die Ergebnisse zusammen:



Die Abarbeitung der Schritte erfolgt sequenziell: Über die Eingabe und Ausgabe wird bestimmt, welcher Schritt mit welcher Treffermenge weiterarbeitet. So konnten wir z.B. bei unserem alternativen Weg wieder ganz vorne mit "mood" beginnen.

1.3.3.2. Das Prinzip der Gewichtungen

Das Ziel war es, den Bands, die als Ergebnisse herauskommen, ein Ranking zu geben, das zeigt, wie groß ihre semantische "Nähe" zum mood *Aggressiv* ist. Das Ranking beeinflussen wir in dieser Suche vor allem an zwei Stellen: ganz zum Schluss gewichten wir in der Zusammenfassung Bands höher, die sowohl über ihre Musikrichtung als auch über Songs gefunden werden. Das trifft hier auf Linkin Park und auf Sex Pistols zu. Das höhere Ranking von Linkin Park resultiert daraus, dass immer wieder von verschiedenen Songs mit dem mood Agressiv zu Linkin Park führt. Da im Datenbestand mehr aggressive Songs von Linkin Park vorhanden sind, soll Linkin Park mit höherem Ranking "belohnt" werden.

1.3.3.3. Zusammenstellung von Such-Pipelines

Die einzelnen Komponenten einer Such-Pipeline, werden im Hauptfenster im Feld *Komponente* in der Reihenfolge ihrer Ausführung dargestellt.

Mit der Schaltfläche *Hinzufügen* können wir eine neue Komponente am Ende der vorhandenen Komponenten anfügen.

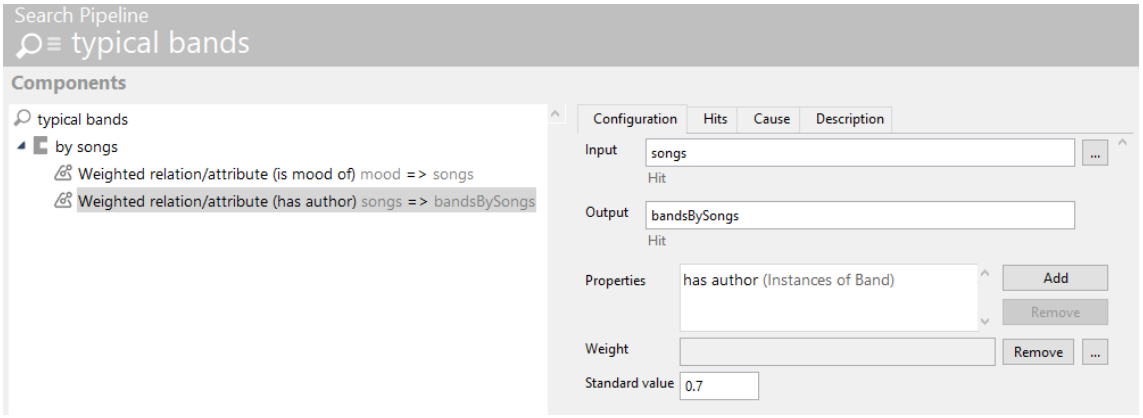
Die Gruppierung mit Blöcken dient allein der Übersichtlichkeit, etwa zur Zusammenfassung mehrerer Komponenten zu einem Funktionsbereich der Such-Pipeline.

Mit den Schaltflächen *Nach oben* und *Nach unten* oder mit Drag&Drop kann die Reihenfolge der Schritte geändert werden.

Mit *Entfernen* wird die ausgewählte Komponente entfernt, inklusive aller evtl. enthaltenen Unterkomponenten. Auf der rechten Seite im Hauptfenster wird die Konfiguration für die ausgewählte Komponente angezeigt.

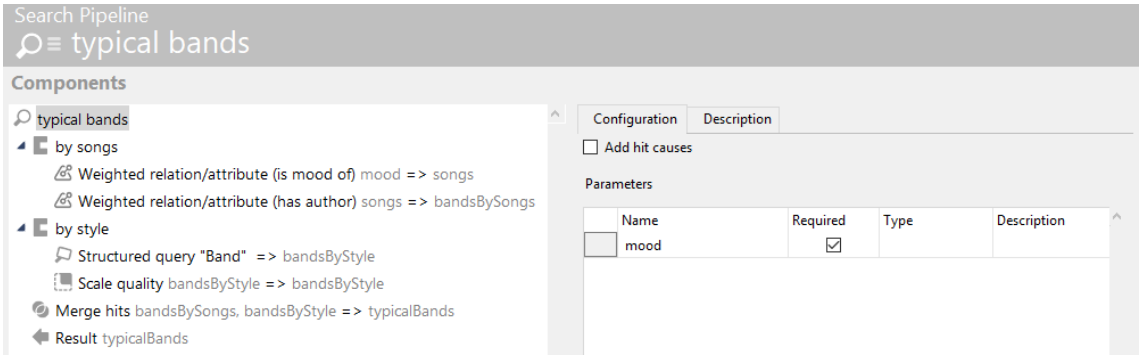
1.3.3.3.1. Konfiguration einer Komponente

Auf der rechten Seite des Hauptfensters lässt sich unter dem Reiter *Konfiguration* eine gewählte Komponente konfigurieren: Die meisten Komponenten benötigen eine Eingabe. Diese kommt i.d.R. von einem vorangegangenen Schritt. So gibt die erste Komponente in unserem Beispiel ihr Ergebnis unter der Variable "songs" an die nächsten Komponenten weiter, diese geht von da aus zu den Bands und gibt das Ergebnis wiederum als "bandsThroughSongs" an die nächsten Schritte weiter:



Über die Eingabe- und Ausgabe-Variable können wir auch in späteren Schritten wieder neu bei den ersten Ergebnissen aufsetzen, wie wir schon im letzten Abschnitt gesehen haben.

Als globale Einstellungen für die Suche legen wir die Eingabeparameter fest. Unter dem Namen, den wir hier vergeben, können wir in den einzelnen Schritten unserer Such-Pipeline dann auf diese Eingaben zugreifen. In unserem Beispiel ist der Mood, zu dem wir typische Bands ermitteln, der Eingabeparameter.



Einige Komponenten erlauben eine Abweichungen von der Standard-Verarbeitungsreihenfolge:

Einzelverarbeitung

Elemente einer Menge wie z.B. Treffer einer Suche kann man auch einzeln verarbeiten. Dies ist z.B.

praktisch, wenn man zu Suchtreffern eine individuelle Umgebung von benachbarten Objekten aufsammeln will. Bei der Einzelverarbeitung wird jedes Element in der bei Einzeltreffer konfigurierten Variable gespeichert und die Unterkomponenten ausgeführt.

Bedingung Parameter gesetzt

Diese Komponente führt weitere Unterkomponenten nur dann aus, wenn vorgegebene Parameter gesetzt sind. Der Wert ist dabei unerheblich. Mit Hinzufügen kann man eine neue Unterkomponente hinzufügen.

KPath-Bedingung

Mit einer KPath-Bedingung können wir bestimmen, dass die Unterkomponenten nur dann ausgeführt werden, wenn eine in KPath formulierte Bedingung erfüllt ist. Falls die Bedingung nicht erfüllt ist, wird die Eingabe übernommen. KPath ist im Kapitel *KPath* des technischen Handbuchs beschrieben.

Ergebnis

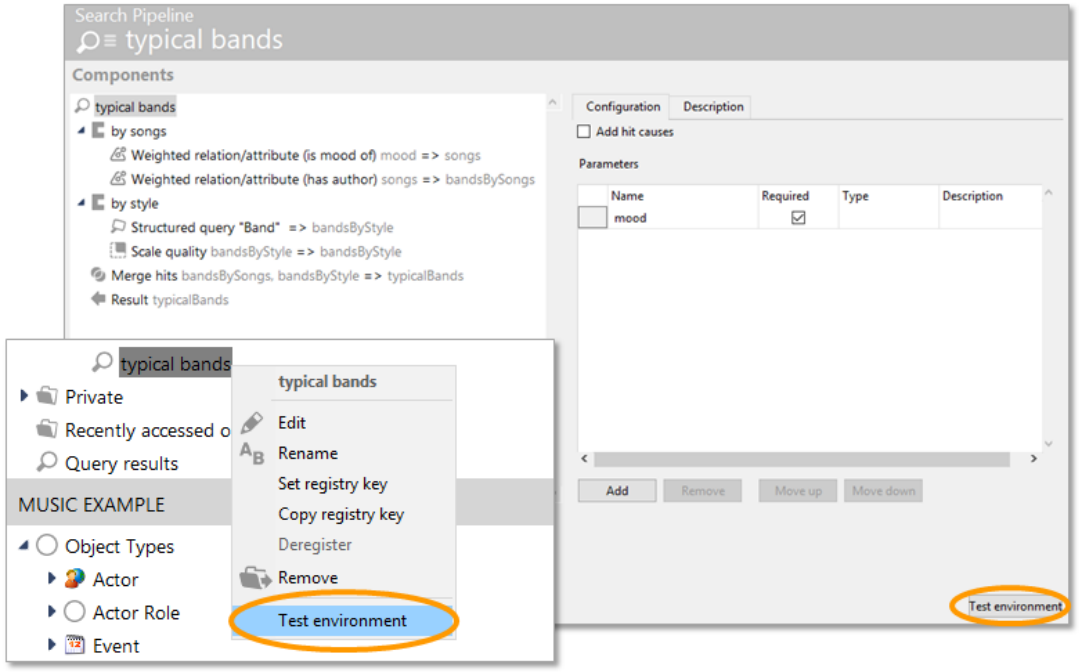
An beliebigen Stellen der Suche können wir die Suche abbrechen und ein Ergebnis zurückgeben. Diese Komponente ist unter anderem für das Testen der Such-Pipeline nützlich.

Block-Komponente

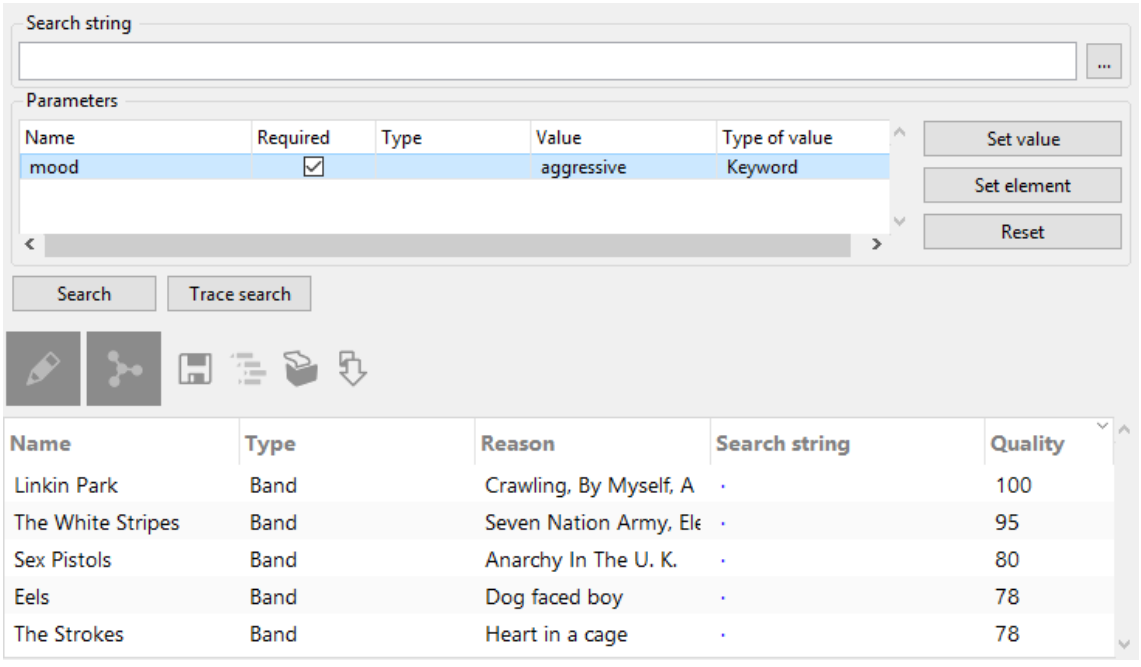
Die Block-Komponente, die wir auch in unserem Beispiel verwendet haben, gruppiert eine Menge von Einzelschritten. Um bei umfangreicheren Konfigurationen den Überblick zu wahren, können wir zudem auf dem Reiter *Beschreibung* den Namen der Komponente ändern sowie einen Kommentar hinzufügen. Weder die Block-Komponente noch die Beschreibung haben funktionale Auswirkungen. Beides dient lediglich der "Lesbarkeit" der Such-Pipeline.

1.3.3.3.2. Testumgebung

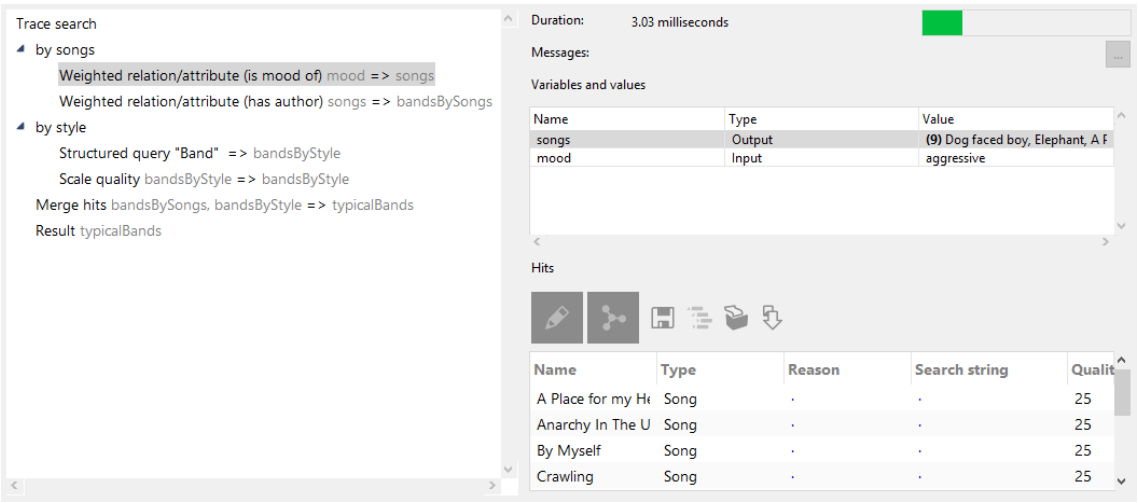
Auf die Testumgebung kann an folgenden Stellen zugegriffen werden:



Mit der Testumgebung können wir die Arbeitsweise der Suche analysieren. Der obere Teil enthält die Sucheingabe, der untere Teil das Ergebnis. Die Sucheingabe kann ein Suchtext oder ein Element der semantischen Graph-Datenbank sein, je nachdem welche geforderten und optionalen Eingabeparameter wir global in der Such-Pipeline definiert haben. Wenn wir ein Element der semantischen Graph-Datenbank als Startpunkt eingeben wollen, selektieren wir die entsprechende Parameter-Zeile und fügen, je nach Typ einen Attributwert oder ein Element der semantischen Graph-Datenbank hinzu.



Auf dem Reiter *Suchablauf* wird ein Protokoll der Suche angezeigt. Dieses besteht im wesentlich aus der Belegung der Ausgabevariablen sowie der Dauer zur Ausführung der einzelnen Komponenten.



1.3.3.3.3. Verrechnungsmöglichkeiten

Bei einigen Komponenten ist es möglich, mehrere Qualitätswerte zu einem einzigen Qualitätswert zusammenzufassen — z.B. bei *Treffer zusammenfassen*, aber auch beim Traversieren von Relationen (siehe Beispiel oben). Dabei stehen folgende Berechnungsmethoden zur Verfügung:

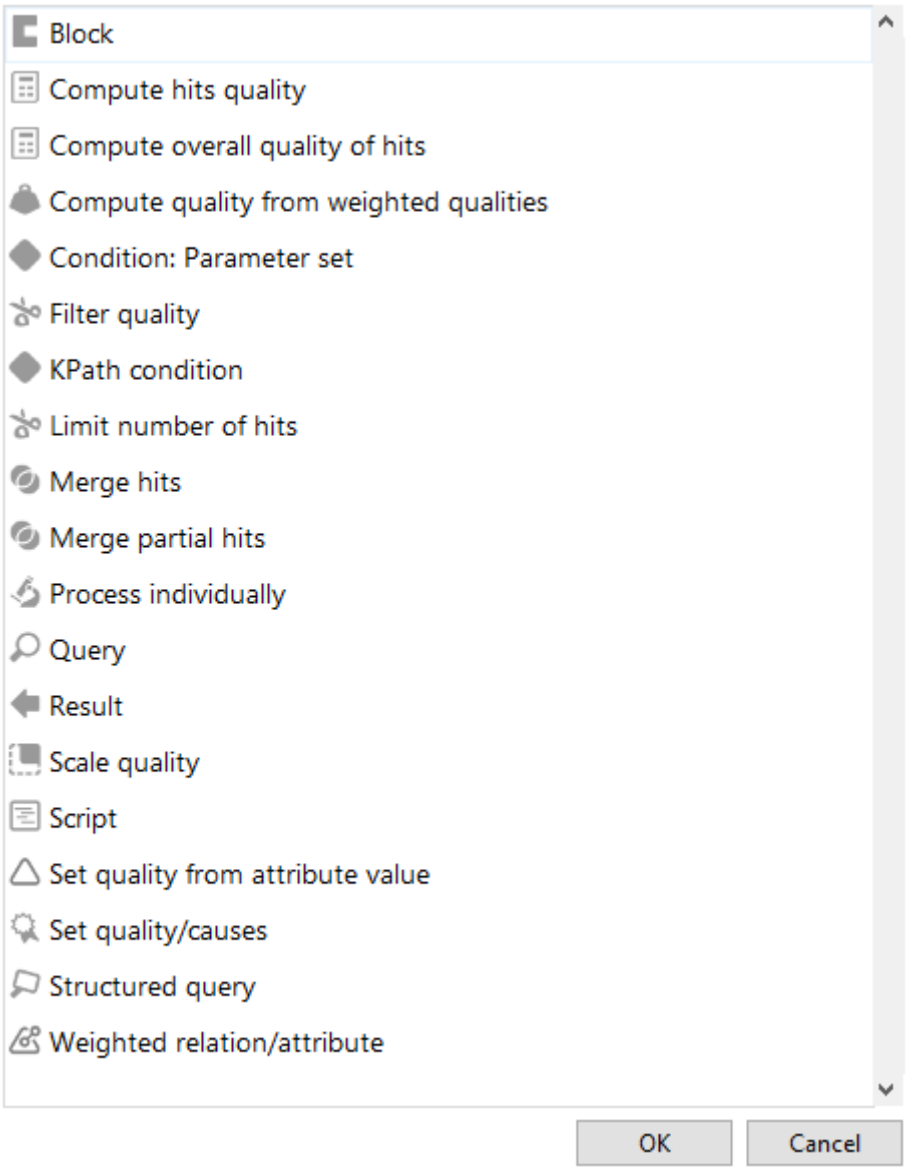
- Addieren / Multiplizieren
- Arithmetischer Mittelwert / Median
- Minimum / Maximum
- Ranking

Die Option *Ranking* passt immer dann, wenn wir aus Einzelhinweisen ein Gesamtbild zusammenfügen möchten, z.B. wenn wir viele, zumindest teilweise unabhängige Wege — am Ende noch mit unterschiedlicher Länge — zu einer "Gesamtnähe" verrechnen wollen. Mit der Ranking-Verrechnung sorgen wir dafür, dass alle positiven Hinweise (alle unabhängigen Wege) die Ähnlichkeit immer weiter erhöhen, ohne dass die 100% überschritten werden.

In der Such-Pipeline werden Qualitätswerte immer als Fließkommazahlen angegeben. Der Wert 1 entspricht dabei einer Qualität von 100%.

1.3.3.4. Die einzelnen Komponenten

Die hinzufügbaren Elemente einer Such-Pipeline enthalten entweder eine strukturierende Funktion, eine abfragende Funktion, eine logische Funktion oder Funktionen zur Berechnung von Qualitäten:



1.3.3.4.1. Block

Der Block dient dazu, aufeinanderfolgende Elemente optisch zusammenzufassen. Auf diese Weise kann vor allem bei großen Such-Pipelines eine übersichtliche Struktur geschaffen werden. Das Zuordnen von Elementen zu Blöcken erfolgt durch einfaches Drag&Drop. Blöcke haben keinen Einfluss auf das Ergebnis einer Such-Pipeline.

1.3.3.4.2. Gewichtete Relationen und Attribute

Ausgehend von semantischen Objekten können wir mit diesem Schritt den Graph traversieren und Relationsziele oder Attribute aufsammeln. Dazu müssen wir den Typ der Relation oder des Attributs angeben.

HINWEIS	Ausgabe sind nur noch die aufgesammelten Ziele, nicht mehr die

> Ausgangsmenge. Wenn diese angezeigt werden sollen, müssen wir anschließend unter dem Reiter *Hits* die Option *Ausgangstreffer zur Ausgabe hinzufügen* wählen.

Bei der Traversierung einer Relation kann die Gewichtung der Treffer beeinflusst werden. Nehmen wir an, wir wollen den "Ausgangs-mood" unserer Beispielsuche um "Unter-moods" semantisch erweitern. Aber diese Indirektion soll sich in einem Ranking niederschlagen: Verbindungen zu Bands, die über die Untermoods laufen, sollen nicht so stark zählen wie Verbindungen über einen Ausgangsmood. Zu diesem Zweck können wir das Relation-Entlanggehen mit einem pauschalen Wert — z.B. 0,5 — belegen, und mit dem Eingangsgewicht verrechnen, beispielsweise multiplizieren. Dann zählen die in diesem Schritt hinzugefügten Untermoods nur halb so viel wie die direkten.

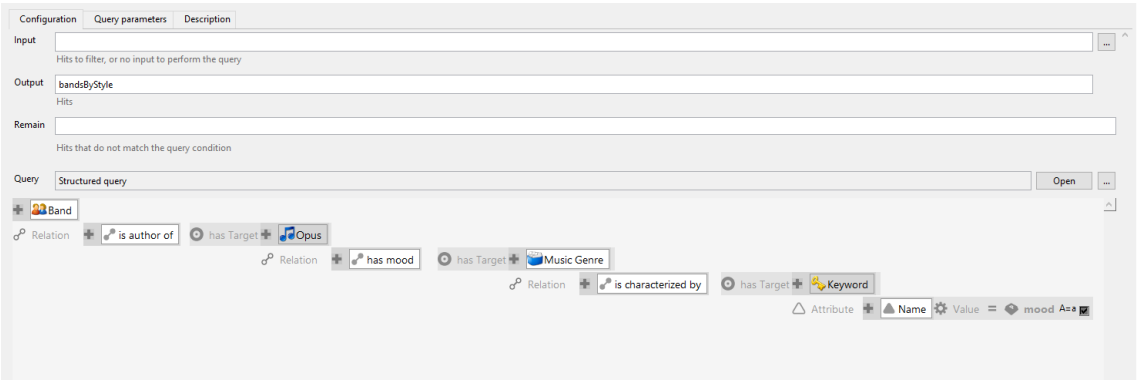
Statt eines pauschalen Gewichts für das Entlanggehen der Relation zu vergeben, können wir den Wert auch aus einer Metaeigenschaft des Basistyps Fließkommazahl der ausgewählten Relation auslesen. Falls dieses Attribut nicht vorhanden ist oder keines konfiguriert wurde, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen. Die Treffererzeugung kann detaillierter konfiguriert werden: Bei Relationen kann optional auch für die Relationsquelle (statt für das Relationsziel) ein neuer Treffer erzeugt werden.

Wenn eine Relation als Eigenschaft ausgewählt wurde und Treffer für Relationsziele erzeugt werden, können wir optional auch die Relation transitiv verfolgen. Bei jedem Schritt verringert sich der Qualitätswert, bis der angegebene Schwellwert unterschritten wird. Falls ein Objekt mehr Relationen hat als bei maximaler Fanout angegeben, werden diese Relationen nicht verfolgt. Je höher der Dämpfungsfaktor ist, desto stärker wird der Qualitätswert verringert.

1.3.3.4.3. Strukturabfrage

Mit Strukturabfrage-Komponenten können wir entweder semantische Objekte suchen / von einer bestehende Menge zu anderen Objekten gehen (wie mit der gewichteten Relation) oder eine Menge filtern.

Wenn wir Objekte suchen, leiten wir unsere Ausgangsmenge von Treffern aus einem der vorhergehenden Schritte über den Parameternamen in die Suche ein. (Allgemein: Innerhalb der Expertensuche können Variablen der Such-Pipeline wie z.B. search-String über Parameter referenziert werden). Die Eingabe bleibt in diesem Fall leer.



Zur Filterung geben wir dagegen als Eingabe eine Menge von Objekten an. In der Ausgabe sind alle Objekte enthalten, auf die die Suchbedingung zutrifft. Objekte, die nicht zur Suchbedingung passen, können optional in einer weiteren Variable (Rest) gespeichert werden.

Wir können die Strukturabfrage entweder direkt in der Komponente ad hoc definieren oder eine bestehende Strukturabfrage verwenden.

HINWEIS	Wenn eine bestehende Suche ausgewählt wurde, wird keine Kopie angelegt, Änderungen, die wir an für Zwecke der Such-Pipeline an der Strukturabfrage vornehmen, ändern sie auch für alle anderen Verwendungen.

1.3.3.4.4. Abfrage

Mit der Komponente *Suche* können einfache Suchen, Volltextsuchen und andere Such-Pipelines ausgeführt werden. Einfache Suchen und Volltextsuchen werden dabei mit einer Zeichenkette versehen, z.B. mit dem *searchString*: Das ist ein Parameter, der in allen Such-Pipelines zur Verfügung steht um die Nutzereingabe zu verarbeiten. Die Treffermenge der aufgerufenen Suche belegt die Ausgabe dieser Komponente.

Über das Einbinden von Such-Pipelines in anderen Such-Pipelines können wir häufiger auftretende Teilschritte ausfaktorisieren. Anderen Such-Pipelines können mehrere Parameter übergeben werden und ganze Treffermengen übergeben. Mit eingebundenen Such-Pipelines können wir auch mehrere Parameter austauschen, d.h. wir können in der eingebundenen Suche auf jede Teilschrittausgabe der umgebenden Suche zugreifen und umgekehrt. Wenn wir auf *Ausgewählte Parameter* gehen, können wir diese auch umbenennen, falls wir z.B. eine Treffermenge aus der eingebundenen Suche nutzen wollen, aber den Namen schon verwendet haben. Oder wir können, um solche Konflikte zu vermeiden, nur einen Teil der Parameter aus der eingebundenen Suche übernehmen.

1.3.3.4.5. Treffer zusammenfassen

Mit dieser Komponente können wir mehrere Treffermengen aus unterschiedlichen vorangegangenen Schritten zusammenfassen. Folgende Methoden zur Zusammenfassung stehen zur Verfügung:

Vereinigungsmenge

Alle Treffer, die in mindestens einer der Mengen vorkommen, werden als Ergebnis ausgegeben

Schnittmenge

Nur Treffer, die in allen Mengen vorkommen, werden als Ergebnis ausgegeben.

Bei Vereinigungsmengen und Schnittmengen kommt es vor, dass ein semantisches Objekt in mehreren Treffermengen vorkommt und zu einem Gesamttreffer mit neuer Trefferqualität verrechnet werden muss. Hier stehen wieder die erwähnten Verrechnungsmethoden zur Verfügung.

Differenz

Eine der Treffermengen muss als Ausgangsmenge definiert werden. Von dieser werden die anderen Mengen abgezogen.

Symmetrische Differenz

Die Ergebnismenge besteht aus den Objekten, die nur in genau einer Teilmenge enthalten sind (= alles außer dem Schnitt, bei zwei Mengen).

Es können drei unterschiedliche Arten von Gesamttreffern erzeugt werden. Die Auswahl ist insbesondere dann relevant, wenn die Teiltreffer zusätzliche Informationen tragen.

- Einheitliche Treffer erzeugen, ursprüngliche Treffer als Ursache merken: Es werden neue Treffer erzeugt, die den ursprünglichen Treffer als Ursache enthalten.
- Ursprüngliche Treffer erweitern: Der ursprüngliche Treffer wird kopiert und erhält einen neuen Qualitätswert. Falls mehrere Treffer für dasselbe semantische Objekt vorliegen, wird ein beliebiger Treffer gewählt.
- Einheitliche Treffer erzeugen: Es wird ein neuer Treffer erzeugt. Die Eigenschaften des ursprünglichen Treffers gehen verloren.

1.3.3.4.6. Bedingung: Parameter gesetzt

Dieses Element dient dazu, die Auswertung der untergeordneten Suchbestandteile nur dann zu gestatten, wenn bestimmte Parameter gesetzt sind. Die dazu benötigten Parameter werden als Bedingung im Konfigurations-Reiter bestimmt. Wenn die Parameterwerte zum Zeitpunkt der Abfrage nicht gesetzt sind, dann werden alle untergeordneten Teile übersrpungen und die Suche für die nachfolgenden Teile fortgesetzt.

1.3.3.4.7. Einzelverarbeitung

Die Einzelverarbeitung wird zur Aufteilung einer Treffermenge (= Array aus Hits) verwendet, um die einzelnen Treffer (= n-tes Element des Hit Arrays) als Eingabewert für einen nachfolgenden Abfragebaustein verwenden zu können, der wiederum nur ein Hit-Element auf einmal verarbeiten kann.

Die Treffermenge einer vorhergehenden Abfrage soll mithilfe einer einfachen Abfrage weiterverarbeitet werden. Hierzu verarbeiten wir die einzelnen Treffer wie folgt: Die Treffermenge, welche in Form eines Arrays übergeben wird, wird mithilfe der Einzelverarbeitung in einzelne Treffer zerlegt (siehe hierzu Kapitel *Inhaltsmodell "Hit"*). Ein einzelner Treffer besteht selbst wiederum aus einem Array mit dem semantischen Element, der Treffer-Qualität, der Treffer-Ursache und weiteren evtl. benutzerdefinierten Abfrageeigenschaften. Mittels Skript wird der Name des semantischen Elements des Hits ausgelesen und in Form einer Zeichenkette ausgegeben. Der Ausgabewert kann anschließend von der nachfolgenden Abfrage entgegengenommen werden, welche eine Zeichenkette als Eingabewert erwartet. Die Treffer je Eingabewert der Abfrage werden mithilfe des Abfragebausteins *Teiltreffer zusammenfassen* wieder zu einer Treffermenge

zusammengeführt.

HINWEIS	Um die Treffer aus einer Einzelverarbeitung wieder zusammenzufassen und die Gesamtqualität korrekt zu berechnen, wird der Abfragebaustein *Teiltreffer zusammenfassen* benötigt.

1.3.3.4.8. Teiltreffer zusammenfassen

Bei der Einzelverarbeitung ist es öfters notwendig, eine Gesamtmenge aus Teiltreffern zu erzeugen. Dieses ermöglicht die Komponente *Teiltreffer zusammenfassen*. Diese fasst alle Treffer einer oder mehrerer Teiltreffermengen zusammen. Der Unterschied zu *Treffer zusammenfassen* liegt darin, dass die Zusammenfassung erst am Ende erfolgt, nicht für jede einzelne Teiltreffermenge. Dies ist insbesondere bei der Berechnung der Qualität relevant, da Treffer zusammenfassen z.B. bei *Median* falsche Werte liefern würde.

1.3.3.4.9. Skript

Eine Such-Pipeline kann ein JavaScript enthalten. Dieses kann auf die Variablen der Such-Pipeline zugreifen. Außerdem kann ein Skript mehrere Parameter an die Such-Pipeline übergeben. Das Ergebnis des Skripts wird als Ergebnis der Komponente verwendet.

Die JavaScript-API ist in einem separaten Handbuch beschrieben.

1.3.3.4.10. Qualität aus Attributwert übernehmen

Für Treffer können wir den Qualitätswert aus einem Attribut des semantischen Objekts übernehmen. Falls das Objekt nicht genau ein solches Attribut besitzt, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen.

1.3.3.4.11. Gesamtqualität aus gewichteten Qualitäten berechnen

Um die Qualität eines Suchtreffers anzupassen, kann es hilfreich sein, aus einzelnen Teilqualitäten einen Gesamtwert zu berechnen. Die Qualitäten müssen dabei als Zahlenwerte vorliegen. Aus diesen Werten wird eine neue Gesamtqualität berechnet.

1.3.3.4.12. Gesamtqualität einer Treffermenge berechnen

Aus den einzelnen Qualitätswerten einer Treffermenge kann man eine Gesamtqualität berechnen.

1.3.3.4.13. Qualität beschränken

Treffermengen können wir auf Treffer beschränken, deren Qualitätswert innerhalb vorgegebener Schranken (Minimum und Maximum) liegen. Im Normalfall möchten wir Treffer, die unterhalb einer bestimmten Qualitätsschwelle liegen, ausfiltern.

1.3.3.4.14. Trefferanzahl beschränken

Falls die Gesamtzahl einer Treffermenge begrenzt werden soll, können wir die Komponente *Trefferanzahl beschränken* hinzufügen. Mit der Option *Treffer gleicher Qualität nicht zerteilen* verhindern wir, dass bei mehreren Treffern mit gleicher Qualität eine willkürliche Auswahl erfolgt, um die Gesamtzahl einzuhalten. Wir erhalten dann mehr Treffer als vorgegeben.

Für einige sehr spezielle Fälle können wir die Treffer auch zufällig auswählen lassen, z.B. wenn wir eine große Menge an Treffern gleicher Qualität haben und eine Vorschau generieren wollen.

1.3.3.4.15. Qualität skalieren

Die Qualitätswerte einer Treffermenge kann skaliert werden. Es wird eine neue Treffermenge mit skalierten Qualitätswerten berechnet. Die Berechnung erfolgt ein zwei Schritten:

1. Die Qualitätswerte der Treffer werden begrenzt. Die Grenzwerte können entweder festgelegt oder berechnet werden. Bei der Berechnung werden der minimale und der maximale Wert der Treffer ermittelt. Falls die Grenzen vorgegeben werden und ein Treffer einen Qualitätswert außerhalb der Grenzwerte hat, wird der Wert auf den Grenzwert beschränkt. Falls man solche Treffer entfernen will, muss man die Komponente Qualität beschränken vorschalten. Beispiel: Abbilden von Prozentwerten auf Schulnoten. 30% ist Durchschnitt, über 90% ist Highscore. Die Werte können innerhalb von 30% bis 90% linear skaliert werden.
2. Anschließend werden die Qualitätswerte linear skaliert. Treffer mit dem minimalen/maximalen Eingangswert erhalten den minimalen/maximalen skalierten Wert.

1.3.3.4.16. Trefferqualität berechnen

Mit Hilfe eines KPath-Ausdrucks wird für einen Treffer ein neuer Treffer mit berechneter Qualität erzeugt. Der KPath-Ausdruck wird ausgehend von der Eingabe berechnet.

1.3.3.4.17. Ergebnis

Das *Ergebnis*-Element wird verwendet, um festzulegen, an welcher Position die Such-Pipeline endet und welcher Parameterwert als Ergebnis ausgegeben wird. Alles, was sich unterhalb des Ergebnis-Elements befindet, wird nicht ausgewertet.

Dies ist insbesondere nützlich, wenn vereinzelte Abfragebausteine vorübergehend nicht benötigt werden: Der nicht benötigte Abfragebaustein wird einfach unterhalb des Ergebnis-Elements "geparkt".

1.3.3.5. KPath

Mit KPath können Objekte im Wissensnetz adressiert werden. Die Notation ist an XPath angelehnt, unterscheidet sich allerdings in einigen Punkten.

Die einzelnen Elemente des Ausdrucks sind normalerweise mit einem Slash / voneinander getrennt. Beginnt der KPath-Ausdruck mit einem /, startet die Auswertung beim Wurzelbegriff, andernfalls beim aktuellen Objekt (hängt vom Kontext des Aufrufes ab). Wenn ein Element keinem der

aufgeführten Elemente in Tabelle entspricht, wird es als Name eines Unterbegriffs interpretiert. Einfache Namen können ohne Anführungszeichen angegeben werden.

Bei Angabe einer Sprache muss diese über ihr Kürzel gemäß ISO 639-2 ("ger" für Deutsch, "eng" für Englisch, …) angegeben werden.

`@Name`
: Attribut "Name"

`//Buch\Faust/~Autor`
: Relation "Autor" des Buches "Faust"

`//$Artefakt$/Book{eng}`
: Unterbegriff "Book" (englischer Name) des Begriffs "Artefakt" (interner Name)

`//Buch\*[~Autor/target()/@Name = "Goethe"]`
: Alle Bücher, die von Goethe geschrieben wurden

1.3.3.5.1. Namen

In Verbindung mit `@`, `/`, `//`, `\` und `\\` können folgende Arten von Namen verwendet werden:

Name	Beschreibung
`name`	Name in der Standardsprache. Ohne Anführungszeichen muss der Name mit einem Buchstaben, Stern oder Underscore beginnen und darf keine Whitespaces oder für andere Ausdrücke verwendeten Sonderzeichen enthalten. Der Name muss folgenden regulären Ausdruck erfüllen: `[a-zA-Z_*][^/(){}$%{}[],~@§#+-'"s|^&]*` Der besseren Lesbarkeit halber wurde das Escape-Zeichen `\` weggelassen
`"name"` oder `'name'`	Genügt der Name nicht den obigen Anforderungen, so muss er in einfache oder doppelte Anführungszeichen gesetzt werden. Hierbei dient das Backslash-Zeichen `\` als Escape-Zeichen für etwaig enthaltene Anführungszeichen, z.B. `Wendy's`.
`namede`	Name in der angegebenen Sprache "lang"
`$name$`, `$"name"$`	Interner Name
`§name§`, `§"name"§`	Systemname

Name	Beschreibung
`#ID42_1013`	ID des Objektes

Namen sind nicht durch Variablen ersetzbar, müssen also direkt im Skript stehen.

1.3.3.5.2. Operatoren

Zahlenwerte können mit den Operatoren `+`, `-`, `*`, oder `/` verknüpft werden.

Bei `*`, `-` und `/` muss mindestens ein Leerzeichen auf beiden Seiten des Operators stehen.

Klammerung wird unterstützt, z.B. ergibt `(5 + 3) * 4` den Wert 32.

Summe der Relationen von Goethe und Schiller:

```
\\Goethe/~*/size() + \\Schiller/~*/size()
```

Der Operator `+` kann auch zur Verknüpfung von Zeichenketten verwendet werden:

```
//Person\Goethe + " schrieb " + //Buch\Faust
```

ergibt

```
Goethe schrieb Faust
```

Mit dem unären Operator `!` kann ein boolscher Ausdruck negiert werden, z.B.

```
!1=2
```

Für manche Operatoren ist auch eine alternative Schreibweise möglich, die nur aus alphabetischen Zeichen besteht, z.B. `eq` für den Gleichheitsoperator. Bei diesen Schreibweisen muss mindestens ein Leerzeichen zwischen Operator und Operanden stehen. Groß-/Kleinschreibung wird unterschieden, nur klein geschrieben wird der Operator erkannt.

Mögliche Operatoren sind (in absteigender Präzedenz):

Operator	Alternative Schreibweise	Bedeutung
!	not	Negation (unärer Operator)
*		Multiplikation
/		Division
+		Addition, Verknüpfung (nur Zeichenketten)
-		Subtraktion
<	lt	Kleiner als
>	gt	Größer als
□	le	Kleiner als oder gleich
>=	ge	Größer als oder gleich
=	eq	Gleich
!=	ne	Ungleich
^^	xor	Exklusives Oder (Logischer Operator)
&&	and	Und (Logischer Operator)

Da KScript auf XML aufsetzt, müssen für die Angabe der Operatoren `&&`, `<` oder `⇐` die Zeichen `<` und `&` durch die Entities `<` bzw. `&` ersetzt oder die alternative Schreibweise verwendet werden.

Beispiel "Und":

```
<Path path="var(left) &amp;&amp; var(right)"/>
<Path path="var(left) and var(right)"/>
```

Beispiel "Kleiner als":

```
<Path path="var(left) &lt; var(right)"/>
<Path path="var(left) lt var(right)"/>
```

1.3.3.5.3. Bedingungen

Es können Bedingungen in folgender Form angegeben werden:

```
path1[path2]path3
```

Auf alle Elemente aus `path1`, die Bedingung `path2` erfüllen, wird `path3` angewendet. Um die Bedingung `path2` zu formulieren, können die Vergleichsoperatoren (siehe voriger Abschnitt) verwendet werden. Boolesche Ausdrücke können mit den boolschen Operatoren verknüpft werden.

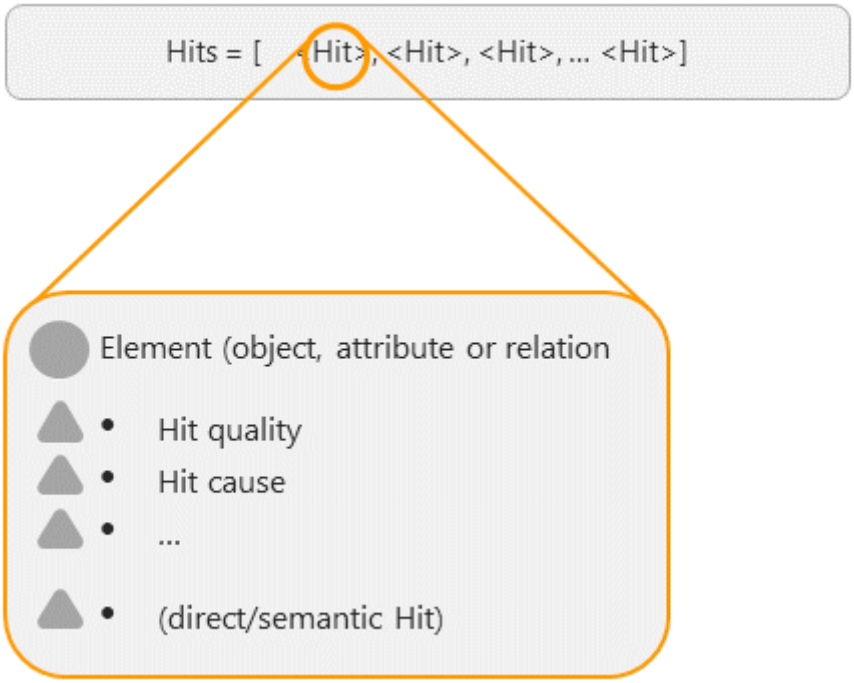
Namen aller Bücher, die von Goethe geschrieben wurden

```
//Buch\*[~Autor/target()/@Name = "Goethe"]/@Name/value()
```

1.3.4. Inhaltsmodell "Hit"

Damit für Suchabfragen sowohl Qualität als auch Ursachen mitverarbeitet und transportiert werden können, gibt es das Inhaltsmodell des Typs "Hit". Ein "Hit" kann als Container verstanden werden, der das Element inklusive mehrerer Eigenschaften zusammenfasst und temporär für den Kontext zur Verfügung stellt. Die enthaltenen Eigenschaften sind beispielsweise berechnete Treffer-Qualität, Treffer-Ursache, ChangeLog-Eintrag etc.

In Such-Pipelines stehen die Inhaltsmodelle "Hit" und "Hits" zur Verfügung. Der Typ "Hits" ist dabei ein Array aus mehreren "Hit"-Elementen:



1.3.4.1. Metaattribute der Hits

Außer dem semantischen Element werden in einem Hit folgende Metaattribute transportiert:

Treffer-Qualität (quality)

Kann in einer Such-Pipeline durch das Setzen einer Qualität einen Wert zwischen 0 und 1 annehmen; die Treffer einer Strukturabfrage enthalten per Default den Wert 1

Treffer-Ursache (cause)

Bezeichnet das Eingangs-Element, das zum Treffer geführt hat und um welchem Typ es sich handelt

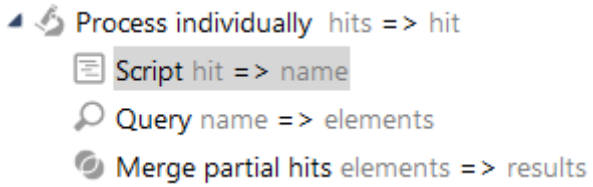
Treffer-Ursache (snippet)

Bezeichnet den Inhalt bzw. den Suchbegriff, der zum Treffer geführt hat

Detaillierte Informationen zu den Metaattributen sind in der JavaScript-API dokumentiert.

1.3.4.2. Verwendung von Hits in Such-Pipelines

Wenn in einer Such-Pipeline eine Treffermenge mithilfe einer einfachen Abfrage verarbeitet werden soll, so ist aus Gründen der zum Array zusammengesetzten Treffermenge eine Einzelverarbeitung notwendig: Abfragen können einen einzelnen "Hit" in Form eines Strings verarbeiten, jedoch keine "Hits" (= Array). Die Umwandlung eines "Hit" in einen String kann wiederum durch ein Skript erfolgen, das der einfachen Abfrage vorangestellt ist.



Beispiel-Skript für die Umwandlung eines Hit in einen String:

```
function search(input, inputVariables, outputVariables) {
  return input.element().name();
}
```

1.3.4.3. Verwendung von Hits in Tabellen

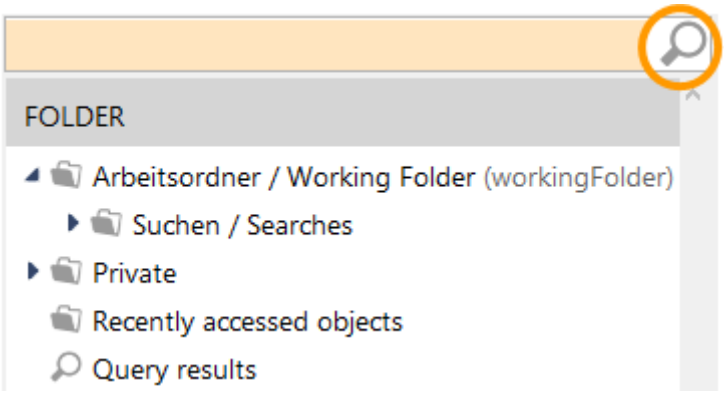
Für eine Tabelle steht in der übergeordneten Suchkonfiguration die Option *Hits verwenden* zur Verfügung. Diese Option bestimmt, ob für die Anzeige von Suchergebnissen das gesamte Hit-Element (semantisches Element + Metaattribute) weitergereicht werden soll oder nur das semantische Element.

1.3.4.4. Verarbeitung von Hits in Tabellen per Skript

Wenn die Suchergebnisse per Skript weiterverarbeitet werden sollen, bestimmt die Option *Hits verwenden*, ob das Suchergebnis wie ein Hit behandelt werden soll: Das Skript bekommt als JavaScript-Objekt entweder $k.SemanticElement oder $k.Hit zur Verarbeitung weitergereicht.

1.3.5. Die Suche im Knowledge-Builder

Bis auf die Strukturabfragen, die in den Ordnern angelegt und auch dort ausgeführt werden, können alle Suchen in der Kopfzeile des Knowledge-Builders für die interne Nutzung verfügbar gemacht werden.



Dazu müssen wir eine vorkonfigurierte Suche nur in das Suchfeld in der Knowledge-Builder-Kopfzeile hineinziehen. Stehen dort mehrere Suchen zur Auswahl, kann durch Klick auf das Lupensymbol aus einem PullDown-Menu die gewünschte Suche ausgewählt werden. Das Sucheingabefeld berücksichtigt dabei immer den zuletzt ausgeführten Suchmodus.

Entfernen können wir die Suche über die globalen Einstellungen. Hier können wir auch die Reihenfolge der verschiedenen Suchen im Menu ändern.

1.3.6. Spezialfälle

1.3.6.1. Volltextsuche mit Lucene

Die Volltextsuche lässt sich auch alternativ über den externen Indexer *Lucene* betreiben. Die Konfiguration der Suche ist dann analog zur normalen Volltextsuche, d.h. es können wiederum Attribute in die Suche konfiguriert werden, die an den Lucene-Index angeschlossen sind; der Suchvorgang erfolgt ebenfalls analog.

1.3.6.2. Suche mit regulären Ausdrücken

Reguläre Ausdrücke sind ein mächtiges Mittel, um je nach Aufgabe Datenbestände nach komplexen Suchausdrücken zu durchforsten.

Suche mit regulärem Ausdruck	Treffer
The [CF]all	The Call, The Fall
Car.	Cars
Car.*	Cars, Caravans, Carmen etc.
[^R]oom	Doom, Loom etc. (aber nicht Room)

Als Sucheingaben unterstützt i-views dabei den auch aus Perl bekannten Standard, der z.B. im Wikipedia-Artikel zu regulären Ausdrücken beschrieben ist.

1.3.6.3. Suche in Ordnern

Die Suche in Ordnern sucht in Ordnernamen und -inhalten:

- Ordner, deren Name auf die Sucheingabe passt
- Ordner, die Objekte enthalten, die auf die Sucheingabe passen
- Expertensuchen, die Elemente enthalten, auf die die Sucheingabe passt
- Skripte, in denen die Sucheingabe auftaucht
- Rechte- und Triggerdefinitionen, die Elemente enthalten, auf die die Sucheingabe passt

Mit der Sucheingabe "#obsolete" kann gezielt nach Verwendungen von gelöschten Objekten (z.B. in Rechten, Triggern, Suchen) gesucht werden. Bei der Konfiguration der Suche kann die Menge der zu durchsuchenden Ordner eingeschränkt werden. Außerdem kann die Option "Nach Objektnamen in Ordnern suchen" deaktiviert werden. Dies ist hilfreich, wenn man nicht nach semantischen Objekten in Ordnern suchen will, da bei umfangreichen Ordnern (z. B. gespeicherte Suchergebnisse) die Suche nach Objektnamen sehr lange dauern kann.

1.3.6.4. Suche nach Duplikaten

Nach Importen oder aus anderen Gründen wie bspw. der Qualitätssicherung kann der Fall vorliegen, dass nach Duplikaten — also doppelt vorkommenden semantischen Elementen — gesucht werden muss. Hierzu gibt es eine Lösung in Form einer gesonderten Strukturabfrage-Konstellation.

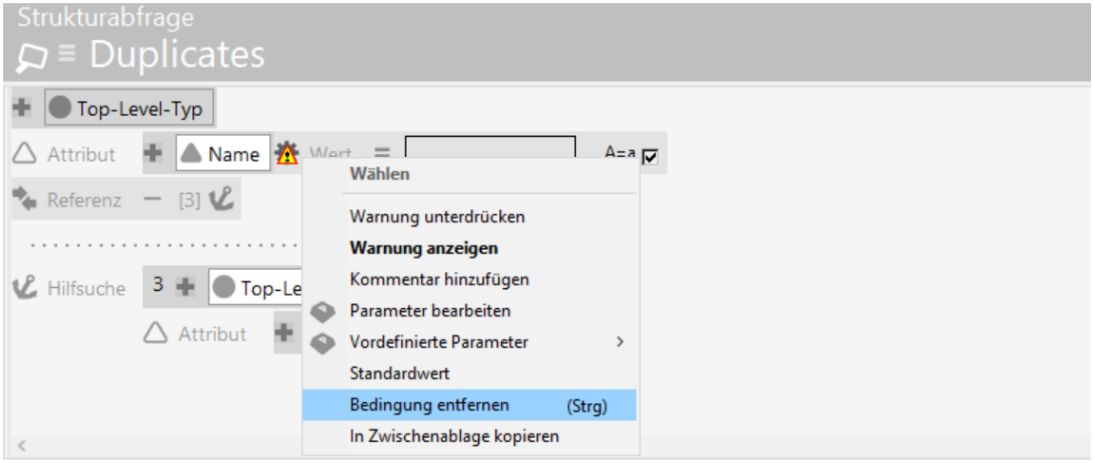
HINWEIS	Da die Strukturabfrage sich im gezeigten Beispiel ohne weitere Typeinschränkung auf alle Elemente des Knowledge-Graphs bezieht (Objekte von Top-Level-Typ), kann der Suchvorgang bei großen Beständen sehr lange dauern. Hier bietet es sich an, die Abfrage auf den spezifischsten Untertyp zu beschränken.

Prinzipiell ist die Strukturabfrage so aufgebaut, dass nach **unterschiedlichen** Objekten gesucht wird, die denselben Wert in ihrem identifizierenden Attribut aufweisen wie ein jeweils anderes Objekt (hier: Objekte mit demselben Namen).

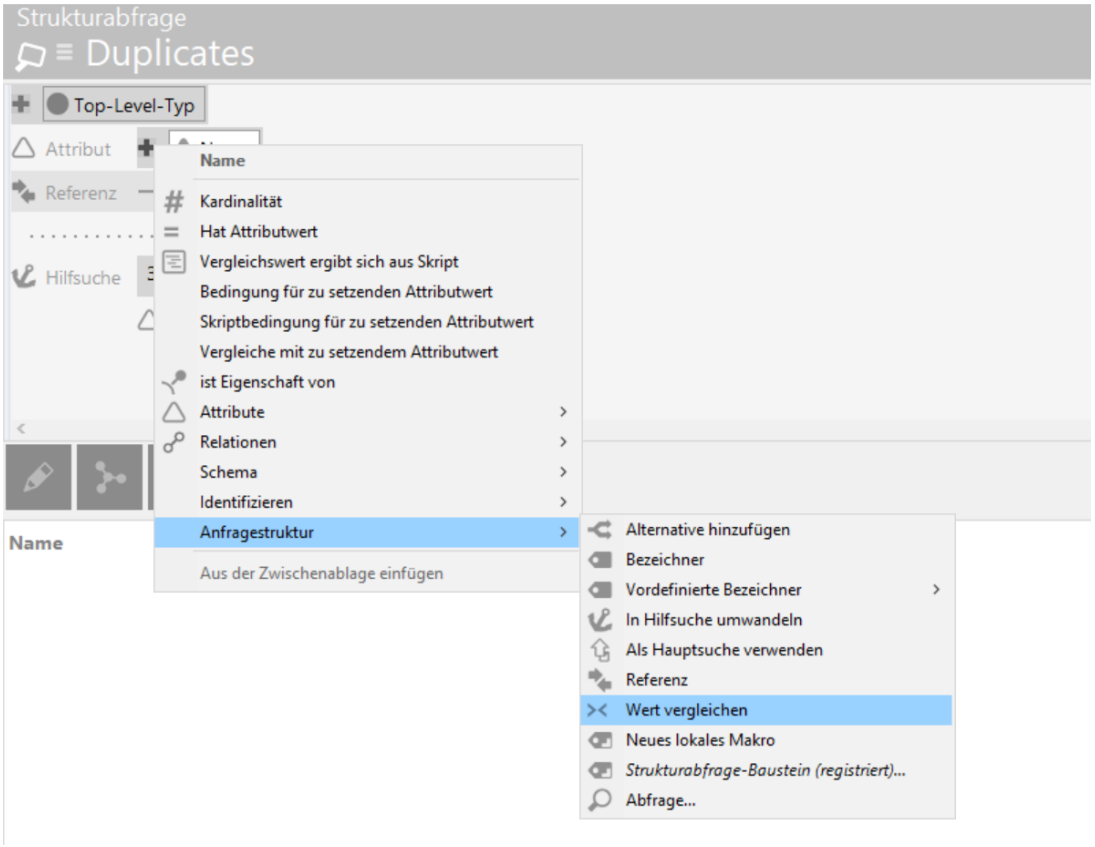
Die Abfrage nach Dubletten wird wie folgt aufgebaut:

1. Abfrage nach Objekte des gesuchten Untertyps konfigurieren, dabei identifizierendes Attribut (hier: Primärname) wählen.
2. Ausgehend vom gesuchten Objekt eine Hilfssuche anlegen, wobei durch Verwendung einer Negativ-Referenz (Vergleichsoperator "ist nicht") ausgeschlossen wird, dass es sich um ein- und dasselbe Objekt handelt:

3. Zum Abgleich der Attributwerte müssen zunächst die Werte-Felder entfernt werden:



4. Nachdem die Werte-Felder entfernt sind, steht per Kontextmenü die Option "Wert vergleichen" zur Verfügung. Anschließend als zu vergleichenden Wert das identifizierende Attribut des Objekts der Hilfssuche wählen:



Ergebnis: Strukturabfrage nach Duplikaten.

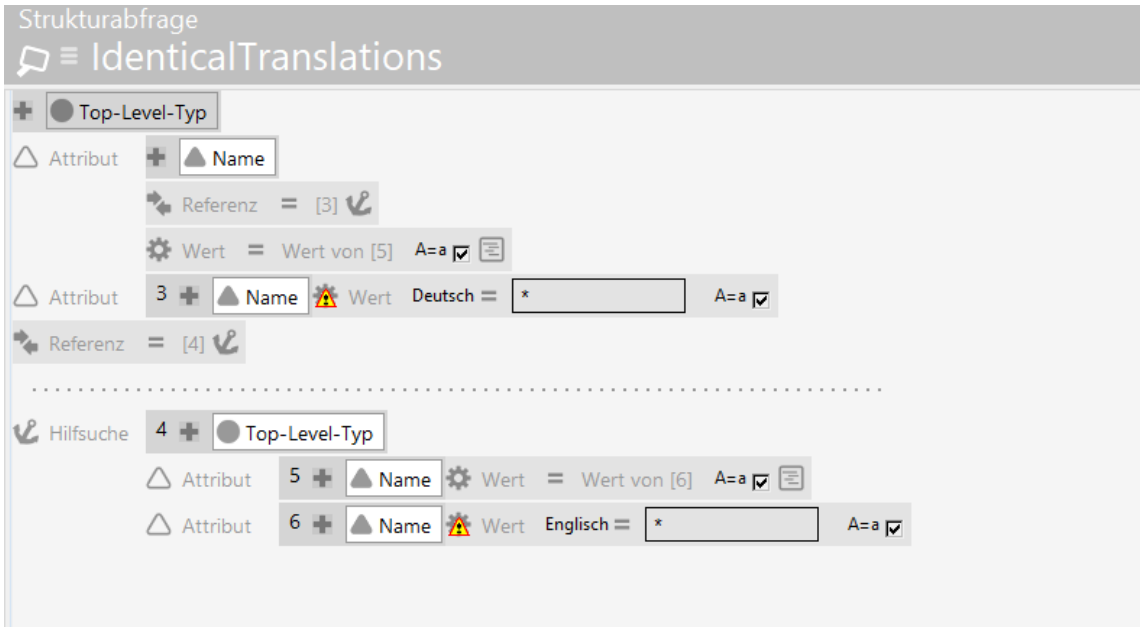
1.3.6.5. Suche nach identischen Übersetzungen

Ähnlich wie die Suche nach Duplikaten wird auch die Suche nach Objekten mit identischen

Übersetzungen mithilfe von Referenzen und Attributwertvergleichen aufgebaut.

HINWEIS	Da die Strukturabfrage sich im gezeigten Beispiel ohne weitere Typeinschränkung auf alle Elemente des Knowledge-Graphs bezieht (Objekte von Top-Level-Typ), kann der Suchvorgang bei großen Beständen sehr lange dauern. Hier bietet es sich an, die Abfrage auf den spezifischsten Untertyp zu beschränken.

Der Unterschied dieser Abfrage zur Abfrage nach Duplikaten besteht darin, dass es sich diesmal um ein- und dasselbe Objekt handelt, mit dem Zusatz des identischen Attributwerts in unterschiedlichen Übersetzungen desselben Attributs:



1.3.7. Graph Query Language (GQL)

Die Graph Query Language (GQL) ist ein ISO-Standard zu Abfrage und Modifikation von Graph-Datenbanken, die an die proprietäre Sprache Cypher angelehnt ist.

GQL ermöglicht es, Muster für zu suchende Knoten, Kanten und Eigenschaften kompakt zu formulieren. Das Ergebnis besteht aus einer Liste von Feldern, die Eigenschaften oder berechnete Werte enthalten.

Webseite: https://www.gqlstandards.org

HINWEIS	Datenmodifizierende Statements werden derzeit noch nicht unterstützt.
HINWEIS	Der GQL-Standard ist jung und es existieren noch wenige Referenzimplementierungen. Es kann daher vorkommen, dass das Verhalten der i-views-Implementierung in einzelnen Punkten vom Standard abweicht. Durch zukünftige Revisionen des Standards kann es zudem zu Änderungen des

> vorgeschriebenen Verhaltens kommen. In solchen Fällen gilt der Standard als maßgeblich, und abweichendes Verhalten kann jederzeit — zum Beispiel in Patch-Releases — ohne Ankündigung korrigiert werden.
>
> Dies gilt nicht für i-views-spezifische Spracherweiterungen, da diese außerhalb des GQL-Standards liegen.

1.3.7.1. Aufbau

Folgende Query findet alle produzierten Produkte, deren Gewicht mindestens den Wert 10 hat, und gibt nach Gewicht sortiert Hersteller, Produkt und Gewicht zurück:

```
MATCH (m:manufacturer)-[:produces]->(p:product)
WHERE p.weight >= 10
RETURN m.name as manufacturer, p.name as product, p.weight as weight
ORDER BY weight
```

Ergebnis

manufacturer	product	weight
ACME	Artificial Rock	50
ACME	Anvil	230

`MATCH` listet einen oder mehrere Pfade auf, die im Graph gesucht werden sollen. Ein Pfad besteht abwechselnd aus einem **Knoten** in runden Klammern und einer **Kante** in eckigen Klammern. Anfang und Ende müssen ein Knoten sein.

Knoten sind normalerweise Objekte im Knowledge-Graph, und Kanten entsprechen Relationen.

Knoten und Kanten bestehen aus einer optionalen Variable und einem optionalen Label, die durch einen Doppelpunkt getrennt werden. Bei `m:manufacturer` ist `m` die Variable und `manufacturer` das Label. Bei der Kante `[:produces]` wurde die Variable weggelassen.

Das Label gibt den Typ der zu suchenden semantischen Elemente an. Standardmäßig ist das Label der interne Name des Typs. Sie können aber auch eine eigene Label-Konfiguration einrichten.

Mit der Variable können Sie den Knoten beziehungsweise die Kante in der Query später erneut referenzieren.

`WHERE` gibt zusätzliche Bedingungen für die zu suchenden Knoten und Kanten an. Der Ausdruck `p.weight >= 10` referenziert über die Variable `p` die Knoten der Produkte. `weight` ist das Label einer Eigenschaft der Produkte.

Bedingungen können nur Knoten und Kanten referenzieren, die im MATCH-Teil aufgelistet wurden, es können keine zusätzlichen Knoten und Kanten abgefragt werden.

`RETURN` gibt eine Liste von Elementen an, die im Ergebnis enthalten sein sollen. Neben Eigenschaften können mit Hilfe von Ausdrücken auch Werte berechnet werden. Für jedes in RETURN aufgelistete Element wird im Ergebnis ein Feld ausgegeben. Der Name des Felds kann durch `AS name` bestimmt werden. Die Rückgabe entspricht einer Tabelle mit einer Spalte pro Feld und einer Zeile pro zum Muster passender Belegung der Felder.

Das optionale `ORDER BY` sortiert die Ergebnisse. Es können auch Eigenschaften/Werte verwendet werden, die nicht Teil von RETURN sind.

1.3.7.2. Abgrenzung zu Strukturabfragen

- Im Gegensatz zu Strukturabfragen werden GQL-Suchen textuell formuliert. Die Schema-Unterstützung ist minimal und wird in der Regel erst zur Laufzeit ausgewertet.
- Das Ergebnis besteht nicht aus einer unsortierten Menge an semantischen Elementen, sondern aus einer Liste von Belegungen der in `RETURN` angegebenen Feld-Elemente. Die Liste kann durch `ORDER BY` sortiert werden.

1.3.7.3. GQL-Suchen ausführen

Es gibt mehrere Möglichkeiten, GQL-Queries im Knowledge-Graph auszuführen:

- Workbench
- GQL-Suchen als Ordnerelemente oder registrierte Suchen
- REST-API
- JavaScript-API

1.3.7.3.1. Workbench

In der Workbench können Sie beliebige GQL-Suchen formulieren und ausführen. Sie öffnen die Workbench im Hauptmenü unter ***Werkzeuge > GQL Workbench***.

1.3.7.3.2. GQL-Suchen als Ordnerelemente oder registrierte Suchen

Beim Anlegen einer neuen Suche als Ordnerelement oder registrierte Suche können Sie im Dialog in der Liste ***GQL-Abfrage*** auswählen.

Die Suche können Sie dann wie andere Arten von Suchen (Strukturabfrage usw.) ausführen. GQL unterscheidet sich allerdings von anderen Sucharten darin, dass GQL eine Tabelle als Ergebnis hat, während andere Suchen eine Menge von semantischen Elementen oder Treffer-Objekten liefern.

Um zu anderen Suchen kompatibel zu sein, werden bei GQL-Suchen, die über die Standard-APIs/Werkzeuge aufgerufen werden, die Belegungen der Variablen in den zurückgegebenen Zeilen gesammelt zurückgegeben.

Bei der Suche

```
MATCH (m:manufacturer)-[:produces]->(p:product)-[:contains]->(c:component)
WHERE c.weight > 1000
RETURN m.name, p.name
```

werden Hersteller (`m`) und Produkte (`p`) als Ergebnis zurückgegeben.

Der Zugriff auf die Ergebnis-Tabelle ist nur mit der dedizierten GQL-API möglich.

1.3.7.4. Pfad-Muster

1.3.7.4.1. Knoten

Im einfachsten Fall besteht bei `MATCH` das Pfad-Muster aus einem einzelnen Knoten. Knoten bestehen aus runden Klammern `()`, die eine optionale Variable und ein optionales Label enthalten.

Falls ein Label angegeben wird, muss ein Doppelpunkt `:` vor dem Label stehen, auch wenn keine Variable angegeben wird: `(:company)`.

Das folgende Muster weist der Variablen `m` alle Knoten mit dem Label `manufacturer` zu. Das Label entspricht einem Typ im Knowledge-Graph, und die Knoten umfassen alle Objekte dieses Typs und aller Untertypen.

```
MATCH (m:manufacturer) RETURN m
```

1.3.7.4.2. Labels

Das Label entspricht normalerweise dem internen Namen eines Typs des Knowledge-Graphs.

Labels können beliebige Zeichen enthalten. Allerdings müssen sie in doppelten Anführungszeichen (`"`) oder Akzentzeichen (`` ` ``) eingeschlossen werden, wenn sie nicht ausschließlich aus Buchstaben, Ziffern oder Unterstrich (`_`) bestehen. Der erste Buchstabe darf keine Ziffer sein. Dies entspricht einem Unicode-Identifier. Das einfache Anführungszeichen `'` kann nicht verwendet werden.

```
MATCH (p:"base.product") RETURN p
```

Labels können kombiniert werden:

- Vereinigung: `MATCH (:corporation|association)`
- Schnittmenge: `MATCH (:product&cataloged)`
- Ausnahmen, normalerweise nur in Verbindung mit Schnittmenge: `MATCH (:company&!association)`

Das Label ist optional und kann auch weggelassen werden. Verzichten Sie darauf nur, wenn die Knoten/Kanten-Menge durch weitere Pfad-Muster eingeschränkt wird. In folgender Query ist die Menge der Knoten `m` schon durch die Kante `:produces` eingeschränkt:

```
MATCH (m)-[:produces]->(p:product) RETURN m, p
```

Die folgende Query ohne Label adressiert den gesamten Knowledge-Graph und ist deshalb zu vermeiden:

```
MATCH (m) RETURN m
```

1.3.7.4.3. Kanten

Kanten verbinden Knoten im Pfad-Muster und entsprechen Relationen im Knowledge-Graph. Kanten werden durch eckige Klammern `[]` repräsentiert, ansonsten gelten dieselben Regeln wie für Knoten. Auch Kanten können einer Variable zugewiesen und zurückgegeben werden.

Folgendes Muster findet alle Relation vom Relationstyp mit dem internen Namen `produces` und gibt diese zurück:

```
MATCH (:company)-[p:produces]->(:product) RETURN p
```

Das Label ist optional. Verzichten Sie bei Kanten jedoch nicht darauf. Ohne Label werden alle Relationen zwischen den Knoten ermittelt, einschließlich Systemrelationen und Abkürzungsrelationen — das ist sehr aufwändig, auch wenn die Knoten Labels haben.

Kanten können in GQL ungerichtet oder gerichtet sein. Bei gerichteten Kanten steht `<` oder `>` vor dem Zielknoten. Bei ungerichteten Kanten enthält der Pfad keine spitze Klammern.

Es ist auch erlaubt, beide Richtungen anzugeben, was in der GQL-Umsetzung im Knowledge-Graph keinen Unterschied macht.

Muster für "*Komponente c1 ist Teil von c2*"

```
MATCH (c1:component)-[:partOf]->(c2:component) RETURN c1, c2
```

Muster für "*Komponente c2 ist Teil von c1*"

```
MATCH (c1:component)<-[:partOf]-(c2:component) RETURN c1, c2
```

Muster für "*Komponente c1 ist Teil von oder hat Teil c2*", einmal als Kante ohne Richtung formuliert,

einmal als Kante in beide Richtungen

```
MATCH (c1:component)-[:partOf]-(c2:component) RETURN c1, c2
```

```
MATCH (c1:component)<-[:partOf]->(c2:component) RETURN c1, c2
```

Kanten verbinden Knoten, die im Pfad immer angegeben werden müssen. Minimal reicht dazu ein leeres Paar runder Klammern: `()`.

```
MATCH ()-[p:produces]->() RETURN p
```

Pfade können mehrere Kanten enthalten:

```
MATCH (c:component)-[:partOf]->(p:product)<-[:produces]-(m:manufacturer)
RETURN c,p,m
```

1.3.7.4.4. Eigenschaftsfilter in Mustern

Neben dem Label können Knoten und Kanten im Muster direkt mit Eigenschaftsbedingungen versehen werden. Dies ist eine kompakte Alternative zur `WHERE`-Klausel.

Eigenschaftsfilter mit Record-Syntax

Ein Record in geschweifte Klammern nach dem Label legt fest, dass die angegebenen Eigenschaften exakt den angegebenen Werten entsprechen müssen.

```
MATCH (p:product {category: 'electronics', inStock: TRUE})
RETURN p.name, p.price
```

Für Kanten gilt dieselbe Syntax:

```
MATCH (p:person)-[:knows {since: 2020}]->(q:person)
RETURN p.name, q.name
```

Eingebettete WHERE-Klausel

Für komplexere Bedingungen kann eine `WHERE`-Klausel direkt im Knoten- oder Kantenmuster angegeben werden. Dies erlaubt beliebige Ausdrücke, nicht nur Gleichheitsvergleiche.

```
MATCH (p:product WHERE p.price < 100 AND p.rating >= 4)
RETURN p.name, p.price, p.rating
```

```
MATCH (p:person)-[r:knows WHERE r.since > 2015]->(q:person)
RETURN p.name, q.name, r.since
```

1.3.7.4.5. Muster mit mehreren Pfaden

Ein Pfad ist immer eine lineare Abfolge von Knoten und Kanten, das heißt, es gibt keine Abzweigungen. Um komplexere Muster zu formulieren, können mehrere Pfade angegeben werden. Pfade können dabei Knoten und Kanten aus vorherigen Pfaden referenzieren.

Wiederverwendung von Variablen

Wird dieselbe Variable in mehreren Pfaden verwendet, wird sie immer auf dasselbe Element gebunden. Das bedeutet, dass alle Pfade, die die Variable verwenden, auf dasselbe Knoten- oder Kanten-Objekt verweisen müssen. Auf diese Weise können Sie Bedingungen über mehrere Pfade hinweg ausdrücken.

```
MATCH (p:person)-[:owns]->(t:product),
      (p)-[:worksAt]->(c:company)
RETURN p.name, t.name, c.name
```

Die Variable p tritt in beiden Pfaden auf und stellt sicher, dass Produkt und Unternehmen zur selben Person gehören. Ohne diese Verknüpfung würde jede Kombination aus Person, Produkt und Unternehmen zurückgegeben.

Beispiel mit mehreren verknüpften Pfaden

Folgende Query findet Kombinationen aus Komponente, Produkt, Hersteller und Qualifikation, für die gilt:

- Die Komponente ist Teil des Produkts
- Das Produkt wird vom Hersteller produziert
- Das Produkt benötigt die Qualifikation

```
MATCH
  (c:component)-[:partOf]->(p:product)<-[:produces]-(m:manufacturer),
  (p)-[:requires]->(q:qualification)
RETURN
  c.name as component, p.name as product, m.name as company, q.name as
```

```
qualification
```

Die Variable p im zweiten Pfad referenziert das Produkt p aus dem ersten Pfad.

Falls es mehrere Komponenten, Hersteller, Produkte oder Qualifikationen gibt, werden alle Kombinationen einzeln aufgelistet:

component	product	company	qualification
Compressor	Jet Motor	ACME	Technician
Compressor	Jet Motor	ACME	Pilot
Turbine	Jet Motor	ACME	Technician
Turbine	Jet Motor	ACME	Pilot

1.3.7.4.6. Quantifizierte Pfade

Pfad-Muster können mit einem Quantifizierer versehen werden, um Pfade variabler Länge zu suchen. Der Quantifizierer steht in geschweiften Klammern hinter dem zu wiederholenden Muster-Element und gibt die Anzahl der erlaubten Wiederholungen an. Jedes einzelne Knoten- oder Kanten-Element kann quantifiziert werden, ebenso wie ganze Teilpfade in runden Klammern.

Quantifizierer	Bedeutung
`{n}`	Genau n Wiederholungen
`{n,m}`	Mindestens n, höchstens m Wiederholungen
`{n,}`	Mindestens n Wiederholungen (offen nach oben)
`{,m}`	Höchstens m Wiederholungen

```
MATCH (p:person)-[:knows]->{1,3}(q:person)
RETURN p.name, q.name
```

Dieses Muster findet Personen p und q, die über eine bis drei knows-Kanten verbunden sind. Die Kante :knows wird quantifiziert; die dabei implizit entstehenden Zwischenknoten sind anonym. Der explizit angegebene Zielknoten (q:person) ist nicht Teil der Quantifizierung und bezeichnet immer einen einzelnen Knoten.

Ganze Teilpfade in runden Klammern können ebenfalls quantifiziert werden:

```
MATCH (p:person)(-[:owns]->(:item)<-[:wants]-(:person)){1,3}(q:person)
RETURN p.name, q.name
```

Dieses Muster findet Personen `p`, die über ein bis drei Schritte aus "besitzt einen Gegenstand, den eine Person haben möchte" mit einer Zielperson `q` verbunden sind.

HINWEIS	Bei offenen Quantifizierern ohne Obergrenze (`{n,}`) müssen Sie sicherstellen, dass der Graph keine Zyklen enthält oder die Suche anderweitig eingeschränkt wird, da sonst sehr lange Pfade gesucht werden.

Gruppen-Variablen

Variablen, die innerhalb des quantifizierten Teils eines Musters definiert werden, sind außerhalb des Musters als Gruppen-Variablen verfügbar. Eine Gruppen-Variable enthält eine Liste aller Werte, die über alle Wiederholungen des Musters gebunden wurden. Darauf können alle Listen-Funktionen angewendet werden.

```
MATCH (p:person)-[r:knows]->{1,3}(q:person)
RETURN p.name, q.name, SIZE(r) AS hops
```

Hier enthält `r` eine Liste aller `knows`-Kanten entlang des Pfads, und `SIZE(r)` gibt die Anzahl der Hops zurück.

1.3.7.4.7. Optionale Pfade

Ein Pfad-Muster kann mit `?` als optional markiert werden. Das Muster wird entweder einmal oder gar nicht gefunden. Wie bei quantifizierten Pfaden kann jedes einzelne Element oder ein geklammeter Teilpfad optional gemacht werden.

Im Gegensatz zu quantifizierten Pfaden entstehen durch `?` keine Gruppen-Variablen. Variablen im optionalen Teil sind einzelne Werte, die `NULL` sind, wenn das Muster nicht gefunden wird — die Ergebniszeile selbst bleibt erhalten.

```
MATCH (p:person)-[r:manages]->?(q:person)
RETURN p.name, q.name, r
```

Wenn keine `manages`-Kante existiert, enthält `r` den Wert `NULL` und `p` und `q` beziehen sich auf dieselbe Person.

1.3.7.5. Werte

Folgende literale Wertetypen sind vom GQL-Standard definiert:

Zeichenketten

Zeichenketten werden durch einfache oder doppelte Anführungszeichen eingeschlossen: `'Some string'` oder `"Another string"`.

HINWEIS

Das Anführungszeichen selbst kann in der Zeichenkette durch einen vorangestellten Backslash angegeben werden: `"A \"quote\""`

Zahlen

Mögliche Notationen sind

- Ganzzahlen, (`42`)
- Dezimalzahlen (`-1.23`)
- Fließkommazahlen mit Mantisse und Exponent und optionaler Genauigkeit (`-1.4e6`, `2.34e9d`)
- hexadezimale, oktale und binäre Ganzzahlen (`0x1f`, `0o7`, `0b101`)

HINWEIS

Im Knowledge-Graph werden nur Ganzzahlen und Fließkommazahlen mit doppelter Genauigkeit unterstützt, alle anderen Formate werden konvertiert.

Null

Der Wert `NULL` entspricht einem nicht vorhanden Wert, kann aber auch als literaler Wert verwendet werden.

Wahrheitswerte

Neben `TRUE` und `FALSE` wird in GQL `NULL` als unbekannter Wahrheitswert verwendet.

WARNUNG

GQL definiert zusätzlich `UNKNOWN` als Schlüsselwort für einen unbekannten Wahrheitswert. `UNKNOWN` ist allerdings kein gültiger literaler Wert und kann ausschließlich beim Operator `IS [NOT]` verwendet werden.

Allerdings kann hier ebenfalls `NULL` verwendet werden: `IS [NOT] NULL` entspricht `IS [NOT] UNKNOWN`. Auf den Einsatz von `UNKNOWN` können Sie deshalb verzichten.

Datum/Uhrzeit

In GQL sind Datum, Uhrzeit sowie Datum und Uhrzeit (auch Zeitstempel genannt) definiert. Werte mit Uhrzeit können entweder mit oder ohne Zeitzone angegeben werden.

GQL orientiert sich bei der Syntax an den Standards `ISO 8601-1:2019` und `ISO 8601-2:2019`.

Wertetyp	Notation	Wert
Datum	`DATE "2025-02-28"`	28. Februar 2025
Datum	`DATE "20240229"`	29. Februar 2024
Uhrzeit	`TIME "12:34:56"`	12:34:56

Wertetyp	Notation	Wert
Uhrzeit	`TIME "23:59"`	23:59:00
Datum und Uhrzeit	`DATETIME "2025-02-23T13:45"`	23.2.2025 13:45:00
Datum und Uhrzeit	`TIMESTAMP "2025-03-05T02:03:04"`	5.3.2025 02:03:04

HINWEIS Im Knowledge-Graph werden keine Zeitzonen unterstützt, bei Zeitangaben mit Zeitzone wird die Zeit in lokale Zeit umgerechnet.

Zeitspannen

In GQL können Zeitspannen auf Datums- oder Uhrzeit-Ebene angegeben werden.

GQL orientiert sich bei der Syntax an den Standards `ISO 8601-1:2019` und `ISO 8601-2:2019`.

Notation	Dauer
`DURATION "P1Y"`	1 Jahr
`DURATION "P2M"`	2 Monate
`DURATION "P30D"`	30 Tage
`DURATION "P1Y2M"`	1 Jahr 2 Monate
`DURATION "PT4H"`	4 Stunden
`DURATION "PT120M"`	120 Minuten
`DURATION "PT30S"`	30 Sekunden

Listen

Listen sind geordnete Folgen von Werten beliebiger Typen. Eine Liste wird durch eckige Klammern eingeschlossen und die Elemente werden durch Komma getrennt.

```
['ACME', 'Mustermann GmbH']
[1, 2, 3]
[DATE "2025-01-01", DATE "2025-06-01"]
```

Eine leere Liste wird als `[]` geschrieben. Die Elemente einer Liste müssen nicht alle vom gleichen Typ sein.

HINWEIS Ein Zugriff auf Listenelemente über einen Index wird in GQL nicht unterstützt.

Records

Records sind Sammlungen von benannten Feldern. Ein Record wird durch geschweifte Klammern eingeschlossen; jedes Feld besteht aus einem Schlüssel, einem Doppelpunkt und

einem Wert; mehrere Felder werden durch Komma getrennt.

```
{companyName: 'ACME'}
{searchString: name, maxResults: 10}
{distanceValue: '100km'}
```

Schlüssel folgen denselben Regeln wie Eigenschaftsnamen: Sie müssen in doppelten Anführungszeichen oder Akzentzeichen eingeschlossen werden, wenn sie nicht ausschließlich aus Buchstaben, Ziffern oder Unterstrich bestehen. Records werden unter anderem beim Aufruf von Prozeduren mit `CALL` zur Parameterübergabe verwendet.

1.3.7.6. Ausdrücke

Bedingungen, zurückzugebende Werte und Werte für die Sortierung werden durch Ausdrücke formuliert.

HINWEIS	Im folgenden bezeichnen in spitze Klammern eingeschlossene Bezeichner wie zum Beispiel `<value>` einen Ausdruck.

1.3.7.6.1. Literale Werte

Jeder gültige literale Wert ist auch ein Ausdruck. Siehe Werte.

1.3.7.6.2. Variablen-Referenzen

Variable des Pfad-Musters können über ihren Namen referenziert werden. Ergebnis des Ausdrucks ist die Belegung der Variable, also ein Knoten, Kante oder Pfad.

```
MATCH p1 = (m:manufacturer)-[r:produces]->(p:product)
RETURN m, r, p, p1
```

Im Beispiel sind `m`, `r` und `p` Variablen-Referenzen, die auf den Hersteller-Knoten, die Kante und den Produkt-Knoten verweisen. `p1` ist eine Pfad-Variable, die den gesamten Pfad enthält.

HINWEIS	`path` ist ein reserviertes Schlüsselwort in GQL und kann nicht als Variablenname verwendet werden.

1.3.7.6.3. Eigenschaften

Eigenschaften entsprechen Attributen im Knowledge-Graph.

Analog zu den Labels entsprechen die Namen von Eigenschaften normalerweise den internen Namen von Attributtypen. Die Namen müssen ebenfalls in doppelten Anführungszeichen (") oder Akzentzeichen (`) eingeschlossen werden, wenn sie nicht ausschließlich aus Buchstaben, Ziffern

oder Unterstrich (_) bestehen.

HINWEIS	Im Gegensatz zu Labels können Eigenschaftsnamen nicht mit &, | oder ! kombiniert werden.

Falls ein Knoten oder Kante die angegebene Eigenschaft nicht hat, wird als Wert NULL zurückgegeben.

```
MATCH (m:manufacturer)-[:produces]->(p:product)
RETURN m.name, m.isin, p.name, p.id
```

1.3.7.6.4. Operatoren

Das Ergebnis eines Operators hängt vom Typ der beteiligten Werte ab.

Allgemeine Operatoren

Operator	Ergebnis
+	Addiert numerische oder chronologische Werte.
-	Subtrahiert numerische oder chronologische Werte.
*	Multipliziert numerische Werte oder Zeitspannen
/	Dividiert numerische Werte oder Zeitspannen
||	Verknüpft Listen oder Zeichenketten

Vergleichsoperatoren und Prädikate

Vergleichsoperatoren liefern einen Boolschen Wert. Bei binären Operatoren müssen die Werte vergleichbar sein.

HINWEIS	Momentan müssen die verglichenen Werte vom selben Typ sein, das GQL-Feature *GA04 Universal comparison* wird nicht unterstützt.

Operator	Ergebnis
=	Gleich
<>	Ungleich
<, <=, >, >=	Kleiner (gleich), größer (gleich)
<boolean> IS [NOT] <truth value>	Vergleich den Wert einen Boolschen Wert mit einem Wahrheitswert (TRUE, FALSE, UNKNOWN). UNKNOWN entspricht einem NULL-Wert (siehe Ausdrücke mit Null-Werten).

Operator	Ergebnis
`<value> IS [NOT] NULL`	True wenn der Wert (k)ein `NULL`-Wert ist
`<node_or_edge> IS [NOT] LABELED <label>`	Prüft zur Laufzeit, ob ein Knoten oder eine Kante ein bestimmtes Label hat.
`PROPERTY_EXISTS(<node_or_edge>, <identifier>)`	Prüft, ob ein Knoten oder eine Kante die angegebene Eigenschaft besitzt.
`SAME(<element1>, <element2>, …)`	Prüft, ob alle angegebenen Variablen auf dasselbe Graph-Element verweisen.
`ALL_DIFFERENT(<element1>, <element2>, …)`	Prüft, ob alle angegebenen Variablen auf paarweise verschiedene Graph-Elemente verweisen.

Logik-Operatoren

Folgende Logik-Operatoren können nur auf Boolsche Werte angewendet werden.

Operator	Ergebnis
`AND`	Boolsches Und
`OR`	Boolsches Oder
`XOR`	Boolsches Exclusiv-Oder
`NOT`	Boolsche Negation

HINWEIS

Im Gegensatz zu JavaScript gibt es in GQL kein "truthy" oder "falsy", andere Werte wie zum Beispiel Zahlen werden deshalb nicht implizit in Boolsche Werte umgewandelt.

1.3.7.6.5. EXISTS-Ausdruck

Mit `EXISTS { … }` kann geprüft werden, ob ein Muster im Graph vorhanden ist, ohne die gefundenen Elemente zurückzugeben. Das Ergebnis ist ein Boolscher Wert und kann direkt in einer `WHERE`- oder `FILTER`-Klausel verwendet werden. Mit `NOT EXISTS { … }` wird geprüft, ob kein passendes Muster vorhanden ist.

```
MATCH (p:product)
WHERE EXISTS {
  MATCH (p)-[:contains]->(c:component)
  WHERE c.weight > 10
}
```

```
RETURN p.name
```

Dieses Muster gibt alle Produkte zurück, die mindestens eine Komponente mit einem Gewicht über 10 enthalten.

```
MATCH (p:person)
WHERE NOT EXISTS {
  MATCH (p)-[:owns]->(:product)
}
RETURN p.name
```

Dieses Muster gibt alle Personen zurück, die kein Produkt besitzen.

HINWEIS	`EXISTS` ist eine Alternative zu `OPTIONAL MATCH` in Fällen, in denen nur das Vorhandensein eines Musters relevant ist und keine Eigenschaften der gefundenen Elemente benötigt werden.

1.3.7.6.6. CASE-Ausdruck

Mit einem `CASE`-Ausdruck können abhängig von Bedingungen unterschiedliche Werte zurückgegeben werden. Ein `CASE`-Ausdruck besteht aus einer oder mehreren `WHEN`-Klauseln und einem optionalen `ELSE`-Zweig. Trifft keine `WHEN`-Klausel zu und ist kein `ELSE` angegeben, ist das Ergebnis `NULL`.

Einfache Form

In der einfachen Form wird ein Ausdruck mit einer Liste von Werten auf Gleichheit verglichen.

```
MATCH (p:product)
RETURN p.name,
  CASE p.category
    WHEN 'electronics' THEN 'Elektronik'
    WHEN 'furniture' THEN 'Möbel'
    ELSE 'Sonstiges'
  END AS categoryLabel
```

Die einfache Form kann auch mit einem Prädikat verwendet werden. Der Ausdruck nach `CASE` dient dabei implizit als linke Seite des Prädikats.

```
MATCH (p:product)
RETURN p.name, p.price,
  CASE p.price
```

```
    WHEN < 10 THEN 'günstig'
    WHEN < 100 THEN 'mittel'
    WHEN IS NULL THEN 'unbekannt'
    ELSE 'teuer'
  END AS priceCategory
```

Gesuchte Form

In der gesuchten Form wird jede `WHEN`-Klausel als eigenständige Boolsche Bedingung ausgewertet. Dies erlaubt beliebige Ausdrücke, nicht nur Vergleiche eines einzelnen Werts.

```
MATCH (p:product)
RETURN p.name,
  CASE
    WHEN p.price < 10 AND p.rating >= 4 THEN 'Empfehlung'
    WHEN p.price >= 10 AND p.rating >= 4 THEN 'Gut, aber teuer'
    ELSE 'Keine Empfehlung'
  END AS recommendation
```

1.3.7.6.7. Funktionen

Funktionen sind Teil der GQL-Syntax, es gibt keinen generischen Funktionsaufruf. Dadurch unterscheiden sich einige Funktionen syntaktisch von den restlichen Funktionen.

HINWEIS

Wenn ein Funktionsargument den Wert `NULL` hat, ist das Ergebnis der Funktion ebenfalls `NULL`. Dies gilt nur für übergebene Argumente — optionale Argumente, die weggelassen werden, lösen dieses Verhalten nicht aus.

Mathematische Funktionen

Funktion	Ergebnis
`ABS(<value>)`	Absoluter numerischer Wert
`FLOOR(<value>)`	Größte ganze Zahl, die kleiner oder gleich dem Argument ist
`CEIL(<value>)`	Kleinste ganze Zahl, die größer oder gleich dem Argument ist
`MOD(<dividend>, <divisor>)`	Modulus (Rest der ganzzahligen Division)
`SQRT(<value>)`	Quadratwurzel
`POWER(<base>, <exponent>)`	Potenz
`EXP(<value>)`	Exponentialfunktion (e hoch `<value>`)

Funktion	Ergebnis
`LN(<value>)`	Natürlicher Logarithmus
`LOG(<base>, <value>)`	Logarithmus zur angegebenen Basis
`LOG10(<value>)`	Dekadischer Logarithmus
`SIN(<value>), COS(<value>), TAN(<value>), COT(<value>)`	Trigonometrische Funktionen (Argument in Radiant)
`ASIN(<value>), ACOS(<value>), ATAN(<value>)`	Inverse trigonometrische Funktionen, Ergebnis in Radiant
`SINH(<value>), COSH(<value>), TANH(<value>)`	Hyperbolische Funktionen
`DEGREES(<value>), RADIANS(<value>)`	Umrechnung zwischen Radiant und Grad

Zeichenketten-Funktionen

Funktion	Ergebnis
`UPPER(<string>)`	Zeichenkette in Großbuchstaben
`LOWER(<string>)`	Zeichenkette in Kleinbuchstaben
`NORMALIZE(<string> [, NFC|NFD|NFKC|NFKD])`	Unicode-Normalisierung der Zeichenkette in die angegebene Normalform (Standard: `NFC`).
`TRIM(<string>)`	Zeichenkette ohne führende und nachgestellte Leerzeichen. Über optionale Argumente kann gesteuert werden, welche Zeichen entfernt werden und von welcher Seite: `TRIM(LEADING '#' FROM <string>)`, `TRIM(TRAILING FROM <string>)`, `TRIM(BOTH '.' FROM <string>)`. Ohne Angabe gilt `BOTH` und Leerzeichen als Trennzeichen.
`LTRIM(<string> [, <chars>]), RTRIM(<string> [, <chars>]), BTRIM(<string> [, <chars>])`	Zeichenkette ohne führende (`LTRIM`), nachgestellte (`RTRIM`) oder beidseitige (`BTRIM`) Vorkommen der angegebenen Zeichenkette. Ohne Angabe von `<chars>` werden Leerzeichen entfernt.
`CHAR_LENGTH(<string>)`	Länge der Zeichenkette in Zeichen

Funktion	Ergebnis
`LEFT(<string>, <n>)`	Die ersten `<n>` Zeichen der Zeichenkette
`RIGHT(<string>, <n>)`	Die letzten `<n>` Zeichen der Zeichenkette

Listen-Funktionen

Funktion	Ergebnis
`SIZE(<list>)`	Anzahl der Elemente in der Liste
`TRIM(<list>, <n>)`	Liste der ersten `<n>` Elemente der Liste

Graph-Funktionen

Funktion	Ergebnis
`ELEMENT_ID(<node_or_edge>)`	Interne ID des Knotens oder der Kante als Zeichenkette
`PATH_LENGTH(<path>)`	Anzahl der Kanten im Pfad
`ELEMENTS(<path>)`	Elemente des Pfads als Liste (Knoten, Kanten und Teilpfade)

Datum/Uhrzeit-Funktionen

Funktion	Ergebnis
`CURRENT_DATE, DATE()`	Aktuelles Datum
`DATE(<string>)`	Datum aus einer Zeichenkette (ISO-8601-Format)
`DATE({year} | {year, month} | {year, month, day})`	Datum aus einzelnen Komponenten
`CURRENT_TIME, LOCAL_TIME()`	Aktuelle Uhrzeit
`LOCAL_TIME(<string>)`	Uhrzeit aus einer Zeichenkette (ISO-8601-Format)

Funktion	Ergebnis
`LOCAL_TIME({hour} | {hour, minute} | {hour, minute, second} | {hour, minute, second, millisecond|microsecond|nanosecond})`	Uhrzeit aus einzelnen Komponenten
`CURRENT_TIMESTAMP, LOCAL_DATETIME()`	Aktuelles Datum und Uhrzeit
`LOCAL_DATETIME(<string>)`	Datum und Uhrzeit aus einer Zeichenkette (ISO-8601-Format)
`LOCAL_DATETIME({year, month, day} + beliebige gültige Uhrzeit-Komponenten)`	Datum und Uhrzeit aus einzelnen Komponenten
`DURATION(<string>)`	Zeitspanne aus einer Zeichenkette (ISO-8601-Format)
`DURATION({year} | {year, month} | {year, month, day} | {hour} | …)`	Zeitspanne aus einzelnen Komponenten
`DURATION_BETWEEN(<value1>, <value2>)`	Zeitspanne zwischen zwei vergleichbaren Datum/Uhrzeit-Werten
`ABS(<duration>)`	Absoluter Wert einer Zeitspanne

HINWEIS

Aggregationsfunktionen wie `COUNT`, `SUM`, `AVG` usw. sind in `RETURN`-Ausdrücken verfügbar und in Rückgabe von Werten beschrieben.

i-views-spezifische Funktionen (`IV_COMPARE`, `IV_VALUE`, `IV_ACCESS_PARAMETER`) sind in i-views-spezifische Spracherweiterungen beschrieben.

1.3.7.6.8. Ausdrücke mit Null-Werten

Das Ergebnis von Funktionen oder Operatoren ist in fast allen Fällen `NULL`, wenn eines der Argumente `NULL` ist. Bei der Addition `p.netWeight + p.packagingWeight` zum Beispiel ist das Ergebnis `NULL`, wenn der Wert `netWeight` gleich `NULL` ist, unabhängig vom Wert von `packagingWeight`.

Wenn ein Wert als Wahrheitswert ausgewertet wird, entspricht `NULL` dem Wert `FALSE`. Bei Boolschen Operatoren bleibt `NULL` erhalten, `NULL AND TRUE` zum Beispiel ergibt `NULL` und wird als `FALSE` ausgewertet.

HINWEIS

Wenn Sie prüfen möchten, ob ein Wert `NULL` ist, dürfen Sie nicht den Vergleichsoperator verwenden, da das Ergebnis von `NULL = NULL` der Wert `NULL` ist.

Verwenden Sie stattdessen den `IS`-Operator: `v IS NULL` oder `v IS NOT NULL`.

Für den Umgang mit `NULL`-Werten stehen zudem zwei spezielle Funktionen zur Verfügung:

COALESCE

`COALESCE(<value1>, <value2>, …)`

Gibt den ersten Wert aus der Argumentliste zurück, der nicht `NULL` ist. Sind alle Argumente `NULL`, ist das Ergebnis `NULL`. Nützlich, um Fallback-Werte für fehlende Eigenschaften anzugeben.

```
MATCH (p:product)
RETURN p.name, COALESCE(p.salePrice, p.price) AS effectivePrice
```

NULLIF

`NULLIF(<value1>, <value2>)`

Gibt `NULL` zurück, wenn beide Argumente gleich sind, andernfalls den ersten Wert. Nützlich, um bestimmte Werte als fehlend zu behandeln.

```
MATCH (p:product)
RETURN p.name, NULLIF(p.status, 'discontinued') AS activeStatus
```

1.3.7.7. Rückgabe von Werten

Bei `RETURN` geben Sie eine Liste von zurückzugebenden Werten an. Jeder Wert wird durch einen Ausdruck bestimmt.

1.3.7.7.1. Rückgabe von Graph-Elementen

Bei Rückgabe von Pfad-Variable werden die referenzierten Graph-Elemente (Knoten, Kanten, Pfade) in JSON-artiger Syntax zurückgegeben.

Knoten

Objekt mit Eigenschaft `_id` mit der ID des semantischen Elements

Kante

Objekt mit den Eigenschaften `_from` mit der ID der Quelle und `_to` mit der ID des Ziels der Relation

HINWEIS	Die ID der Relation wird nicht aufgeführt, da die ID in einigen Relations-Indizes nicht enthalten ist und es Mehraufwand bedeuten würde, die ID zu ermitteln.

Pfad

Objekt mit den Eigenschaften `_nodes` (Knoten des Pfads) und `_edges` (Kanten des Pfads)

1.3.7.7.2. Benennung der Felder

Pro Ausdruck wird ein Feld zum Ergebnis hinzugefügt. Der Name des Felds ist standardmäßig der Ausdruck selbst. Alternativ können Sie durch `AS` einen anderen Namen bestimmen:

```
MATCH (c:component) RETURN c.id AS "Component ID"
```

Der Name muss in doppelte Anführungszeichen (") oder Akzentzeichen (`) eingeschlossen werden, wenn er nicht ausschließlich aus Buchstaben, Ziffern oder Unterstrich (_) besteht.

Feldnamen müssen eindeutig sein, bei Duplikaten wird ein Fehler gemeldet.

1.3.7.7.3. Rückgabe aller Variablen

Mit `RETURN *` werden alle Variablen zurückgegeben, die im `MATCH`-Teil der Abfrage referenziert wurden.

```
MATCH (m:manufacturer)-[:produces]->(p:product)
RETURN *
```

m	p
{ _id: manufacturer-1 }	{ _id: product-1 }
{ _id: manufacturer-1 }	{ _id: product-2 }

1.3.7.7.4. Redundante Ergebnisse

Wenn nicht alle Knoten des Musters zurückgegeben werden, oder die zurückgegebenen Eigenschaften von verschiedenen Knoten gleich sind, enthält das Ergebnis eventuelle Duplikate.

```
MATCH
  (c:component)-[:partOf]->(:product)<-[:produces]-(m:manufacturer)
RETURN
  c.name as component, m.name as company
```

component	company
Compressor	ACME
Compressor	ACME
Turbine	ACME
Turbine	ACME

Durch die Verwendung von `RETURN DISTINCT` statt `RETURN` können redundante Ergebnisse entfernt werden.

```
MATCH
  (c:component)-[:partOf]->(:product)<-[:produces]-(m:manufacturer)
RETURN DISTINCT
  c.name as component, m.name as company
```

component	company
Compressor	ACME
Turbine	ACME

1.3.7.7.5. Aggregation

Aggregationsfunktionen fassen mehrere Zeilen des Zwischenergebnisses zu einem einzigen Wert zusammen. Sie können in `RETURN`-Ausdrücken verwendet werden.

Funktion	Ergebnis
`COUNT(<variable>)`	Anzahl der Zeilen, in denen die Variable nicht `NULL` ist
`COLLECT_LIST(<ausdruck>)`	Liste aller Werte des Ausdrucks über alle Zeilen
`SUM(<ausdruck>)`	Summe aller numerischen Werte des Ausdrucks über alle Zeilen
`AVG(<ausdruck>)`	Arithmetisches Mittel aller numerischen Werte des Ausdrucks über alle Zeilen
`MIN(<ausdruck>)`	Kleinster Wert des Ausdrucks über alle Zeilen
`MAX(<ausdruck>)`	Größter Wert des Ausdrucks über alle Zeilen
`STDDEV_SAMP(<ausdruck>)`	Stichproben-Standardabweichung der numerischen Werte des Ausdrucks über alle Zeilen
`STDDEV_POP(<ausdruck>)`	Populations-Standardabweichung der numerischen Werte des Ausdrucks über alle Zeilen

Gruppierung mit GROUP BY

Ohne `GROUP BY` beziehen sich Aggregationsfunktionen auf alle Zeilen des Zwischenergebnisses und liefern genau eine Ergebniszeile.

Mit `GROUP BY` werden die Zeilen anhand der angegebenen Ausdrücke gruppiert. Aggregationsfunktionen werden dann pro Gruppe ausgewertet und liefern je eine Ergebniszeile pro Gruppe. `GROUP BY` wird nach `RETURN` angegeben.

```
MATCH (c:company)-[:produces]->(p:product)
RETURN c.name AS company, COUNT(p) AS productCount
GROUP BY c
```

company	productCount
ACME	3
Mustermann GmbH	1

Alle Ausdrücke in `RETURN`, die keine Aggregationsfunktionen sind, müssen in `GROUP BY` aufgeführt sein oder müssen sich aus den gruppierten Variablen ableiten lassen.

HINWEIS

Da Aggregation nur in `RETURN`-Ausdrücken möglich ist, können Sie auf aggregierte Werte nicht direkt in einer `WHERE`-Bedingung filtern. Dazu muss ein nachgeschaltetes `NEXT` mit `FILTER` verwendet werden (siehe Kombination von Suchen).

1.3.7.7.6. Sortierung mit ORDER BY

Die Ergebniszeilen können mit `ORDER BY` sortiert werden. `ORDER BY` wird nach `RETURN` angegeben und enthält eine kommagetrennte Liste von Ausdrücken, nach denen sortiert wird.

```
MATCH (p:product)
RETURN p.name, p.weight
ORDER BY p.weight
```

Es können auch Eigenschaften angegeben werden, die nicht Teil von `RETURN` sind.

```
MATCH (m:manufacturer)-[:produces]->(p:product)
RETURN m.name AS manufacturer, p.name AS product
ORDER BY m.name, p.name
```

Sortierrichtung

Standardmäßig wird aufsteigend sortiert. Die Richtung kann explizit mit `ASC` bzw. `ASCENDING` (aufsteigend) oder `DESC` bzw. `DESCENDING` (absteigend) angegeben werden. Die Richtung gilt jeweils für den vorangehenden Ausdruck.

```
MATCH (p:product)
RETURN p.name, p.price, p.weight
ORDER BY p.price DESCENDING, p.name ASCENDING
```

NULL-Werte beim Sortieren

`NULL`-Werte werden beim Sortieren standardmäßig als kleiner als alle anderen Werte behandelt: bei aufsteigender Sortierung erscheinen sie am Anfang, bei absteigender Sortierung am Ende.

Dieses Verhalten kann mit `NULLS FIRST` oder `NULLS LAST` gesteuert werden.

```
MATCH (p:product)
RETURN p.name, p.weight
ORDER BY p.weight ASCENDING NULLS LAST
```

Seitenweise Ausgabe mit LIMIT und OFFSET

Mit `LIMIT` kann die Anzahl der zurückgegebenen Zeilen beschränkt werden. Mit `OFFSET` kann eine Anzahl von Zeilen am Anfang des Ergebnisses übersprungen werden. Beide Angaben sind optional und werden nach `ORDER BY` angegeben, wobei `OFFSET` vor `LIMIT` stehen muss.

```
MATCH (p:product)
RETURN p.name, p.price
ORDER BY p.price
OFFSET 20 LIMIT 10
```

Dieses Beispiel gibt die Zeilen 21 bis 30 zurück und eignet sich damit zur seitenweisen Ausgabe von Ergebnissen.

HINWEIS	`LIMIT` und `OFFSET` können auch ohne `ORDER BY` verwendet werden. Die Reihenfolge der Ergebnisse ist dann aber nicht definiert, was für seitenweise Ausgaben in der Regel ungeeignet ist.

1.3.7.7.7. Suchen ohne Rückgabe

Eine Query muss mit RETURN beendet werden. Bei rein schreibenden Queries, bei denen Sie keine Rückgabe benötigen, können Sie statt `RETURN` auch `FINISH` ohne weitere Argumente verwenden.

1.3.7.8. Kombination von Suchen

In der einfachsten Form bestehen GQL-Suchen nur aus

```
MATCH ...
WHERE ...
RETURN ...
```

Für komplexere Suchen können mehrere MATCH-Bedingungen angegeben oder mehrere Ergebnis-Tabellen kombiniert werden.

1.3.7.8.1. Lineare Suchen

Eine lineare Suche besteht aus einer Reihe von Match-Bedingungen (`MATCH … WHERE…` , `FILTER`), gefolgt von der Rückgabe mit `RETURN` und optionaler Sortierung mit `ORDER`.

Zusätzliche Bedingungen mit FILTER

Bedingungen, die Sie im `WHERE`-Teil angeben, können alternativ auch in einer `FILTER`-Anweisung angegeben werden. Im Gegensatz zu `WHERE` sind mehrere `FILTER`-Anweisungen möglich. Bei komplexeren Bedingungen kann eine Aufteilung auf mehrere `FILTER` die Übersichtlichkeit erhöhen.

HINWEIS	Die Performance ist bei einem einzelnen `WHERE` meistens besser, da mehrere `FILTER`-Anweisungen derzeit nicht optimiert, sondern linear hintereinander ausgeführt werden.

```
MATCH (m:manual)-[:describes]->(p:product)
FILTER p.weight > 1000
FILTER m.language = 'en'
RETURN m.title,p.id
```

Diese Suche ist äquivalent zu

```
MATCH (m:manual)-[:describes]->(p:product)
WHERE p.weight > 1000 AND m.language = 'en'
RETURN m.title,p.id
```

Kombination mit MATCH

Bei der linearen Suche können mehrere `MATCH`-Teilsuchen mit jeweils einer optionalen `WHERE` -Bedingung formuliert werden:

```
MATCH (p:product)
WHERE p.weight > 1000
MATCH (m:manual)-[:describes]->(p)
WHERE m.language = 'en'
RETURN m.title,p.id
```

Diese Suche ist äquivalent zu

```
MATCH (m:manual)-[:describes]->(p:product)
WHERE p.weight > 1000 AND m.language = 'en'
RETURN m.title,p.id
```

Bei komplexeren Suchen kann eine Aufteilung auf mehrere Teilsuchen die Übersichtlichkeit erhöhen.

Bei mehreren Teilsuchen wird bei jeder Teilsuche außer der ersten das Ergebnis mit dem Ergebnis der vorherigen Teilsuche zusammengefasst. Die optional nachfolgende `WHERE`-Bedingung wird auf das zusammengefasste Ergebnis angewendet. Man kann sich deshalb auch auf Variablen der vorherigen Teilsuchen beziehen.

Die Ergebnisse werden anhand der gemeinsamen Felder der Teilsuchen zusammengefasst. Im Beispiel oben ist das die Variable `p`. Falls es keine gemeinsamen Felder gibt, wird ein kartesisches Produkt gebildet.

HINWEIS

Die Performance ist bei einer zusammengefassten Suche meistens besser als bei einer auf mehrere `MATCH`-Bedingungen aufgeteilten Suche.

Mehrere `MATCH`-Bedingungen ohne gemeinsame Felder sollten unbedingt vermieden werden, da ein kartesisches Produkt einen hohen Speicher- und Zeitbedarf hat.

OPTIONAL MATCH

Eine besondere Form von MATCH ist `OPTIONAL MATCH`. Damit können optionale Eigenschaften zum Ergebnis hinzugefügt werden.

Falls die Suchbedingungen auf keine Elemente des Graphen zutreffen, ist das Ergebnis der Teilsuche eine Tabelle mit einer Zeile mit `NULL`-Werten, statt einer leeren Tabelle. Bei der Zusammenfassung mit der vorherigen Teilsuche führt das dazu, dass die Ergebnisse der vorherigen Teilsuche erhalten bleiben.

HINWEIS

Es werden nur Eigenschaften hinzugefügt, keine neuen Zeilen. Zur Vereinigung von Teilsuchen dient UNION

Beispiel:

```
MATCH (p:product)
WHERE p.weight > 10000
OPTIONAL MATCH (p:product)
WHERE p.tag = 'heavy'
RETURN p.name as product, p.weight as weight, p.tag as tag
```

Das Ergebnis enthält auch Zeilen, in denen Tag-Feld leer ist:

product	weight	tag
Anvil A1	17200	heavy
PowerMole 3000	27150	

Ohne `OPTIONAL` enthält das Ergebnis nur Zeilen, in denen das Feld `tag` den Wert `heavy` hat:

```
MATCH (p:product)
WHERE p.weight > 10000
MATCH (p:product)
WHERE p.tag = 'heavy'
RETURN p.name as product, p.weight as weight, p.tag as tag
```

product	weight	tag
Anvil A1	17200	heavy

1.3.7.8.2. Zusammengesetzte Suchen

Eine lineare Suche liefert als Ergebnis eine Tabelle mit den in `RETURN` definierten Feldern. Die Ergebnisse von zwei linearen Suchen können mit verschiedenen Operationen zu einer Tabelle kombiniert werden.

Der grundlegende Aufbau einer zusammengesetzten Suche ist `<Lineare Suche 1> <OPERATOR> <Lineare Suche 2>`. Die Operatoren verknüpfen immer genau zwei Tabellen. Wenn Sie mehrere Operatoren angeben, werden diese einzeln der Reihe nach ausgeführt.

Mengen-Operatoren

Bei Mengen-Operatoren werden die einzelnen Zeilen der beiden Tabellen mit Mengen-Operatoren verknüpft.

Die Tabellen müssen dazu die selben Felder haben. Die Reihenfolge der Felder ist nicht relevant. Falls sich die Felder unterscheiden, führt die Suche zu einer Fehlermeldung.

Duplikate können mit dem Schlüsselwort `DISTINCT` hinter dem Operator entfernt werden (siehe Beispiel bei UNION)

UNION

Bei `UNION` werden die Zeilen beider Tabellen zur Ergebnistabelle hinzugefügt.

```
MATCH (p:product)
WHERE p.weight > 10000
RETURN p.name as product, p.weight as weight, p.tag as tag
UNION
MATCH (p:product)
WHERE p.tag = 'heavy'
RETURN p.name as product, p.weight as weight, p.tag as tag
```

product	weight	tag
Anvil A1	17200	heavy
Anvil A1	17200	heavy
PowerMole 3000	27150	

Durch die Vereinigung enthält das Ergebnis Duplikate. Diese können durch `UNION DISTINCT` entfernt werden.

```
MATCH (p:product)
WHERE p.weight > 10000
RETURN p.name as product, p.weight as weight, p.tag as tag
UNION DISTINCT
MATCH (p:product)
WHERE p.tag = 'heavy'
RETURN p.name as product, p.weight as weight, p.tag as tag
```

product	weight	tag
Anvil A1	17200	heavy
PowerMole 3000	27150	

INTERSECT

Bei `INTERSECT` werden Zeilen zur Ergebnistabelle hinzugefügt, die in beiden Ergebnistabellen vorhanden sind.

EXCEPT

Bei `EXCEPT` werden die Zeilen aus der ersten Tabelle hinzugefügt, die nicht in der zweiten Tabelle enthalten sind.

HINWEIS	Durch `INTERSECT` und `EXCEPT` entstehen keine Duplikate, aber `DISTINCT` kann trotzdem verwendet werden, zum Beispiel, falls die erste Tabelle Duplikate enthält

1.3.7.8.3. OTHERWISE

`<Lineare Suche 1> OTHERWISE <Lineare Suche 2>` gibt das Ergebnis der ersten Suche zurück, sofern es nicht leer ist. Die zweite Suche wird nicht ausgeführt.

Bei einem leeren Ergebnis wird dagegen das Ergebnis der zweiten Suche zurückgegeben.

Im Gegensatz zu `UNION`, `INTERSECT` und `EXCEPT` müssen die beiden Tabellen nicht diesselben Felder haben.

```
MATCH (p:product)-[:producedBy]->(m:company)
WHERE m.name = $manufacturer OR m.id = $manufacturer
RETURN p.name as product, m.id as manufacturer-id, p.producer as producer-
tag
OTHERWISE
MATCH (p:product)
WHERE p.producer = $manufacturer
RETURN p.name as product, p.producer as producer-tag
```

Suche mit `$manufacturer` = "M173621"

product	manufacturer-id	producer-tag
Anvil A1	M173621	heavy-industries
PowerMole 3000	M173621	heavy-corp

Suche mit `$manufacturer` = "heavy-corp"

product	producer-tag
PowerMole 3000	heavy-corp

1.3.7.8.4. NEXT

Mehrere lineare Suchen können mit `MATCH` hintereinander geschaltet werden. Die Teilergebnistabellen werden durch ein kartesisches Produkt verknüpft. Sie können sich auf Variablen

der vorhergehenden Teilsuchen beziehen.

```
MATCH (p:product) WHERE p.rating > 4
RETURN p
NEXT
MATCH (p) where p.price < 1000
RETURN p.name, p.rating, p.price
```

Es ist auch möglich, auf aggregierte Werte zuzugreifen, was im `WHERE`-Teil nicht möglich ist.

```
MATCH (c:company)-[:produces]->(p:product)
RETURN c.name as company, COUNT(p) as products
GROUP BY c
NEXT
FILTER products >= 2
RETURN company, products
```

1.3.7.8.5. CALL

Mit `CALL` werden aus einer GQL-Abfrage heraus andere Abfragen aufgerufen. GQL unterscheided zwischen eingebetteten und benannten Prozeduren. Prozeduren werden zeilenweise basierend auf dem aktuellen Zwischenergebnis ausgewertet. Liefert die aufgerufene Prozedur mehr als ein Ergebnis, wird für jede Ergebniszeile eine Zeile in die Ergebnis-Tabelle des Aufrufers eingefügt. Im Umkehrschluss bedeutet dies auch, dass ein leeres Ergebnis der aufgerufenen Prozedur die zugrundeliegende Zeile in der Ergebnis-Tabelle des Aufrufers entfernt.

HINWEIS	Alle Prozeduren werden zeilenweise basierend auf dem aktuellen Zwischenergebnis ausgewertet. Dies kann erhebliche Performanz-Einbußen zur Folge haben. Häufig ist es daher besser, auf einen Prozedur-Aufruf zu verzichten und die Anweisungen in ein vorhergehendes `MATCH` oder `FILTER` zu integrieren.

Eingebettete Prozeduren

Eingebettete Prozeduren sind gekapselte GQL-Abfragen, die direkt in einer anderen Abfrage integriert sind. Dies ermöglicht eine logische Strukturierung der Abfrage. Dabei können Sie steuern, welche Variablen der eingebetteten Prozedur übergeben werden und welche Rückgabewerte in die Ergebnistabelle der Gesamtabfrage einfließen. Variablen, die einer Prozedur nicht übergeben werden, sind für diese nicht sichtbar und können innerhalb der Prozedur auch neu definiert werden.

Das folgende Beispiel nutzt eine eingebettete Prozedur zum Bestimmen des Besitzers jedes Unternehmens `c` aus dem vorherigen `MATCH`. Die Variable `p` wird der Prozedur nicht

übergeben und kann daher innerhalb der Prozedur für Personen neu vergeben werden. Bei der Rückgabe der Personen aus der Prozedur muss ein Alias vergeben werden, da sich `p` im äußeren Kontext auf Produkte bezieht.

```
MATCH (c:company)-[:produces]->(p:product)
CALL (c) {
  MATCH (c)-[:ownedBy]->(p:person)
  RETURN p AS owner
}
RETURN c AS company, owner, COLLECT_LIST(p) AS products
GROUP BY c
```

Die Liste der übergebenen Variablen (`(c)`) ist optional. Wenn sie nicht angegeben ist, werden implizit alle Variablen des äußeren Kontexts übergeben. Eine leere Variablenliste `()` stellt der Prozedur hingegen keine existierende Variable zur Verfügung.

HINWEIS

Die GQL-Spezifikation verbietet eingebetteten Prozeduren die Rückgabe von Variablen, die bereits im äußeren Kontext existieren. In obigem Beispiel wäre daher weder `RETURN p` noch `RETURN *` erlaubt, da die Variable `p` im äußeren Kontext für Produkte verwendet wird.

Aufruf externer Abfragen

Über den Aufruf benannter Prozeduren können andere im Knowledge-Graph gespeicherte Abfragen in eine GQL-Abfrage integriert werden. Dabei kann es sich sowohl um weitere GQL-Abfragen als auch andere Suchen, wie zum Beispiel Strukturabfragen oder Volltextsuchen, handeln. Auch für benannte Prozeduren wird das Schlüsselwort `CALL` genutzt. Dafür wird der Registrierungsschlüssel der aufzurufenden Abfrage benötigt. Mit dem Schlüsselwort `YIELD` legen Sie fest, welche Spalten der aufgerufenen Prozedur in der weiteren Abfrage verfügbar sein sollen.

Angenommen die Abfrage aus dem Beispiel Eingebettete Prozeduren ist unter dem Registrierungsschlüssel `productsByCompany` registriert, kann sie über folgenden `CALL`-Aufruf in eine neue Abfrage integriert werden.

```
CALL productsByCompany()
YIELD company, products
RETURN company, SIZE(products) AS productCount
```

HINWEIS

Die GQL-Spezifikation geht davon aus, dass die Typen und Namen der Felder, die eine aufgerufene Prozedur zurückgibt, statisch bekannt sind. Die Syntax erlaubt daher auch Aufrufe ohne `YIELD`, wobei implizit alle Felder der

> aufgerufenen Prozedur übernommen werden. Dies ist in i-views aktuell nicht möglich, da die Rückgabe einer aufgerufenen GQL-Prozedur erst zur Laufzeit bekannt ist.

Parametrisierung

Definiert die aufgerufene Prozedur Parameter, können diese in Form eines Records übergeben werden. Innerhalb des Parameter-Records muss jeder Schlüssel dem Namen eines Parameters entsprechen. Die folgenden Beispiele nutzen `FOR … IN` zur Iteration über eine Liste von Werten — siehe Iteration mit FOR.

Abfrage mit dem Registrierungsschlüssel `productsByCompanyWithName` und Parameter `$companyName`:

```
MATCH (c:company)-[:produces]->(p:product)
WHERE c.name = $companyName
RETURN p AS product
```

Aufruf als benannte Prozedur mit Parameter `$companyName`:

```
FOR name IN ['ACME', 'Mustermann GmbH']
CALL productsByCompanyWithName({companyName: name})
YIELD product
RETURN product, name AS `manufactured by`
```

Das Ergebnis ist eine Tabelle von Produkten der beiden Unternehmen.

Definiert die aufgerufene Prozedur Parameter, die nicht explizit beim Aufruf im Parameter-Record übergeben werden, wird ersatzweise der gleichnamige Parameterwert der aufrufenden GQL-Abfrage genutzt. Ist der Parameter für die aufrufende Abfrage nicht gesetzt, kommt es zu einem Laufzeitfehler.

Die aufgerufene Abfrage `productsByCompanyWithName` definiert den Parameter `$companyName`. Da beim `CALL` keine Parameter gesetzt werden, wird der Wert aus dem Parameter `$companyName` des Aufrufers übernommen.

```
CALL productsByCompanyWithName()
YIELD product
RETURN product, $companyName AS manufacturedBy
```

Aufruf von Nicht-GQL-Abfragen

Handelt es sich bei der aufgerufenen Prozedur nicht um eine GQL-Abfrage, sondern zum Beipiel um eine Strukturabfrage, beinhaltet die zurückgegebene Tabelle genau eine Spalte mit dem Namen `_result`. Diese Spalte enthält die Ergebnis-Objekte, die in GQL den Typ `NODE` haben.

HINWEIS	Der Aufruf von Suchen, die Treffer produzieren, die nicht auf Knowledge-Graph-Objekten basieren, ist derzeit nicht möglich. In diesem Fall liefert der Aufruf eine leere Ergebnistabelle.

Parametrisierung funktioniert identisch zum Aufruf von GQL-Abfragen mithilfe eines Parameter-Records. Manche Abfragetypen, zum Beispiel Volltextsuchen, erfordern zwingend einen Parameter mit dem Namen `searchString`, der die zu suchende Zeichenkette enthält.

Im folgenden Beispiel wird eine Volltextsuche nach Zeitungsartikeln durchgeführt, die einen der beiden Unternehmensnamen enthalten. Die Rückgabe ist eine Tabelle mit Name des Unternehmens und Name und Datum jedes gefundenen Zeitungsartikels.

```
FOR name IN ['ACME', 'Mustermann GmbH']
CALL pressReleaseFulltextSearch({searchString: name})
YIELD _result AS pressRelease
RETURN name, pressRelease.name, pressRelease.releaseDate
```

OPTIONAL CALL

Liefert eine aufgerufene Prozedur eine leere Ergebnis-Tabelle wird standardmäßig die gesamte Zeile, für die die Prozedur aufgerufen wird, aus dem Gesamtergebnis entfernt. Ist dies nicht gewünscht, können Sie `CALL` mit dem Zusatz `OPTIONAL` aufrufen. Alle Felder des `YIELD`-Ausdrucks werden dann mit `NULL` belegt, wenn die aufgerufene Prozedur für die Zeile ein leeres Ergebnis liefert.

Wenn "ACME" in keinem Zeitungsartikel erwähnt wird, enthält die Ergebnistabelle des folgenden GQL-Programms trotzdem eine Zeile mit `name` "ACME", in der `release` und `date` mit `NULL` belegt sind.

```
FOR name IN ['ACME', 'Mustermann GmbH']
OPTIONAL CALL pressReleaseFulltextSearch({searchString: name})
YIELD _result AS pressRelease
RETURN name, pressRelease.name AS release, pressRelease.releaseDate AS
date
```

1.3.7.9. Weitere Anweisungen

Dieser Abschnitt beschreibt Anweisungen, die keinem der anderen Kapitel zugeordnet sind: Variablenzuweisung mit LET und VALUE sowie Iteration mit FOR.

1.3.7.9.1. Variablenzuweisung mit LET und VALUE

LET

Mit LET können Ausdrücke an Variablen gebunden werden, um sie in der weiteren Abfrage mehrfach verwenden zu können. Mehrere Zuweisungen können mit Komma verkettet werden.

```
MATCH (p:product)
LET discounted = p.price * 0.9, label = UPPER(p.name)
WHERE discounted < 100
RETURN label, p.price, discounted
ORDER BY discounted
```

Soll die Bindung nur auf einen einzelnen Ausdruck beschränkt sein, kann die LET … IN … END -Form verwendet werden. Die Variablen sind dann ausschließlich im Ausdruck zwischen IN und END sichtbar.

```
MATCH (p:product)
RETURN p.name,
  LET discounted = p.price * 0.9 IN
    CASE WHEN discounted < 100 THEN discounted ELSE p.price END
  END
```

VALUE

VALUE ist eine alternative Form der Variablenzuweisung, die ausschließlich am Anfang eines Prozedur-Rumpfes verwendet werden kann. Im Gegensatz zu LET kann pro VALUE-Anweisung nur eine Variable definiert werden; es sind aber mehrere aufeinanderfolgende VALUE-Anweisungen erlaubt.

```
VALUE minRating = 4
VALUE category = 'electronics'
MATCH (p:product)
WHERE p.rating >= minRating AND p.category = category
RETURN p.name, p.rating
```

1.3.7.9.2. Iteration mit FOR

Mit `FOR` kann über die Elemente einer Liste iteriert werden. Für jedes Element der Liste wird eine Zeile in das Zwischenergebnis eingefügt.

```
FOR name IN ['ACME', 'Mustermann GmbH']
RETURN name
```

name
ACME
Mustermann GmbH

`FOR` kann mit nachfolgenden Anweisungen kombiniert werden, die sich auf die Iterationsvariable beziehen.

```
FOR name IN ['ACME', 'Mustermann GmbH']
MATCH (c:company)
WHERE c.name = name
RETURN c.name, c.foundingYear
```

Die Liste kann auch eine berechnete Größe sein, zum Beispiel das Ergebnis von `COLLECT_LIST` in einer vorangehenden Teilsuche:

```
MATCH (c:company)-[:produces]->(p:product)
RETURN COLLECT_LIST(p.name) AS productNames
GROUP BY c
NEXT
FOR name IN productNames
RETURN name
```

1.3.7.10. Suchparameter

GQL-Suchen können Parameter definieren, deren Werte zur Laufzeit übergeben werden. Ein Parameter wird im Abfragetext durch ein vorangestelltes `$` gekennzeichnet.

```
MATCH (p:product)
WHERE p.rating > $minRating AND p.category = $category
RETURN p.name, p.rating
```

Parameter können überall dort verwendet werden, wo ein literaler Wert stehen kann: in `WHERE`

-Bedingungen, `RETURN`-Ausdrücken, `ORDER BY` und als Argumente von Funktionen.

HINWEIS	Wird ein Parameter in einer Abfrage verwendet, aber zur Laufzeit kein Wert dafür übergeben, führt dies zu einem Fehler. Es gibt keine implizite Deaktivierung von Bedingungen bei fehlenden Parametern.

1.3.7.10.1. Übergabe von Parameterwerten

Wie Parameter übergeben werden, hängt von der Ausführungsumgebung ab:

REST-API

Parameter werden als Request-Parameter (GET, POST mit Text-Body) oder im JSON-Objekt `queryParameters` (POST mit Form-Daten) übergeben. Siehe GQL REST API.

JavaScript-API

Parameter werden als Objekt an `perform()` übergeben, wobei die Eigenschaftsnamen den Parameternamen entsprechen. Siehe GQL JavaScript API.

Workbench

Parameter können direkt in der Workbench eingegeben werden.

1.3.7.10.2. Typen von Parameterwerten

Parameterwerte werden als Zeichenketten übergeben und implizit in den für den Kontext passenden Typ konvertiert. Bei der JavaScript-API können Werte auch als native JavaScript-Typen übergeben werden.

HINWEIS	Listen als Parameterwerte sind nicht transparent: wenn ein Array-Wert übergeben wird, muss er in der Abfrage explizit als Liste ausgewertet werden, zum Beispiel mit `FOR … IN $myListParam`.

1.3.7.11. i-views-spezifische Spracherweiterungen

1.3.7.11.1. Abbildung von i-views-Attributtypen auf GQL-Wertetypen

Nicht alle i-views-Attributtypen haben eine direkte Entsprechung im GQL-Standard. Die folgende Tabelle zeigt, wie i-views-Attributtypen auf GQL-Wertetypen abgebildet werden.

i-views-Attributtyp	GQL-Wertetyp	Hinweis
Zeichenkette	String	Mehrsprachige Werte: siehe [gql-iviews-specifics-multilingual-attributes]
Ganzzahl	Integer	
Fließkommazahl	Float	Nur doppelte Genauigkeit
Boolesch	`TRUE` / `FALSE`	

i-views-Attributtyp	GQL-Wertetyp	Hinweis
Datum	`DATE`	
Zeit	`TIME`	
Datum und Uhrzeit	`DATETIME` / `TIMESTAMP`	
Flexible Zeit	`DATE`, `TIME` oder `DATETIME`	Je nach gespeichertem Wert
Internet-Verknüpfung	String	Mehrsprachige Werte: siehe [gql-iviews-specifics-multilingual-attributes]
Auswahl	String	
Datei	String	Dateiname wird als Zeichenkette ausgegeben
Intervall		Nur als Zeichenkette ausgebbar
Geometrie / Geographische Position		Nur als Zeichenkette ausgebbar (WKT-Format)
Farbwert		Nur als Zeichenkette ausgebbar

1.3.7.11.2. Erweiterte Wertvergleiche

i-views bietet mehr Vergleichsoperatoren an, als sich im GQL-Standard wiederfinden. So erlaubt i-views zum Beispiel Vergleiche mit regulären Ausdrücken, Platzhaltern und ignorierter Groß-/Kleinschreibung, sowie Vergleichsoperatoren für spezielle Wertetypen wie Intervalle oder Geometrien.

Mithilfe der Funktion `IV_COMPARE` ist der Zugriff auf alle nativen Operatoren von i-views möglich. Die Funktion erwartet zwei zu vergleichende Argumente, sowie den Namen eines Vergleichsoperators als Zeichenkette. `IV_COMPARE` liefert entweder `TRUE` oder `FALSE` als Rückgabe. Die Funktion kann auch direkt in Kombination mit einer `WHERE`-Klausel ohne zusätzlichen Vergleich mit `TRUE` oder `FALSE` genutzt werden.

```
MATCH (c:city)
WHERE IV_COMPARE(c.name, '*b*', 'equal')
RETURN c.name
```

c.name
Berlin
Hamburg
…

```
MATCH (c:city)
WHERE IV_COMPARE(c.name, '(Bad )?H(a|o)mburg', 'equalRegex')
RETURN c.name
```

c.name
Hamburg
Bad Homburg
…

Die vollständige Liste der Operatoren mit ihren jeweiligen Bezeichnern entnehmen Sie dem technischen Handbuch.

Optionale zusätzliche Parameter können mithilfe eines Records als viertes Argument übergeben werden. Die möglichen Parameter hängen vom jeweiligen Operator ab.

Operator	Parameter	Standardwert	Beschreibung
Operatoren für Zeichenkettenvergleiche	`allowWildcards`	`TRUE`	`FALSE` deaktiviert `*` und `?` als Platzhalter für beliebige Zeichen
	`ignoreCase`	`TRUE`	`FALSE` aktiviert die Berücksichtigung von Groß-/Kleinschreibung
Operatoren für Abstandsvergleiche	`distanceValue`	`0`	Maximale Distanz, die die beiden Werte zueinander haben dürfen

In diesem Beispiel werden Städte gefunden, deren geodesische Distanz zu Darmstadt maximal 100 Kilometer beträgt. Für den Wertetyp Geometrie gibt es in GQL aktuell keine Entsprechung, es wäre mit Standardoperatoren also nicht möglich, überhaupt einen Vergleich mit solchen Werten durchzuführen. `IV_COMPARE` unterstützt aber auch solche Wertetypen und führt eine automatische Konvertierung der Zeichenketten-Repräsentation des Vergleichswerts durch.

```
MATCH (c:city)
WHERE IV_COMPARE(
  c.area,
  'SRID=4326;POINT(8.6512 49.8728)',
  'geodesicDistance',
```

```
  { distanceValue: '100km' }
)
RETURN c.name
```

HINWEIS

Mindestens einer der zu vergleichenden Werte in `IV_COMPARE` muss eine Referenz auf ein Attribut sein. Es ist nicht möglich, zwei literale Werte mit `IV_COMPARE` zu vergleichen.

1.3.7.11.3. Mehrwertige Attribute

Allgemein werden die Attribute von Objekten des Knowledge-Graphs auf Eigenschaften in GQL abgebildet. Dem GQL-Standard folgend hat jede Eigenschaft entweder genau einen materiellen Wert oder sie ist `NULL`. i-views unterstützt hingegen mehrwertige Attribute. Obwohl GQL Listentypen für Eigenschaften unterstützt, werden mehrwertige Attribute bewusst nicht auf Listen abgebildet, da der Umgang mit Listen in GQL vergleichsweise kompliziert ist. Mehrwertige Attribute haben zum Beispiel keine inhärente Sortierung der Attributwerte, ein Vergleich mit einer sortierten Liste wäre also im Allgemeinen nicht möglich. Stattdessen werden mehrwertige Attribute analog zu mehrwertigen Relationen (Kanten) im Abfrageergebnis auf eine dynamische Anzahl Zeilen aufgeteilt.

```
MATCH (p:person)
RETURN p.name, p.nickname, UPPER(p.nickname) AS uppercase
```

p.name	p.nickname	uppercase
"Jonathan"	"Jon"	"JON"
"Jonathan"	"Nathan"	"NATHAN"
"Jonathan"	"Nat"	"NAT"

In jeder Zeile beziehen sich alle Referenzen auf das mehrwertige Attribut auf denselben Wert. Dies bedeutet insbesondere, dass bei einer Referenz auf das Attribut in einer `WHERE`-Klausel auch die Zeilen für nicht passende Attributwerte entfernt werden.

```
MATCH (p:person)
WHERE p.nickname = "Nathan"
RETURN p.name, p.nickname, UPPER(p.nickname) AS uppercase
```

p.name	p.nickname	uppercase
"Jonathan"	"Nathan"	"NATHAN"

1.3.7.11.4. Mehrsprachige Attributwerte

Allgemein werden die Attribute von Objekten des Knowledge-Graphs auf Eigenschaften in GQL abgebildet. Dem GQL-Standard folgend hat jede Eigenschaft entweder genau einen materiellen Wert oder sie ist `NULL`. i-views unterstützt hingegen mehrsprachige Attributwerte für Zeichenketten-, URL- und Dateiattribute. Um möglichst standardkompatibel zu sein, liefert die Abfrage eines mehrsprachigen Attributs den Wert, der am besten zur derzeitigen Sprache der Anwendung passt. Das kann zum Beispiel die Systemsprache sein, im Kontext einer Web-Anwendung aber auch die vom Browser bevorzugte Sprache. Dies gilt sowohl für die Abfrage in `WHERE`-Klauseln als auch für die Rückgabewerte in `RETURN`-Anweisungen.

Die folgende Abfrage findet nur in einem deutschen Anwendungskontext Ergebnisse, da `c.name` standardmäßig den zur Anwendungssprache passenden Wert liefert. Die Rückgabe gibt den Namen in der Anwendungssprache zurück.

```
MATCH (c:city)
WHERE c.name = 'Köln'
RETURN c.name
```

Um gezielt auf unterschiedliche Übersetzungen des Attributwerts zuzugreifen, gibt es zwei i-views-spezifische Spracherweiterungen.

Sprachzugriff per Qualifikator

Ist die gewünschte Sprache statisch bekannt, nutzen Sie einen Qualifikator an der Eigenschaftsreferenz. Dieser wird über eine Raute (`\#`) eingeleitet und von einem ISO-639-1- (`de`), ISO-639-2- (`ger`) oder Locale-Identifikator (`de_DE`) gefolgt. Wird ein Identifikator abgefragt, für den kein übersetzter Attributwert vorliegt, wird `NULL` ausgegeben.

Die folgende Abfrage trifft unabhängig von der Anwendungssprache auf den deutschen Stadtnamen zu. Die Rückgabe gibt den Namen in der Anwendungssprache, auf Deutsch und auf US-Englisch zurück (sofern eine entsprechende Übersetzung vorliegt).

```
MATCH (c:city)
WHERE c.name\#de = 'Köln'
RETURN c.name, c.name\#de, c.name\#en_US
```

Sprachzugriff mit `IV_VALUE`

Die Funktion `IV_VALUE` unterstützt als optionales zweites Argument die Angabe einer Sprache als String. Auch hier wird ein ISO-639-1- (`'de'`), ISO-639-2- (`'ger'`) oder Locale-Identifikator (`'de_DE'`) erwartet. Diese Variante ist etwas weniger kompakt, erlaubt jedoch auch die Verwendung von Variablen oder Eigenschaftsreferenzen zur dynamischen Ausgabe von Übersetzungen.

Die folgende Abfrage trifft unabhängig von der Anwendungssprache auf den deutschen Stadtnamen zu. Die Rückgabe gibt drei Zeilen zurück: den deutschen, den US-englischen und den muttersprachlichen Wert.

```
MATCH (c:city)
WHERE IV_VALUE(c.name, 'de') = 'Köln'
FOR lang IN ['de', 'en_US', city.nativeLanguageID]
RETURN IV_VALUE(c.name, lang)
```

Mithilfe der `IV_VALUE`-Funktion ist es außerdem möglich, den übersetzten Wert von einem Attribut-Knoten abzufragen.

Die folgende Abfrage gibt alle deutschen Werte des Namensattributs zurück.

```
MATCH (nameAttribute:name)
RETURN IV_VALUE(nameAttribute, 'de')
```

1.3.7.11.5. Zugriff auf Metaeigenschaften

Das Graphmodell von GQL sieht Knoten ("Nodes") und Kanten ("Edges") vor, die jeweils Eigenschaften ("Properties") haben können. Kanten verbinden grundsätzlich zwei Knoten. Weder Kanten noch Eigenschaften können selbst Quelle einer Kante oder Eigenschaft sein. Das Graphmodell von i-views erlaubt solche Metaeigenschaften. Alle Elemente des Knowledge-Graphs (Objekte, Relationen, Attribute und Typen) haben Schema, das die Definition von Attributen und Relationen erlaubt.

In dieser GQL-Implementierung können daher auch Attribute und Relationen als Knoten behandelt werden. Sie sind somit genau wie Objekte Elemente "erster Klasse".

Das Namensattribut mit dem internen Namen `name` kann wie jedes andere Objekt als Knoten gefunden werden.

```
MATCH (n:name)
RETURN n
```

Auf diese Weise ist es möglich, auf Metaeigenschaften mithilfe der normalen GQL-Syntax zuzugreifen.

Zugriff auf das Metaattribut `date` der Relation `editedBy`:

```
MATCH (r:editedBy)
RETURN r.date
```

Zugriff auf die Metarelation `signedOffBy` der Relation `editedBy` und das Meta-Metaattribut `date`:

```
MATCH (r1:editedBy)-[r2:signedOffBy]->(p:person)
RETURN p.name, r2.date
```

Im Zusammenspiel mit der restlichen GQL-Syntax ist es manchmal notwendig, eine Kante innerhalb einer Abfrage als Knoten umzudeuten. Dies wird über den `\#node`-Qualifikator ermöglicht:

In diesem Beispiel wird `r` zunächst als Identifikator einer Kante genutzt, deren Ziel Bestandteil der Ausgabe sein soll. Um nun zusätzlich auf eine Metarelation von `r` zuzugreifen, muss `r` mit `r\#node` als Knoten refenziert werden.

```
MATCH
  (doc:document)-[r:editedBy]->(editor:person),
  (r\#node)-[:signedOffBy]->(signer:person)
RETURN
  doc.name, editor.name, signer.name
```

1.3.7.11.6. Zugriff auf Systemrelationen

Systemrelationen können wie normale Relationen als Kanten in der Abfrage verwendet werden. Jede Systemrelation verfügt über ein vordefiniertes Label mit dem Präfix `iv_`, über das sie referenzierbar ist (`-[r:iv_hasTarget]→`).

Systemrelation	Label
Definiert für	`iv_hasDomain`
Definiert für Objekte von	`iv_domainOfInstancesOf`
Definiert für Untertypen von	`iv_domainOfSubtypesOf`
Domäne von	`iv_isDomainOf`
Ergänzt Objekt	`iv_hasSupplementalInstance`
Ergänzt Objekte von	`iv_isSupplementalInstanceOf`
Erweitert Objekt	`iv_extends`
Erweitert Objekte von	`iv_extendsInstancesOf`
Hat Erweiterung	`iv_extendedBy`
Hat Objekt	`iv_hasInstance`
Hat Ziel	`iv_hasTarget`
Inverse Relation	`iv_hasInverseRelation`
Ist Obertyp von	`iv_hasSubType`
Ist Objekt von	`iv_hasType`
Ist Untertyp von	`iv_hasSuperType`
Ist Ziel von	`iv_isTargetOf`
Objekte können erweitert werden durch	`iv_instancesExtendedBy`
Objekte sind Domäne von	`iv_instanceIsDomainOf`
Typen sind Domäne von	`iv_typeIsDomainOf`

1.3.7.11.7. Zugriffsparameter

Je nach Ausführungsumgebung stehen bestimmte Zugriffsparameter wie zum Beispiel der derzeitige Nutzer, das zugegriffene Element oder die Anwendungskonfiguration zur Verfügung. Diese Parameter können mit der Funktion `IV_ACCESS_PARAMETER` abgefragt werden. Die Funktion verlangt eine Zeichenkette als Argument, die den Parameternamen spezifiziert. Da alle Zugriffsparameter Objekte des Knowledge-Graphs referenzieren, ist das Ergebnis der `IV_ACCESS_PARAMETER`-Funktion immer ein Knoten.

Ausgabe aller Zugriffsparameter:

```
FOR accessParam IN
[
  'user', 'accessedObject', 'applicationTopic',
```

```
  'property', 'topic', 'inverseTopic', 'inverseRelation',
'inverseRelationConcept',
  'core', 'primaryProperty', 'primaryTopic', 'primaryCoreTopic',
'inversePrimaryCoreTopic'
]
RETURN
  accessParam, IV_ACCESS_PARAMETER(accessParam)
```

Alle Personen, die vom derzeitigen Nutzer gekannt werden:

```
MATCH (u:person)-[:knows]->(p:person)
WHERE u = IV_ACCESS_PARAMETER("user")
RETURN p.name
```

1.3.7.12. Zu Strukturabfragen äquivalente GQL-Abfragen

Im Folgenden finden Sie einige Strukturabfragen und äquivalente GQL-Abfragen.

HINWEIS	Eine automatische Übersetzung von Strukturabfragen in GQL (oder umgekehrt) ist nicht geplant.

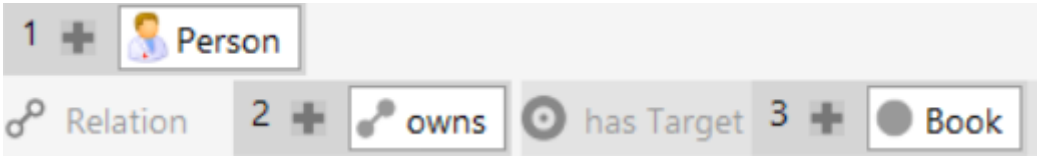
1.3.7.12.1. Eigenschaften

Objekt hat eine Relation

Suche

Personen, die ein Buch besitzen

Strukturabfrage



GQL

```
MATCH (p:person)-[:owns]->(b:book)
RETURN p.name, b.name
```

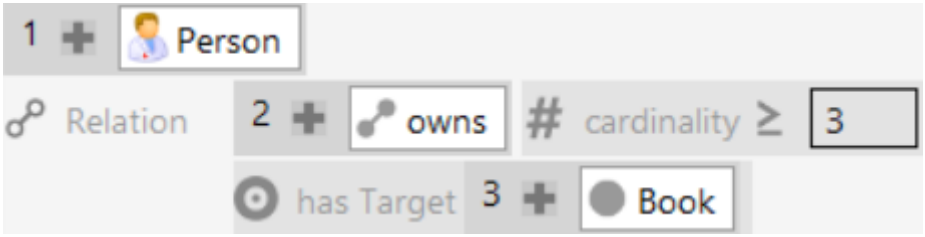
Siehe Kanten

Anzahl Relationen (Kardinalität)

Suche

Personen, die mindestens drei Bücher besitzen

Strukturabfrage



GQL

```
MATCH (p:person)-[o:owns]->(b:book)
RETURN p.name as name, COUNT(o) as owns
GROUP BY p
NEXT
FILTER owns >= 3
RETURN name, owns
```

Die Kardinalität wird durch Aggregation (`GROUP BY`) und Zählen (`COUNT`) realisiert. Da Aggregation nur in `RETURN`-Statements möglich ist, können Sie nicht schon im `WHERE`-Teil zählen. Da Sie im `RETURN` nicht filtern können, wird ein durch `NEXT` verknüpfter `FILTER` nachgeschaltet.

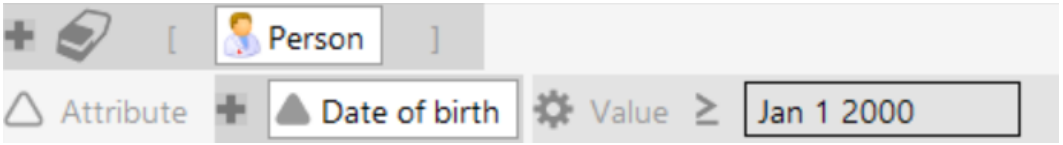
Auf diese Weise erhalten Sie im Ergebnis auch die tatsächliche Anzahl.

Bedingung für Attributwert

Suche

Personen, die ab dem 1.1.2000 geboren wurden

Strukturabfrage



GQL

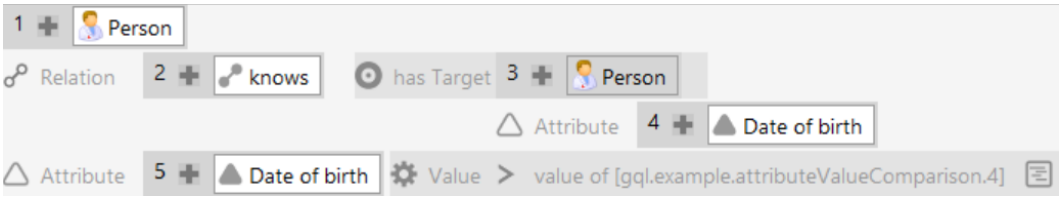
```
MATCH (p:person)
WHERE p.dateOfBirth >= DATE "2000-01-01"
RETURN p.name, p.dateOfBirth
```

Attributwerte vergleichen

Suche

Personen, die eine Person kennen, die älter als sie selbst ist

Strukturabfrage



GQL

```
MATCH (p:person)-[:knows]->(q:person)
WHERE p.dateOfBirth > q.dateOfBirth
RETURN p.name, p.dateOfBirth, q.name, q.dateOfBirth
```

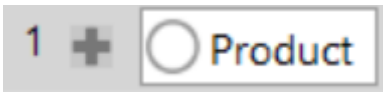
1.3.7.12.2. Typen

Untertypen

Suche

Typ "Produkt" und alle Untertypen

Strukturabfrage



GQL

```
MATCH (p:product\#type)
RETURN p.name
```

Durch den Suffix `\#type` des Labels `product` wird ausgedrückt, dass Untertypen statt Objekte gesucht werden sollen.

Objekte von Typen ohne Vererbung

Suche

Objekte des Typs "Produkt", ohne Objekte von Untertypen von "Produkt"

Strukturabfrage

GQL

```
MATCH (p:product\#exact)
RETURN p.name
```

Die standardmäßig aktive Vererbung wird durch den Suffix `\#exact` beim Label `product` deaktiviert.

Objekte mehrerer Typen

Suche

Objekte der Typen "Produkt" oder "Firma"

Strukturabfrage

GQL

```
MATCH (t:product|company)
RETURN t.name
```

Mit | können mehrere alternative Labels eingegeben werden.

Siehe Labels

1.3.7.12.3. Identifizieren

HINWEIS Beziehen Sie sich in GQL-Abfragen nicht auf interne IDs, da diese beim Transfer von Objekten in andere Graphen nicht erhalten bleiben. Verwenden Sie stattdessen ein identifizierendes Attribut.

Element festlegen

Suche

Personen, die das Produkt *Power Mole 3000* besitzen

Strukturabfrage

GQL

```
MATCH (p:person)-[:owns]->(o:product)
WHERE o."RDF-about" = 'http://example.org/#powerMole3000'
```

```
RETURN p.name, o.name
```

Das Zielobjekt wird hier durch ein identifizierendes Attribut festgelegt.

Element über ID identifizieren

Suche

Personen, die ein über ID-Parameter festgelegtes Produkt besitzen

Strukturabfrage

GQL

```
MATCH (p:person)-[:owns]->(o:product)
WHERE ELEMENT_ID(o) = $id
RETURN p.name, o.name
```

Die Element-ID wird über den Parameter `id` an die GQL-Suche übertragen.

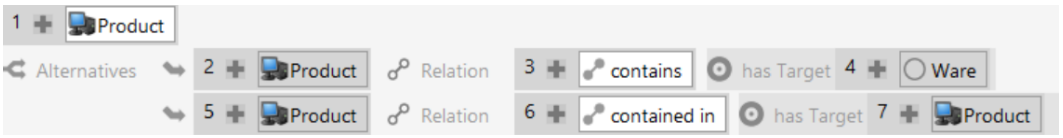
1.3.7.12.4. Struktur

Alternativen

Suche

Produkte, die andere Produkte enthalten oder in anderen Produkten enthalten sind

Strukturabfrage



GQL

```
MATCH (p:product)-[:contains]->(o:product)
RETURN p.name as name, o.name as related
UNION
MATCH (p:product)-[:containedIn]->(o:product)
RETURN p.name as name, o.name as related
```

Im Allgemeinen können Sie Alternativen als `UNION` umsetzen.

Falls die Alternativen nur Attribut-Bedingungen enthalten, können Sie auch im `WHERE`-Teil mit `OR` arbeiten, wie das folgende Beispiel zeigt.

Alternativen (einfach)

Suche

Produkte, eine Bewertung mit Wert >= 4 oder keine Bewertung haben

Strukturabfrage

GQL

```
MATCH (p:product)
WHERE p.rating >= 4 OR p.rating IS NULL
RETURN p.name as name, p.rating as rating
ORDER BY rating DESC
```

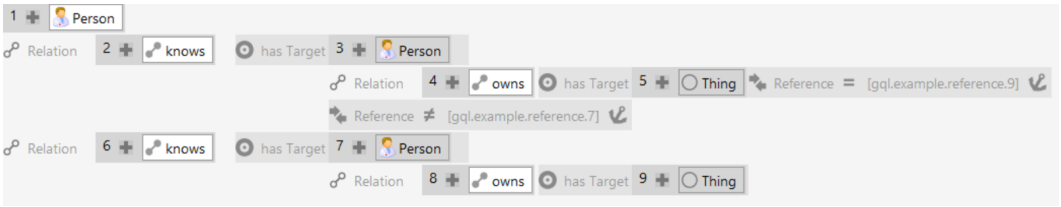
Statt `UNION` wird eine `OR`-Bedingung verwendet (siehe logische Operatoren). Dies ist möglich, wenn nur Attribute und andere Ausdrücke in der Bedingung benötigt werden. Bei Alternativen mit Relationen muss `UNION` verwendet werden.

Referenzen

Suche

Personen, die zwei unterschiedliche Personen kennen, die dasselbe Produkt besitzen

Strukturabfrage



GQL

```
MATCH (p:person)-[:knows]->(q1)-[:owns]->(t),
      (p)-[:knows]->(q2)-[:owns]->(t)
WHERE q1 <> q2
RETURN p.name, q1.name, q2.name, t.name
```

Die Identität kann durch die Verwendung derselben Variable `t` bei beiden Bedingungen erreicht werden. Die Ungleichheit der beiden Personen muss explizit durch `q1 <> q2` ausgedrückt werden.

1.3.7.13. GQL REST API

Über die REST-API können Sie GQL-Abfragen ausführen und das Ergebnis in verschiedenen Formaten ausgeben.

Um die API einzurichten, legen Sie bei einem REST-Service eine neue Resource vom Typ *Built-In Resource* an und geben Sie bei dieser als *REST resource id* einen der vordefinierten GQL-IDs an:

`GQLQueryResource`

: Erlaubt die Ausführung von registrierten und von im Request angegebenen Suchen (Ad-hoc-Suchen).

`GQLRegisteredQueryResource`

: Es können nur registrierte Suchen ausgeführt werden. Requests mit einer Ad-hoc-Suche führen zu einem Fehler.

`GQLQueryValidationResource`

: Validiert die Suche, führt sie aber nicht aus. Bei validen Suchen werden als Ergebnis die Query und Suchparameter im JSON-Format zurückgegeben. Bei Fehlern wird ein `BAD REQUEST` mit einer Fehlermeldung im JSON-Format zurückgegeben.

Die Resource unterstützt `GET`- und `POST`-Requests.

1.3.7.13.1. GET

Es können nur registrierte Suchen aufgerufen werden.

Der Registrierungsschlüssel der Suche wird als Parameter `queryId` angegeben. Alle Parameter außer `queryId`, `query` und `queryParameters` werden als Suchparameter ausgewertet.

Die Parameter können Sie bei der Konfiguration der REST-Method als Query-, Pfad- oder Header-Parameter definieren. Ohne Konfiguration wird ein Query-Parameter erwartet.

HINWEIS	Die Parameter `query` und `queryParameters` werden bei GET-Requests nicht unterstützt.

1.3.7.13.2. POST mit Form-Daten

Bei einem POST-Request mit Content-Type `application/x-www-form-urlencoded` oder `multipart/form-data` können folgende Form-Parameter angegeben werden:

`queryId`

: Der Registrierungsschlüssel einer registrierten Suche

`query`

: Der Quelltext einer GQL-Suche. Bei der Resource `GQLRegisteredQueryResource` führt dieser

Form-Parameter zu einem Fehler, da Ad-hoc-Suchen nicht erlaubt sind.

`queryParameters`

Suchparameter in JSON-Syntax

HINWEIS	Die Parameter `query` und `queryParameters` können nicht gleichzeitig angegeben werden.

1.3.7.13.3. POST ohne Form-Daten

Wenn Sie einen Content-Typ außer den beiden Form-Typen verwenden, wird im Body des POST-Requests der Quelltext einer GQL-Suche erwartet.

HINWEIS	Da es derzeit keinen offiziellen registrierten Content-Type für GQL gibt, geben Sie `text/plain` mit einem Charset an.

Die Suchparameter werden analog zu `GET` als Request-Parameter erwartet.

Bei der Resource `GQLRegisteredQueryResource` führt der Request zu einem Fehler, da Ad-hoc-Suchen nicht erlaubt sind.

1.3.7.13.4. Ausgabe

Über das Header-Feld `Accept` des Requests wird das gewünschte Ausgabeformat ausgewählt.

`text/csv`

Ausgabe als CSV-Datei. Die Tabelle enthält pro in `RETURN` angegebenem Feld eine gleichnamige Spalte.

- Encoding `UTF-8`
- Trennzeichen `;`
- Werte werden in Anführungszeichen eingeschlossen.
- Erste Zeile enthält Überschriften.

`application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`

Ausgabe als Excel-Datei.

`application/json`

Ausgabe als JSON-Datei. Die Datei enthält einen Array von Objekten. Jedes Objekt entspricht einem Suchergebnis und enthält die Felder, die in `RETURN` angegeben wurden.

HINWEIS	Falls der Feldname einen Punkt enthält, enthalten die Eigenschaftsschlüssel ebenfalls einen Punkt. In JSON ist das zwar erlaubt, aber eventuell für die weitere Verarbeitung ungünstig. Wählen Sie mit `AS` einen Namen ohne

> Punkt, beispielsweise `MATCH (p:person) RETURN p.name AS personName`.

`text/plain`

Textuelle Darstellung einer Tabelle. Dieses Format eignet sich zum Testen. Für maschinelle Verarbeitung verwenden Sie ein anderes Format.

1.3.7.13.5. Beispiele

Im Folgenden wird ein REST-Service `gql` mit einer unter dem Pfad `search` registrierten Built-In-Resource `GQLQueryResource` angesprochen.

GET

Aufruf der registrierten Suche `persons` mit Suchparameter `name`

```
GET /gql/search?queryId=persons&name=Goethe
Host: localhost:8815
Accept: application/json
```

Mit curl:

```
curl -H "Accept: application/json"
"http://localhost:8815/gql/search?queryId=persons&name=Goethe"
```

POST mit Form-Daten

Aufruf einer Ad-hoc-Suche:

```
curl -H "Accept: text/csv" -F query="MATCH (p:person) RETURN p.name,
p.birthday" "http://localhost:8815/gql/search"
```

Aufruf der registrierten Suche `persons` mit Suchparameter `name` als Form-Parameter:

```
curl -H "Accept: text/plain" -F queryId="persons" -F name="Fleißer"
"http://localhost:8815/gql/search"
```

Aufruf einer Ad-hoc-Suche mit Suchparametern im JSON-Format:

```
curl -H "Accept: text/plain" -F query="MATCH (p:product) WHERE p.rating
> $rating  RETURN p.name, p.rating" -F "queryParameters={\"rating\":
```

```
3}" "http://localhost:8815/gql/search"
```

POST mit Text-Body

Aufruf einer Ad-hoc-Suche mit Suchparametern als URL-Query-Parameter:

```
curl -H "Accept: text/plain" -H "Content-type: text/plain; charset=utf-
8" -d "MATCH (p:product) WHERE p.rating > $rating  RETURN p.name,
p.rating" "http://localhost:8815/gql/search?rating=4"
```

1.3.7.14. GQL JavaScript API

Über die JavaScript-API können Sie GQL-Abfragen programmatisch ausführen und auf die Ergebnis-Tabelle zugreifen.

1.3.7.14.1. GQL-Abfrage erstellen

Eine GQL-Abfrage wird als Instanz von `$k.GQLQuery` erstellt. Der Konstruktor nimmt optional den Abfragetext als Zeichenkette entgegen.

```
const query = new $k.GQLQuery('MATCH (p:product) RETURN p.name, p.price')
```

Alternativ können Sie den Abfragetext nachträglich mit `setSource()` setzen oder mit `source()` abfragen.

```
const query = new $k.GQLQuery()
query.setSource('MATCH (p:product) RETURN p.name, p.price')
```

Um eine registrierte Suche als GQL-Abfrage auszuführen, rufen Sie `$k.Registry.query()` mit dem Registrierungsschlüssel auf. Das Ergebnis ist ebenfalls ein `$k.GQLQuery`-Objekt.

```
const query = $k.Registry.query('products')
```

Beachten Sie dabei, dass `setSource()` auf registrierten GQL-Abfragen den zugrundeliegenden Quelltext modifiziert und speichert. Dafür wird eine Schreibtransaktion benötigt.

1.3.7.14.2. Parameter setzen

Suchparameter können auf drei Arten gesetzt werden:

`setParameter(parameterId, value)`

Setzt einen einzelnen Parameter anhand seines Namens.

`setParameters(parameters)`

Setzt mehrere Parameter auf einmal über ein Objekt, dessen Schlüssel den Parameternamen entsprechen.

`setAccessParameter(accessParameter, value)`

Setzt einen Zugriffsparameter (z. B. `'user'`, `'topic'`, `'accessedObject'`) auf ein semantisches Element oder eine Liste von semantischen Elementen.

```
query.setParameter('minRating', 4)
query.setParameters({ manufacturer: 'ACME', category: 'electronics' })
```

1.3.7.14.3. Abfrage ausführen

perform()

Die Abfrage wird mit `perform()` ausgeführt. Die Methode nimmt optional einen Suchstring und ein Parameter-Objekt entgegen, das eventuell zuvor gesetzte Parameter überschreibt. Die beiden Argumente können in beliebiger Reihenfolge übergeben werden.

```
const result = query.perform('PowerMole', { manufacturer: 'ACME' })
```

Wenn nur Parameter übergeben werden, lassen Sie den Suchstring weg:

```
const result = query.perform({ manufacturer: 'ACME' })
```

HINWEIS	Fehlende Parameter führen zu einem Laufzeitfehler. Es gibt keine implizite Deaktivierung von Parametern, die nicht übergeben werden.

findElements()

Da `$k.GQLQuery` von `$k.Query` erbt, steht auch `findElements()` zur Verfügung. Die Methode gibt ein unsortiertes Array aller semantischen Elemente zurück, die in einem der `RETURN`-Felder enthalten sind.

```
const elements = query.findElements({ manufacturer: 'ACME' })
```

HINWEIS	Bei Verwendung von `findElements()` gehen wesentliche Vorteile von GQL verloren: die strukturierte Ergebnistabelle, die explizite Sortierung und projizierte Ausdruckswerte stehen nicht mehr zur Verfügung. `findElements()` eignet sich daher nur zur Kompatibilität mit anderen Sucharten, die dieselbe

> Schnittstelle verwenden. Verwenden Sie für GQL-spezifische Auswertungen `perform()`.

1.3.7.14.4. Ergebnis auswerten

Das Ergebnis wird als `$k.GQLQueryResult` zurückgegeben.

Spalten

`columns()`

Gibt ein Array von `$k.GQLQueryResultColumn`-Objekten zurück.

`captions()`

Gibt ein Array der Spaltennamen als Zeichenketten zurück.

Jede Spalte (`$k.GQLQueryResultColumn`) hat folgende Methoden:

`index()`

Gibt den nullbasierten Spaltenindex zurück.

`caption()`

Gibt den Namen der Spalte zurück.

`valueType()`

Gibt den Typ der Spaltenwerte zurück.

Zeilen

`rows()`

Gibt einen Iterator über die Ergebniszeilen zurück.

`size`

Anzahl der Ergebniszeilen.

Jede Zeile (`$k.GQLQueryResultRow`) hat folgende Methoden:

`values()`

Gibt die Zeilenwerte als Array zurück.

`bindings()`

Gibt die Zeilenwerte als Objekt zurück, dessen Schlüssel den `RETURN`-Feldnamen entsprechen.

`structuredBindings()`

Wie `bindings()`, aber Feldnamen mit Punkten werden in verschachtelte Objekte umgewandelt

(z. B. wird `p.name` zu `{p: {name: "…"}}`).

`size`

Anzahl der Spalten in der Zeile.

Werttypen

Die Werte werden als entsprechende JavaScript-Objekte zurückgegeben:

Knoten

Knoten werden als semantische Elemente zurückgegeben.

Kanten

Bei Kanten hängt der Typ davon ab, ob ein Relations-Index verwendet wurde. Es wird entweder eine `$k.Relation` oder eine `$k.VirtualRelation` zurückgegeben.

Attributwerte

Attributwerte werden wie beim direkten Zugriff per `attributeValue()` zurückgegeben.

1.3.7.14.5. Beispiel

Registrierte Suche `products` ausführen und Spalten und Zeilen als Text ausgeben.

```
const result = $k.Registry.query('products').perform({
  manufacturer: 'Drill Corp'
})
const report = new $k.TextDocument()
report.println(`Anzahl Ergebnisse: ${result.size} Zeilen`)
report.println(`Spalten:`)
for (const column of result.columns()) {
  report.println(`${column.index()}. ${column.caption()}: ${column
.valueType()}`)
}
report.println(`Zeilen:`)
for (const row of result.rows()) {
  report.println(row.values().map(v => JSON.stringify(v)).join(','))
}
```

1.4. Ordner und Registrierung

Neben den Objekten und ihren Eigenschaften bauen wir in einem typischen Projekt auch diverse andere Elemente: wir definieren z.B. Abfragen und Importe/Exporte oder schreiben Skripte für spezielle Funktionen. Alles, was wir bauen und konfigurieren, können wir in Ordnern organisieren.

Die Ordner werden geteilt mit allen anderen, die am Projekt arbeiten. Wenn wir das nicht möchten, können wir Dinge im Privatordner ablegen, etwa für Testzwecke. Dieser ist nur für den jeweiligen Nutzer sichtbar.

Eine spezielle Form des Ordners ist die Sammlung semantischer Objekte, in der wir von Hand Objekte z.B. zur späteren Bearbeitung ablegen können. Dazu können wir sie einfach mit Drag & Drop in den Ordner schieben, zudem gibt es Operationen um z.B. Ergebnislisten in Ordnern festzuhalten.

In dem Moment, in dem wir eines dieser Objekte im Knowledge-Graph löschen, wird es ebenfalls in der Sammlung entfernt. Wird ein Objekt aus einer Sammlung entfernt (Klick auf *Aus Ordner entfernen*), dann wird nur das Objekt aus dem Ordner entfernt, das Objekt selbst bleibt erhalten. Wenn im Ordner jedoch auf *Löschen*, auf *Selektierte Objekte löschen* oder auf *Alle Objekte im Ordner löschen* geklickt wird, so wird das semantische Element vollständig gelöscht; es ist dann weder im Ordner noch im Knowledge-Graph auffindbar.

WARNUNG	Die Aktion *Aus Ordner entfernen* hat je nach Anwendungskontext eine andere Bedeutung: Im Fall eines Ordners mit Importmappings bedeutet der Befehl *Aus dem Ordner entfernen* ein Löschen des Importmappings!

Aus Gründen der Performance wird bei Sammlungen semantischer Objekte mit mehr als 100 Einträgen darauf verzichtet, die am besten zum Inhalt passende Tabellenkonfiguration zu ermitteln. Mittels der Kontextmenü-Funktion *Konfiguration der Objektliste bestimmen* können wir das bei Bedarf anfordern.

1.4.1. Registrierung

Abfragen, Skripte etc. können sich gegenseitig aufrufen (eine Abfrage kann in eine andere Abfrage oder in ein Skript eingebaut werden, umgekehrt kann ein Skript von einer Such-Pipeline aus aufgerufen werden). Zu diesem Zweck gibt es Registrierungsschlüssel, mit denen wir Abfragen, Import-/Export-Abbildungen, Skripte und sogar Sammlungen semantischer Objekte und Strukturordner ausstatten können, damit sie anderen Konfigurationen Funktionalität zur Verfügung stellen. Der Registrierungsschlüssel  muss eindeutig sein. Alles, was einen Registrierungsschlüssel hat, wird automatisch in den Ordner *Registrierte Objekte* aufgenommen bzw. in den seinem Typ entsprechenden Unterordner.

1.4.2. Verschieben, Kopieren, Löschen

Nehmen wir an, wir haben in unserem Projekt einen Ordner namens "Playlist-Funktionen" Dieser enthält vielleicht einen Export, einige Skripte und eine Strukturabfrage "ähnliche Songs", die wir in

einem REST-Services benutzen wollen. In dem Moment, in dem wir der Strukturabfrage einen Registrierungsschlüssel geben, wird sie im Ordner *Registrierte Objekte* (Abschnitt *Technik*) aufgenommen. D.h. die Strukturabfrage "ähnliche Songs" taucht im Ordner *Registrierte Objekt* unter *Abfrage* auf. Sie bleibt dort auch, wenn wir sie aus unserem Projekt-Unterordner "Playlist-Funktionen" entfernen. Entfernen wir den Registrierungsschlüssel, fällt die Abfrage automatisch aus der Registratur heraus.

Das Grundprinzip beim Löschen bzw. Entfernen: Abfragen, Importe, Skripte können in ein oder mehreren Ordnern gleichzeitig und müssen in mindestens in einem Ordner enthalten sei. Erst wenn wir also z.B. unsere Abfrage aus dem letzten Ordner entfernen, wird sie tatsächlich gelöscht. Nur dann bittet i-views auch um eine Bestätigung der Löschaktion. Das gleiche gilt für das Entfernen des Registrierungsschlüssels.

Wollen wir in einem Schritt die Abfrage löschen, unabhängig davon, in wie vielen Ordnern sie sich findet, so können wir das nur von der Registratur aus.

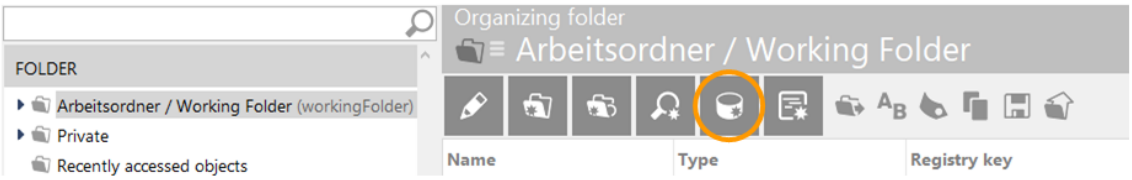
1.4.3. Ordnereinstellungen

In den Ordnereinstellungen können wir für Suchergebnisse, Ordner und Objektlisten (Liste der konkreten Objekte im Hauptfenster des Knowledge-Builders bei Selektion eines Objekttyps auf der linken Seite) Mengengrenzen definieren. *Automatische Abfrage bis zur Anzahl Objekte* gibt an, bis zu welcher Anzahl von Objekte der Ordnerinhalt oder die Objektliste ohne weitere Interaktion durch den Benutzer gezeigt wird. Ist die dort eingestellte Grenze überschritten, bleibt die Liste zunächst leer und in der Statuszeile erscheint die Meldung "Abfrage nicht ausgeführt". Das Ausführen der Suche ohne Eingabe in die Eingabezeile zeigt alle Objekte, zumindest bis die zweite Grenze erreicht ist: *Maximale Anzahl der Abfrageergebnisse* oder *Maximale Ergebnisanzahl in Objektlisten*. Auch in diesem Fall ist die Ergebnisliste leer und die Statuszeile zeigt eine entsprechende Nachricht. Die Suche muss dann weiter eingegrenzt werden, um ein Ergebnis zu erhalten.

1.5. Import und Export

Mit den Abbildungen von Datenquellen können wir Daten aus strukturierten Quellen in i-views importieren und Objekte und ihre Eigenschaften in strukturierter Form exportieren. Die Quellen können Excel/CSV-Tabellen, Datenbanken oder XML-Strukturen sein.

Die Funktionen für den Import und Export decken sich größtenteils und sind daher alle in einem Editor verfügbar. Um auf die Funktionen für den Import und Export zugreifen zu können, muss zunächst ein Ordner (z.B. der Arbeitsordner) ausgewählt werden. Dort kann über die Schaltfläche "Neue Abbildung einer Datenquelle" eine Datenquelle ausgewählt werden, aus der importiert, bzw. in die exportiert werden soll.



Alternativ findet man die Schaltfläche auch im Reiter "TECHNIK" unter "Registrierte Objekte" > "Abbildungen von Datenquellen".

Folgende Schnittstellen und Dateiformate stehen für den Import und Export zur Verfügung:

- CSV-Datei
- Excel-Datei
- XML-Datei
- JSON-Datei
- LDAP
- Elasticsearch
- IAS Knowledge Model
- MySQL-Schnittstelle
- ODBC-Schnittstelle
- Oracle-Schnittstelle
- PostgreSQL-Schnittstelle

Im Folgenden wird anhand einer CSV-Datei beschrieben, wie man einen tabellenorientierten Import/Export anlegt.

1.5.1. Abbildungen von Datenquellen

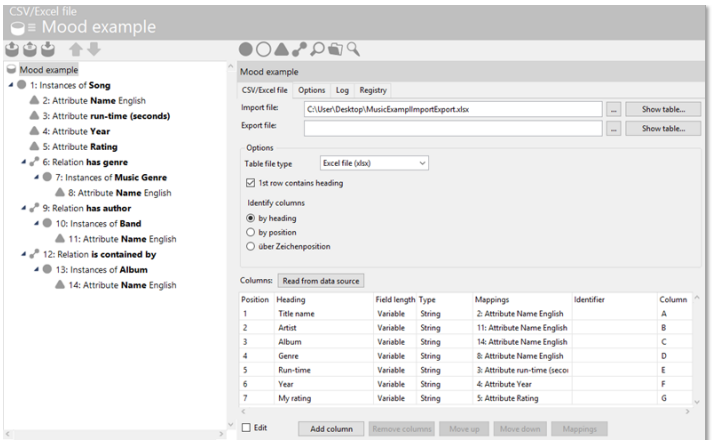
CSV-Dateien sind das Standard-Austauschformat von Tabellenkalkulationstools wie Excel. CSV-Dateien bestehen aus einzelnen Klartext-Zeilen, bei denen die Spalten durch ein fest vorgegebenes Zeichen wie z.B. ein Semikolon getrennt sind.

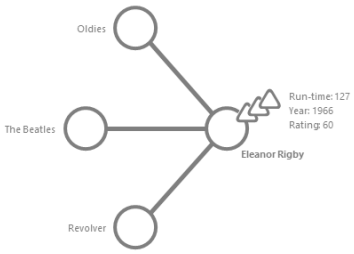
1.5.1.1. Funktionsprinzip

Nehmen wir als erstes Beispiel eine Tabelle mit Songs: Beim Import dieser Tabelle möchten wir für jede Zeile ein neues konkretes Objekt vom Typ Song anlegen. Aus den Inhalten der Spalten B bis G werden Attribute des Songs bzw. Relationen zu anderen Objekten:

	A	B	C	D	E	F	G	H
1	**Title name**	**Artist**	**Album**	**Genre**	**Run-time**	**Year**	**My rating**	
2	The suburbs	Arcade Fire	The suburbs	Postwave	315	2010	60	
3	Ready To Start	Arcade Fire	The suburbs	Postwave	255	2010	80	
4	Modern Man	Arcade Fire	The suburbs	Postwave	279	2010	60	
5	Rococo	Arcade Fire	The suburbs	Postwave	236	2010	40	
6	Empty Room	Arcade Fire	The suburbs	Postwave	171	2010	20	
7	City With No Children	Arcade Fire	The suburbs	Postwave	191	2010	20	
8	Half Light I	Arcade Fire	The suburbs	Postwave	253	2010	20	
9	Half Light II (No Celebration)	Arcade Fire	The suburbs	Postwave	267	2010	40	
10	Suburban War	Arcade Fire	The suburbs	Postwave	281	2010	80	
11	Month Of May	Arcade Fire	The suburbs	Postwave	230	2010	20	
12	Wasted Hours	Arcade Fire	The suburbs	Postwave	200	2010	40	
13	Deep Blue	Arcade Fire	The suburbs	Postwave	268	2010	60	
14	We Used To Wait	Arcade Fire	The suburbs	Postwave	301	2010	100	
15	Sprawl I (Flatland)	Arcade Fire	The suburbs	Postwave	174	2010	40	
16	Sprawl II (Mountains Beyond Mountains)	Arcade Fire	The suburbs	Postwave	318	2010	40	
17	The Suburbs (Continued)	Arcade Fire	The suburbs	Postwave	87	2010	40	
18	Eleanor Rigby	The Beatles	Revolver	Oldies	127	1966	60	
19	For No One	The Beatles	Revolver	Oldies	121	1966	60	
20	Good Day Sunshine	The Beatles	Revolver	Oldies	129	1966	40	
21	Here There And Everywhere	The Beatles	Revolver	Oldies	145	1966	40	
22	I Want To Tell You	The Beatles	Revolver	Oldies	149	1966	40	
23	I'm Only Sleeping	The Beatles	Revolver	Oldies	181	1966	60	
24	Love To You	The Beatles	Revolver	Oldies	181	1966	20	
25	She Said She Said	The Beatles	Revolver	Oldies	157	1966	40	
26	Taxman	The Beatles	Revolver	Oldies	159	1966	20	
27	Tomorrow Never Knows	The Beatles	Revolver	Oldies	177	1966	20	
28	Yellow Submarine	The Beatles	Revolver	Oldies	160	1966	20	
29	About A Girl	Nirvana	MTV Unplugged in NY	Rock	217	1994	60	
30	Jesus Doesn't Want Me For A Su	Nirvana	MTV Unplugged in NY	Rock	277	1994	40	
31	The Man Who Sold The World	Nirvana	MTV Unplugged in NY	Rock	260	1994	80	
32	Pennyroyal Tea	Nirvana	MTV Unplugged in NY	Rock	220	1994	60	
33	Dumb	Nirvana	MTV Unplugged in NY	Rock	172	1994	40	
34	Polly	Nirvana	MTV Unplugged in NY	Rock	196	1994	60	

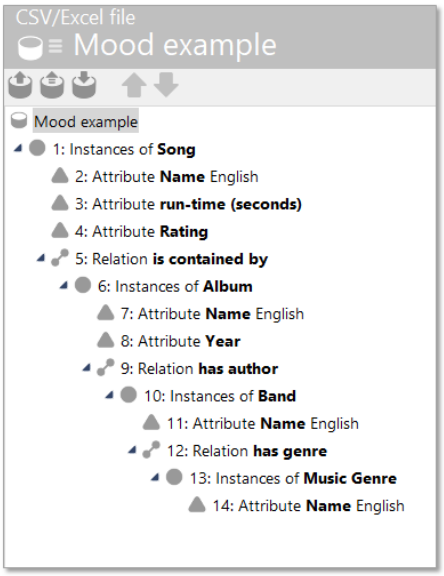
Ausgehend vom Song, bauen wir die Struktur von Attributen, Relationen und Zielobjekten auf, die durch den Import angelegt werden soll (linke Seite). So wird für die Zeile 18 beispielsweise ein Objekt vom Typ Song mit folgenden Attributen und Beziehungen angelegt:

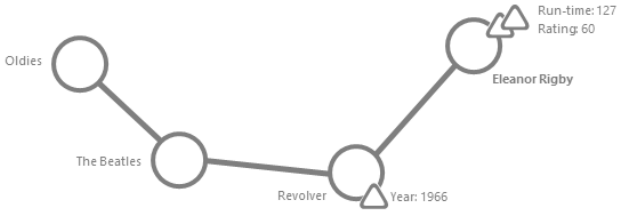




Wir können uns aber auch dafür entscheiden, die Angaben aus der Tabelle anders zu verteilen also

z.B. Erscheinungsjahr und Interpret dem Album zuordnen und das Genre wiederum dem Interpreten. Eine Zeile bildet immer noch einen Kontext, muss deswegen aber nicht zu genau einem Objekt gehören:

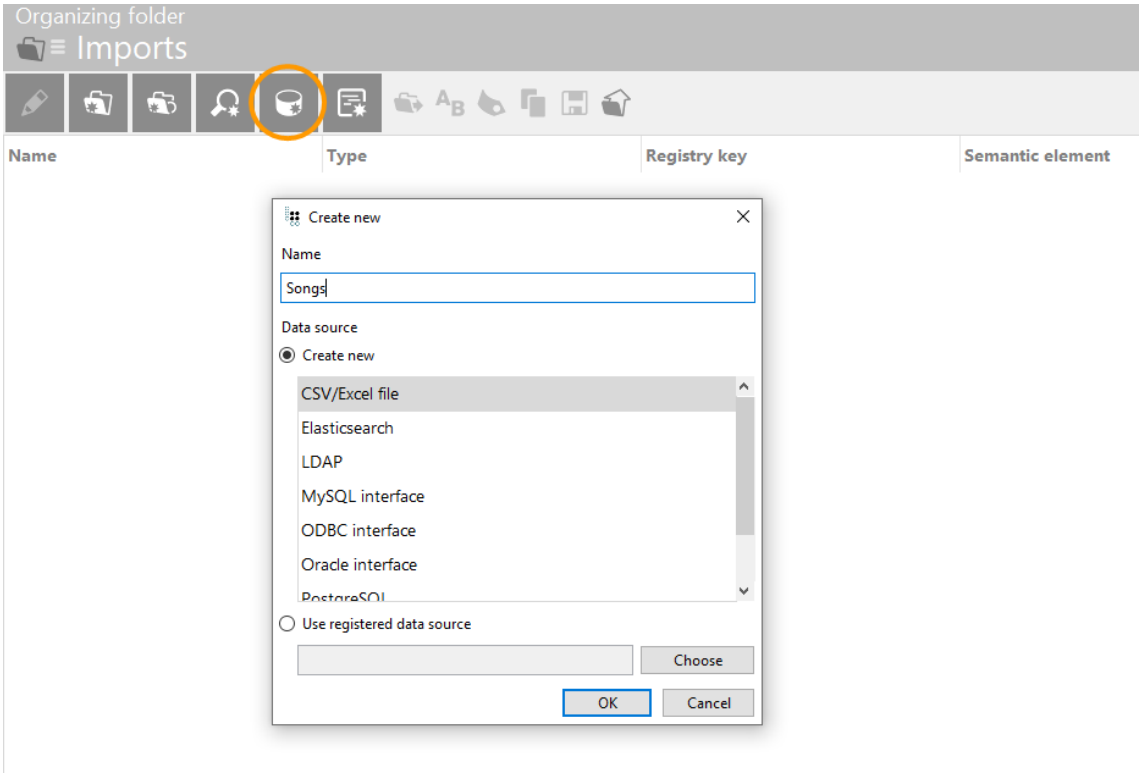




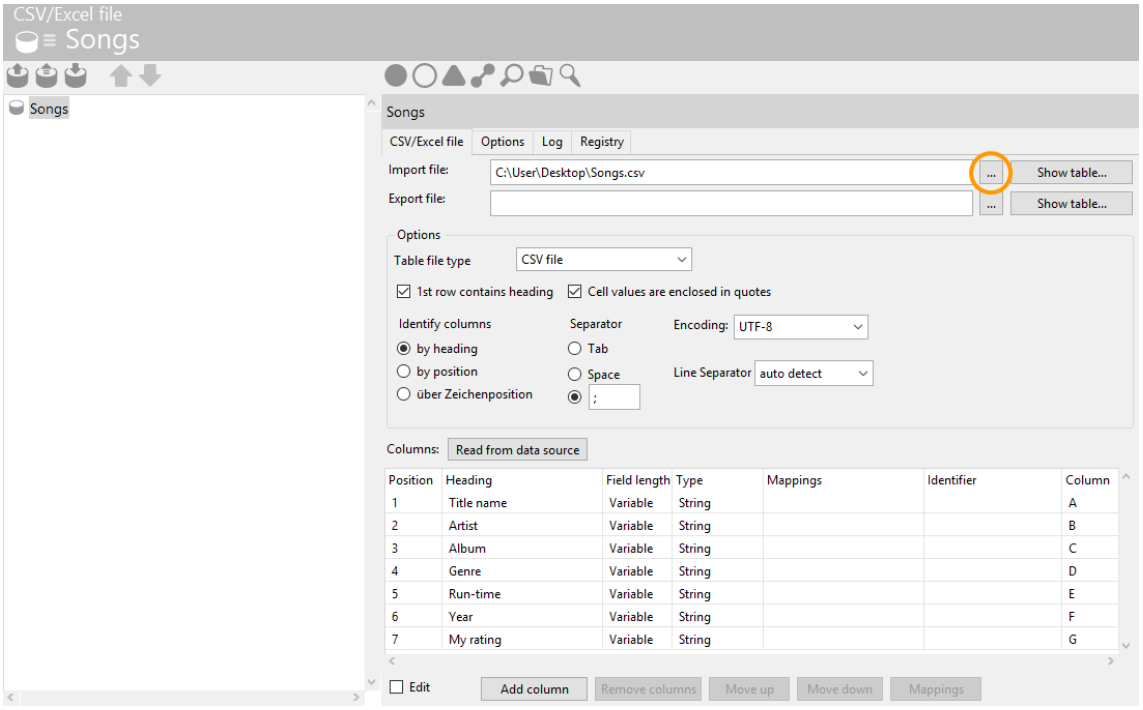
Überall, wo wir in unserem Beispiel neue konkrete Objekte als Relationsziele aufbauen, müssen wir immer mindestens ein Attribut zu diesem Objekt angeben, hier jeweils das Namensattribut, mit dem wir das entsprechende Objekt identifizieren können.

1.5.1.2. Datenquelle — Auswahl und Optionen

Nachdem wir, die Schaltfläche "*Neue Abbildung einer Datenquelle*" ausgewählt haben, öffnet sich ein Dialog, mit dem wir die Art der Datenquelle und den Namen der Abbildung angeben müssen. Haben wir die Datenquelle bereits in der semantischen Graph-Datenbank registriert, können wir diese im unteren Auswahlmenü finden.



Mit der Bestätigung auf "OK" öffnet sich der Editor für den Import und Export. Unter "Import-Datei" können wir den Pfad unserer zu importierenden Datei angeben. Alternativ können wir die Datei auch über den Button rechts daneben auswählen. Sobald die Datei ausgewählt wurde, werden die Spaltenüberschriften und ihre Positionen in der Tabelle ausgelesen und im Feld rechts unten angezeigt. Die Schaltfläche "*Aus Datenquelle lesen*" kann bei eventuellen Änderungen der Datenquelle die Spalten erneut auslesen. Die Spalte "Abbildungen" zeigt uns später jeweils auf welches Attribut die jeweilige Spalte der Tabelle abgebildet wird.



Die Struktur unserer Beispiel-Tabelle entspricht komplett den Standard-Einstellungen, sodass wir unter dem Menüpunkt *Optionen* nichts weiter berücksichtigen müssen. CSV-Dateien können jedoch sehr unterschiedliche Strukturen aufweisen, die mit folgenden Einstellungsmöglichkeiten berücksichtigt werden müssen:

Encoding : Hier wird die Zeichenkodierung der Import-Datei festgelegt. Zur Auswahl stehen ascii, ISO-8859-1, ISO-8859-15, UCS-2, UTF-16, UTF-8 und windows-1252. Ist nichts ausgewählt, wird die Standard-Einstellung übernommen, die der des laufenden Betriebssystems entspricht.

Zeilentrenner : In den meisten Fällen reicht die Einstellung "automatisch erkennen", die auch standardmäßig ausgewählt ist. Sollte man jedoch feststellen, dass Zeilenumbrüche nicht richtig erkannt werden, sollte man die entsprechende korrekte Einstellung manuell auswählen. Zur Auswahl stehen *CR* (*carriage return*, engl. für Wagenrücklauf), *LF* (*line feed*, engl. für Zeilenvorschub), *CR - LF* und *Keine*. Der Standard, der den Zeilenumbruch in einer Textdatei kodiert, ist bei Unix, Linux, Android, Mac OS X, AmigaOS, BSD und weiteren *LF*, bei Windows, DOS, OS/2, CP/M und TOS (Atari) *CR-LF* und bei Mac OS bis Version 9, Apple II und C64 *CR*.

1. Zeile ist Überschrift : Es kann vorkommen, dass die erste Zeile keine Überschrift enthält, was mit dem Entfernen des standardmäßig gesetzten Häkchens bei "*1. Zeile ist Überschrift*" dem System mitgeteilt werden muss.

Werte in Zellen sind in Anführungszeichen eingeschlossen wählt man aus, damit die Anführungszeichen nicht mit importiert werden, wenn man das nicht möchte.

Spalten identifizieren : Ob die Spalten über ihre Überschrift, die Position oder die Zeichenposition identifiziert werden, muss angegeben werden, da ansonsten die Tabelle nicht korrekt erfasst werden kann.

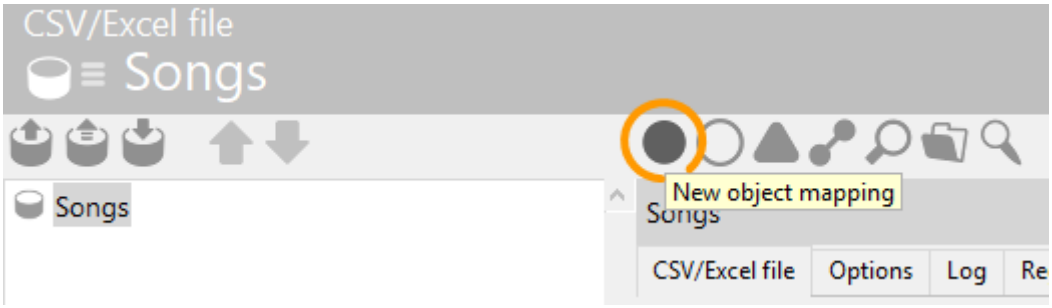
Trennzeichen : Falls ein anderes Trennzeichen als das standardmäßige Semikolon verwendet wird, muss dies ebenfalls angegeben werden, sofern die Spalte nicht über die Zeichenposition identifiziert wird.

Darüber hinaus gelten folgende Regeln: Falls ein Wert der Tabelle das Trennzeichen oder einen Zeilenumbruch enthält, muss der Wert in doppelte Anführungszeichen gestellt werden. Falls der Wert ein Anführungszeichen enthält, muss dieses verdoppelt ("") werden.

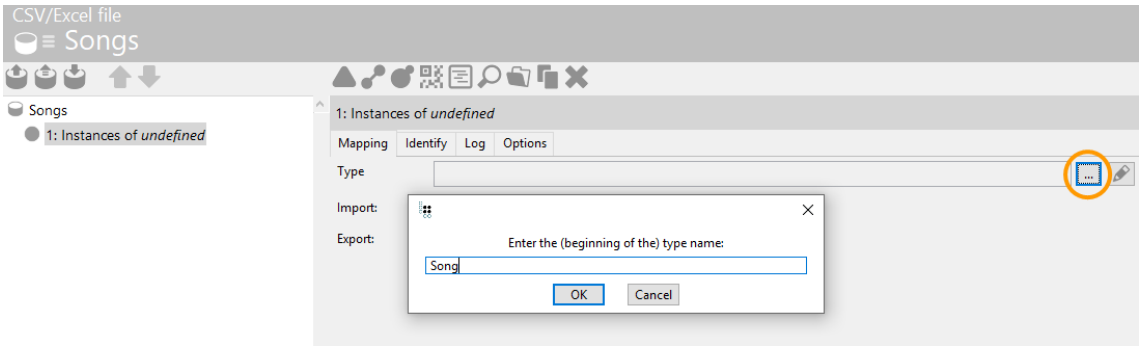
1.5.1.3. Definition von Zielstruktur und Abbildungen

1.5.1.3.1. Die Objektabbildung

Nun fangen wir an die Zielstruktur, die in der semantischen Graph-Datenbank entstehen soll, aufzubauen. In unserem Beispiel beginnen wir mit einer Objektabbildung der Songs. Um ein neues Objekt abzubilden müssen wir den Button "Neue Objektabbildung" bestätigen.



Als nächstes muss der Typ des zu importierenden Objektes angegeben werden.



Im Optionen-Tab an der Objektabbildung gibt es noch weitere spezifische Einstellungen.

Mit Objekten aller Untertypen: Wenn der Haken bei "Mit Objekten aller Untertypen" gesetzt ist, dann werden beim Import auch Objekte aus allen Untertypen von "Song" berücksichtigt. Da dies meistens gewünscht ist, ist der Haken hier standardmäßig gesetzt.

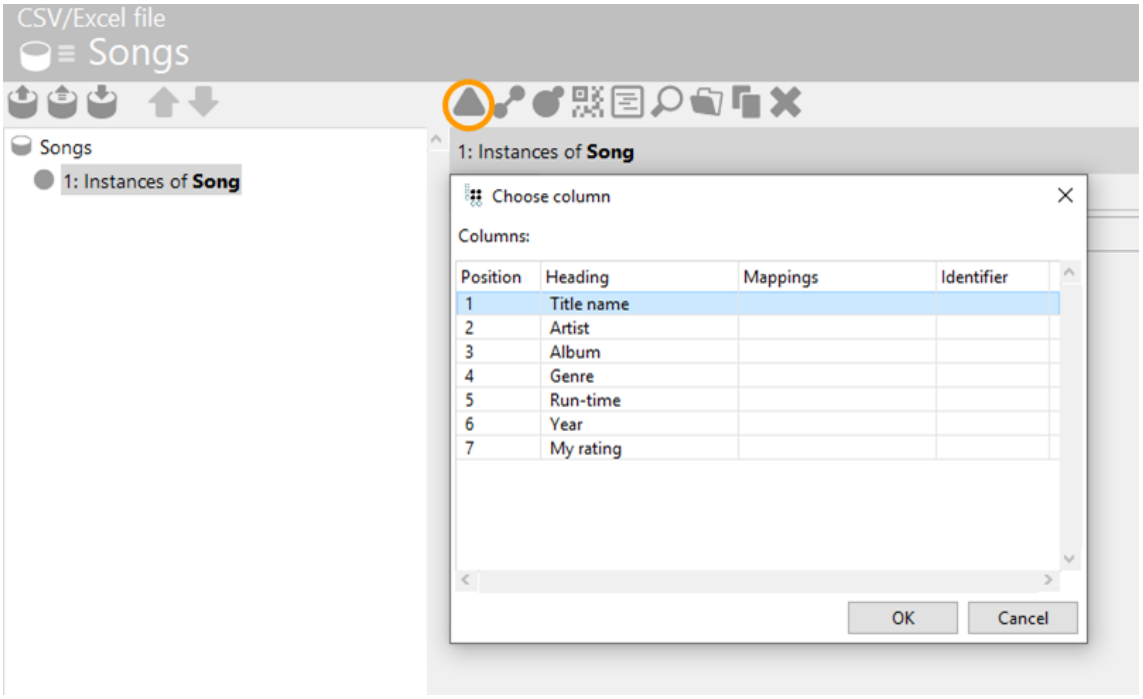
Genauerer Typ wird durch folgende Abbildung spezifiziert: Wenn in der Import-Quelle der genaue Typ identifiziert ist, zu dem das Objekt angelegt werden soll, so kann dieser hier über den Button "Neu..." abgebildet werden. Es muss sich dabei um einen Untertyp des im Tab Abbildung festgelegten Typs handeln.

Mehrere Objekte erlauben: Es kann vorkommen, dass im Knowledge Graph bereits mehrere Objekte mit gleichwertigen identifizierenden Eigenschaften exisitieren (mehrere gleichnamige Objekte). Wenn das Importmapping sich auf diese Objekte beziehen soll, kann ein Mehrdeutigkeitskonflikt entstehen. Wenn für die Option "Mehrere Objekte erlauben" der Haken gesetzt ist, wird für alle diese Objekte der Import trotzdem vollzogen.

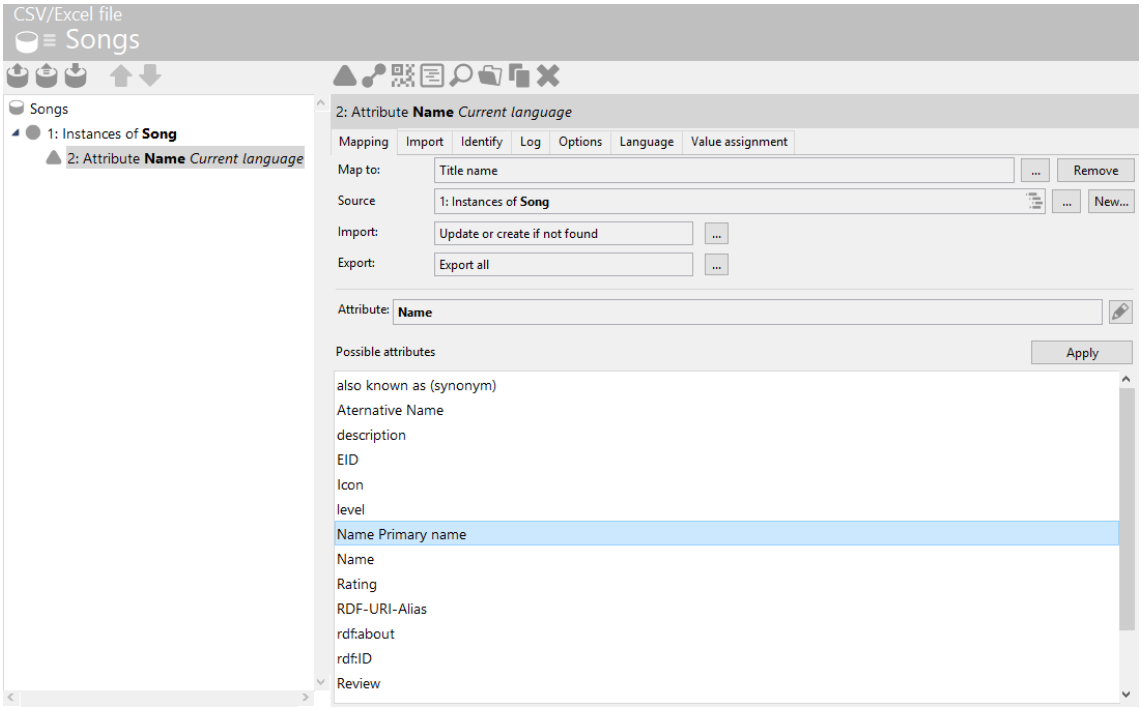
Setzt man den Haken nicht, wird der Import für die mehrfach vorkommenden Objekte nicht durchgeführt und es wird dem Nutzer stattdessen mitgeteilt, dass der Importer das Objekt nicht eindeutig identifizieren kann.

1.5.1.3.2. Die Attributabbildung / Identifizieren von Objekten

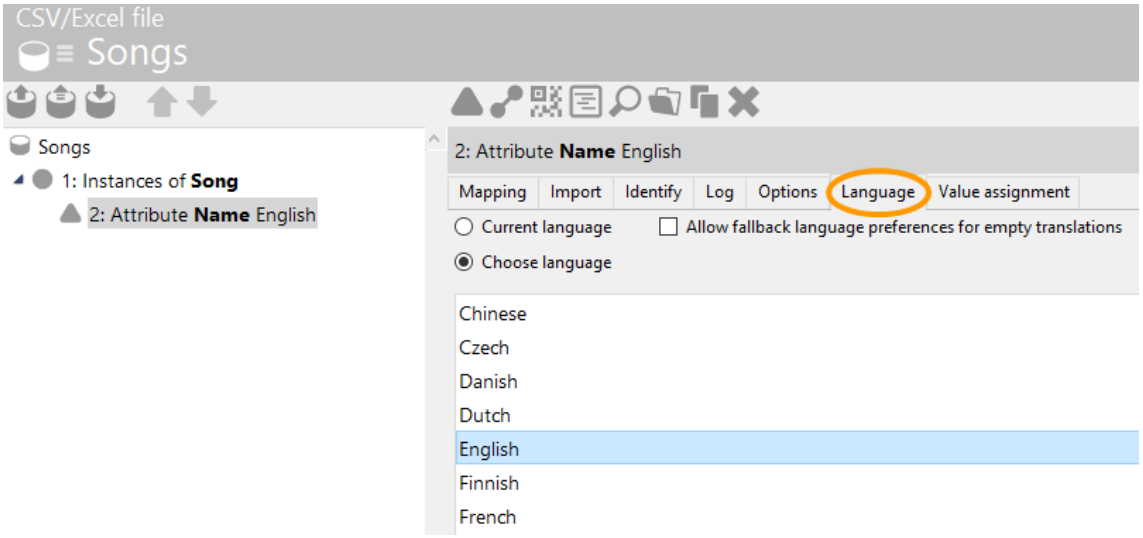
Nun wollen wir die in der Tabelle enthaltenen Informationen mit der Objektabbildung der Songs verknüpfen. Es sind sowohl Attribute zu den einzelnen Songs, als auch Relationen vertreten. Um zunächst den Titelnamen eines Songs in der Abbildung anzulegen, fügen wir der Objektabbildung von Song ein Attribut hinzu. Ein Klick auf die Schaltfläche "*Neue Attributabbildung*" öffnet einen Dialog, mit dem die entsprechende Spalte aus der zu importierenden Tabelle ausgewählt werden muss.



Da dieses Attribut, das erste ist, das wir zu der Objektabbildung von Songs angelegt haben, wird es daraufhin automatisch auf den Namen des Objekts abgebildet, da der Name in der Regel das meistgenutzte Attribut ist.



Das erste zu einem Objekt angelegte Attribut wird zudem automatisch zur **Identifizierung des Objektes** verwendet. Zu beachten ist, dass für Zeichenketten-Attribute eine Sprache gewählt werden kann, sofern für den Attributtyp die Übersetzungen aktiviert wurden. Wenn nichts anderes angegeben wird, wird die automatisch die aktive Sprache (Anzeigesprache des Knowledge-Builders) als Referenz verwendet. Die Sprache wird im Reiter "Sprache" gewählt:

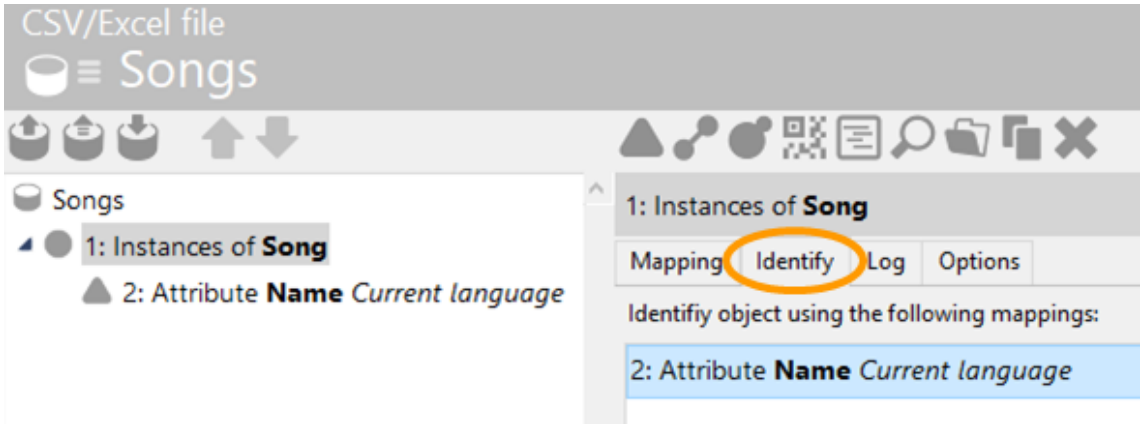


Ein Objekt muss über mindestens ein Attribut identifiziert werden — sei es über seinen Namen oder seine ID oder eine Kombination aus mehreren Attributen (wie bei Personen aus Vor- und Nachname und dem Geburtsdatum)-, damit es in der semantischen Graph-Datenbank eindeutig wiedergefunden werden kann, sollte es bereits vorhanden sein. So wird das ungewollte Anlegen von Dubletten beim Import vermieden.

HINWEIS

Auch Meta-Attribute an Relationen können importiert werden. Hier ist sicherzustellen, dass sowohl Relationsquelle als auch das Relationsziel angegeben und identifiziert sind, da ansonsten die Relation vom Importer ignoriert wird.

Im Reiter "*Identifizieren*" kann das Attribut, das das Objekt identifiziert noch nachträglich geändert oder mehrere Attribute hinzugefügt werden. Zudem kann hier eingestellt werden, ob die Groß- und Kleinschreibung beim Abgleich der Werte beachtet werden soll und ob nach exakt gleichen Werten gesucht werden soll (ohne Indexfilter/Wildcards). Letzteres ist dann relevant, wenn im Index Filter oder Wildcards definiert sind, die z.B. festlegen, dass ein Bindestrich im Index entfallen soll. Der Begriff würde mit Bindestrich nicht gefunden werden, wenn nur über den Index gesucht wird, also müsste in diesem Fall hier der Haken gesetzt werden, sodass nach dem exakt gleichen Wert gesucht wird.



Jetzt können wir dem Objektmapping weitere Attribute hinzufügen, die nicht zur Identifizierung beitragen müssen, z.B. die Dauer eines Songs — dies geschieht wieder über die Schaltfläche "Neue Attributabbildung". (Achtung: Die Objektabbildung "Objekte von Song" muss zunächst wieder ausgewählt werden.) Jetzt wählen wir die Spalte "Dauer" aus der zu importierenden Tabelle aus. Dieses Mal müssen wir das Attribut, auf das die Spalte "Dauer" abgebildet werden soll, manuell auswählen. In dem Feld rechts unten befindet sich die Lister aller im Schema festgelegten möglichen Attribute, die uns für Objekte des Typs "Song" zur Auswahl stehen, darunter auch das Attribut "Dauer".

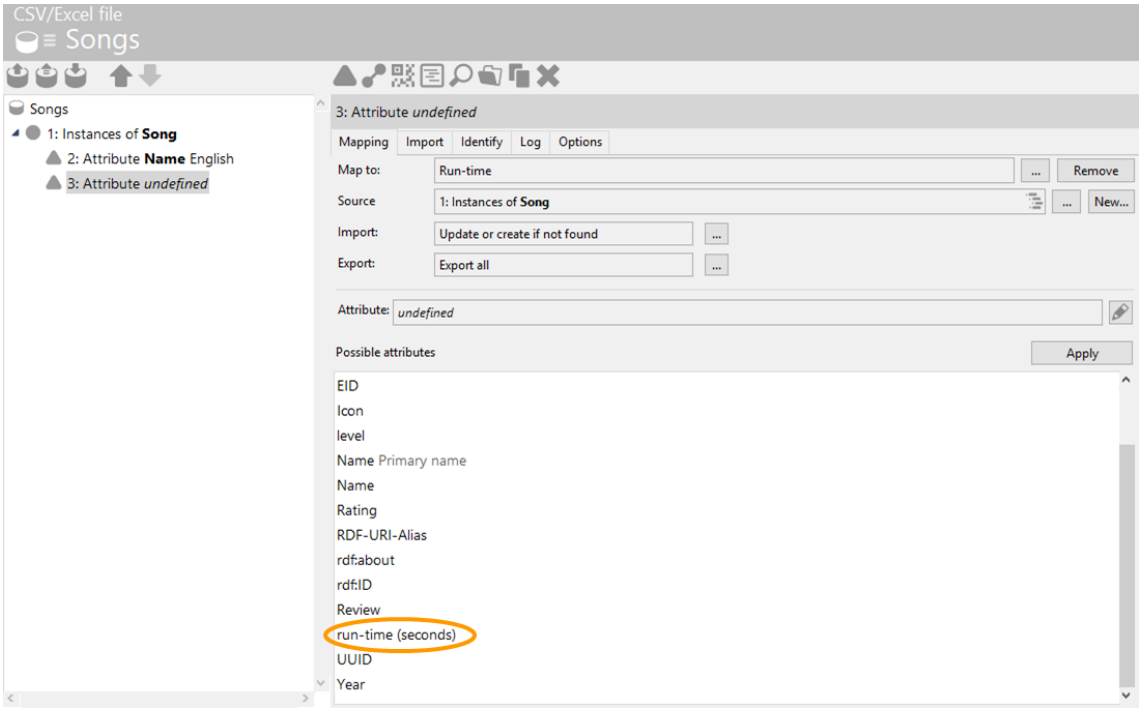


Abbildung von Übersetzungen

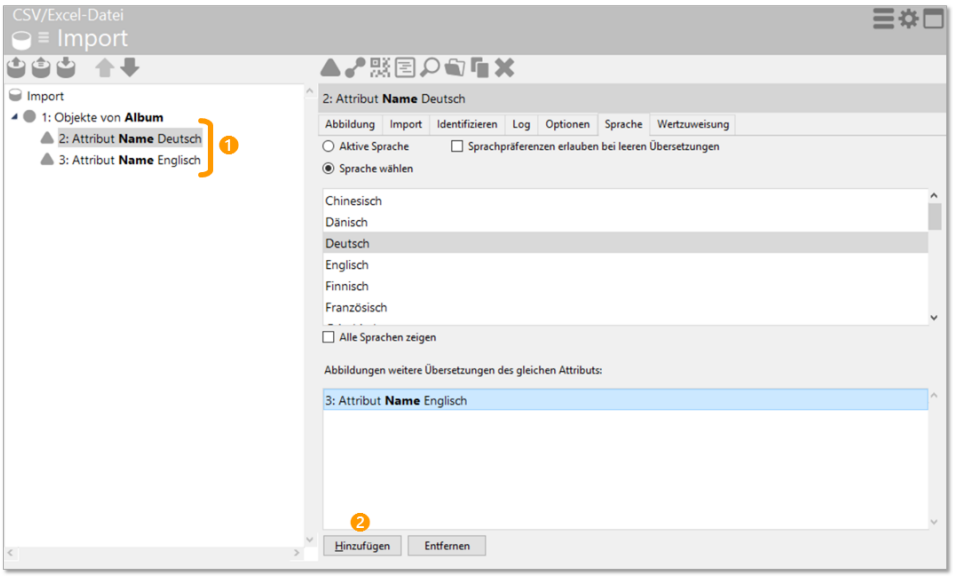
Für Zeichenketten-Attribute mit Übersetzungen wie bspw. dem Primärnamen von Objekten kann festgelegt werden, in welcher Sprache ein Wert importiert werden soll.

Wenn die Attributabbildung für ein übersetztes Attribut neu angelegt wird, so wird als Importsprache automatisch die "Aktive Sprache" verwendet. Die aktive Sprache ist die Sprache, in welcher der Knowledge-Builder gestartet wurde (welche der Sprache der Benutzeroberfläche entspricht).

Soll nun abweichend davon ein Wert in einer bestimmten Sprache importiert werden, so wird dies im Reiter "Sprache" festgelegt durch Auswahl einer Sprache aus der Liste.

Wenn die Datenquelle zugleich mehrere Übersetzungen für ein- und dasselbe Attribut (in derselben Zeile) bereithält, können diese Werte in einem Mapping zeitgleich importiert werden.

Das gleichzeitige Importieren von Übersetzungen eines Attributs funktioniert wie folgt:



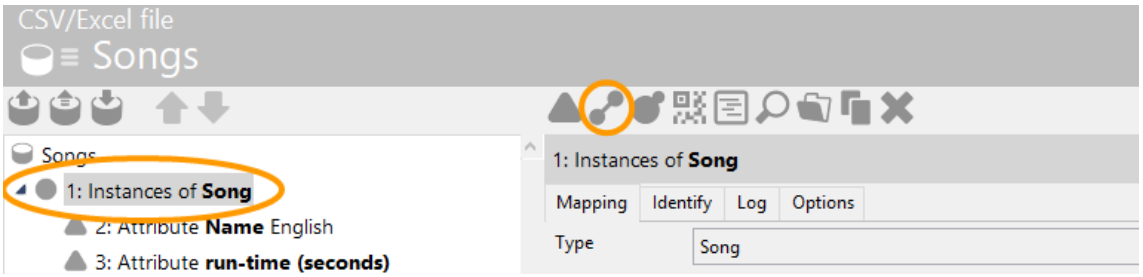
❶ Für jede Sprache eine eigene Attributabbildung desselben Typs anlegen und die Sprache festlegen

❷ Im Reiter "Sprache" für eine der Attributabbildungen die jeweils zugehörigen Attributabbildungen der anderen Sprachen hinzufügen

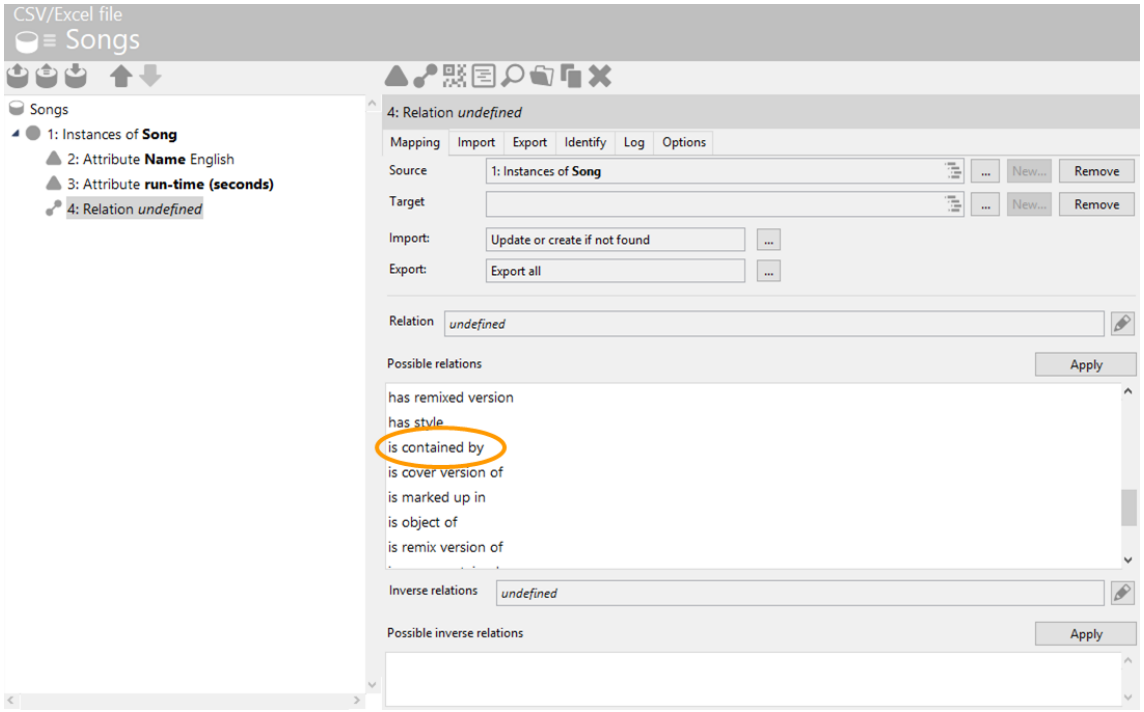
Dies verhindert, dass für jede Übersetzung ein separates Attribut angelegt wird. Zugleich stellt diese Option sicher, dass einander zugehörige Übersetzungen demselben Attribut zugewiesen werden.

1.5.1.3.3. Die Relationsabbildung

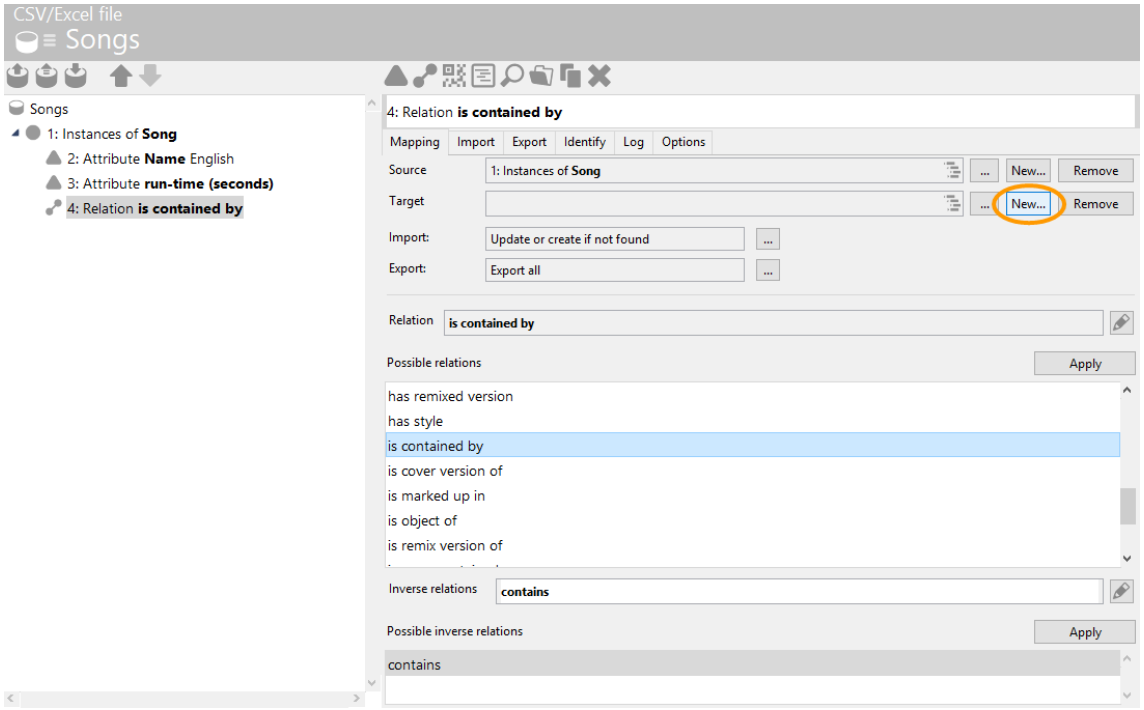
Als nächstes wollen wir das Album abbilden, auf dem der Song sich befindet. Da Alben eigene konkrete Objekte in der semantischen Graph-Datenbank sind, benötigen wir hierfür die Relation, die den Song und das Album verbindet. Um eine Relation abzubilden wählen wir zunächst das Objekt aus, für das die Relation definiert wird und klicken dann auf die Schaltfläche "*Neue Relationsabbildung*".



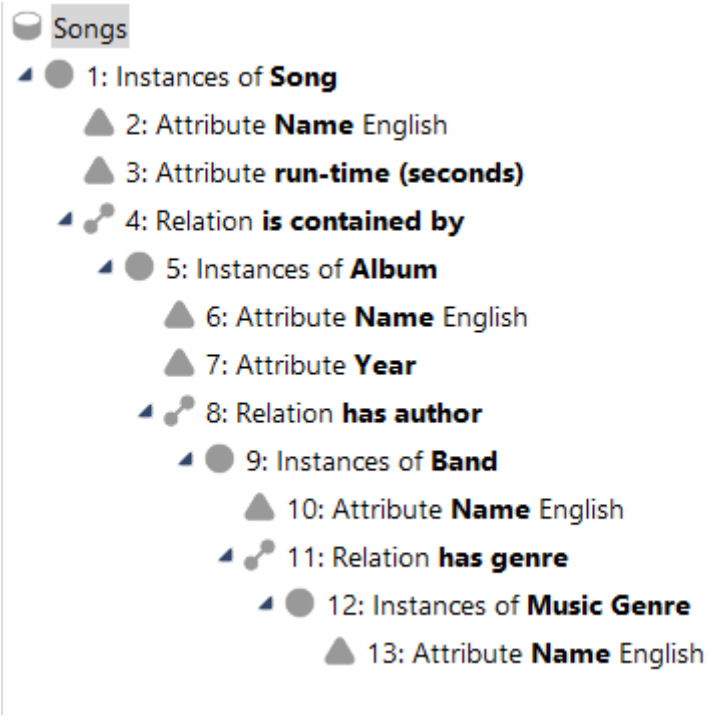
Daraufhin erhalten wir — wie bei den Attributen — eine Liste aller möglichen Relationen, auch die benötigte Relation "*ist enthalten in*" ist selbstverständlich dabei.



Nun müssen wir im nächsten Schritt festlegen, wo aus der Tabelle die Zielobjekte herkommen. Für das Ziel wird eine neue Objektabbildung gebraucht, die über die Schaltfläche "Neu" angelegt wird. Ist der Typ des Zielobjektes eindeutig im Schema definiert, wird dieser automatisch übernommen, ansonsten erscheint eine Liste der möglichen Objekttypen.



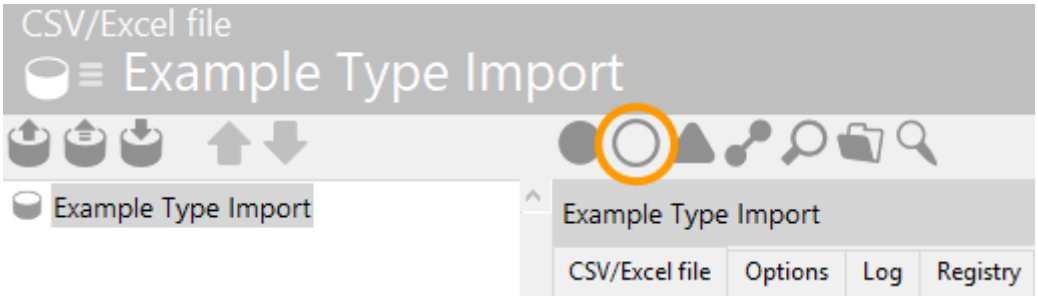
Bei der neuen Objektabbildung müssen wir anschließend wieder das Attribut auswählen, das das Zielobjekt identifiziert usw. So wird die Zielstruktur des Imports aufgebaut.



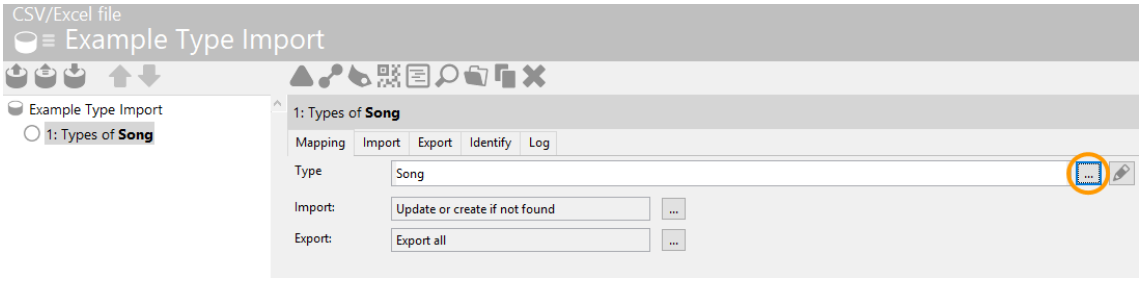
1.5.1.3.4. Die Typabbildung

Auch Typen können importiert und exportiert werden. Nehmen wir beispielhaft an, wir wollten die Genres der Songs als Typen importieren.

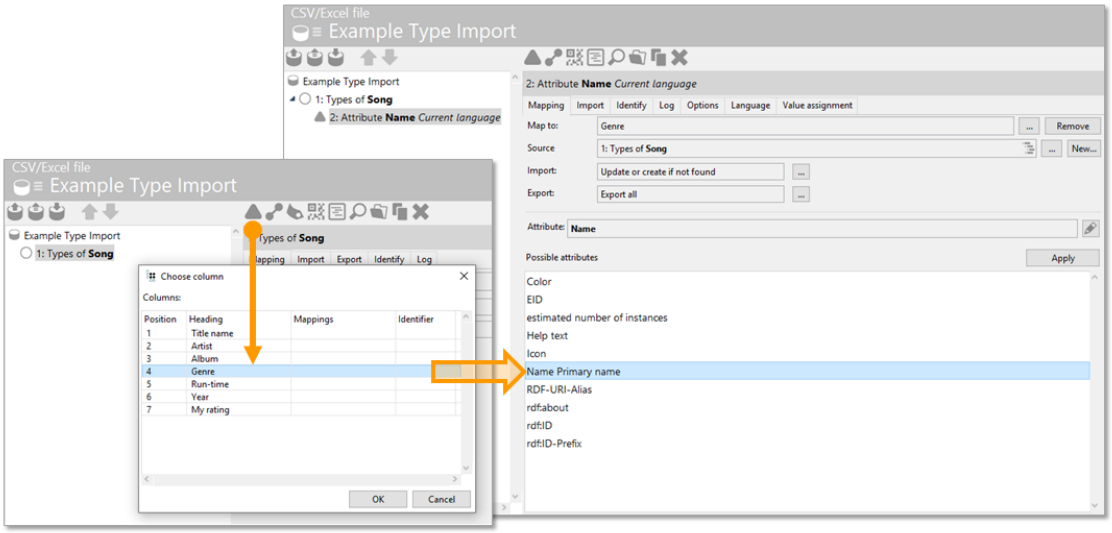
Um einen neuen Typen abzubilden, wählen wir den Button "Neue Typabbildung".



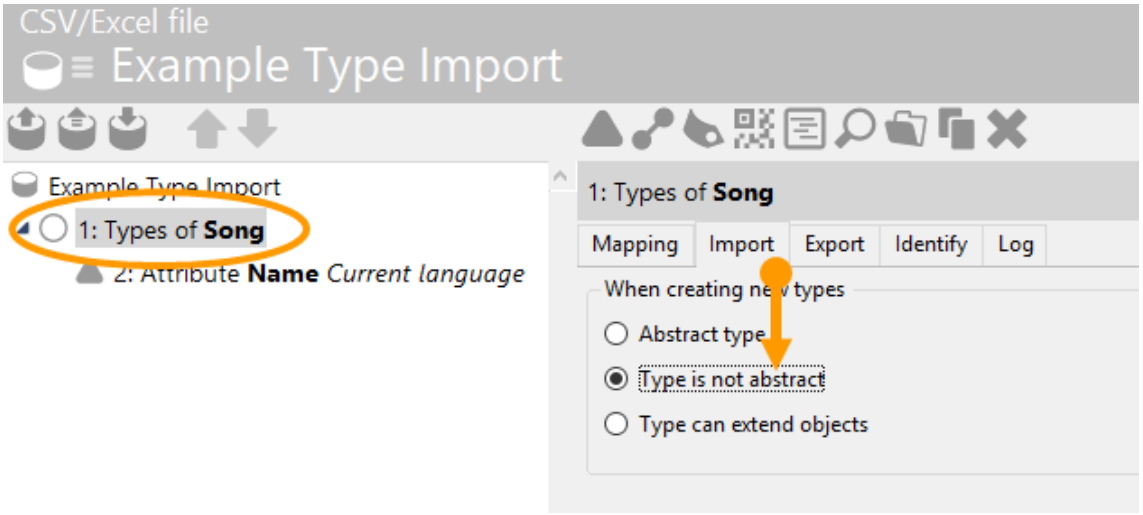
Daraufhin müssen wir den Obertyp der neu anzulegenden Typen angeben, in unserem Beispiel wäre der Obertyp "Song":



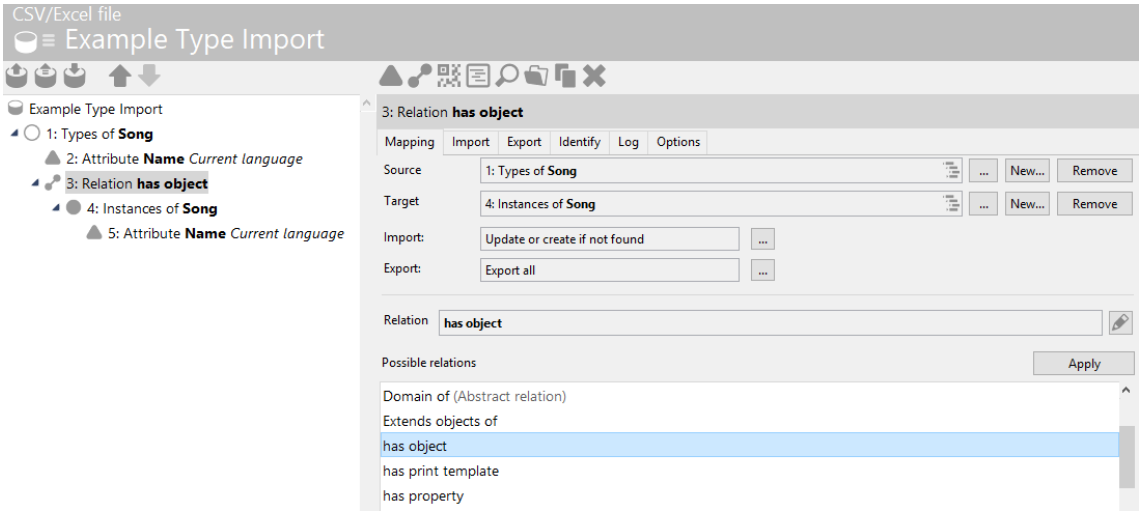
Anschließend müssen wir angeben, aus welcher Spalte der importierten Tabelle der Name unserer neuen Typen entnommen werden soll:



Schließlich müssen wir unter dem Reiter "Import" noch angeben, dass unsere neuen Typen nicht abstrakt sein sollen:



Wenn wir nun die entsprechenden Songs ihren neuen Typen zuordnen wollen, müssen wir die Systemrelation "hat Objekt" verwenden. In älteren Versionen von i-views heißt diese Relation "hat Individuum". Als Ziel wählen wir alle Objekte von Song (inkl. der Untertypen) aus, die sich über das Attribut Name entsprechend der Spalte Songtitel definieren.

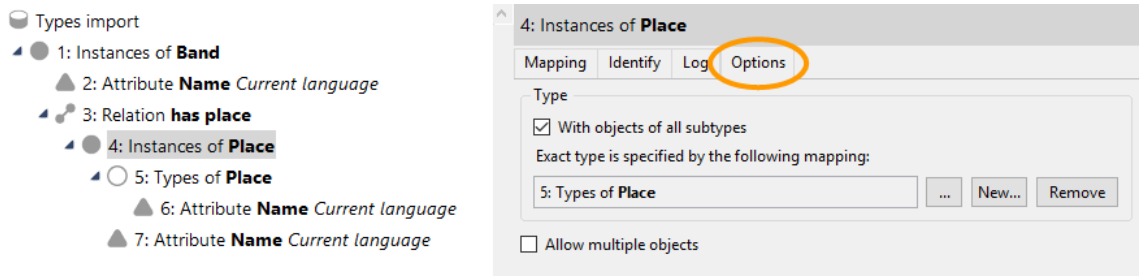


Importieren wir nun diese Abbildung erhalten wir das gewünschte Ergebnis. Die Songs, die bereits in der semantischen Graph-Datenbank vorhanden sind, werden durch die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden" berücksichtigt und unter ihren neuen entsprechenden Typ geschoben, sodass kein Objekt doppelt angelegt wird (siehe Einstellungen des Import-Verhaltens). Zur Erinnerung: Ein konkretes Objekt kann nicht mehreren Typen gleichzeitig angehören.

Es gibt noch einen Spezialfall. Angenommen, wir haben eine Tabelle, in der in einer Spalte verschiedene Typen vorkommen, dann können wir auch dies in unseren Importeinstellungen abbilden.

Person/Band	Origin	Type of location
Paul McCartney	Liverpool	City
The Beatles	Great Britain	Country

Dazu wählen wir die Abbildung der Objekte aus, denen wir die Untertypen zuordnen wollen (in diesem Fall "Objekte von Ort") und wählen dann unter dem Reiter "Optionen" den entsprechenden Obertyp aus.



Wichtig ist auch hier wieder nicht zu vergessen, unter dem Reiter "Import" festzulegen, dass der Typ nicht abstrakt sein soll, damit konkrete Objekte angelegt werden können.

VORSICHT	Angenommen, Liverpool existiert bereits im Knowledge Graph, ist jedoch dem Typ "Ort" zugeordnet, da dieser bis zu diesem Zeitpunkt noch keine

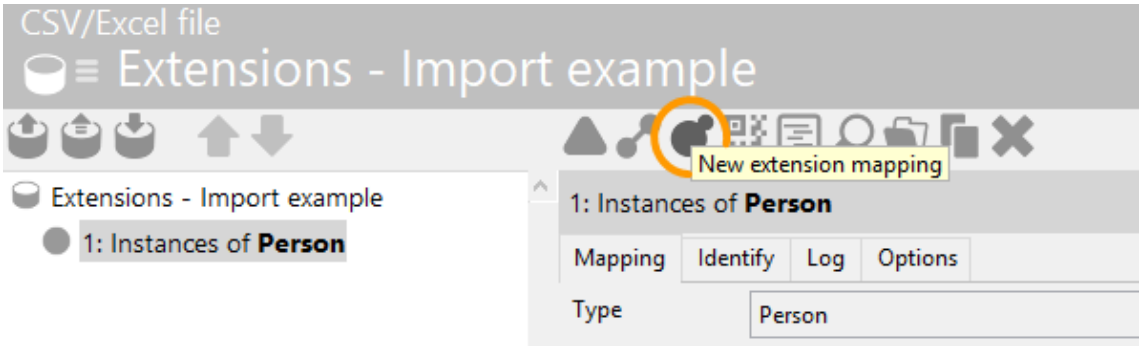
> Untertypen wie "Stadt" und "Land" besessen hat. In diesem Fall wird Liverpool **nicht** unter dem Typ Stadt neu angelegt. Begründung: die Objekte des Typs Ort werden lediglich über das Namensattribut identifiziert, nicht jedoch über den Untertyp.

1.5.1.3.5. Abbildung von Erweiterungen

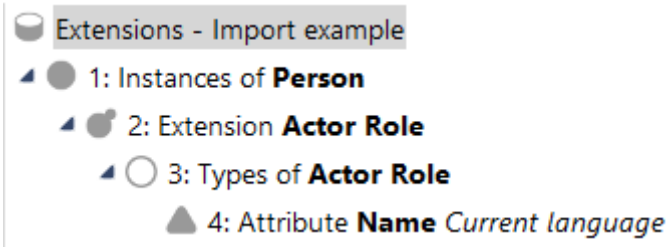
Auch Erweiterungen können importiert und exportiert werden. Angenommen, wir haben eine Tabelle, die die Rolle eines Bandmitglieds in einer Band zeigt:

Person	Band	Rolle
Ron Wood	Faces	Guitarist
Ron Wood	Jeff Beck Group	Bassist
Ron Wood	Rolling Stones	Guitarist

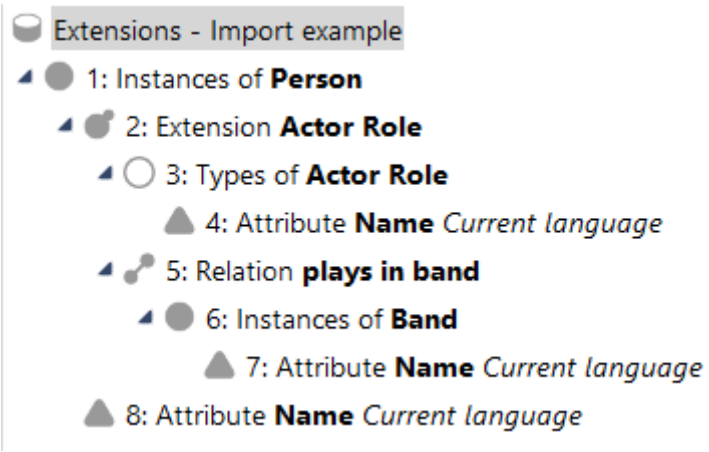
Ron Wood ist Gitarrist bei den Faces und den Rolling Stones aber Bassist bei der Jeff Beck Group. Um dies abzubilden müssen wir das Objekt auswählen, zu dem im Schema eine Erweiterung definiert wurde und dann den Button "Neue Erweiterungsabbildung" betätigen.



Die Abbildung einer Erweiterung fragt — wie eine Objektabbildung — einen zugehörigen Typen ab. Im Schema des Musik-Netzes ist der Typ "Rolle" ein abstrakter Typ. Deswegen muss in der Abbildung definiert werden, dass die Rolle auf Untertypen des Typs "Rolle" abgebildet weden sollen (siehe Die Typabbildung).



Die Relation kann — wie auch bei Objekten und Typen — an der Erweiterung (bzw. an den Untertypen einer Erweiterung) abgebildet werden.



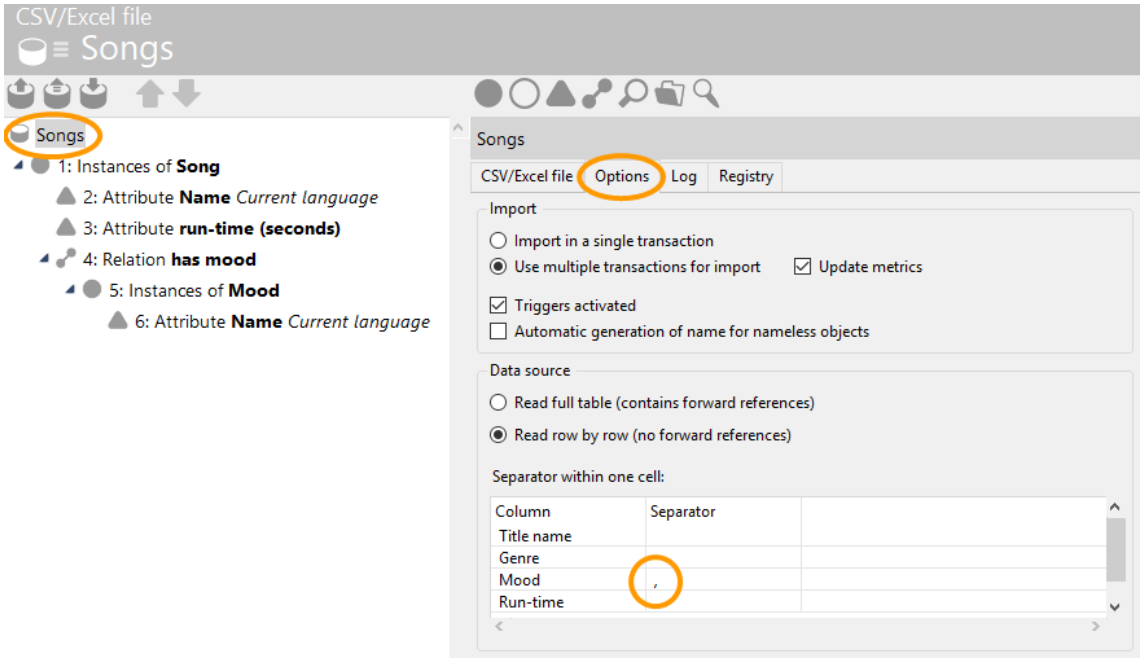
1.5.1.4. Abbildung von mehreren Werten für einen Objekttyp bei einem Objekt

Wenn für einen Objekttyp bei einem Objekt mehrere Werte angegeben sind (in unserem Beispiel etwa mehrere "Moods" für jeden Song), dann gibt es drei Möglichkeiten, wie die Tabelle aussehen kann. Für zwei der drei Möglichkeiten muss der Import angepasst werden, was im Folgenden beschrieben ist.

Möglichkeit 1 — Trennzeichengetrennte Werte: Die einzelnen Werte befinden sich in einer Zelle und sind durch ein Trennzeichen (z.B. ein Komma) getrennt.

	A	B	C	D	E
1	**Title name**	**Genre**	**Mood**	**Run-time**	**Year**
2	Eleanor Rigby	Oldies	reflective, dreamy	127	1966
3	For No One	Oldies	acerbic	121	1966
4	I'm Only Sleeping	Oldies	quirky, mellow	181	1966
5	Yellow Submarine	Oldies	spacey, trippy, playful	160	1966

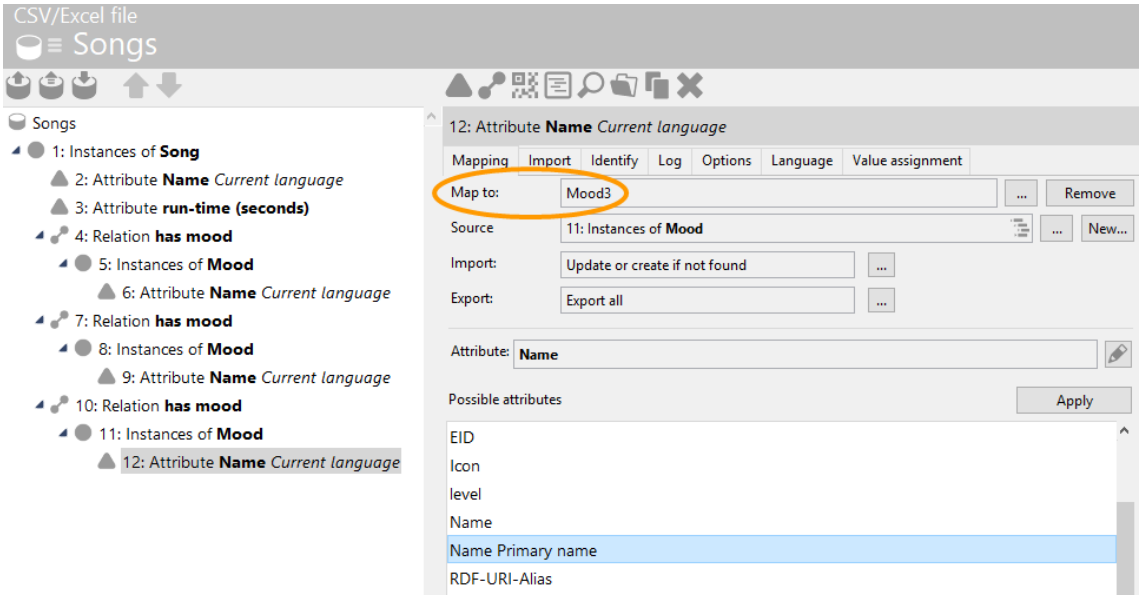
In diesem Fall gehen wir auf die Abbildung der Datenquelle, wo sich die allgemeinen Einstellungen befinden und dort auf den Reiter "*Optionen*". Hier finden wir im unteren Bereich die Einstellungsmöglichkeit, Trennzeichen innerhalb einer Zelle anzugeben. Nun müssen wir nur noch die entsprechende Spalte der zu importierenden Tabelle heraussuchen ("Mood") und in die Spalte "*Trennzeichen*" das verwendete Trennzeichen (",") eingeben.



Möglichkeit 2 — Mehrere Spalten: Die einzelnen Werte befinden sich jeweils in einer eigenen Spalte, wobei nicht jedes Feld ausgefüllt sein muss. Es werden so viele Spalten benötigt, wie maximal Moods pro Song vorhanden sind.

	A	B	C	D	E	F	G
1	**Title name**	**Genre**	**Mood**	**Mood2**	**Mood3**	**Run-time**	**Year**
2	Eleanor Rigby	Oldies	reflective	dreamy		127	1966
3	For No One	Oldies	acerbic			121	1966
4	I'm Only Sleeping	Oldies	quirky	mellow		181	1966
5	Yellow Submarine	Oldies	spacey	trippy	playful	160	1966

In diesem Fall muss die entsprechende Relation so oft angelegt werden, wie Spalten vorhanden sind. In diesem Beispiel müssen demnach die erste Relation auf "Mood1", die zweite Relation auf "Mood2" und die dritte Relation auf "Mood3" abgebildet werden.



Möglichkeit 3 — Mehrere Zeilen: Die einzelnen Werte befinden sich jeweils in einer eigene Zeile. Achtung: Hierbei ist es zwingend nötig, dass die Attribute, die für die Identifizierung des Objektes benötigt werden (in diesem Fall der Titelname), in jeder Zeile auftreten, ansonsten würden die Zeilen als jeweils eigenes Objekt ohne Name gedeutet werden und ein korrekter Import wäre nicht möglich.

	A	B	C	D	E
1	**Title name**	**Genre**	**Mood**	**Run-time**	**Year**
2	Eleanor Rigby	Oldies	reflective	127	1966
3	Eleanor Rigby		dreamy	127	1966
4	For No One	Oldies	acerbic	121	1966
5	I'm Only Sleeping	Oldies	quirky	181	1966
6	I'm Only Sleeping		mellow	181	1966
7	Yellow Submarine	Oldies	spacey	160	1966
8	Yellow Submarine		trippy	160	1966
9	Yellow Submarine		playful	160	1966

In diesem Fall sind keine besonderen Import-Einstellungen nötig, da das System über das identifizierende Attribut das Objekt erkennt und die Relationen korrekt zieht.

1.5.1.5. Einstellungen des Import-Verhaltens

Beim Import-Vorgang wird immer geprüft, ob ein Attribut bereits vorhanden ist. Das "Identifizieren" schließt von Attributen auf die konkreten Objekte. Wenn wir nun im Folgenden von "bereits vorhandenen Attributen" sprechen, dann sind das Attribute, die im Wert genau mit dem Wert aus der Spalte, auf die sie abgebildet sind, übereinstimmen. Wenn wir von bereits vorhandenen Objekten sprechen, dann sind das konkrete Objekte, die durch ein bereits vorhandenes Attribut identifiziert werden.

Beispiel: Wenn in unserem Netz bereits ein Song mit dem Namen "Eleanor Rigby" existiert, dann ist

das Namensattribut (abgebildet auf die Spalte "Titelname" unserer Importtabelle) ein existierendes Attribut und folglich der Song ein existierender Song, solange der Song nur über das Namensattribut identifiziert wird.

Mit den Einstellungen für das Importverhalten können wir steuern, wie der Import auf bereits vorhandene und neue Wissensnetzelemente reagieren soll. Folgende Tabelle zeigt eine Kurzbeschreibung der einzelnen Einstellungen, während die Unterkapitel dieses Kapitels ausführliche und anschauliche Beschreibungen beinhalten.

Einstellung	Kurzbeschreibung
Aktualisieren	Vorhandene Elemente werden überschrieben (aktualisiert), keine neuen Elemente werden angelegt.
Aktualisieren oder neu anlegen wenn nicht vorhanden	Vorhandene Elemente werden überschrieben, sollten keine vorhanden sein, werden sie neu angelegt.
Alle mit selbem Wert löschen (nur bei Eigenschaften verfügbar)	Alle Attributwerte, die mit dem importierten Wert übereinstimmen, werden für die jeweils entsprechenden Objekte gelöscht.
Alle vom selben Typ löschen	Alle Attributwerte des ausgewählten Typs werden für die entsprechenden Objekte gelöscht, unabhängig davon, ob die Werte übereinstimmen oder nicht.
Löschen	Wird verwendet, um genau das eine Element zu löschen.
Neu anlegen	Legt eine neue Eigenschaft/Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist.
Neu anlegen wenn Typ nicht vorhanden (nur bei Attributen verfügbar)	Nur, wenn noch kein Attribut des gewünschten Typs vorhanden ist, wird eines angelegt.
Neu anlegen wenn Wert nicht vorhanden (nur bei Attributen verfügbar)	Nur, wenn noch kein Attribut mit dem zu importierenden Wert vorhanden ist, wird eines angelegt.
Nicht importieren	Kein Import.
Synchronisieren	Um die zu importierenden Inhalte mit den Inhalten der Datenbank zu synchronisieren, werden bei dieser Aktion alle Elemente, die noch nicht vorhanden sind, neu angelegt, alle die sich geändert haben, aktualisiert und alle, die nicht mehr vorhanden sind, gelöscht.

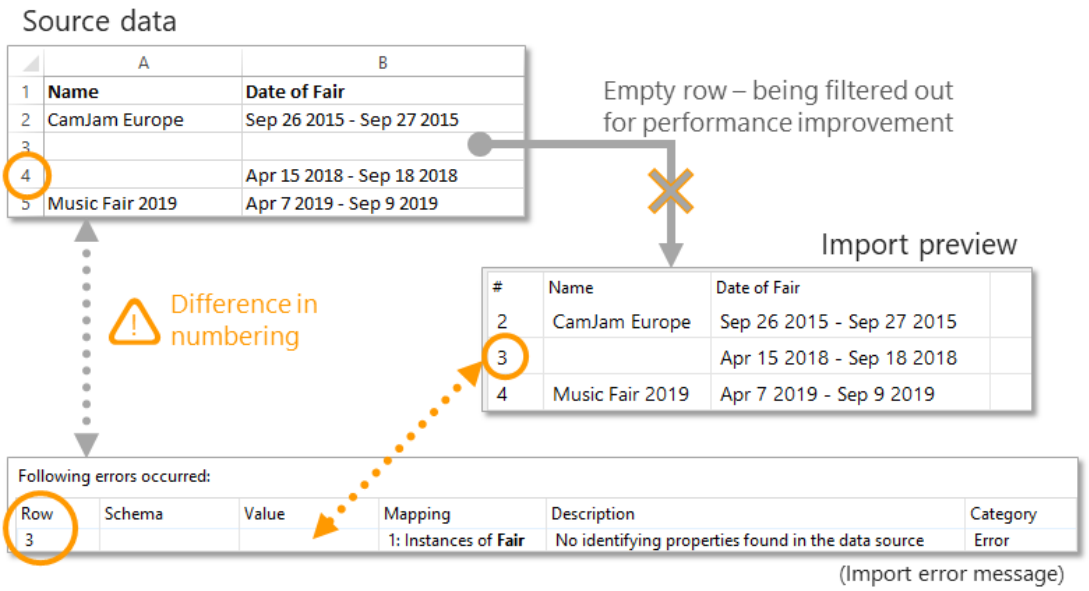
Bei einem Import müssen wir uns für jedes abgebildete Objekt, jede abgebildete Relation und jedes abgebildete Attribut einzeln entscheiden, welche Import-Einstellung wir jeweils verwenden wollen.

HINWEIS	Anders als in anderen Editoren des Knowledge-Buildes "vererbt" sich eine Einstellung nicht an die darunterliegenden Abbildungselemente. Auch die Import-Einstellung für ein Objekt "vererbt" sich nicht an seine Attribute.

Das Importmapping

Wenn aufgrund der Datenlage Fehler auftreten sollten, werden sie nach der Import-Transaktion in gesammelter Form zurückgemeldet. Die jeweilige Fehlermeldung bezieht sich dabei auf die Zeilennummer der Vorschau-Tabelle, welche durch Klick auf "Tabelle anzeigen" betrachtet werden kann.

Wenn in der Quelldatei leere Zeilen vorhanden sind, werden sie vom Importmechanismus herausgefiltert. Aus diesem Grund kann es im Falle leerer Zeilen in der Quelldatei vorkommen, dass die Zeilennummerierung der Quelldatei von der Zeilennummerierung im Importmapping (einschließlich der Vorschau) abweicht.



1.5.1.5.1. Aktualisieren

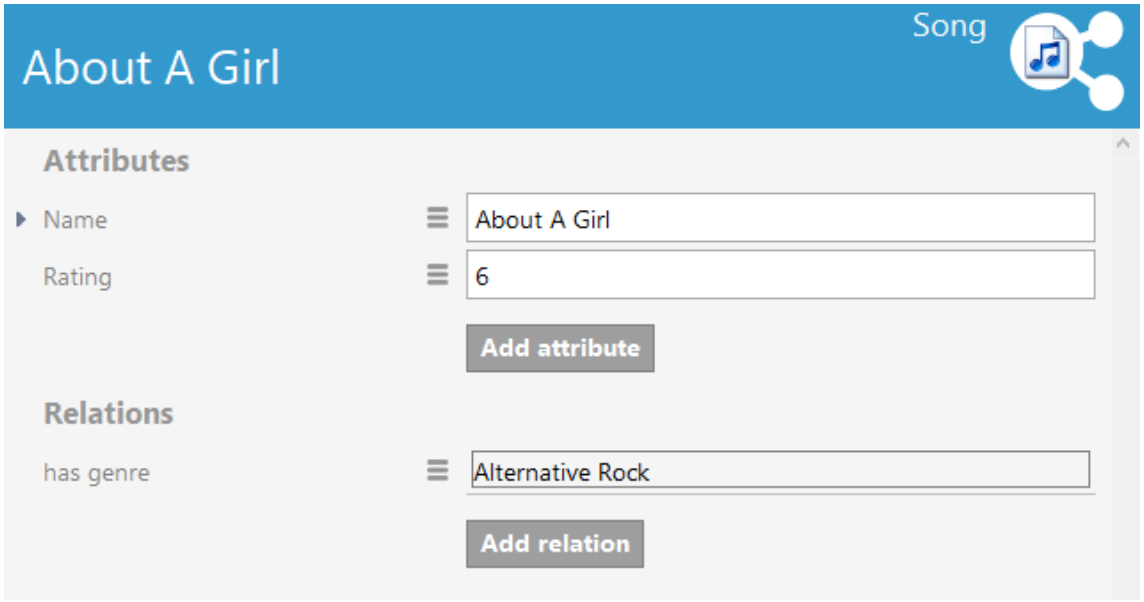
Wird diese Einstellung bei einem **Attribut** angewendet, sorgt sie dafür, dass der Wert aus der Tabelle den Attributwert genau eines bereits existierenden Attributs überschreibt. Mit dieser Einstellung werden keine neuen Attribute angelegt. Falls das Objekt mehr als einen Attributwert des ausgewählten Typs hat, wird kein Wert importiert.

Verwendet man die Einstellung "Aktualisieren" bei einem identifizierenden Attribut, während man bei dem dazugehörigen Objekt die Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet, erscheint die Fehlermeldung "Attribut nicht gefunden", wenn das identifizierende Attribut nicht in i-views vorhanden ist.

Wird "Aktualisieren" bei einem **Objekt** angewendet, sorgt die Einstellung dafür, dass alle Eigenschaften des Objekts durch den Import hinzugefügt, bzw. geändert werden können. Neue Objekte werden nicht angelegt.

Beispiel: Angenommen wir führen eine Datenbank mit unseren Lieblings-Songs. Nun haben wir eine Liste mit Songs, die neue Informationen, enthalten. Wir wollen diese Informationen in unsere

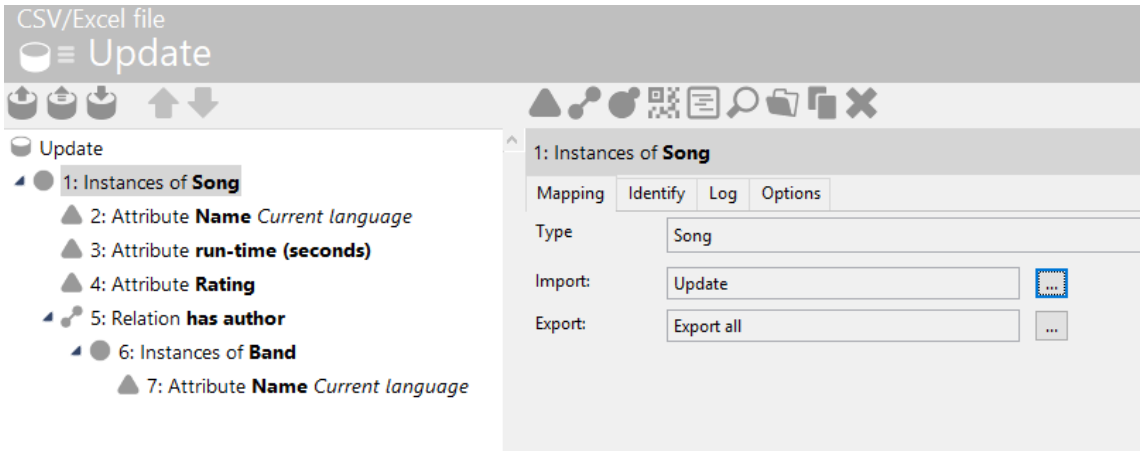
Datenbank bringen, gleichzeitig aber nicht, dass Songs importiert werden, die nicht zu unseren Lieblings-Songs gehören. Hierfür verwenden wir die Einstellung "Aktualisieren".



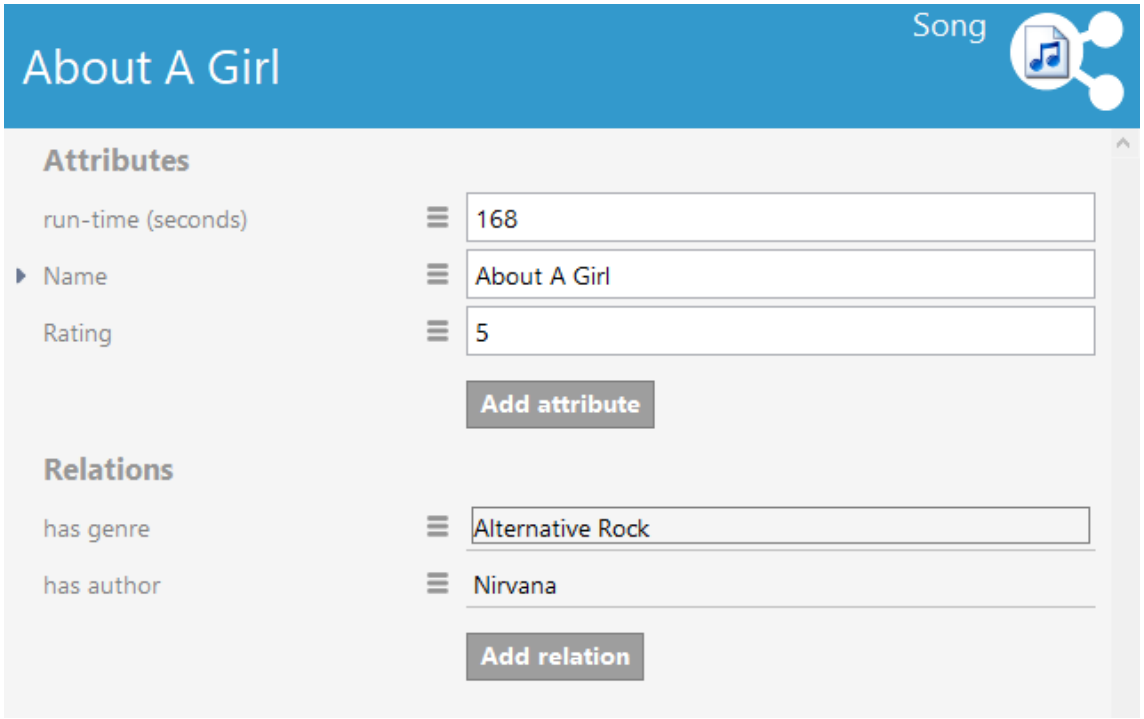
Der Song "About A Girl" ist bereits im Knowledge-Builder vorhanden.

Song	Run-time	Rating	Author
About A Girl	168	5	Nirvana

Die Import-Tabelle enthält Angaben zu Dauer, Wertung und Autor des Songs.



Wir legen für Objekte von Song fest, dass sie aktualisiert werden sollen. Alle Attribute, Relationen und Relationsziele erhalten die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden".



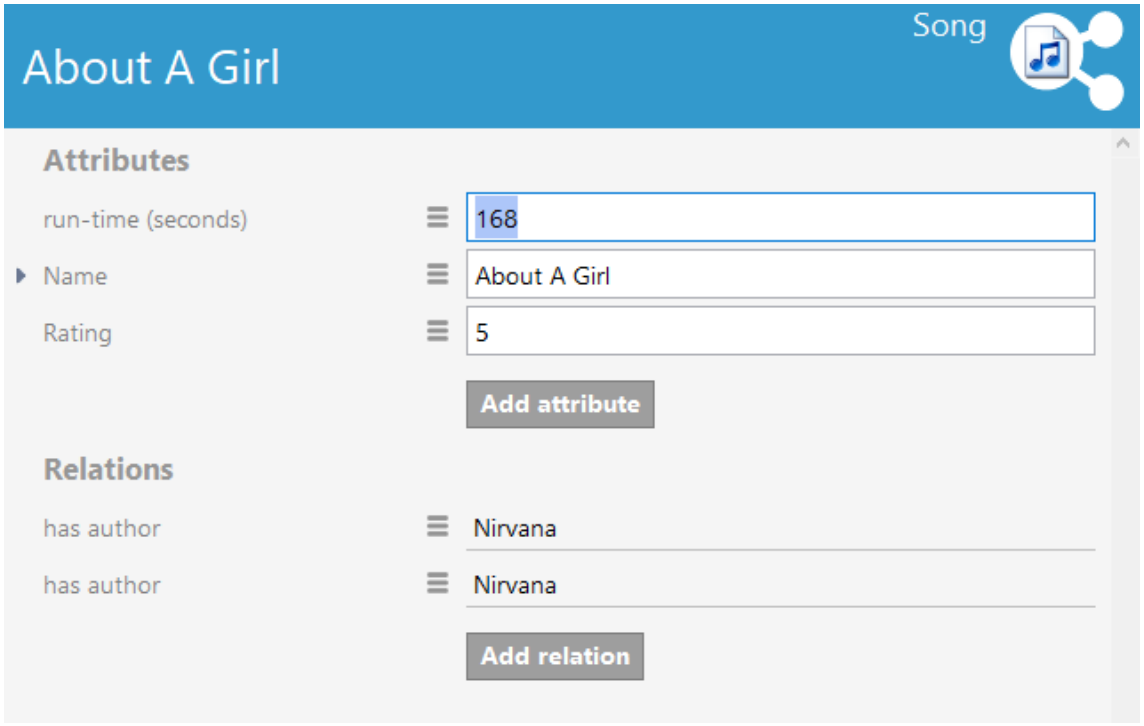
Das Ergebnis: Der Song wurde aktualisiert und hat nun neue Attribute und Relationen erhalten. Bereits vorhandene Eigenschaften wurden aktualisiert (Wertung).

1.5.1.5.2. Aktualisieren oder neu anlegen wenn nicht vorhanden

Diese Import-Einstellung wird in den meisten Fällen benötigt und ist darum als Standard-Einstellung gesetzt. Wenn Elemente bereits vorhanden sind, werden sie aktualisiert. Wenn Elemente noch nicht vorhanden sind, werden sie in der Datenbank neu angelegt.

1.5.1.5.3. Alle mit selben Wert löschen

Diese Import-Einstellung ist nur bei Eigenschaften (Relationen und Attributen) verfügbar und wird nur verwendet, wenn über die Import-Einstellung "Löschen" nicht gelöscht werden kann. Mit "Löschen" kann dann nicht gelöscht werden, wenn eine Relation oder ein Attribut bei einem Objekt mehrmals mit denselben Werten vorkommt. Versucht man es dennoch, erscheint eine Fehlermeldung. Zum Beispiel kann es sein, dass der Song "About A Girl" versehentlich zweimal mit der Band "Nirvana" über die Relation "hat Autor" verknüpft wurde.



In solchen Fällen greift die Import-Einstellung "Löschen" nicht, da sie bei Mehrfachvorkommen nicht weiß, welche der Relationen sie löschen soll. Hier muss also "Alle mit selben Wert löschen" verwendet werden.

1.5.1.5.4. Alle vom selben Typ löschen

Diese Import-Einstellung wird verwendet, wenn alle Attribute, Objekte oder Relationen eines Typs gelöscht werden sollen, unabhängig von den vorhandenen Werten. Im Gegensatz dazu berücksichtigen die Einstellungen "Löschen" und "Alle mit selben Wert löschen" die vorhandenen Werte. Es werden nur die Elemente der Objekte gelöscht, die in der Import-Tabelle vorkommen.

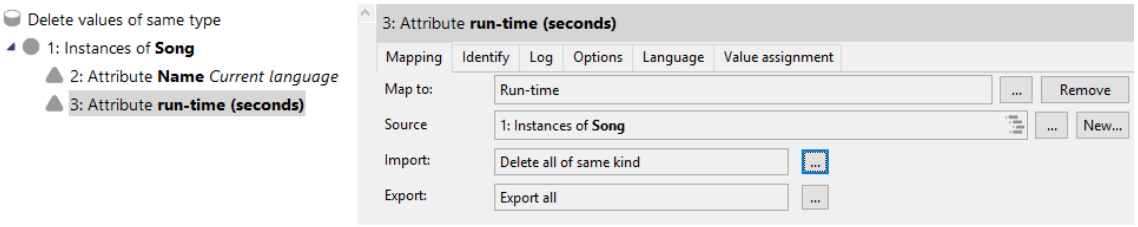
Beispiel: Wir haben eine Import-Tabelle mit Songs und der Dauer der Songs. Wir sehen, dass sich die Dauer in vielen Fällen unterscheidet und beschließen, die Dauer für diese Songs zu löschen, damit wir keinesfalls falsche Angaben haben.

Song	Run-time
19th Nervous Breakdown	113
A Maniac Depressive Named Laughing Boy	300
A Place For My Head	249
About A Girl	168

Die Dauer in der Import-Tabelle unterscheidet sich bei den meisten Songs…

Name	[3] run-time (seconds)
19th Nervous Breakdown	113
A Manic Depressive Named Lauging Boy	300
A Place for my Head	249
About A Girl	168

… von der Dauer der Songs in der Datenbank.



Beim Attribut "Dauer" verwenden wir die Import-Einstellung "Alle vom selben Typ löschen".

Name	run-time (seconds)
19th Nervous Breakdown	
A Manic Depressive Named Lauging Boy	
A Place for my Head	
About A Girl	

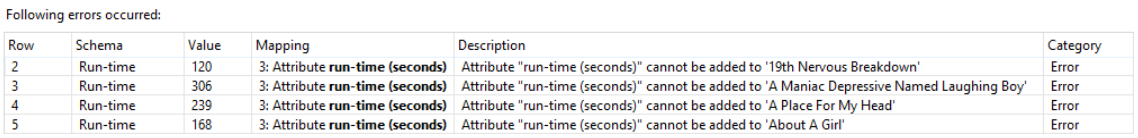
Nach dem Import, sind alle Attributwerte des Attributtyps Dauer für diese 4 Songs gelöscht.

1.5.1.5.5. Löschen

Die Import-Einstellung "Löschen" wird verwendet, um genau das eine Objekt / genau die eine Relation / genau den einen Attributwert zu löschen. Falls keine oder mehrere Objekte / Relationen / Attributwerte mit den zu importierenden Elementen übereinstimmen, erscheint diesbezüglich eine Fehlermeldung und die betroffenen Elemente werden nicht gelöscht.

1.5.1.5.6. Neu anlegen

Diese Import-Einstellung legt eine neue Eigenschaft / ein neues Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist. Einzige Ausnahme: Sollte eine Eigenschaft nur einmal vorkommen (man beachte die Einstellung "kann mehrfach vorkommen" bei der Attributdefinition), so wird das neue Attribut nicht angelegt und eine Fehlermeldung erscheint, die dies mitteilt.


Following errors occurred:

Row	Schema	Value	Mapping	Description	Category
2	Run-time	120	3: Attribute **run-time (seconds)**	Attribute "run-time (seconds)" cannot be added to '19th Nervous Breakdown'	Error
3	Run-time	306	3: Attribute **run-time (seconds)**	Attribute "run-time (seconds)" cannot be added to 'A Maniac Depressive Named Laughing Boy'	Error
4	Run-time	239	3: Attribute **run-time (seconds)**	Attribute "run-time (seconds)" cannot be added to 'A Place For My Head'	Error
5	Run-time	168	3: Attribute **run-time (seconds)**	Attribute "run-time (seconds)" cannot be added to 'About A Girl'	Error

1.5.1.5.7. Neu anlegen wenn Typ nicht vorhanden

Diese Import-Einstellung ist nur bei Attributen verfügbar. Ein neuer Attributwert wird nur angelegt, wenn das entsprechende Attribut noch keinen Wert hat. Die Werte müssen nicht gleich sein, es geht nur um das Vorhandensein, bzw. Nichtvorhandensein irgendeines Wertes des entsprechenden Attributtyps. Der Import mehrerer Attributwerte gleichzeitig auf einen Attributtyp ist nicht möglich, da hier nicht entschieden werden kann welcher der Attributwerte verwendet werden soll.

Beispiel: Angenommen, wir haben eine Import-Tabelle, die Musiker mit ihren alias-Namen beinhaltet. Einige Musiker haben auch mehrere alias-Namen. Hier können wir die Einstellung "Neu anlegen wenn Typ nicht vorhanden" nicht verwenden, da dann alle Musiker mit mehreren alias-Namen keinen erhalten würden.

1.5.1.5.8. Neu anlegen wenn Wert nicht vorhanden

Diese Import-Einstellung ist nur bei Attributen verfügbar. Ein neuer Attributwert wird nur angelegt, wenn das Objekt diesen Wert für das entpsrechende Attribut noch nicht hat.

Beispiel: Nehmen wir noch einmal die Import-Tabelle, die Musiker mit ihren alias-Namen beinhaltet. Hier können wir die Einstellung "Neu anlegen wenn Wert nicht vorhanden" verwenden, da dann die Musiker mit mehreren alias-Namen alle diese alias-Namen erhalten können.

1.5.1.5.9. Nicht importieren

Mit der Import-Einstellung "Nicht importieren" können wir sagen, dass ein Objekt oder eine Eigenschaft nicht importiert werden soll. Dies ist dann nützlich, wenn wir bereits eine Abbildung definiert haben und diese wiederverwenden wollen, jedoch bestimmte Objekte und Eigenschaften nicht noch einmal importieren wollen.

1.5.1.5.10. Synchronisieren

Die Import-Einstellung "Synchronisieren" ist mit Vorsicht zu genießen, denn sie betrifft als einzige Import-Einstellung nicht nur die Objekte und Eigenschaften in i-views, die in ihren Werten mit denen der Import-Tabelle übereinstimmen, sondern darüber hinaus alle Elemente des gleichen Typs in i-views. Wenn man eine Import-Tabelle mit i-views synchronisiert, bedeutet das prinzipiell, dass das Ergebnis in i-views nach dem Import exakt so aussehen soll wie in der Tabelle.

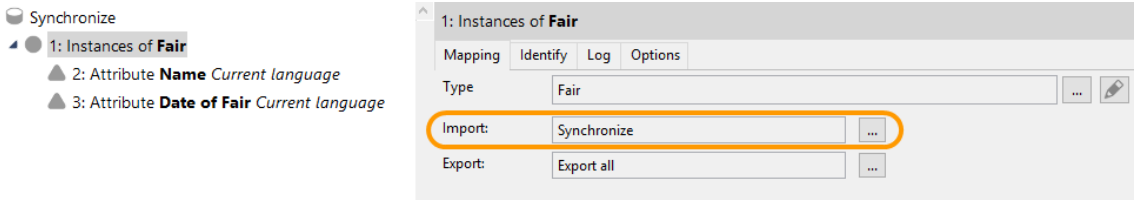
VORSICHT	Wenn Objekte eines Typs synchronisiert werden, werden alle Objekte dieses Typs gelöscht, die nicht in der Import-Tabelle vorkommen. Die Objekte, die vorkommen, werden aktualisiert und die Objekte, die nicht in i-views vorkommen, werden neu angelegt.

Beispiel: Wir wollen die Musikmessen in i-views (links) mit einer Tabelle mit den Messen und ihrem Datum (rechts) synchronisieren:

Name	Date of Fair
CamJam Europe	Sep 26 2015 - Sep 27 2015
choir.com Fair	Apr 1 2015 - Apr 4 2015
Music Fair 2018	Apr 15 2018 - Sep 18 2018
Music Fair 2019	Apr 7 2019 - Sep 9 2019

	A	B
1	**Name**	**Date of Fair**
2	CamJam Europe	Sep 26 2015 - Sep 27 2015
3		
4		Apr 15 2018 - Sep 18 2018
5	Music Fair 2019	Apr 7 2019 - Sep 9 2019

Für die Objekte des Typs Messe wählen wir die Import-Einstellung "Synchronisieren", für die einzelnen Attribute *Name* und *Messedatum* wird die Import-Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet:



Das Attribut Name ist das identifizierende Attribut von Messe. Für das Objekt Musikmesse 2015 fehlt der Name in der Import-Tabelle. Importieren wir die Tabelle so, erhalten wir dazu eine Fehlermeldung:

Following errors occurred:

Row	Schema	Value	Mapping	Description	Category
3			1: Instances of **Fair**	No identifying properties found in the data source	Error

Nach dem Import sehen wir nun, dass durch den Import zwei Objekte entfallen sind, die keine Entsprechung in der Import-Tabelle hatten. Das Datum bei Musikmesse 2016 wurde aktualisiert:

Name	Date of Fair
CamJam Europe	Sep 26 2015 - Sep 27 2015
Music Fair 2019	Apr 7 2019 - Sep 9 2019

Wenn **Attribute** synchronisiert werden gilt folgendes: Wenn ein bestehendes Attribut durch einen Import keinen Wert erhält, wird es für das entsprechende Objekt der Import-Tabelle gelöscht. Wenn das bestehende Attribut einen anderen Wert hat als in der Import-Tabelle, wird es aktualisiert, auch dann, wenn es mehrmals vorkommen darf. Wenn das Attribut noch nicht vorhanden ist, wird es neu angelegt.

Wenn **Relationen** synchronisiert werden, und sie erhalten keinen Wert werden sie für das entsprechende Objekt gelöscht. Wenn die bestehende Relation einen anderen Wert hat, als in der Import-Tabelle, wird sie aktualisiert. Sollte es das Zielobjekt noch nicht in der Datenbank geben, wird es neu angelegt, vorausgesetzt, das Zielobjekt hat eine entsprechende Import-Einstellung zugewiesen bekommen. Kann das Zielobjekt nicht neu angelegt werden, da hier beispielsweise die Import-Einstellung "Aktualisieren" zugewiesen wurde erscheint eine Fehlermeldung, die uns

mitteilt, dass das Zielobjekt nicht gefunden wurde und es wird nicht neu angelegt.

1.5.2. Konfiguration von Datenquellen

1.5.2.1. Tabellenspalten

Bei Abbildungen von Datenbank-Queries sind die Spalten, die zum Import zur Verfügung stehen, durch die Datenbanktabellen bzw. durch das Select-Statement vorgegeben. Beim Abbilden von Dateien können die Spalten mit der Schaltfläche "Aus Datenquelle lesen" aus der Datei übernommen werden. Man kann sie aber auch von Hand angeben. Dann hat man die Wahl, ob man eine Standard-Spalte oder eine virtuelle Eigenschaft anlegen möchte.

Will man aus der semantischen Graph-Datenbank exportieren, muss man die Spalten von Hand eingeben. Es können nur Standard-Spalten, nicht jedoch virtuelle Spalten exportiert werden.

Virtuelle Tabellenspalte / Virtuelle Eigenschaft Virtuelle Spalten sind zusätzliche Spalten, die es erlauben die Inhalte, die wir in einer Spalte der zu importierenden Tabelle vorfinden, mit regulären Ausdrücken zu transformieren. Beispiel: Nehmen wir an in unserer Import-Tabelle steht bei den Jahreszahlen immer ein a.d. dahinter. Das können wir bereinigen, indem wir eine virtuelle Spalte anlegen, die aus der Spalte Jahr nur die ersten 4 Zeichen übernimmt.

Auch beim Export können wir virtuelle Eigenschaften definieren.

Den Ausdruck schreiben wir dabei einfach in die Spaltenüberschrift (in den Namen der Spalte). Dabei werden Teilzeichenketten, die in spitze Klammern <…> eingeschlossen sind, nach den folgenden Regeln ersetzt, wobei *n, n1, n2, …* für die Inhalte anderer Tabellenspalten mit der Spaltennummer n stehen.

Ausdruck	Beschreibung	Beispiel	Eingabe	Ausgabe
<n **p** >	Druckausgabe des Inhalts von Spalte n	Treffer: <1p>	1 (integer)	Treffer: 1
			"keine" (String)	Treffer: "keine"
<n **s** >	Ausgabe der Zeichenkette in Spalte n	Hallo <1s>!	"Peter"	Hallo Peter!
<n **u** >	Ausgabe der Zeichenkette in Spalte n in Großbuchstaben	Hallo <1u>!	"Peter"	Hallo PETER!
<n **l** >	Ausgabe der Zeichenkette in Spalte n in Kleinbuchstaben	Hallo <1l>!	"Peter"	Hallo peter!
<n **c** start-stop>	Teilzeichenkette von Position start bis stop aus Spalte n	<1c3-6>	"Spalten"	alte
		<1c3>	"Spalten"	ten
		<1c3->	"Spalten"	alten

Ausdruck	Beschreibung	Beispiel	Eingabe	Ausgabe
<n **m** regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck zutrifft.	<1m0[0-9]>hi	01	hi
			123	(leer)
		<1m$>test	(leer)	test
			123	(leer)
<n **x** regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck **nicht** zutrifft.	<1x0[0-9]>hallo	01	(leer)
			123	hallo
<n **e** regex>	Selektiert alle Treffer von regex aus dem Inhalt von Spalte n. Einzeltreffer sind im Ergebnis durch Komma voneinander getrennt.	<1eL+>	HELLO WORLD	LL,L
		<1e\d\d\d\d>	02.10.2001	2001
<n **r** regex>	Entfernt alle Treffer von regex aus dem Inhalt von Spalte n	<1rL>	HELLO WORLD	HEO WORD
<n **g** regex>	Überträgt den Inhalt aller Gruppen des regulären Ausdrucks	<1g\(\d)->	+42-13	42
<n **f** format>	Formatiert Zahlen, Datums- und Zeitangaben aus Spalte n gemäß der Formatangabe "format"	<1f#,0.00>	3,1412	3,14
			1.234,50	1234,5
		<1fd/m/y>	1. Mai 1935	1/5/1935
		<1fdd/mmm>	1. Mai 1935	01/Mai

Tabellenspalten können auch unabhängig von ihrer Spaltennummer referenziert werden, indem eigens definierte Bezeichner verwendet werden. Der Vorteil hierin ist, dass bei einer Änderung der Spalten-Reihenfolge der Importtabelle die Zuordnung nicht verloren geht.

Der Bezeichner für die jeweilige Spalte der Importtabelle wird in die Spalte mit der Überschrift *Bezeichner* der Spaltendefinitionstabelle eingetragen. Referenziert werden diese Spalten durch Anlegen einer virtuellen Tabellenspalte, die den Bezeichner als Tabellenspalten-Überschrift enthalten (siehe Beispiel 2).

Ausdruck	Beschreibung	Beispiel	Eingabe	Ausgabe
<$ name **$Ausdruck**>	Referenz auf eine Spalte mittels eindeutigem Spalten-Bezeichner *name* und anschließender Transformierung durch den Ausdruck.Die **$** -Zeichen sind funktioneller Bestandteil der Bezeichner-Syntax.	<$Name$u>	"mp3"	MP3

Mehr Information zu regulären Ausdrücken finden sich unter https://regex101.com/.

Beispiel 1: Verwendung von Ausdrücken (Referenz über Spaltennummer)

Angenommen wir haben eine Import-Tabelle, in der konkrete Objekte ohne Namen vorkommen. In unserem Datenmodell sollen diese Objekte jedoch als eigene Objekte modelliert werden. Ein Beispiel: zu einem Lastpunkt steht in Spalte 88 sein Hauptwert, der Drehmoment. Als Definition unserer virtuellen Spalte, die für den Namen dieses Lastpunktes stehen soll, geben wir also den Ausdruck *Lastpunkt <88s>* ein. Der daraus entstehende Name für einen Lastpunkt mit dem Drehmoment von 850 wäre demnach "Lastpunkt 850".

Wir können die virtuelle Eigenschaft auch nutzen, um einen Usernamen herzustellen, der aus den ersten 4 Buchstaben des Vornamens und des Nachnamens zusammengesetzt ist. Heißt die Person Maximilian Mustermann und wir definieren die virtuelle Spalte mit dem entsprechenden Ausdruck <1c1-4><2c1-4>, erhalten wir das Ergebnis "MaxiMust".

Die virtuelle Eigenschaft kann auch dazu genutzt werden, einem User beim Import ein initiales Passwort anzulegen. Der Ausdruck könnte *Pass4<2s>* lauten. Das daraus resultierende Passwort für Maximilian Mustermann wäre "Pass4Mustermann".

Ein etwas umfangreicheres Beispiel zeigt, wie die virtuelle Eigenschaft dazu genutzt werden kann, Objekten eine Obergruppe von Nummern zuzuordnen:

#	Media	Item number	Title	Artist	Genre	<1mCD><2c1-3>000	<1xCD><1xMD><2c1-4>00	Playlist Summer 2019
2	CD	010000	The suburbs	Arcade Fire	Postwave	010000		Playlist Summer 2019
3	CD	010100	Modern Man	Arcade Fire	Postwave	010000		Playlist Summer 2019
4	LP	010101	Empty Room	Arcade Fire	Postwave		010100	Playlist Summer 2019
5	MP3	010102	Half Light I	Arcade Fire	Postwave		010100	Playlist Summer 2019
6	OGG	010103	Half Light II	Arcade Fire	Postwave		010100	Playlist Summer 2019
7	LP	010104	Month Of Day	Arcade Fire	Postwave		010100	Playlist Summer 2019
8	LP	010105	Deep Blue	Arcade Fire	Postwave		010100	Playlist Summer 2019
9	LP	010106	Eleanor Rigby	The Beatles	Oldies		010100	Playlist Summer 2019
10	LP	010107	I Want To Tell You	The Beatles	Oldies		010100	Playlist Summer 2019
11	LP	010109	I'm Only Sleeping	The Beatles	Oldies		010100	Playlist Summer 2019
12	LP	010110	Love To You	The Beatles	Oldies		010100	Playlist Summer 2019
13	MD	010200	Taxman	The Beatles	Oldies			Playlist Summer 2019
14	LP	010201	About A Girl	Nirvana	Rock		010200	Playlist Summer 2019

Die drei rechten Spalten sind virtuelle Spalten.

<1mUG>: In die erste der virtuellen Spalten wird die Nummer der Obergruppe des Objekts nur geschrieben, wenn der Begriff "CD" (für Compact Disc) in der ersten Spalte für das Objekt vorkommt.

<2c1-3>000: Die Nummer die in die Spalte geschrieben werden soll, setzt sich aus den ersten drei Zeichen der zweiten Spalte zusammen und drei Nullen.

<1xCD><1xMD>: Nur wenn die erste Spalte die Begriffe "CD" und "MD" nicht enthält, wird die Nummer der Obergruppe des Objekts in die Spalte geschrieben.

<2c1-4>00: Die Nummer, die in die Spalte geschrieben werden soll, setzt sich aus den ersten vier Zeichen der zweiten Spalte zusammen.

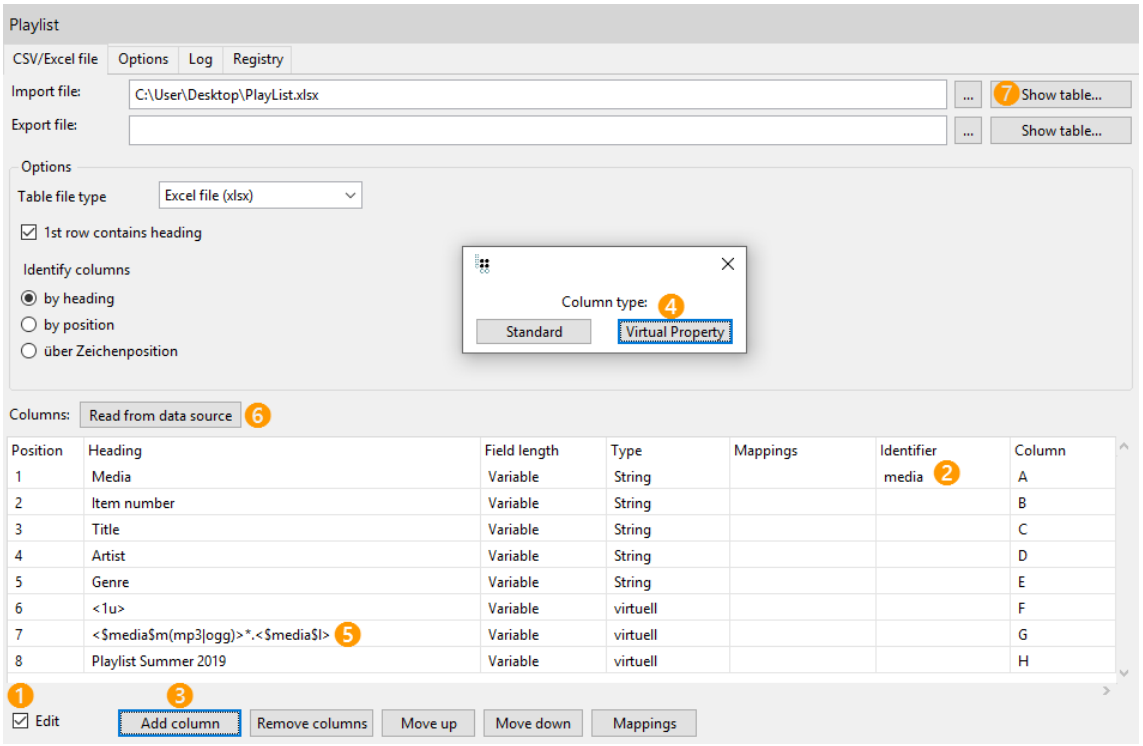
Playlist Summer 2019: Dieser Ausdruck wird für alle Objekte in die Spalte geschrieben.

Beispiel 2: Verwendung individueller Bezeichner (in Kombination mit regulären Ausdrücken)

Im folgenden Beispiel werden die Inhalte der Spalte Media mittels virtueller Spalten in Großbuchstaben und in Dateinamenserweiterungen transformiert: Spalte 6 referenziert mittels Spaltennummer, Spalte 7 referenziert mittels Spaltenbezeichner.

Für das Einrichten der Spaltenbezeichner und der virtuellen Spalten wie folgt vorgehen:

1. Zuerst den "Bearbeiten"-Modus aktivieren
2. Der Spalte den Spaltenbezeichner vergeben (Spaltenbezeichner "media" wird immer der Spalte mit der Überschrift "Media" zugeordnet bleiben)
3. Auf die Schaltfläche "Spalte hinzufügen" klicken
4. Option "Virtuelle Eigenschaft" auswählen
5. Als Spaltenüberschrift den Spaltenbezeichner (in Kombination mit dem regulären Ausdruck) eingeben
6. Um sicherzustellen, dass die aktuellen Daten geladen sind, auf die Schaltfläche "Aus Datenquelle lesen" klicken
7. Zum Anzeigen des Ergebnisses auf "Tabelle anzeigen" klicken



Ein Klick auf die Vorschau zeigt die transformierten Spalteneinträge:

#	Media	Item number	Title	Artist	Genre	<1u>	<$media$m(mp3|ogg)>*.<$media$I>	Playlist Summer 2019
2	CD	010000	The suburbs	Arcade Fire	Postwave	CD		Playlist Summer 2019
3	CD	010100	Modern Man	Arcade Fire	Postwave	CD		Playlist Summer 2019
4	LP	010101	Empty Room	Arcade Fire	Postwave	LP		Playlist Summer 2019
5	mp3	010102	Half Light I	Arcade Fire	Postwave	MP3	*.mp3	Playlist Summer 2019
6	ogg	010103	Half Light II	Arcade Fire	Postwave	OGG	*.ogg	Playlist Summer 2019
7	LP	010104	Month Of Day	Arcade Fire	Postwave	LP		Playlist Summer 2019
8	LP	010105	Deep Blue	Arcade Fire	Postwave	LP		Playlist Summer 2019
9	LP	010106	Eleanor Rigby	The Beatles	Oldies	LP		Playlist Summer 2019
10	LP	010107	I Want To Tell You	The Beatles	Oldies	LP		Playlist Summer 2019
11	LP	010109	I'm Only Sleeping	The Beatles	Oldies	LP		Playlist Summer 2019
12	LP	010110	Love To You	The Beatles	Oldies	LP		Playlist Summer 2019
13	MD	010200	Taxman	The Beatles	Oldies	MD		Playlist Summer 2019
14	LP	010201	About A Girl	Nirvana	Rock	LP		Playlist Summer 2019

Folgende Abbildung zeigt die Auswirkung von vertauschten Spalten einer Importtabelle: Während bei der alleinigen Verwendung von Spaltennummern (<1u>) die falsche Spalte transformiert wird, bleibt bei Verwendung eines Bezeichners mit nachgelagertem regulären Ausdruck (<$Comp$u>) der Inhalt gleich:

#	Item number	Media	Genre	Title	Artist	<1u>	<$media$m(mp3|ogg)>*.<$media$l>	Playlist Summer 2019
2	010000	CD	Postwave	The suburbs	Arcade Fire	010000		Playlist Summer 2019
3	010100	CD	Postwave	Modern Man	Arcade Fire	010100		Playlist Summer 2019
4	010101	LP	Postwave	Empty Room	Arcade Fire	010101		Playlist Summer 2019
5	010102	mp3	Postwave	Half Light I	Arcade Fire	010102	*.mp3	Playlist Summer 2019
6	010103	ogg	Postwave	Half Light II	Arcade Fire	010103	*.ogg	Playlist Summer 2019
7	010104	LP	Postwave	Month Of Day	Arcade Fire	010104		Playlist Summer 2019
8	010105	LP	Postwave	Deep Blue	Arcade Fire	010105		Playlist Summer 2019
9	010106	LP	Oldies	Eleanor Rigby	The Beatles	010106		Playlist Summer 2019
10	010107	LP	Oldies	I Want To Tell You	The Beatles	010107		Playlist Summer 2019
11	010109	LP	Oldies	I'm Only Sleeping	The Beatles	010109		Playlist Summer 2019
12	010110	LP	Oldies	Love To You	The Beatles	010110		Playlist Summer 2019
13	010200	MD	Oldies	Taxman	The Beatles	010200		Playlist Summer 2019
14	010201	LP	Rock	About A Girl	Nirvana	010201		Playlist Summer 2019

Funktionsweise und Reihenfolge der regulären Ausdrücke

Die regulären Ausdrücke des vorangegangenen Beispiels wirken wie folgt:

- Der reguläre Ausdruck "m(mp3|ogg)" passt auf alle Einträge mit den Zeichenketten-Folgen "mp3" oder "ogg"
- Die Zeichen "*." außerhalb der spitzen Klammern werden einfach in der Reihenfolge ihrere Erscheinung als Zeichenkette hinzugefügt
- Der reguläre Ausdruck <$media$l> trnsformiert alle Buchstaben in Kleinbuchstaben

Bezüglich der Reihenfolge der regulären Ausdrücke ist es wichtig, den *filternden* Ausdruck vor den *transformierenden* Ausdruck zu positionieren:

<$media$m(mp3|ogg)> filtert zuerst die Einträge, welche dann anschließend durch <$media$l> transformiert werden.

Der koplette Ausdruck <$media$m(mp3|ogg)>*.<$media$l> liefert das beabsichtigte Ergebnis, während eine abweichende Reihenfolge *.<$media$l><$media$m(mp3|ogg)> der Ausdrücke dazu führt, dass alle Einträge transformiert werden.

Weil die transformierenden Ausdrücke zeitgleich als Ausgabe funktionieren, wird der nachfolgende Ausdruck nicht mehr berücksichtigt, was zu den eher unüblichen Musik-Dateinamenerweiterungen *.lp, *.cd und *.md führt:

#	Media	Item number	Title	Artist	Genre	<1u>	*.<$media$l><$media$m(mp3|ogg)>	Playlist Summer 2019
2	CD	010000	The suburbs	Arcade Fire	Postwave	CD	*.cd	Playlist Summer 2019
3	CD	010100	Modern Man	Arcade Fire	Postwave	CD	*.cd	Playlist Summer 2019
4	LP	010101	Empty Room	Arcade Fire	Postwave	LP	*.lp	Playlist Summer 2019
5	mp3	010102	Half Light I	Arcade Fire	Postwave	MP3	*.mp3	Playlist Summer 2019
6	ogg	010103	Half Light II	Arcade Fire	Postwave	OGG	*.ogg	Playlist Summer 2019
7	LP	010104	Month Of Day	Arcade Fire	Postwave	LP	*.lp	Playlist Summer 2019
8	LP	010105	Deep Blue	Arcade Fire	Postwave	LP	*.lp	Playlist Summer 2019
9	LP	010106	Eleanor Rigby	The Beatles	Oldies	LP	*.lp	Playlist Summer 2019
10	LP	010107	I Want To Tell Yo	The Beatles	Oldies	LP	*.lp	Playlist Summer 2019
11	LP	010109	I'm Only Sleeping	The Beatles	Oldies	LP	*.lp	Playlist Summer 2019
12	LP	010110	Love To You	The Beatles	Oldies	LP	*.lp	Playlist Summer 2019
13	MD	010200	Taxman	The Beatles	Oldies	MD	*.md	Playlist Summer 2019
14	LP	010201	About A Girl	Nirvana	Rock	LP	*.lp	Playlist Summer 2019

1.5.2.2. Konfiguration weiterer tabellenorientierter Datenquellen

Datenbanken

In einem Mapping für eine PostgreSQL-, Oracle- oder ODBC-Schnittstelle müssen die Datenbank, der Benutzer und das Passwort angegeben werden.

Angabe der Datenbank

Die Angabe für die Datenbank setzt sich aus Name des Host, dem Port und dem Namen der Datenbank zusammen. Die Syntax lautet.

Datenbank-System	Angabe der Datenbank
PostgreSQL	hostname:port_datenbank
Oracle	//hostname:[port][/datenbankService]
ODBC	Name der konfigurierten Datenquelle
MySQL	Getrennte Konfiguration von Datenbank und Hostname

Benutzername und Passwort konfigurieren

Benutzername und Passwort werden so angegeben, wie sie in der Datenbank abgelegt sind. Unter dem Punkt Tabelle kann die Tabelle angegeben werden, die importiert werden soll. Für den Import besteht aber auch die Möglichkeit, dass unter dem Punkt "Query" eine Query formuliert wird, die angibt, welche Daten importiert werden sollen.

Encoding

Handelt es sich um ein PostgreSQL-Mapping, dann kann auf dem Reiter "Encoding" das Encoding angegeben werden.

Spezielle Anforderungen der Oracle-Schnittstelle

Die Funktion zum direkten Import aus einer Oracle Datenbank setzt voraus, dass auf dem importierenden Rechner bestimmte Laufzeit-Bibliotheken installiert sind.

Direkt benötigt wird das "Oracle Call Interface" (OCI) und zwar in einer Version, die laut Oracle zu dem Datenbankserver passt, der angesprochen werden soll. D.h., um eine Oracle 11i Datenbank anzusprechen, sollte auf dem importierenden Rechner das OCI in Version 11 installiert sein. Das OCI lässt sich am einfachsten installieren, wenn man den "Oracle Database Instant Client" installiert. Die Package Version "Basic" ist ausreichend. Der Client ist entweder vom Serverbetreiber zu bekommen oder von Oracle nach Registrierung unter http://www.oracle.com/technology/tech/oci/index.html ladbar.

Nach der Installation ist sicher zu stellen, dass die Bibliothek für den importierenden Client auffindbar ist, entweder indem sie im gleichen Verzeichnis liegt oder für das entsprechende Betriebssystem passende Umgebungsvariablen definiert werden (ist beim OCI dokumentiert).

Je nach Betriebssystem auf dem der Import stattfinden soll, sind weitere Bibliotheken notwendig, die nicht immer installiert sind.

- MS Windows: neben der benötigten "oci.dll" sind noch zwei weitere Bibliotheken notwendig: advapi32.dll (Erweitertes Windows 32 Base-API) und mscvr71.dll (Microsoft C Runtime Library)

Bis auf den XML-Import/Export sind alle Importe/Exporte tabellenorientiert und unterscheiden sich nur in der Konfiguration der Quelle. Für die Beschreibung einer tabellenorientierten Abbildung kann das Beispiel der CSV-Datei herangezogen werden.

1.5.2.3. Abbildung einer XML-Datei

Das Prinzip von XML-Dateien ist, die unterschiedlichen Angaben zu einem Datensatz über Tags (<>) explizit zu machen (nicht über Tabellenspalten). Dementsprechend sind Tags auch die Grundlage der Abbildung beim Import von XML-Strukturen in i-views.

Ein Beispiel: Nehmen wir an, unsere Liste von Songs liegt als XML-Datei vor:

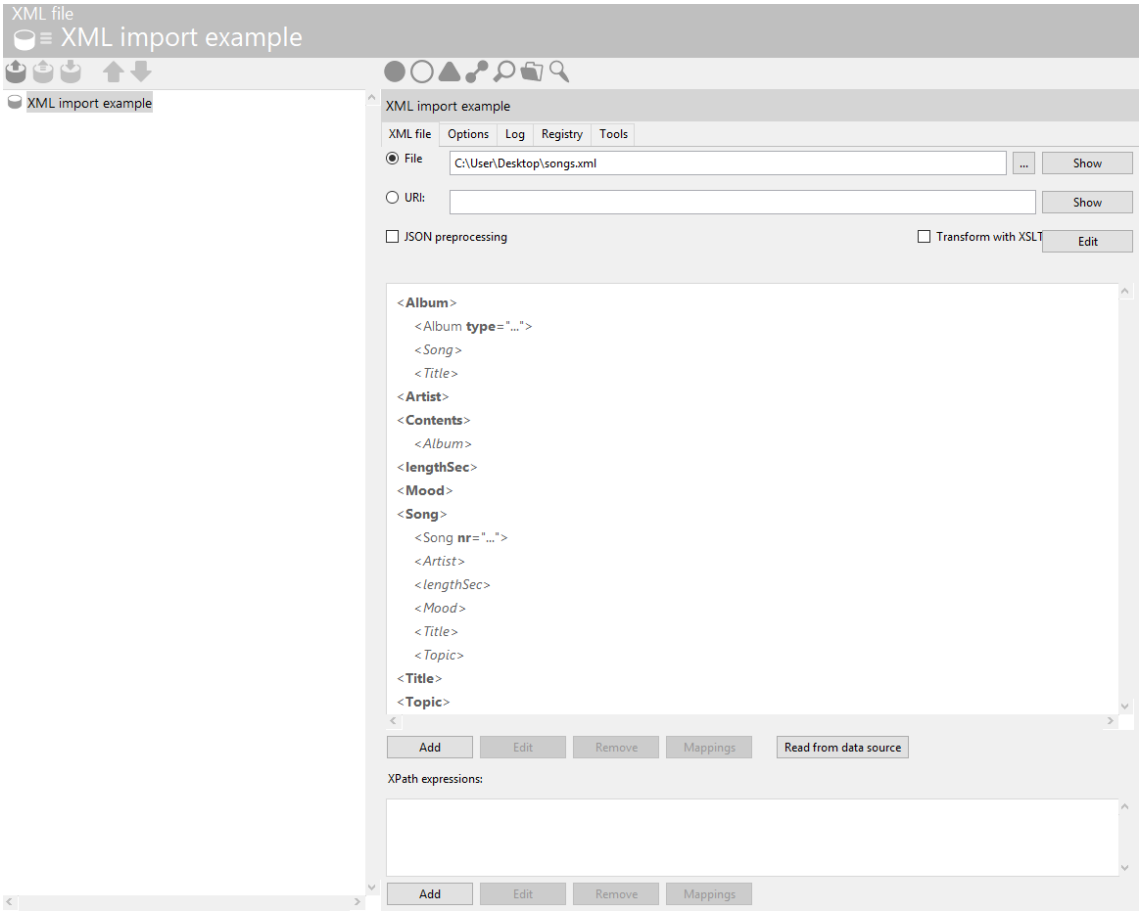
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Inhalt>
    <Album type="Oldie">
        <Title>Revolver</Title>
        <Song nr="1">
            <Title>Eleanor Rigby</Title>
            <lengthSec>127</lengthSec>
            <Interpret>The Beatles</Interpret>
            <Thema>Mental illness</Thema>
            <Mood>Dreamy</Mood>
            <Mood>Reflective</Mood>
        </Song>
        [...]
    </Album>
    [...]
</Inhalt>
```

Wenn wir nun diese XML-Datei importieren wollen, wählen wir bei der Auswahl des Typs der Datenquelle "XML-Datei" aus, wodurch sich der Editor für den Im- und Export von XML-Dateien öffnet. Bereits in der Angabe des Dateistandortes gibt es Unterschiede zum Editor für CSV-Dateien. Wir können nun zwischen einem lokalen Dateipfad und der Angabe einer URI wählen.

JSON preprocessing ermöglicht das Umwandeln einer JSON-Datei in XML vor dem eigentlichen Import.

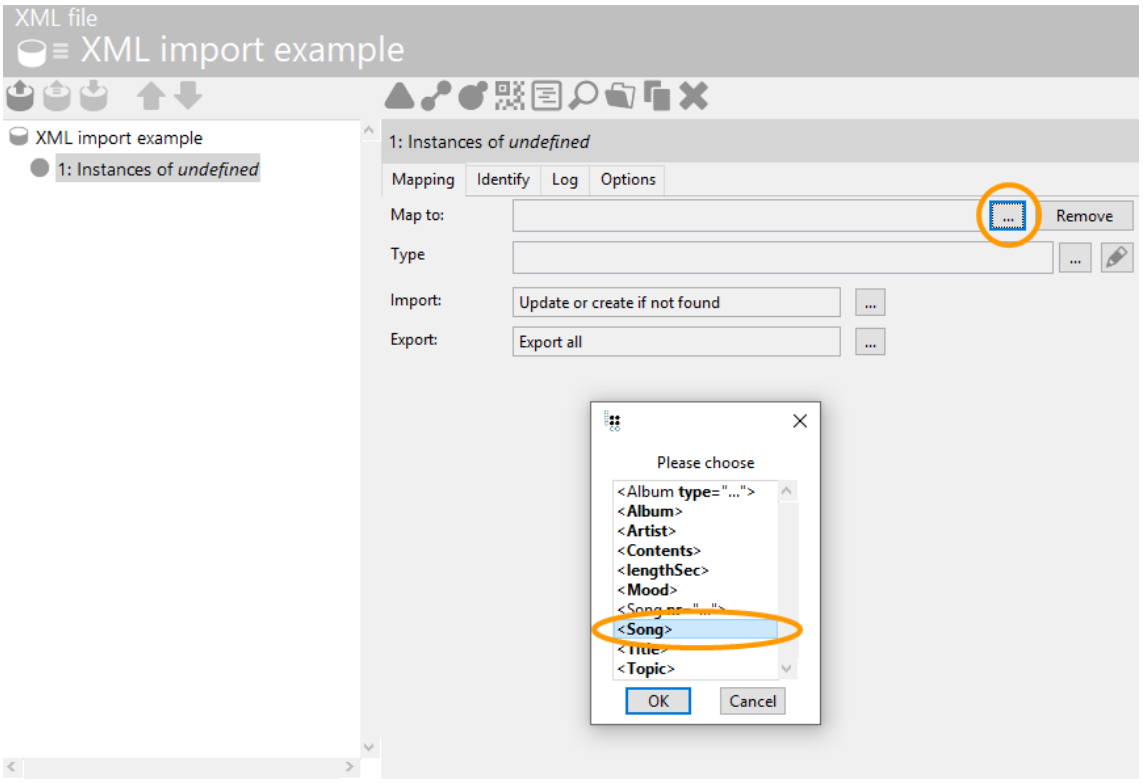
Mit XSTL transformieren kann man auswählen, wenn man die XML-Daten aus der ausgewählten XML-Datei noch vor dem Import in andere XML-Daten umwandeln möchte, um beispielsweise die Struktur zu ändern oder einzelne Werte weiter aufzutrennen. Über die Schaltfläche "Bearbeiten" öffnet sich die XML-Datei, in der man die Änderungen mittels XSLT definieren kann.

Ist die Datei ausgewählt, können wir mit dem Button "Aus Datenquelle lesen" die XML-Struktur auslesen lassen, die uns daraufhin in rechten Fenster angezeigt wird.

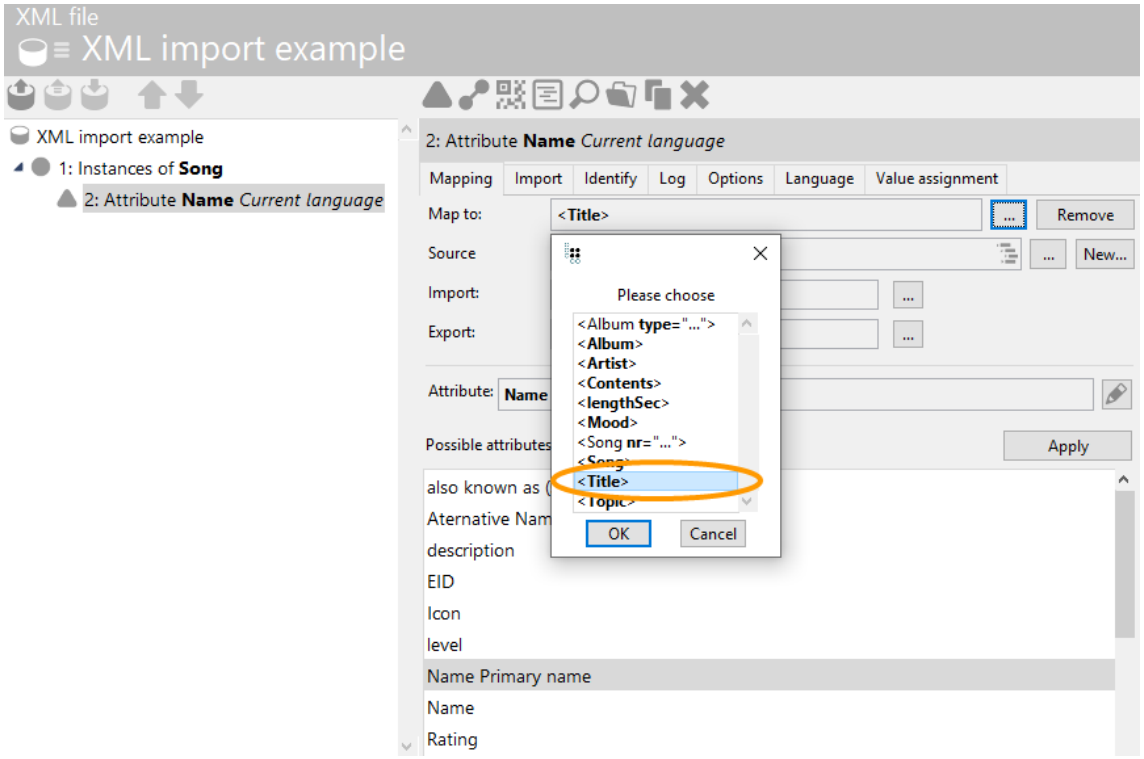


Wir wollen die einzelnen Songs unserer Liste importieren. Darum legen wir eine neue Objektabbildung an und wählen über den Button bei "Abbilden auf" den Tag *<Song>* aus. Im Gegensatz zum CSV-Import, bei dem nur Attributwerte eine Entsprechung in der CSV-Tabelle finden und eine einzelne Zeile für ein Objekt steht, sodass auch nur die Attributwerte abgebildet werden müssen, erfolgt das Mapping der semantischen Objekte hier über die XML-Struktur. Darum müssen auch für alle abzubildenden Objekte jeweils ein entsprechendes Tag der XML-Datei angegeben

werden.



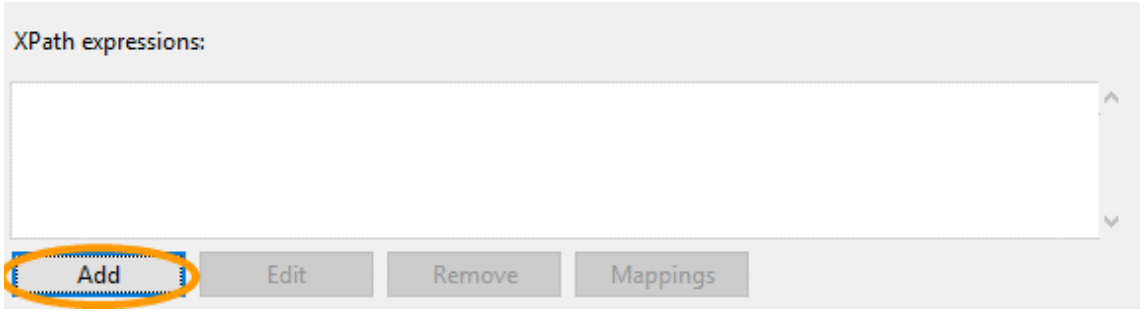
Wie in unserem Beispiel, sind die Tags ohne Kontext nicht immer eindeutig: <Title> wird sowohl für Titel von Alben als auch für Songtitel verwendet. Erst in Kombination mit dem umgebenden Tag wird der Objekttyp klar. Oft laufen der Kontext der XML-Struktur und der Kontext der Abbildungshierarchie synchron: Da wir nun bereits festgelegt haben, dass die Objekte auf den Tag <Song> abgebildet werden sollen, ist durch die XML-Struktur klar, welcher <Title>-Tag nun gemeint ist, wenn wir <Title> mit dem Namensattribut von Songs mappen. Dort, wo Abbildungshierarchie und Tagstruktur nicht parallel laufen, können wir im XML-Import zusätzlich zu den in der XML-Datei vorkommenden Tags Ketten bilden — mit XPath.



Wie auch beim CSV-Import muss über den Reiter "Identifizieren" bei der Objektabbildung festgelegt werden, durch welche Attributwerte das Objekt in der semantischen Graph-Datenbank identifiziert werden soll. Das erste angelegte Attribut für ein Objekt wird auch hier wieder automatisch als identifizierendes Attribut verwendet.

Möglichkeiten mit XPath-Ausdrücken

Angenommen wir würden nur Songs aus Alben des Musik-Stils "Oldie" importieren wollen. In unserem XML-Dokument ist die Information über den Musik-Stil direkt im Album-Tag angegeben unter *type="…"*. Im Editor müssen wir also einen XPath-Ausdruck definieren, der den Pfad im XML-Dokument beschreibt, der nur diejenigen Songs enthält, die aus Oldie-Alben stammen. Im rechten unteren Bereich des Editors finden wir ein Feld zum hinzufügen von XPath-Ausdrücken.

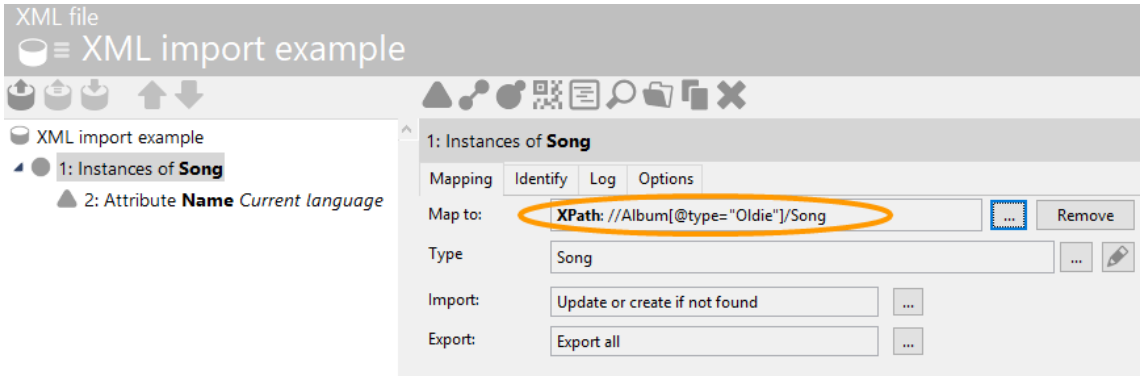


Der passende XPath-Ausdruck lautet: `//Album[@type="Oldie"]/Song`

Erklärung im Einzelnen:

- `//Album`: Selektiert alle Alben, wobei es keine Rolle spielt, wo sie sich im Dokument befinden.
- `Album[@type="Oldie"]`: Selektiert alle Alben vom Typ "Oldie"
- `Album/Song`: Selektiert alle Songs, die Subelemente von Alben sind.

Diesen Ausdruck können wir nun verwenden, um eine Entsprechung für die Objektabbildung der Songs zu definieren.



Mit XPath stehen uns außerdem viele weitere nützliche Selektions-Funktionen zur Verfügung. So können wir beispielsweise Elemente über ihre Position im Dokument selektieren, Vergleichsoperatoren einsetzen, sowie alternative Pfade angeben.

Grundlegende Tips für den XML-Import

- Einen absoluten Pfad für einen Ausgangsknoten verwenden.
- Alle anderen Pfade relativ zum absoluten Pfad fomulieren.
- Ein inkrementeller Import ist nur möglich, wenn das XML-Dokument keine Kreuzreferenzen aufweist. In diesem Fall den Knoten mit absolutem Pfad als partittionierendes Element angeben (Option auf dem zweiten Reiter des Importmappings).
- Wenn die Struktur sich in die Tiefe auffächert, dann empfiehlt sich ein Import von den Elementen in tieferer Ebene hin zu den Elementen höherer Ebene, da zum Aufbau etwaiger Objekthierarchien die Elternelemente eindeutig sind, während es die Kindelemente nicht sind.
- Bei komplexeren XML-Dokumenten kann es sich lohnen, zunächst alle Objekte inklusive ihrer identifizierenden Attribute zu importieren, um in einem zweiten Durchlauf die Beziehungen zwischen den Objekten aufbauen zu können, da die Objekte dann alle auffindbar sind.

Alternative: XML Import-Mapping einer XML-basierten RDF-Datei

Wenn die RDF-Datei mit dem herkömmlichen RDF-Importmechanismus nicht korrekt import werden kann, weil das Schema des Wissensnetzes zu spezifisch ist oder die RDF-Datei zu spezifisch ist oder wenn Schema-Teile fehlen, dann kann als Alternative das XML-Importmapping verwendet werden.

In einigen Fällen kann es vorkommen, dass die automatisierte Zuordnung von RDF-Tag-Inhalten nicht ausreichend detailliert für die Importanforderungen sind und deshalb XPath-Ausdrücke für

eine dedizierte Wertzuweisung benötigt werden.

HINWEIS	Bei der Verwendung von XPAth-Ausdrücken wird der Namensraum (welche auf dem Qualifier aufbaut) nicht berücksichtigt.

Input RDF-XML	XPath	Bedeutung
	//	Oberste Ebene der RDF-Datei
	../	Eine Ebene nach oben
	../../xyz	Zwei Ebenen aufwärts, von dort aus zu einem Knoten namens "xyz"
`<rdf:label>`	/label/	Tag "label"
`<rdf:prefLabel xml:lang="en"> Beispiel </rdf:prefLabel>`	prefLabel[@lang="en"]	Knoten mit attribute und bestimmtem Attributwert
	ancestor::termEntry/attribute::id	Übergeordneter Knoten der aktuellen Ebene mit dem Namen ("termEntry") und dem Attribut ("id")
	/myparent/mychild[text()]	Text zwischen bestimmten Tags

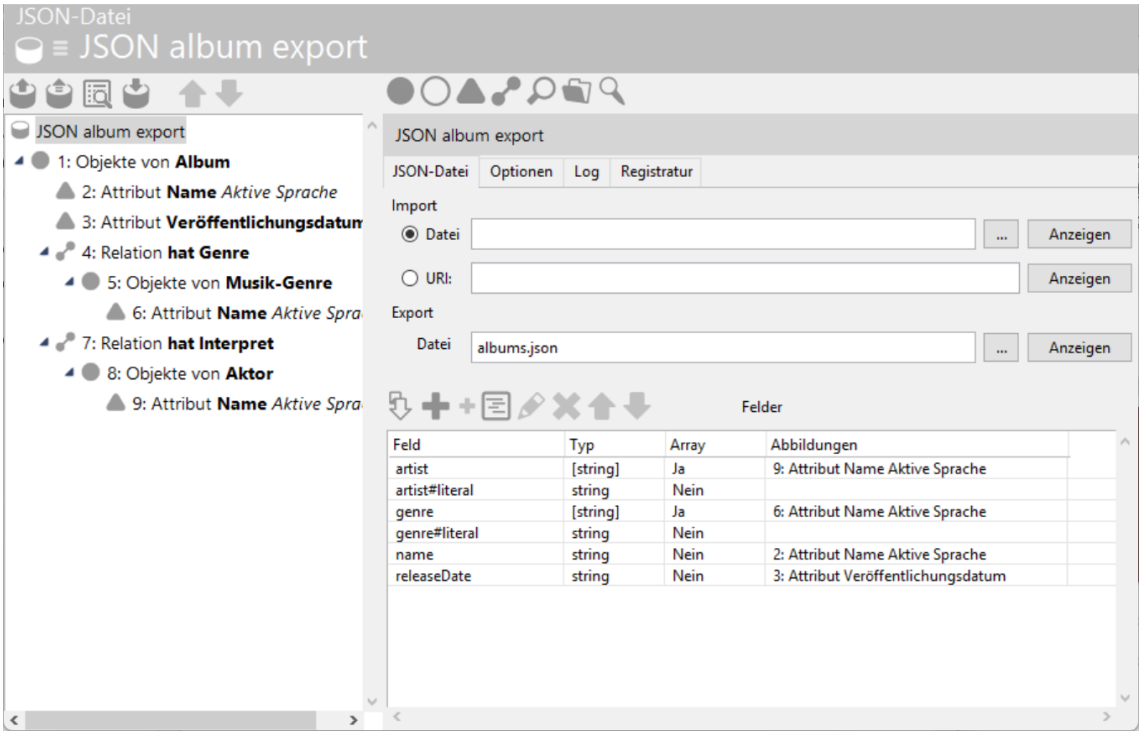
1.5.2.4. Abbildung von JSON-Dateien

Für den Import und Export von JSON-Dateien müssen die einzelnen Bestandteile (Objekte, Arrays usw.) als Felder definiert werden.

Objekte haben wiederum Sub-Felder. Wenn bei einer Abbildung ein Feld angegeben wurde, können bei nachgeordneten Abbildungen nur direkte Sub-Felder angegeben werden.

Beim Export von mehreren Elementen wird auf oberster Ebene grundsätzlich ein Array von Objekten exportiert. Es müssen keine entsprechenden Felder konfiguriert werden.

Beispiel:



Das in der Abbildung gezeigte Mapping erzeugt beim Export folgenden Inhalt:

```
[
  {
    "name": "Das Geheimnis",
    "releaseDate": "01.10.2013",
    "artist": ["Traumhaus"],
    "genre": ["Progressive Rock"]
  }, {
    "name": "The Raven that Refused to Sing and other Stories",
    "releaseDate": "26.02.2013",
    "artist": ["Steven Wilson"],
    "genre": ["Progressive Rock", "Art Rock"]
  }
]
```

1.5.2.4.1. Script-Felder

Alternativ zu Feldern können beim Import auch Scripte verwendet werden. Auf dem Reiter **JSON-Datei** legt man dazu mit **Skript hinzufügen** ein neues Script-Feld an. Man kann ein bestehendes Script auswählen oder ein neues anlegen.

Beim Export wird dieses Script mit einem Objekt der Klasse `$k. ExternalDataObject` als Parameter aufgerufen. Dieses ermöglicht mit der Methode `properties()` den Zugriff auf den gerade importierten Ausschnitt der Daten.

Beim Export müssen stattdessen Script-Abbildungen verwendet werden.

Beispiel: Aus einer JSON-Datei soll aus dem optionalen Sub-Feld `extra` das Datum aus `releaseDate` extrahiert werden.

```
[{
    "name": "Ausgeliefert",
    "artist": ["Traumhaus"],
    "genre": ["Progressive Rock"],
    "extra": {
        "releaseDate": "1.1.2014",
    }
  }
]
```

Das Script sieht folgendermaßen aus:

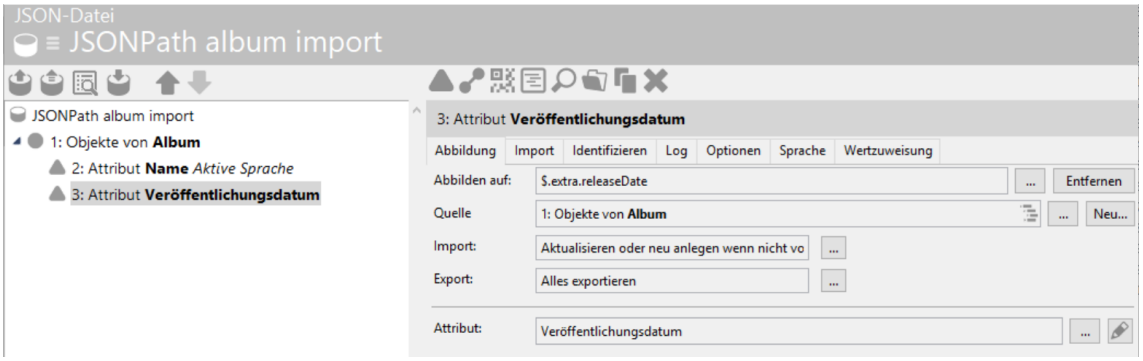
```
function getValue(dataSourceValue) {
  let dateString = dataSourceValue.properties()["extra"]?.["releaseDate"]
  return dateString ? $k.Date.parse(dateString) : undefined
}
```

1.5.2.4.2. JSONPath

Um auf Unterelement zuzugreifen oder die JSON-Inhalte zu filtern, kann beim Import auch auf JSONPath zurückgegriffen werden. Dies ist ein mit XPath oder CSS-Selektoren vergleichbarer Mechanismus, um bestimmte Elemente zu adressieren. Formal ist JSONPath durch RFC 9535: *JSONPath: Query Expressions for JSON* spezifiziert.

Auf dem Reiter **JSON-Datei** legt man dazu mit **JSONPath hinzufügen** ein neues Element an. Im Dialog gibt man dazu direkt den JSONPath an. Es werden nur Ausdrücke akzeptiert, deren Syntax den Anforderungen von RFC 9535 entsprechen.

Um einen Wert mit JSONPath zu bestimmen, bildet man das Attribut auf ein JSONPath-Feld ab:



Beispiele:

Sub-Feld `releaseDate` von `extra`	`$.extra.releaseDate`
Das erste Genre	`$.genre[0]`

Um Elemente zu filtern, bildet man ein Objekt auf einen filternden JSONPath ab:

Beispiele:

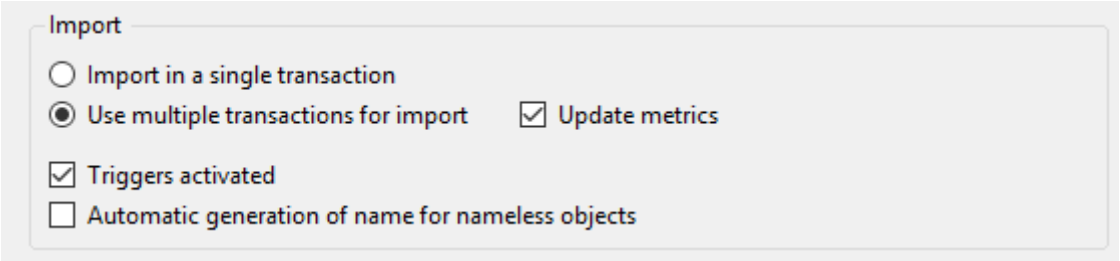
Alle Alben mit Release-Datum `01.10.1994`	`$[?@.releaseDate == "01.10.1994"]`
Alle Alben von Steven Wilson (geschachtelt, da `artist` ein Array ist)	`$[?@.artist[?@ == "Steven Wilson"]]`

JSONPath-Felder sind unabhängig vom Schema, d.h. sie können überall angewendet werden, und man muss die Felder für das Ergebnis ggf. von Hand anlegen.

1.5.3. Weitere Optionen, Log und Registratur

1.5.3.1. Weitere Optionen beim Import

Im Reiter "Optionen" stehen uns folgende Funktionen zur Auswahl, die unabhängig von der Datenquelle sind:



In einer Transaktion importieren : Ist langsamer als ein Import mit mehreren Transaktionen und sollte darum nur dann verwendet werden, wenn es beim Import mit mehreren Transaktionen ansonsten zum Konfliktfall kommen kann, wenn viele Personen zur selben Zeit im Knowledge-

Builder arbeiten oder wenn man Daten importieren will, bei denen es eine Rolle spielt, dass sie nicht getrennt voneinander betrachtet werden. Beispiel 1: Es erfolgt stündlich ein Import mit dem Status der Auslastung von Maschinen. Die addierten Werte der Auslastung dürfen einen gewissen Wert nicht übersteigen, da es ansonsten eventuell zum Stromausfall kommen kann. Damit diese Regel (z.B. mithilfe eines Skripts) berücksichtigt werden kann, müssen alle Werte gemeinsam betrachtet und dann importiert werden. Beispiel 2: Es erfolgt ein Import mit Personen, von denen höchstens eine den Hauptschlüssel haben kann, weil nur ein Hauptschlüssel existiert. Auch hier muss der Import in einer Transaktion erfolgen, da bei mehreren Transaktionen der Fehler übersehen werden könnte, dass bei zwei Personen das Attribut für den Hauptschlüsselbesitz gesetzt wurde.

Mehrere Transaktionen verwenden : Standardeinstellung für einen schnellen Import.

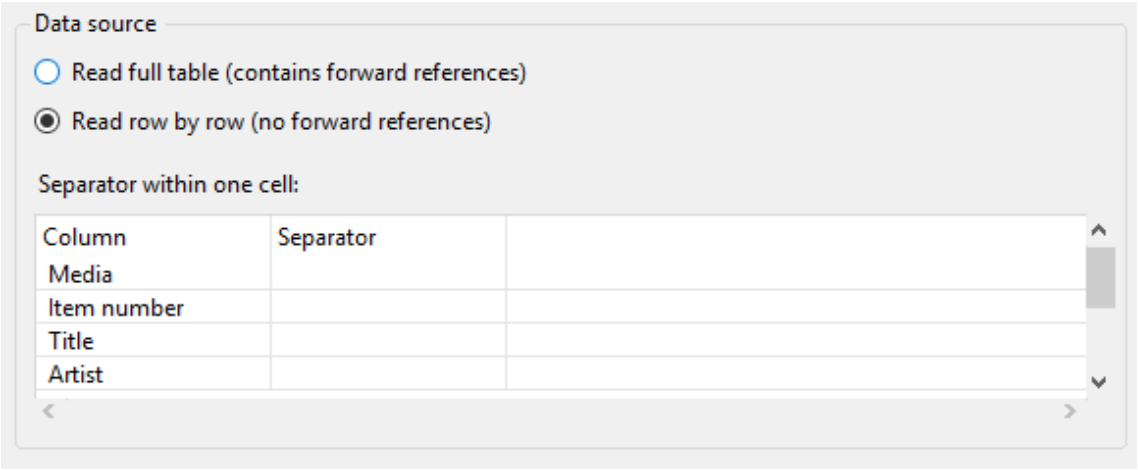
Journaling : Das Journaling sollte verwendet werden, wenn extrem viele Daten mit einem Import gelöscht oder geändert werden. Erst nach jeweils 4.096 Einträgen (die Zahl ist variabel), sollen die Änderungen, bzw. Löschungen für diese Einträge auch am Index vorgenommen werden. Dadurch wird der Import beschleunigt, da nicht für jede einzelne Änderung/Löschung der Index herangezogen werden muss, sondern spätestens nach 4.096 Veränderungen diese gemeinsam in den Index übernommen werden.

Metriken aktualisieren : Die Metriken sollten aktualisiert werden, wenn der Import eine große Auswirkung auf die Menge von Objekttypen oder Eigenschaftstypen hat, wenn also sehr viele Objekte oder Eigenschaften eines Typs der semantischen Graph-Datenbank hinzugefügt werden. Würden die Metriken nicht aktualisiert, könnte dies negative Auswirkungen auf die Performance von Suchen haben, in denen die entsprechenden Typen eine Rolle spielen.

Trigger aktiviert : Ob Trigger beim Import aktiviert sein sollen oder nicht kann hier über das Häkchen bestimmt werden. Falls es gewünscht ist, dass ein Trigger greift und ein anderer nicht, müssen zwei verschiedene Abbildungen mit den entsprechenden semantischen Elementen definiert werden. Informationen zu Triggern stehen im Kapitel Trigger zur Verfügung.

Automatische Namensgenerierung für namenlose Objekte : Ermöglicht die automatische Namensgenerierung für namenlose Objekte.

Liegt eine tabellenorientierte Quelle vor, können wir folgende Einstellungen vornehmen:



Komplette Tabelle einlesen : Obwohl es länger dauern kann, die komplette Tabelle auf einmal einzulesen, macht es Sinn diese Option auszuwählen, wenn Vorwärtsreferenzen vorhanden sind, d.h., wenn Relationen zwischen den zu importierenden Objekten gezogen werden sollen. In diesem Fall müssen nämlich beide Objekte bereits vorhanden sein, was beim zeilenweisen Einlesen der Tabelle nicht der Fall ist. Zudem ist die Fortschrittsanzeige genauer als beim zeilenweisen Einlesen.

Tabelle zeilenweise einlesen : Das zeilenweise Einlesen der Tabelle sollte immer dann verwendet werden, wenn keine Querreferenzen in der Tabelle vorhanden sind, da der Import so schneller geht.

Trennzeichen innerhalb einer Zelle: siehe [mehrere-werte].

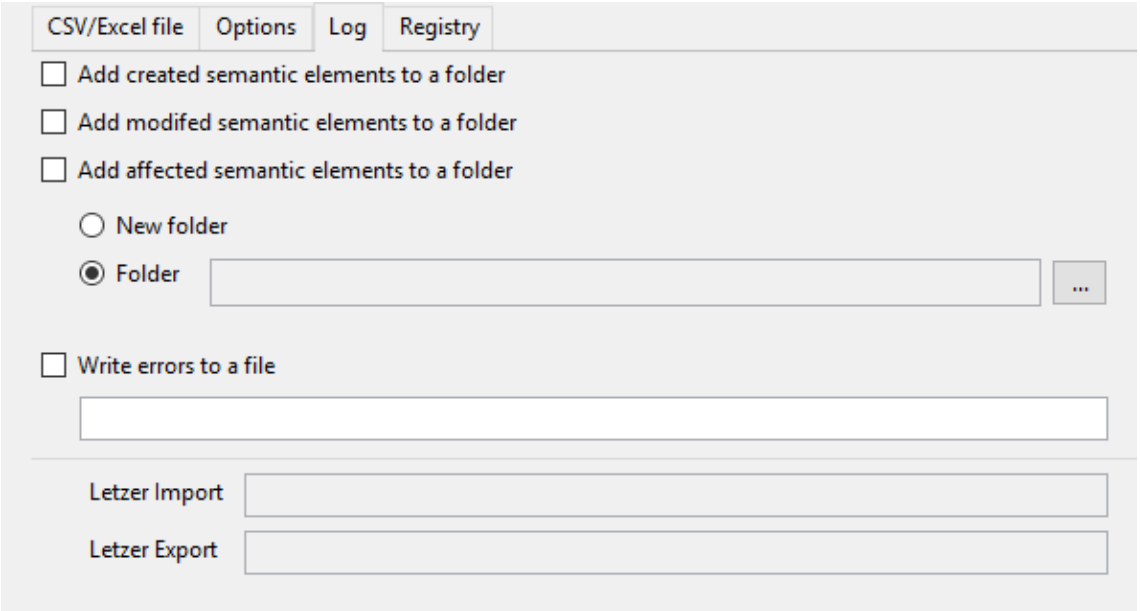
Liegt eine XML-basierte Datenquelle vor, stehen uns folgende Funktionen zur Verfügung:

Inkrementeller XML-Import: Der XML-Import erfolgt schrittweise. Die Schritte werden durch das partitionierende Element festgelegt.

DTD einlesen: Liest die Dokumenttypdefinition (DTD) ein **.

1.5.3.2. Log

Die Funktionen im Reiter "Log" ermöglichen Änderungen, die beim Import vorgenommen werden, verfolgen zu können.

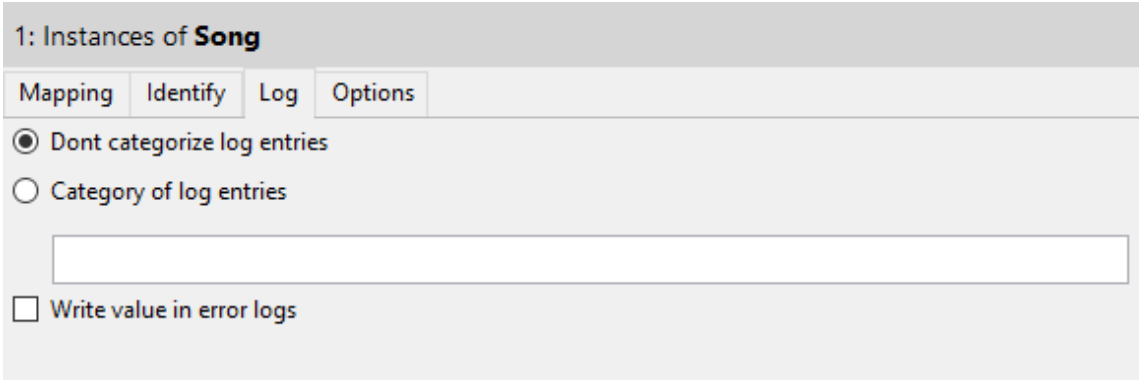


Erzeugte Wissensnetzelemente in einen Ordner stellen: Werden neue Objekte, Typen oder Eigenschaften durch den Import erzeugt, können diese in einen Ordner in der semantischen Graph-Datenbank gestellt werden.

Veränderte Wissensnetzelemente in einen Ordner stellen: Alle Eigenschaften oder Objekte, deren Eigenschaften sich durch den Import geändert haben, können in einen Ordner gestellt werden.

Fehlermeldungen in eine Datei schreiben: Beim Import können Fehler auftreten (z.B. kann es sein, dass ein identifizierendes Attribut für mehrere Objekte vorkam und daher das Objekt nicht eindeutig identifiziert werden konnte). Diese Fehler werden standardmäßig nach einem Import in einem Fenster angezeigt und man hat dann die Möglichkeit den Fehlerbericht zu speichern. Wenn dies automatisch passieren soll, kann hier der Haken gesetzt und eine Datei angegeben werden.

Letzter Import / Letzter Export : Hier werden das Datum und die Uhrzeit des zuletzt vorgenommen Imports und des zuletzt vorgenommen Export angezeigt.



Auch bei den einzelnen Abbildungs-Objekten ist der Reiter "Log" verfügbar. Hier kann bei Bedarf eine Kategorie für Logeinträge eingetragen werden. Zudem kann festgelegt werden, dass der Wert des entsprechenden Objekts / der entsprechenden Eigenschaft in den Fehlerlog geschrieben

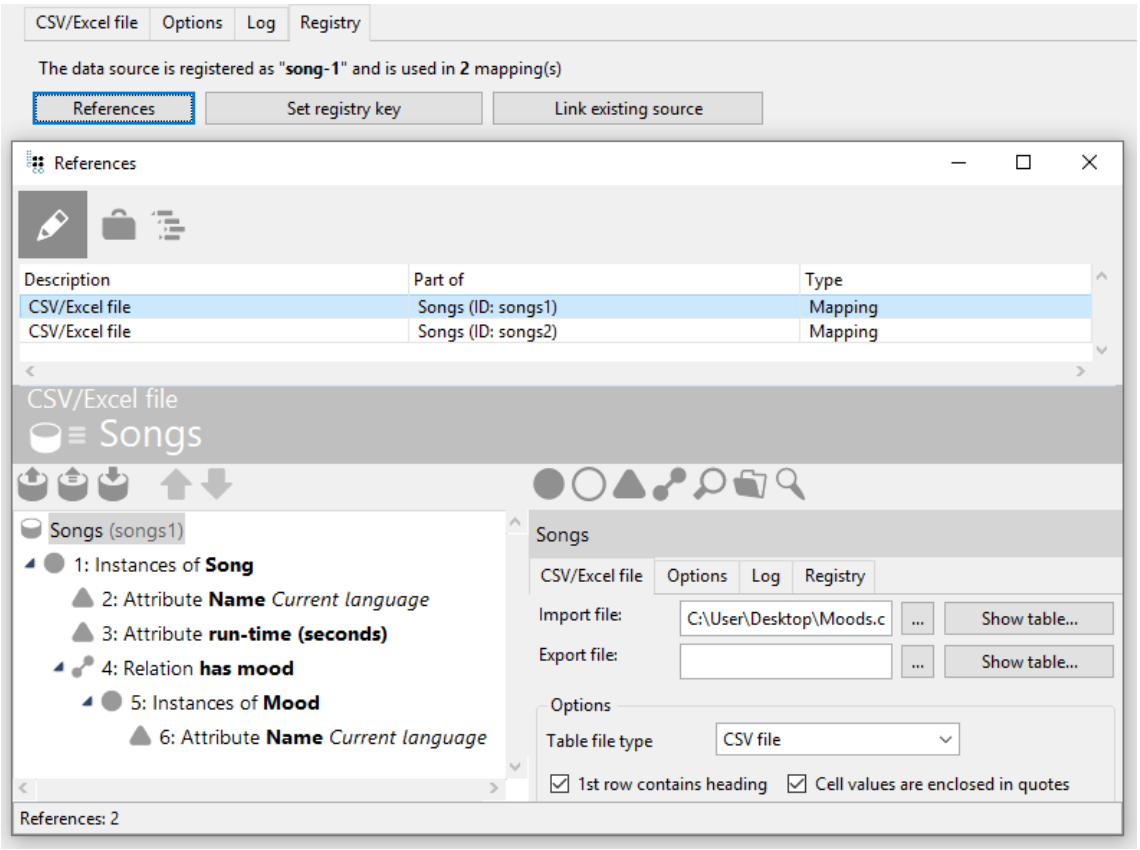
werden soll. Dies ist standardmäßig nicht aktiviert, um das Offenlegen sensibler Daten (z.B. Passwörter) zu vermeiden.

1.5.3.3. Registratur

Unter dem Reiter "Registratur" findet man die Funktion "Registrierungsschlüssel setzen" mit der man die Datenquelle für andere Importe und Exporte registrieren kann.

Die Funktion "Bestehende Quelle verknüpfen" ermöglicht die Wiederverwendung einer registrierten Quelle.

Unter "Verwendungen" kann man einsehen, wo eine Datenquelle noch verwendet wird:



1.5.4. Attributtypen und -formate

Eine häufig auftretende Aufgabe einer Attributabbildung ist der Import bestimmter Daten von konkreten Objekten, beispielsweise von Personen: Telefonnummer, Geburtsdatum etc.

Beim Import von Attributen, für die i-views ein bestimmtes Format verwendet (z.B. Datum), müssen die Einträge der zu importierenden Spalte in einer Form vorliegen, die von i-views unterstützt wird. Beispielsweise kann in ein Attributfeld vom Typ Datum keine Zeichenkette der Form "abcde..." importiert werden; in einem solchen Fall wird für das entsprechende Objekt kein Wert importiert.

Die folgende Tabelle listet die von i-views unterstützten Formate beim Import von Attributen auf.

Ein Tabellenwert "ja" oder "1" wird also beispielsweise korrekt als boolescher Attributwert (bei einem entsprechend definierten Attribut) importiert, ein Wert wie "ein" oder ähnliches hingegen nicht.

Attribut	Unterstützte Werte-Formate
Auswahl	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Boolesch	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Datei	Das Importieren von Dateien (z.B. Bildern) ist möglich. Dazu muss entweder der absolute Pfad zur Datei angegeben werden oder die zu importierenden Dateien müssen im gleichen (oder einem anzugebenden Unterverzeichnis) liegen wie die Import-Datei.
Datum	• <day> <monthName> <year>, z. B. 5 April 1982, 5-APR-1982 • <monthName> <day> <year>, z. B. April 5, 1982 • <monthNumber> <day> <year>, z. B. 4/5/1982 Das Trennzeichen zwischen <day>, <monthName> und <year> kann z.B. ein Leerzeichen, ein Komma oder ein Bindestrich sein (es sind aber noch weitere Zeichen möglich). Gültige Monatsnamen sind: * "Januar", "Februar", "März", "April", "Mai", "Juni", "Juli", "August", "September", "Oktober", "November", "Dezember" * "Jan", "Feb", "Mrz", "Mär", "Apr", "Mai", "Jun", "Jul", "Aug", "Sep", "Okt", "Nov", "Dez". Achtung: Zweistellige Jahreszahlen xy werden zu 20xy expandiert (aus 4/5/82 wird also 4/5/2082). Wenn das Mapping auf "Frei definierbares Format" eingestellt ist, können folgende Tokens verwendet werden: YYYY und YY (Jahr), MM und M (Monatsnummer), MMMM (Monatsname), MMM (abgekürzter Monatsname), DD und D (Tag)
Datum und Uhrzeit	Für Datum und Uhrzeit siehe die jeweiligen Attribute. Das Datum muss vor der Uhrzeit stehen. Wenn die Uhrzeit weggelassen wird, wird 0:00 verwendet.
Farbe	Import nicht möglich.
Festkommazahl	Import möglich.
Ganzzahl	• Ganzzahlen beliebiger Größe • Fließkommazahlen (mit Punkt getrennt), z.B. 1.82. Die Zahlen werden beim Import gerundet.
Internet-Verknüpfung	Jede beliebige URL möglich.

Attribut	Unterstützte Werte-Formate
Uhrzeit	<hour>: <minute>: <second> <am/pm>, z.B. 8:23 pm (wird zu 20:23:00) <minute>, <second> und <am/pm> können weggelassen werden. Wenn das Mapping auf "Frei definiertes Format" eingestellt ist, könne folgende Tokens verwendet werden: hh und h (Stunde), mm und m (Minute), ss und s (Sekunde), mmm (Millisekunde)
Zeichenkette	Jede beliebige Zeichenkette. Es wird keine Dekodierung vorgenommen.

Boolesche Attribute und Auswahlattribute

Auswahl- oder boolesche Attribute können nur Werte aus einer vorgegebenen Menge annehmen; bei Auswahlattributen ist dies eine vorgegebene Liste, bei booleschen Attributen das Wertepaar `ja` /`nein` in Form eines Klickfelds. Beim Import dieser Attribute kann angegeben werden, wie die Werte aus der Import-Tabelle in Attributwerte der semantischen Graph-Datenbank übersetzt werden. Zum einen können die Werte so, wie sie in der Tabelle stehen, übernommen werden; entsprechen sie keiner der in der semantischen Graph-Datenbank definierten möglichen Werte des Attributs, werden sie nicht importiert. Zum anderen können Wertzuweisungen zwischen Tabellenwerten und Attributwerten, die dann importiert werden, festgelegt werden.

1.5.5. Konfiguration des Exports

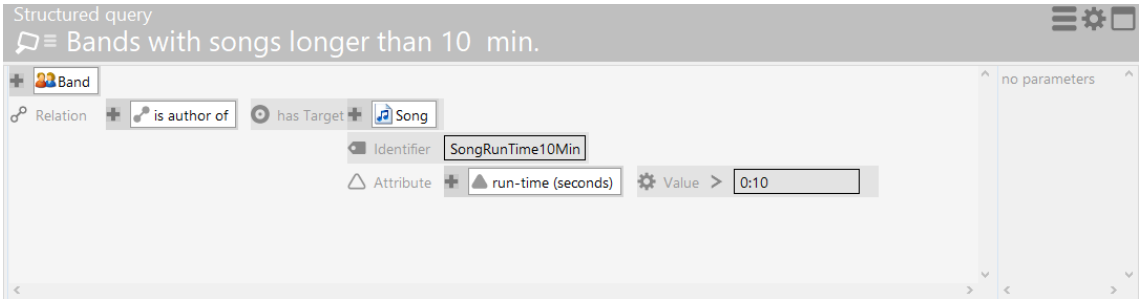
Der Export von Daten aus einer semantischen Graph-Datenbank in eine Tabelle wird in demselben Editor wie der Import und ganz analog vorbereitet:

1. Im einem Tabellen-Mapping-Ordner im Hauptfenster wird ein neues Mapping angelegt.
2. Im Tabellen-Mapping-Editor wird die zu erzeugende Datei angegeben.

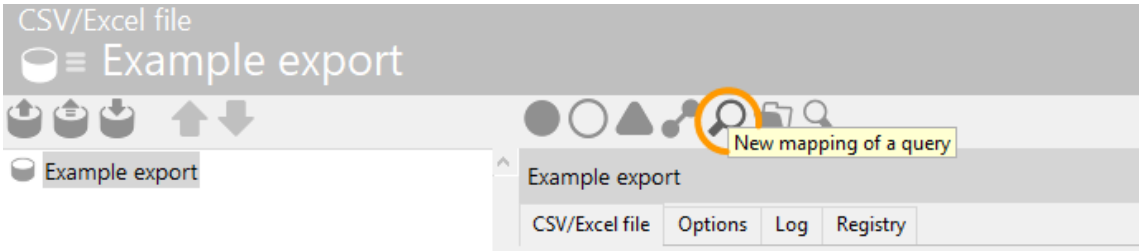
Der Unterschied zum Import liegt darin, dass die Spalten jetzt nicht aus der Tabelle eingelesen werden, sondern im Tabellen-Mapping-Editor angelegt werden müssen. Da der Import- und der Export-Editor derselbe sind, muss man beim Anlegen einer neuen Spalte zunächst auswählen, ob es sich um eine *Standard*- Spalte oder eine *Virtuelle Eigenschaft* handelt. Virtuelle Eigenschaften sind bei einem Export jedoch nicht verwendbar.

1.5.5.1. Export von Strukturabfragen

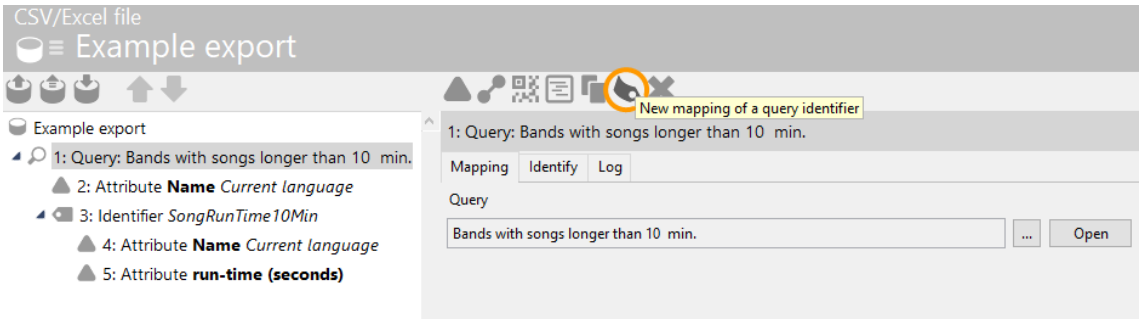
Es besteht die Möglichkeit das Ergebnis einer Strukturabfrage zu exportieren. Diese Vorgehensweise bietet sich an, wenn nur bestimmte Objekte, die durch eine Suche eingeschränkt werden, exportiert werden sollen. Nehmen wir als Beispiel an, wir wollten alle Bands, die Songs geschrieben haben, die länger als 10 min. dauern, exportieren. Dafür müssen wir zunächst eine Strukturabfrage definieren, die die gewünschten Objekte zusammenstellt.



Auf diese Strukturabfrage greifen wir dann von der Konfiguration des Exports aus zu. Dazu wählen wir im Kopf der Mapping-Konfiguration anstelle einer Objekt-Abbildung die Abbildung einer Abfrage. Die Strukturabfrage benötigt einen Registrierungsschlüssel, um auf sie zugreifen zu können.

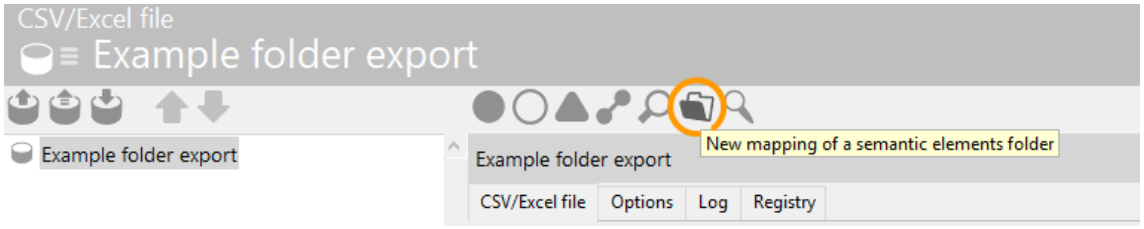


Damit werden nur noch die Ergebnisse der Strukturabfrage exportiert. Für diese Objekte können wir jetzt wieder Eigenschaften angeben, die in den Export mit aufgenommen werden sollen: z.B. Gründungsjahr der Band, Mitglieder und Songs. Jetzt kann es aber vorkommen, das wir von den Bands, die wir so zusammengestellt haben, nicht alle Songs exportieren wollen, sondern gerade nur die, die auch dem Suchkriterium entsprechen, in unserem Beispiel die Songs über 10 min. Dazu können wir die einzelnen Suchbedingungen in der Strukturabfrage mit Bezeichnern belegen. Diese Bezeichner können dann wiederum in der Export-Definition angesprochen werden.



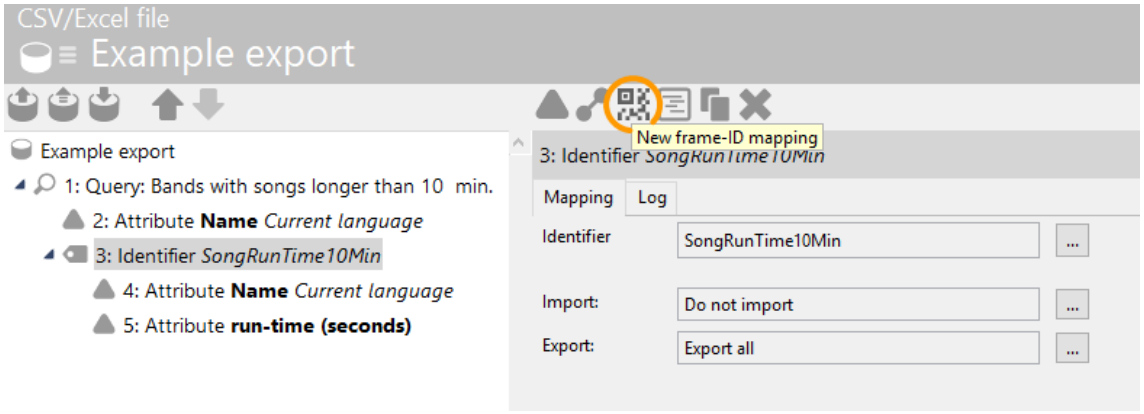
1.5.5.2. Export von Sammlungen semantischer Objekte

Auch Sammlungen semantischer Objekte können exportiert werden. Diese brauchen ebenfalls einen Registrierungsschlüssel, den man unter TECHNIK > Strukturordner setzen kann.



1.5.5.3. Export der Frame-ID

Die Abbildung der Frame-ID ermöglicht es uns, die in der semantischen Graph-Datenbank für ein Wissensnetzelement vergebene ID, zu exportieren. Hierzu wählen wir einfach das Objekt, den Typ oder die Eigenschaft aus, für die wir die ID brauchen und wählen dann den Button "Neue Abbildung der Frame-ID":

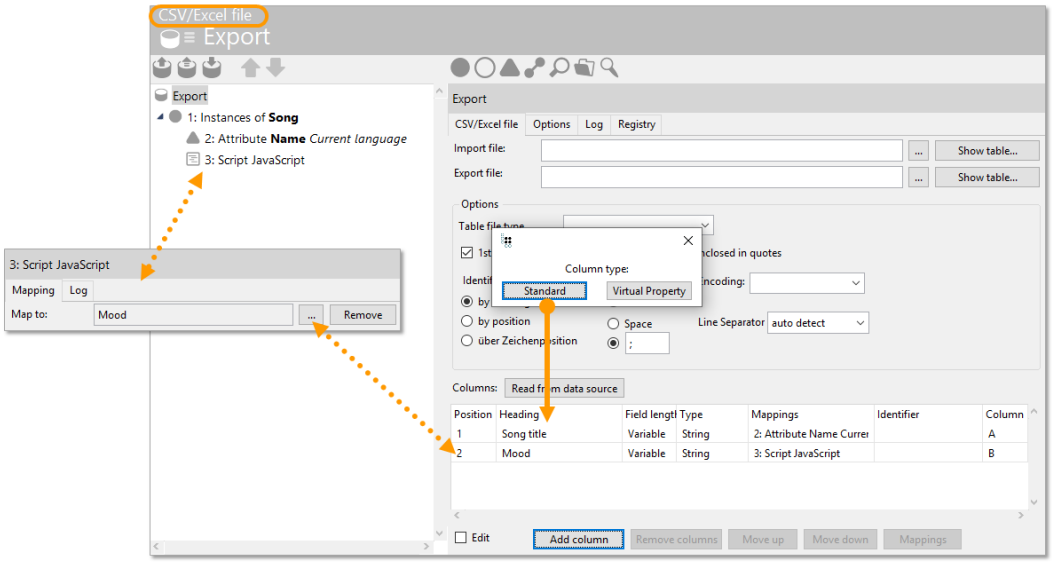


Wir können außerdem entscheiden, ob wir die ID im Format eines Strings wollen (ID123_456) oder, ob wir sie als 64 Bit Integer ausgegeben haben wollen.

1.5.5.4. Export mithilfe von Skripten

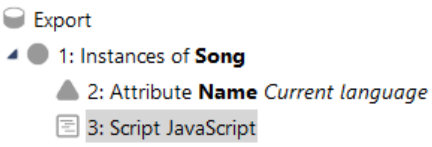
Schließlich steht uns beim Export noch ein weiteres mächtiges Werkzeug zur Verfügung: die Skriptabbildung.

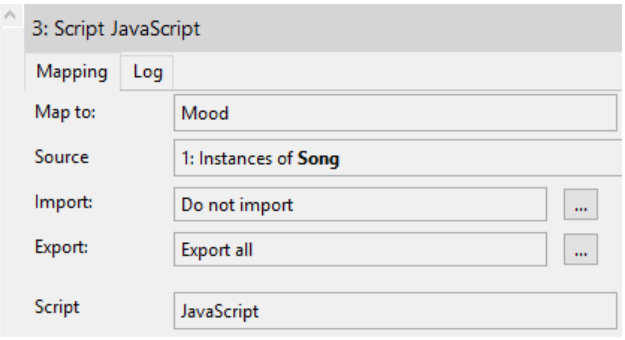
Zunächst spezifizieren wir die Spalten für die zu exportierenden Eigenschaften. An der Exportabbildung der jeweiligen Eigenschaft erfolgt dann die Spaltenzuweisung ("Abbilden auf"):



Die Skriptabbildung findet beispielsweise dann Verwendung, wenn wir drei Attribute aus der semantischen Graph-Datenbank zu einer ID zusammensetzen wollen. Allerdings kann es sein, dass der Export dann langsamer ist. (Bei einem Import könnte man dies einfacher über eine virtuelle Eigenschaft abbilden. Die Verwendung von virtuellen Eigenschaften wird im Kapitel Tabellenspalten erklärt.)

Der folgende Fall ist ein weiteres Beispiel für die Verwendung eines Skripts bei einem Export. Es zeigt wie mehrere Eigenschaften mit einem Trennzeichen in eine Zelle geschrieben werden können. In diesem Fall wollen wir eine Tabelle erzeugen, die in der ersten Spalte die Songnamen und in der zweiten Spalte alle Stimmungen der Songs mit Komma getrennt aufführt:



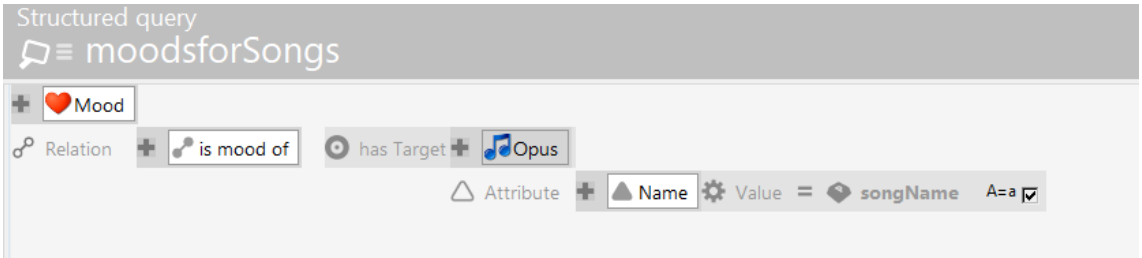


Um die zweite Spalte zu erzeugen benötigen wir folgendes Skript:

```
function exportValueOf(element) {
    var mood = "";
    var relTargets = $k.Registry.query("moodsforSongs").findElements({
        songName: element.attributeValue("objectName")
    });
    if(relTargets && relTargets.length > 0) {
```

```
        for(var i=0; i < (relTargets.length-1); i++) {
            mood += relTargets[i].attributeValue("objectName") + ", ";
        }
        mood += relTargets[relTargets.length-1].attributeValue("
objectName");
    }
    return mood;
}
```

Das Skript beinhaltet folgende Strukturabfrage (Registrierungsschlüssel: "stimmungenZuSongs"):



Über den Ausdruck "findElements" können wir auf einen Parameter (hier "songName") innerhalb der Abfrage zugreifen. "objektName" ist der interne Name des Namensattributs in diesem semantischen Modell.

Innerhalb der if-Anweisung sagen wir, dass wenn ein Element mehrere Relationsziele hat, diese durch ein Komma getrennt dargestellt werden sollen. Nach dem letzten Relationsziel, das die Schleife durchläuft, soll keine Komma mehr stehen. Auch wenn ein Element nur ein Relationsziel hat, wird dies demnach ohne Komma dargestellt.

Das Ergebnis ist eine Liste der Songs mit allen ihren Stimmungen, die durch Komma getrennt in der zweiten Spalte der Tabelle stehen:

Song title	Mood
Black Country Rock	
19th Nervous Breakdown	
A Maniac Depressive Named Laughing Boy	
A Place For My Head	aggressive
All the Madmen	
Bipolar	
Bleed It Out	
Bleed Like Me	
Breaking The Habit	
By Myself	aggressive
Back To Black	dramatic, bittersweet, swinging
China Girl (Bowie)	melancholic/dull, cold
Climbing up the Walls	
Crawling	aggressive
Creep	anthemic, elegiac, dramatic, lethargic, melancholic/dull
Digging In The Dirt	

1.5.5.5. Export-Aktionen bei Datenbankexporten

Die Abbildung der Eigenschaften eines Objekts wird für einen Export in eine Datenbank genauso vorgenommen wie für einen Import und wie für alle anderen Mappings. Einzig ist für den Export die Export-Aktion zu bestimmen. Diese gibt an, welche Art von Query in der Datenbank ausgeführt werden soll. Es stehen drei Export-Aktionen zur Verfügung:

Folgende Aktionen stehen in dem sich öffnenden Auswahldialog zur Verfügung:

- **Datensätze in Tabelle neu anlegen** : Es werden neue Datensätze in der Datenbanktabelle hinzugefügt. Diese Aktion entspricht einem INSERT.
- **Existierende Datensätze aktualisieren** : Die Datensätze werden über eine ID in der Tabelle identifiziert. Sie werden nur überschrieben, wenn der Wert sich geändert hat. Gibt es keinen passenden Datensatz, dann wird ein neuer hinzugefügt. Diese Aktion entspricht einem UPDATE.
- **Tabelleninhalt beim Export überschreiben** : Alle Datensätze werden erst gelöscht und dann neu geschrieben. Diese Aktion entspricht einem DELETE auf der ganzen Tabelle mit folgendem INSERT.

1.5.6. RDF-Import und -Export

RDF ist ein Standardformat für semantische Datenmodelle. Mit dem RDF-Import und -Export können Daten des Knowledge-Graphs mit anderen Anwendungen ausgetauscht, aber auch Daten von einem Knowledge-Graph in einen anderen übertragen werden.

Beim Import werden die Formate RDF/XML und Turtle unterstützt, beim Export RDF/XML.

Für weitergehende Informationen über den RDF-Standard siehe W3C.

1.5.6.1. Grundlagen

Um die Inhalte in einer RDF-Datei zu referenzieren, wird eine URI verwendet, die im folgenden *RDF-URI* genannt wird. Diese RDF-URI wird oft aus einer Basis-URI und einer ID zusammengesetzt, die im folgenden *RDF-ID* genannt wird:

Beispiel:

Basis-URI	`http://www.example.org/heavytools#`
RDF-ID	`mini2000`
Resultierende URI	`http://www.example.org/heavytools#mini2000`

Es können auch URI-Namensräume benannt werden. Komplette URIs können dann als Kombination aus Benennung und RDF-ID angegeben werden.

Namensräume in RDF-XML

```
<rdf:RDF
  xml:base="http://www.example.org/heavytools#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ht="http://www.example.org/heavytools#">

  <ht:excavator rdf:ID="mini2000">
    <rdfs:label>Mini 2000</rdfs:label>
  </ht:excavator>
</rdf:RDF>
```

Attributwerte werden als Zeichenkette dargestellt. Beim Import in und Export aus einem Knowledge-Graph werden die Zeichenketten gemäß XSD-Datentypen kodiert, beispielsweise wird ein Datum gemäß `xs:date` dargestellt.

Datum in RDF-XML

```
<ht:dateOfManufacture>2003-04-05</ht:dateOfManufacture>
```

Die optionale Sprache wird gemäß RFC 3066 angegeben. *Deutsch* wird beispielsweise als `de` kodiert:

Attribut mit Sprachangabe in RDF-XML

```
<rdfs:label xml:lang="de">Bagger</rdf:prefLabel>
```

Bei Relationen wird die URI des Relationsziels angegeben. Diese kann anhand der Basis-URI verkürzt werden:

Relation mit verkürzter URI in RDF-XML

```
<ht:uses rdf:resource="#excavator"/
```

HINWEIS

RDF/XML erlaubt bei `rdf:resource` nicht die Verwendung von Namensräumen, folgendes ist keine gültige URI:

```
<ht:uses rdf:resource="ht:excavator"/
```

1.5.6.2. Identifizierung von Objekten im Knowledge-Graph

Objekte können über folgende Eigenschaften identifiziert werden:

Attribut *rdf:about*

Vollständige URI des Objekts.

Attribut *rdf:ID*

Verkürzte URI. Die ID bildet zusammen mit einer Basis-URI eine komplette URI.

Attribut *RDF-URI-Alias*

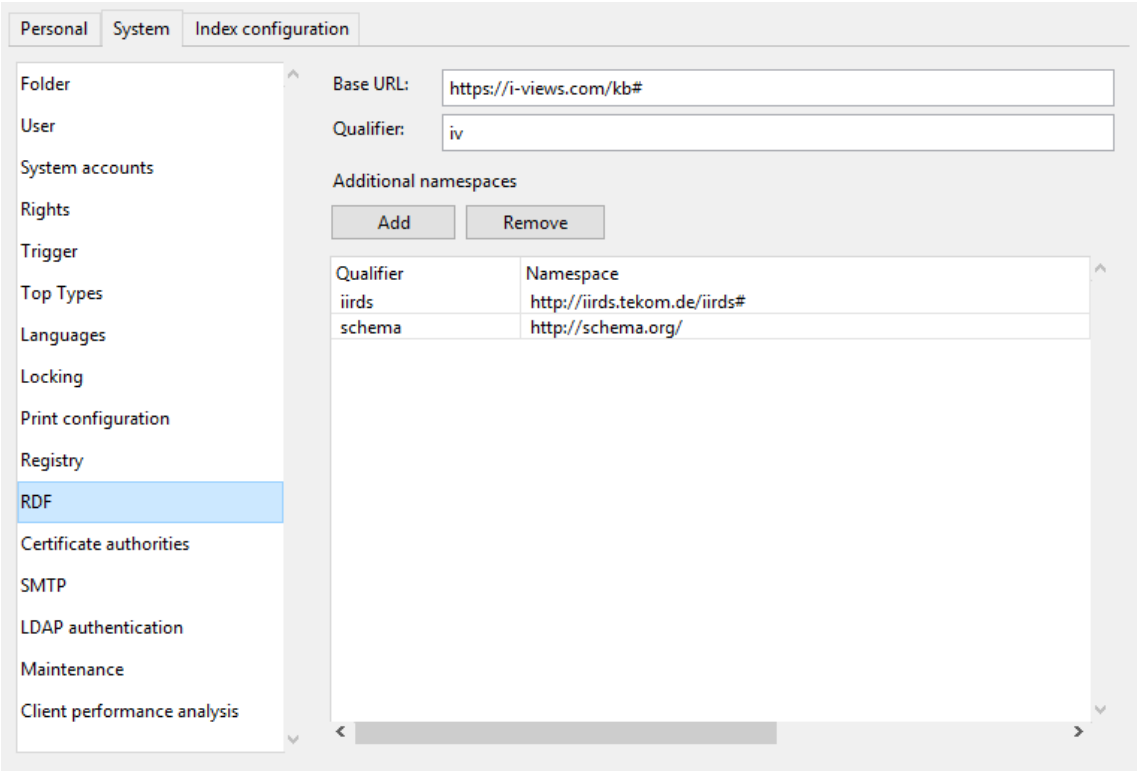
Zusätzliche URI, die beim Import zur Identifizierung verwendet wird.

Frame-ID

Interne, unveränderliche ID eines Objekts. Kann nur verwendet werden, wenn man Daten von einem Knowledge-Graphh in eine Kopie überträgt, zum Beispiel von einer Entwicklungs- in eine Produktionsumgebung.

1.5.6.3. Globale Einstellungen

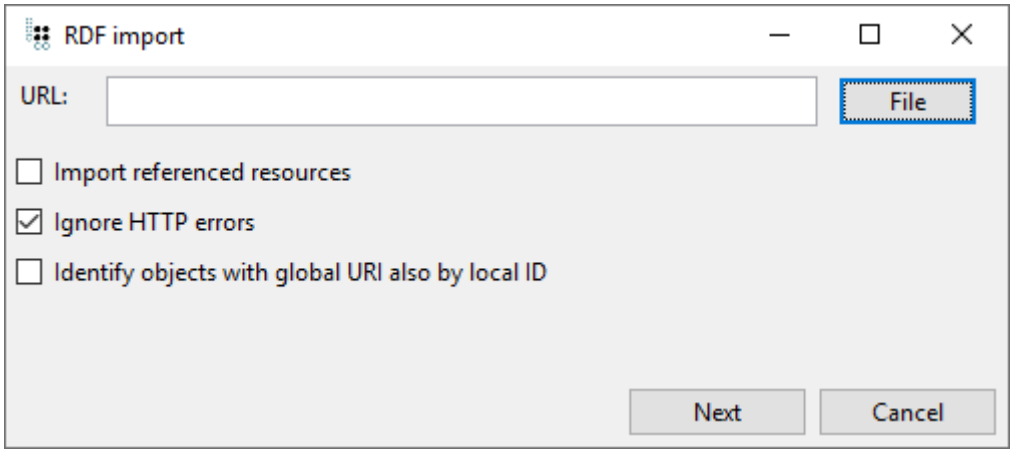
Die Basis-URL des Knowledge-Graphs wird in den globalen Einstellungen des Knowledge-Builders eingestellt und ist sowohl für den Import als auch den Export gültig:



Die Angaben bei "Zusätzliche Namespaces" gelten nur für den Export und werden dort zur Verkürzung von URLs verwendet.

1.5.6.4. RDF-Import

Der RDF-Import wird im Hauptmenü über *Werkzeuge* > *RDF* > *RDF-Import* gestartet. Im darauffolgenden Dialogfenster kann die zu importierende Datei ausgewählt werden:



Optionen

Referenzierte Ressourcen importieren

Wenn diese Option gewählt wird, werden alle in der RDF-Datei referenzierten Ressourcen mit importiert.

HINWEIS	Zu beachten ist, dass die referenzierten Ressourcen wiederum selbst weitere referenzierte Ressourcen enthalten können und dadurch unkontrolliert Daten importiert werden

HTTP-Fehler ignorieren

Der Knowledge-Builder gibt Fehlermeldungen aus, wenn das RDF-Label "namespace" in der URL der RDF-Datei fehlt. Es wird dann nur die Namespace http-URL berücksichtigt. Ein Aktivieren dieser Option unterdrückt die Fehlermeldungen.

Objekte mit globaler URI auch durch lokale ID identifizieren

Wenn diese Option aktiviert ist, werden Objekte anhand der RDF-ID auch dann identifiziert, wenn die Basis-URI nicht übereinstimmt. Diese Optional sollte nur in Ausnahmefällen aktiviert werden.

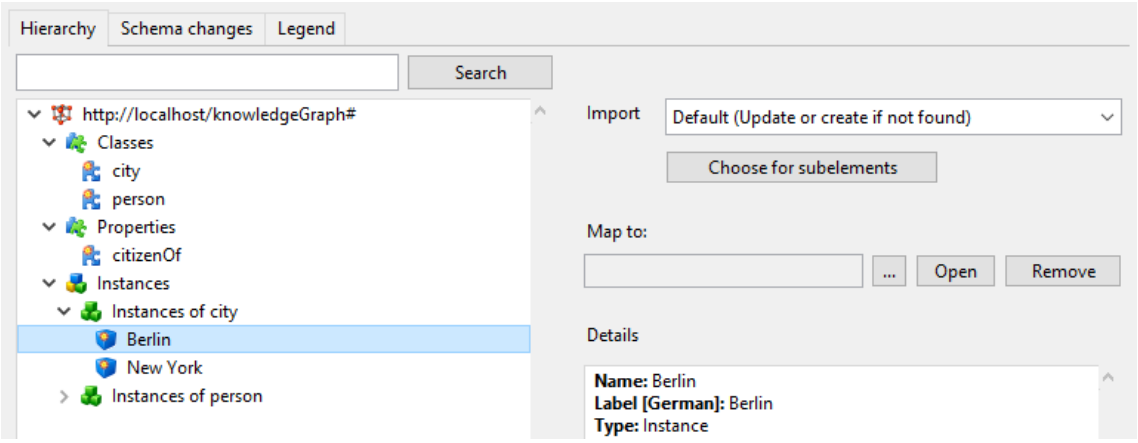
1.5.6.4.1. Import konfigurieren

Nach dem Einlesen der zu importierenden Datei wird eine Übersicht über die eingelesenen Daten angezeigt. Zu diesem Zeitpunkt sind die Daten noch nicht importiert.

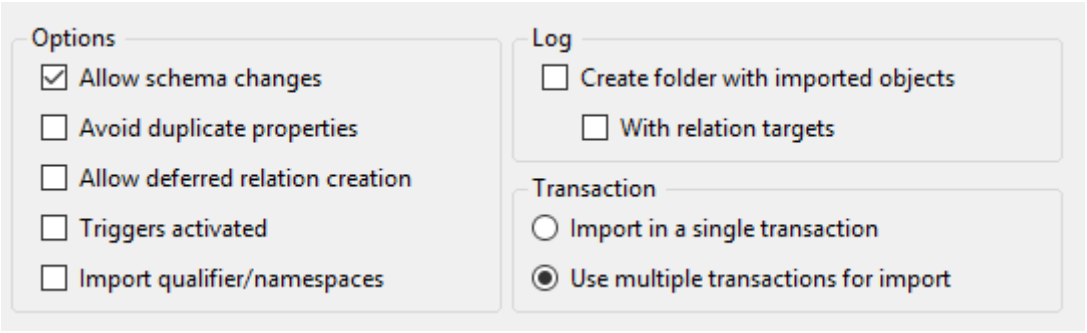
Der Reiter *Schemaänderungen* listet alle geplanten Änderungen am Schema auf. Der Reiter *Legende* enthält eine Erklärung der verwendeten Symbole.

Manuelle Zuordnung

Objekte werden anhand der im Abschnitt Identifizierung von Objekten im Knowledge-Graph beschriebenen Eigenschaften identifiziert. Die Zuordnung kann manuell angepasst werden, indem man in der Hierarchie ein Objekt auswählt und auf der rechten Seite bei *Abbilden auf* ein Objekt auswählt.



Optionen



Änderungen am Schema erlauben

Wenn Änderungen am Schema vorgesehen sind, sollte diese Option aktiviert werden.

Eigenschaftsduplikate vermeiden

In RDF-Dateien ist eine Identifizierung von Eigenschaften nicht üblich. Um zu verhindern, dass sich bei einem Import von bereits vorhandenen Objekten Duplikate von Eigenschaften anhäufen, kann man diese Option aktivieren. Vorhandene Eigenschaften eines Typs werden dann anhand des Werts (bei Attributen) beziehungsweise des Relationsziels (bei Relationen) identifiziert.

Verweise auf abwesende Ressourcen

Wenn Daten aus öffentlichen Quellen importiert werden, kann das Attribut `RDF-refersTo` als Ersatz-Verweis für (vorübergehend nicht verfügbare) Abhängigkeiten verwendet werden. Dieses Attribut dient im Fall eines nachgelagerten Importes für eine Identifizierung der Quellen. Insbesondere ist die nützlich, wenn in der RDF-Datei leere Teile mit URL ohne Typendefinition vorliegen.

Trigger aktiviert

Standardgemäß sind Trigger-Funktionen während des Importes nicht aktiv. Wenn dennoch Trigger benötigt werden, können die durch diese Option aktiviert werden.

Qualifier/Namespaces importieren

Diese Option ist dann sinnvoll, wenn eine RDF-Datei reimportiert wird, also aus dem gleichen Wissensnetz stammt. Wenn eine RDF-Datei mit fremdem Namensraum vorliegt, sollte diese Option nicht verwendet werden.

Log-Optionen

Ordner mit importierten Objekten anlegen

Diese Option erlaubt es, in einem im Arbeitsordner befindlichen Wissensnetz-Ordner die importierten Elemente nach dem Import zu inspizieren.

Mit Relationszielen

Wenn die RDF-Datei neue Objekte mit Relationen enthält, deren Relationsziele bereits im Wissensnetz existieren, so werden die Relationsziele im Ordner der importierten Objekte enthalten sein.

Transaktion

In einer Transaktion importieren

Alle Daten werden in einer Transaktion eingespielt. Bei Transaktions-Konflikten wird der Knowledge-Graph nicht geändert.

Mehrere Transaktionen verwenden

Diese Option wird empfohlen, wenn die RDF-Datei eine große Datenmenge enthält. Bei Transaktions-Konflikten werden dann nur Teile der Datei importiert.

Weitere Möglichleiten des RDF-Import/Export

RDF-Dateien können auch über die REST-Schnittstelle mithilfe eines JavaScript-Funktion importiert werden. Weitere Informationen sind hierzu in der Dokumentation zur JavaScript-API zu finden: https://documentation.i-views.com/6.1/javascript-api/$k.RDFImporter.html

1.5.6.5. RDF-Export

Es können der gesamte Knowledge-Graph oder einzelne Elemente als RDF-Datei exportiert werden.

Gesamter Knowledge-Graph

Der RDF-Export des gesamten Knowledge-Graphs wird im Hauptmenü über *Werkzeuge > RDF > RDF-Export* gestartet.

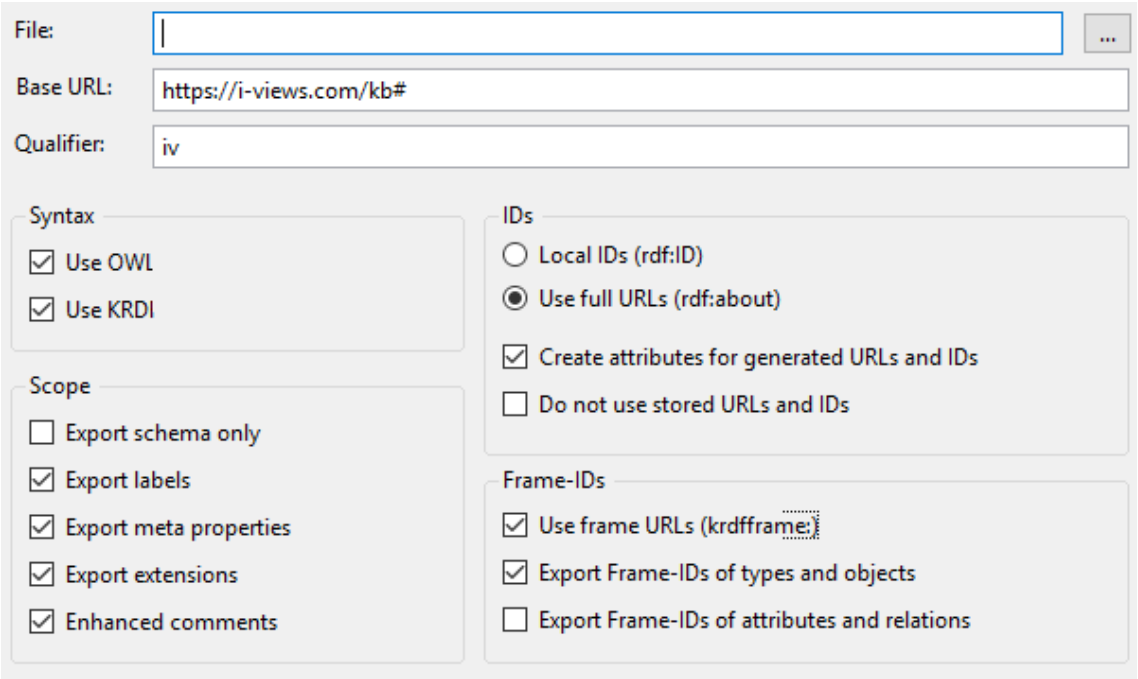
Einzelne Objekte

Einzelne Objekte können exportiert werden, indem man im Kontextmenü des Objekts *RDF-Export* auswählt.

Elementen aus einem Sammlung semantischer Elemente

Mehrere Objekte können exportiert werden, indem man eine Sammlung semantischer Elemente anlegt und im Kontextmenü der Sammlung *RDF-Export* auswählt.

1.5.6.5.1. RDF-Export Einstellungen



Syntax

OWL verwenden

Weil der OWL-Standard (web ontology language) mehr Optionen zur Verfügung stellt als die konventionelle RDF-Syntax, ist diese Option stets zu empfehlen. Ausnahme hiervon ist der Fall, dass die RDF-Datei für ein anderes System wiederverwendet wird, das OWL nicht verarbeitet.

KRDF verwenden

Die KRDF-Syntax enthält zusätzliche Elemente zur Repräsentation von Knowledge-Graph-spezifischen Features wie zum Beispiel

- Schema von Eigenschaften mit Typen als Domänen
- Erweiterungen

HINWEIS	Für den Transfer zwischen Knowledge-Graphen sollte diese Option unbedingt aktiviert werden, für den Austausch mit externen System dagegen deaktiviert.

Umfang

Nur Schema exportieren

Diese Funktion dient dazu, im Falle eines beabsichtigten Schema-Transfers nur das Schema des Knowledge Graphen zu exportieren, ohne Instanzen.

Namen als rdfs:label exportieren

Labels werden nicht als Attribute exportiert, sondern in Form eines Label-Tags mit der Syntax `<rdfs:label xml:lang="de">`.

Metaeigenschaften exportieren

Im Sinne der offiziellen RDF Spezifikation ist die Repräsentation von Meta-Eigenschaften nicht vorgesehen. Dennoch können Meta-Eigenschaften als Konstrukt aus einer Aussage über eine andere Aussage (*Reifikation*) interpretiert werden.

Erweiterungen exportieren

Ermöglicht den Export von Erweiterungen semantischer Elemente.

Erweiterte Kommentare

Erzeugung erweiterter XML-Kommentare. Der zu exportierenden RDF-Datei werden Kommentare hinzugefügt, die das Dokument nach objects (Instanzen), related objects (Relationszielen) und referenced schema (referenziertem Schema) gliedern und zusätzlich Aussagen zu abgebildeten Relationen mit Ausgangs- und Zielobjekt enthält.

IDs

IDs verwenden (rdf:ID)

Exportiert verkürzte RDF-IDs zur Identifizierung von Objekten, sofern möglich.

URLs verwenden (rdf:about)

Exportiert vollständig URLs zur Identifizierung von Objekten.

Attribute für generierte URLs und IDs anlegen

Die generierten RDF-IDs/URIs werden in den Attributen *rdf:ID* oder *rdf:about* gespeichert.

Gespeichert URLs und IDs nicht verwenden

Es werden neue RDF-IDs/URIs generiert, statt auf im Attribut *rdf:ID* oder *rdf:about* gespeicherte Werte zurückzugreifen.

Frame-IDs

Frame-IDs von Typen und Objekten exportieren

Die Frame-ID wird als zusätzliche Eigenschaft `krdf:frameID` exportiert.

RDF-IDs von Eigenschaften exportieren

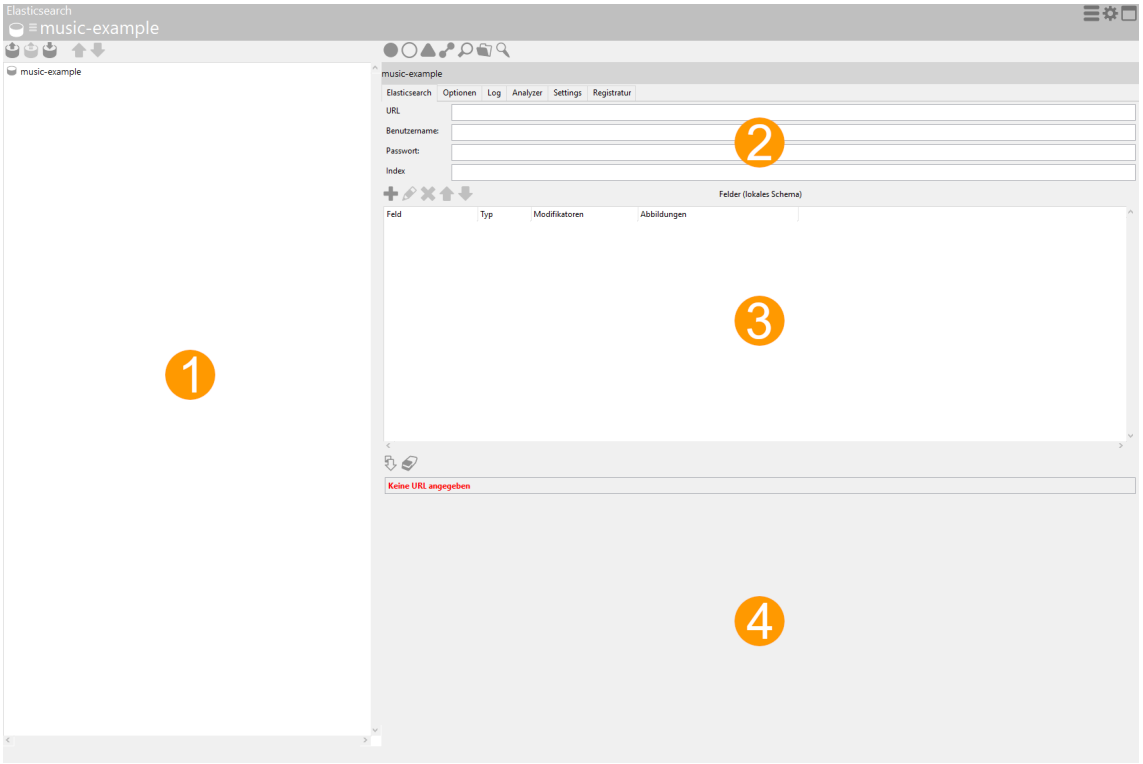
Exportiert bei allen Eigenschaften eine Reifikations-ID (*rdf:ID*). Wenn diese Option nicht aktiviert ist, wird nur bei Bedard eine Reifikations-ID exportiert.

1.5.7. Externer Index in Elasticsearch

Elasticsearch ist eine Open-Source-Suchmaschine auf Basis von Lucene, die für das Indexieren, Durchsuchen und Analysieren von großen Datenmengen entwickelt wurde. Besonders effizient sind hierbei Werte- und Volltextsuche. In i-views besteht die Möglichkeit, die Daten aus dem Wissensnetz mithilfe eines Mappings nach Elasticsearch zu exportieren. Dadurch entsteht ein externer Index, auf den zahlreiche Suchfunktionen und -optionen angewendet werden können. Weitere Informationen zu Elasticsearch gibt es hier auf der offiziellen Seite.

1.5.7.1. Anlegen einer Abbildung

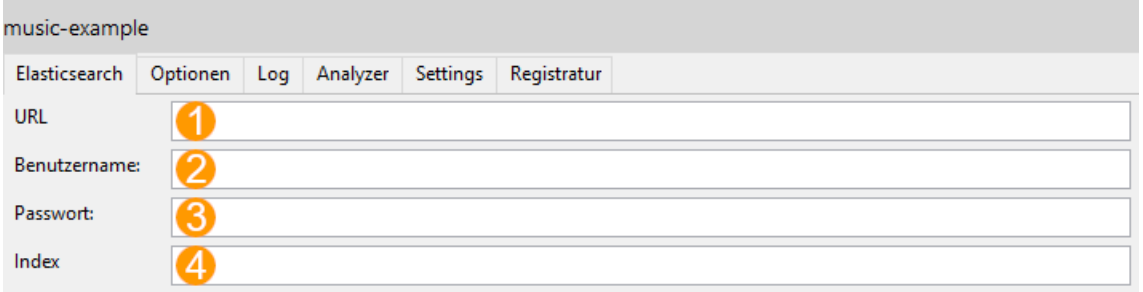
Um die Schnittstelle von i-views zu Elasticsearch nutzen zu können, muss als Erstes eine neue Datenabbildung angelegt werden. Dafür muss im Arbeitsordner mit dem Button **Neue Abbildung einer Datenquelle** die Datenquelle "Elasticsearch" ausgewählt und ein Name definiert werden (siehe 1.5.1.2). Nach dem Bestätigen öffnet sich anschließend die Konfigurationsansicht:



Die Ansicht lässt sich in vier Bereiche unterteilen:

1. Mapping
2. Metadaten
3. die definierten Felder (lokales Schema)
4. später die Übersicht der Felder in Elasticsearch (Schema in Elasticsearch)

Zu Beginn sollten alle nötigen Metadaten für den externen Index angegeben werden.

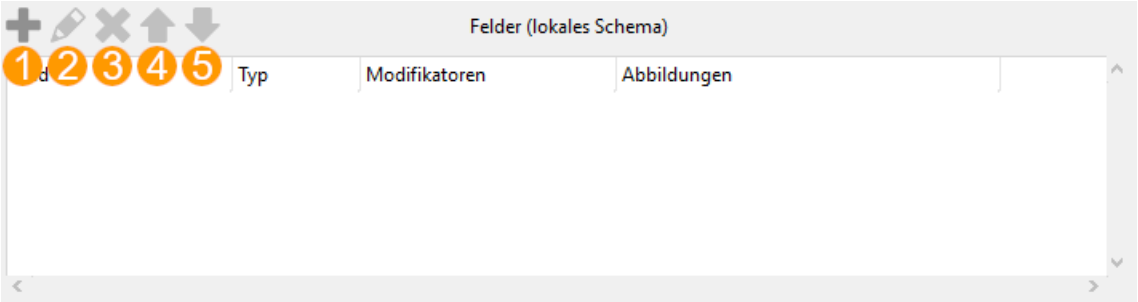


1. (obligatorisch) Die URL, über die der externe Index in Elasticsearch gehostet werden soll
2. (optional) Benutzername, wenn ein Benutzer definiert ist
3. (optional) Passwort, wenn ein Benutzer definiert ist
4. (obligatorisch) Der gewünschte Name des externen Index

Nachdem alle obligatorischen Felder ausgefüllt wurden, können nun die gewünschten Felder lokal definiert werden.

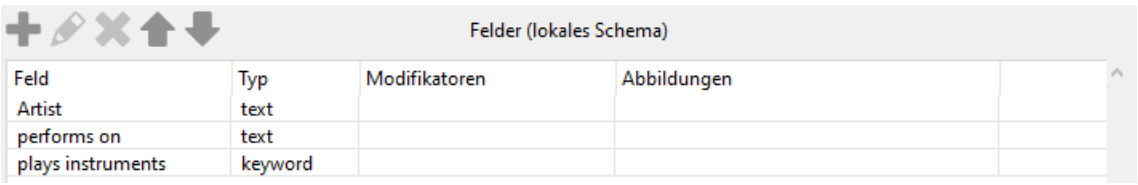
1.5.7.2. Lokales Schema

Das Schema, welches zuerst lokal erstellt wird, legt fest, wie der externe Index auf Elasticsearch aussehen soll. Es werden Felder definiert, die festlegen, wie die Daten letztendlich gespeichert werden. Dabei stellen die Felder die Spalten des externen Index dar. Für jedes Feld muss grundsätzlich ein Name und der Datentyp definiert werden. Dies ist in der folgenden Ansicht möglich:



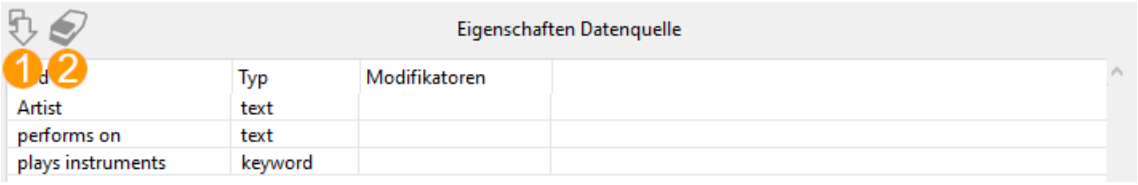
1. **Hinzufügen** Legt ein neues Feld mit einem definierten Namen und Typ an.
2. **Bearbeiten** Name, Typ und Modifikatoren des ausgewählten Feldes können geändert werden. Als Modifikator kann für ein Feld ein Analyzer spezifiziert werden, was in einem späteren Abschnitt thematisiert wird (siehe 1.5.5.9).
3. **Löschen** Löscht das/die ausgewählte(n) Feld(er).
4. **Felder aus Datenquelle importieren** Überschreibt das lokale Schema mit dem Schema in Elasticsearch.
5. **Eigenschaften in Datenquelle exportieren** Lädt das lokale Schema in das Schema auf der Seite von Elasticsearch.

Das lokale Schema, welches alle Felder beinhaltet, definiert, wie der externe Index aussieht. Ein solches Schema könnte wie folgt aussehen:



Nachdem ein neues Schema definiert wurde, kann mit dem Pfeil nach unten **Eigenschaften in**

Datenquelle exportieren das lokale Schema nach Elasticsearch exportiert werden. Dadurch öffnet sich die Ansicht der Felder in Elasticsearch, wo das übertragene Schema angezeigt wird:



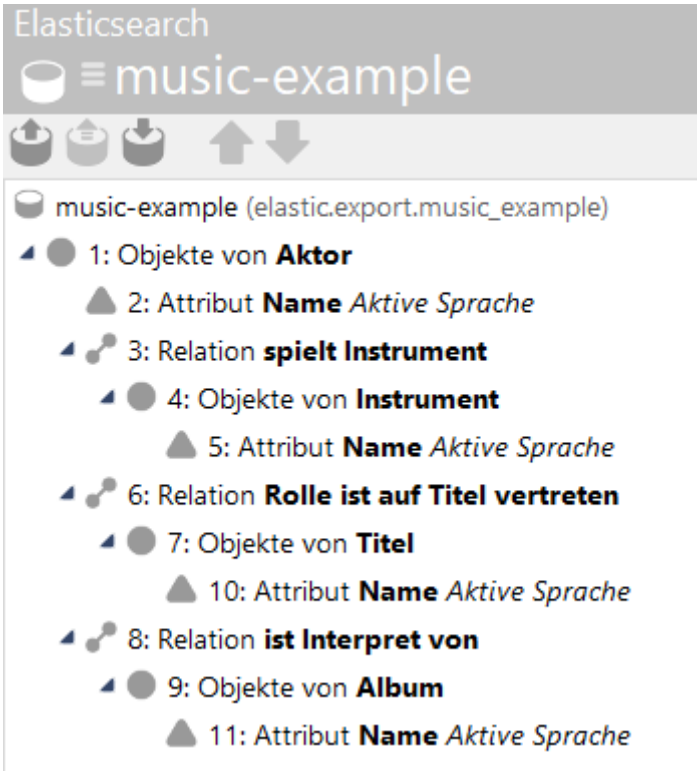
1. **Aktualisieren (F5)** Das Schema wird aus Elasticsearch neu geladen.
2. **Alle Felder zurücksetzen** Der gesamte externe Index wird zurückgesetzt. Alle exportierten Daten werden gelöscht. Nur das Schema bleibt bestehen.

HINWEIS	Generell sind die Felder ein sehr empfindlicher Teil, der während des Betriebs kaum oder am besten gar nicht geändert werden sollte. Werden Felder bearbeitet, während schon ein externer Index besteht, könnte dies zu Anomalien führen. Daher sollte der externe Index bei großen Änderungen im Schema immer neu angelegt werden, was mithilfe des Radiergummis **Alle Felder zurücksetzen** erfolgt.

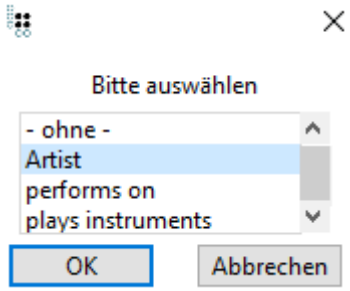
Nachdem das gewünschte Schema erstellt wurde, sollte nun ein Mapping definiert werden.

1.5.7.3. Mapping

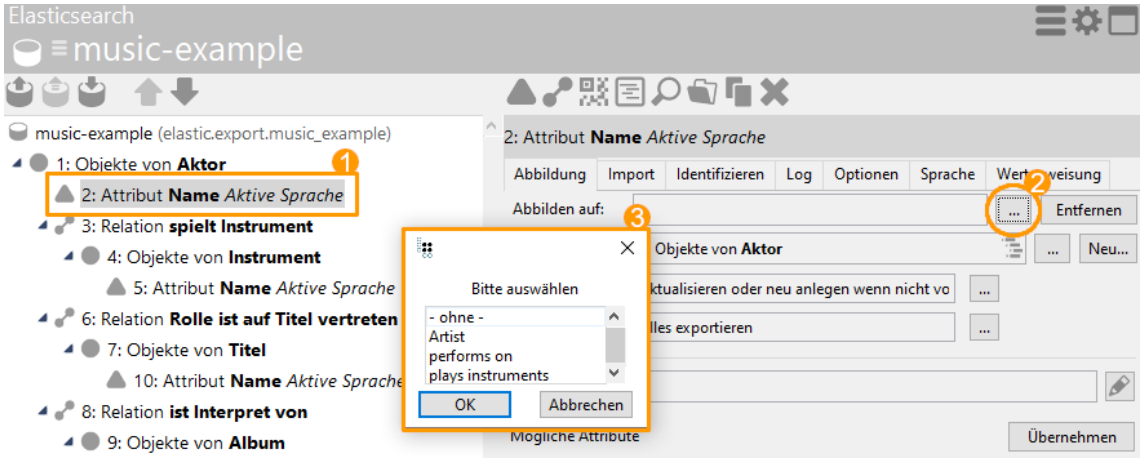
Im Mapping wird festgelegt, welche Daten aus dem Wissensnetz letztendlich zum externen Index exportiert werden. Dafür wird im Bereich des Mappings die Datenquelle ausgewählt. Danach kann Schritt für Schritt der Aufbau des Mappings erfolgen (siehe 1.5.1.3). Ein Beispiel für ein Mapping könnte wie folgt aussehen:



Wurde das lokale Schema zuvor schon erstellt, wird während dem Erstellen des Mappings bei jedem Teil ein Auswahldialog geöffnet, in dem ein Feld ausgewählt werden kann:



Hier kann der aktuell erstellte Teil des Mappings einem Feld für den Export zugewiesen werden. Wird **- ohne -** ausgewählt oder besteht noch kein lokales Schema, so kann auch nachträglich eine Zuweisung erfolgen. Dafür muss ein Teil im Mapping ausgewählt werden und bei **Abbilden auf** das gewünschte Feld ausgewählt werden:

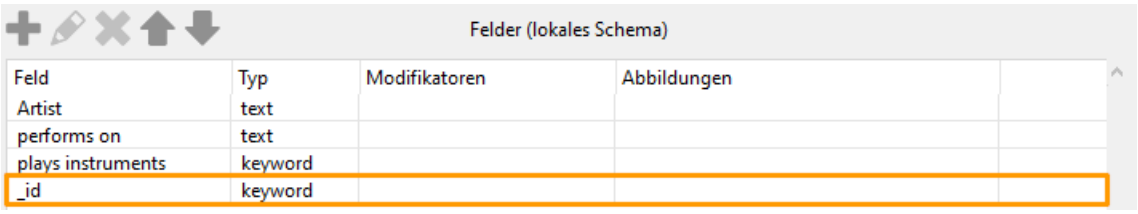


Nachdem das Mapping definiert und dem lokalen Schema zugeordnet wurde, sollte für einen reibungslosen Betrieb eine eindeutige Identifikation der zu exportierenden Objekte konfiguriert werden.

1.5.7.4. Identifikation

Für eine eindeutige Identifikation der exportierten Objekte generiert Elasticsearch automatisch eindeutige Bezeichner (IDs) für jedes Objekt. Jedoch empfiehlt es sich, die bereits vorhandenen IDs der Objekte im Wissensnetz zu verwenden, um für eine eindeutige Identifikation zu sorgen. Somit besteht eine direkte Verbindung von den Objekten im Wissensnetz zu den Einträgen im externen Index, was alle weiteren Vorgänge erleichtert.

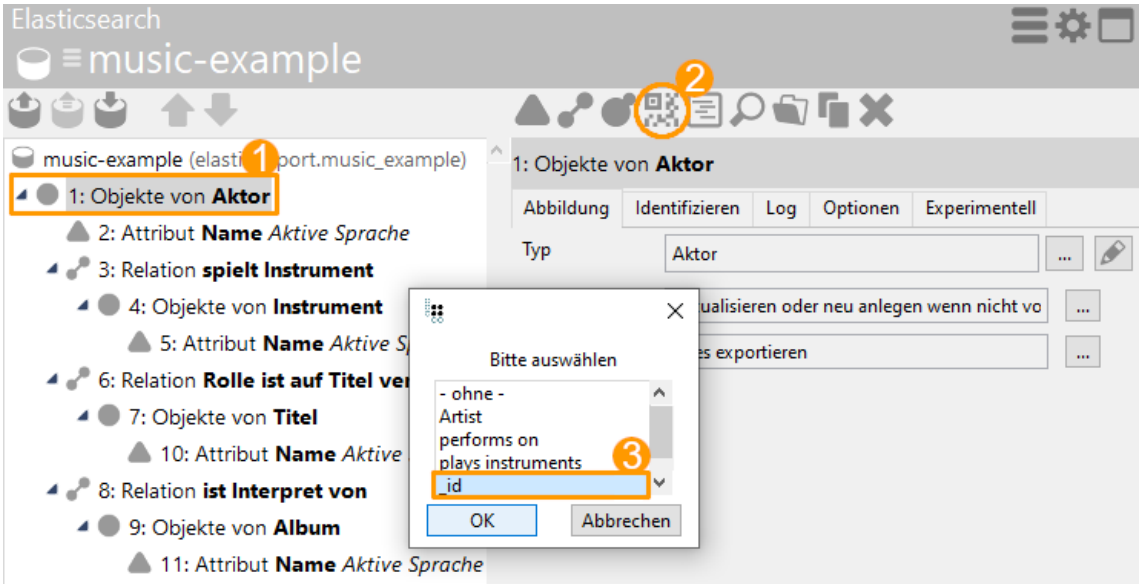
Dieser Zustand kann erreicht werden, indem nach Exportieren des lokalen Schemas das Schema aus Elasticsearch wieder importiert wird. Dadurch wird im lokalen Schema ein neues Feld mit dem Namen "_id" angezeigt. Dieses Feld wird automatisch von Elasticsearch erzeugt und zum Speichern der IDs verwendet.



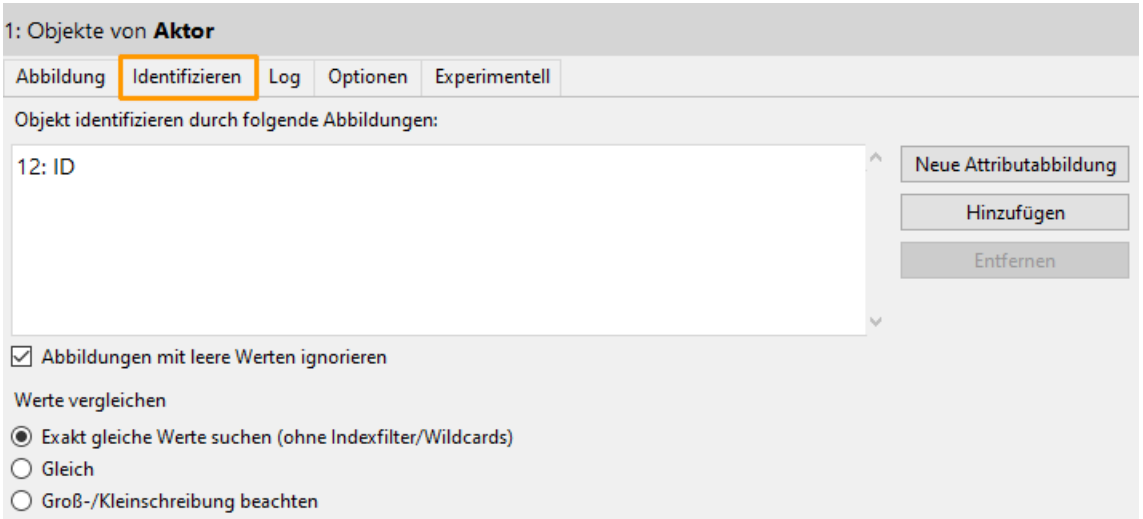
HINWEIS

Das ID-Feld sollte nur für die Abbildung von eindeutigen IDs aus dem Wissensnetz verwendet werden. Werden andere Abbildungen auf das Feld gemappt, beeinträchtigt dies die Funktionsweise des externen Index erheblich oder sorgt gar für einen totalen Funktionsverlust!

Zuvor wurde das von Elasticsearch bereitgestellte ID-Feld lediglich in das lokale Schema geladen. Jedoch muss erst für das Root-Objekt des Mappings eine neue Abbildung der ID hinzugefügt werden:



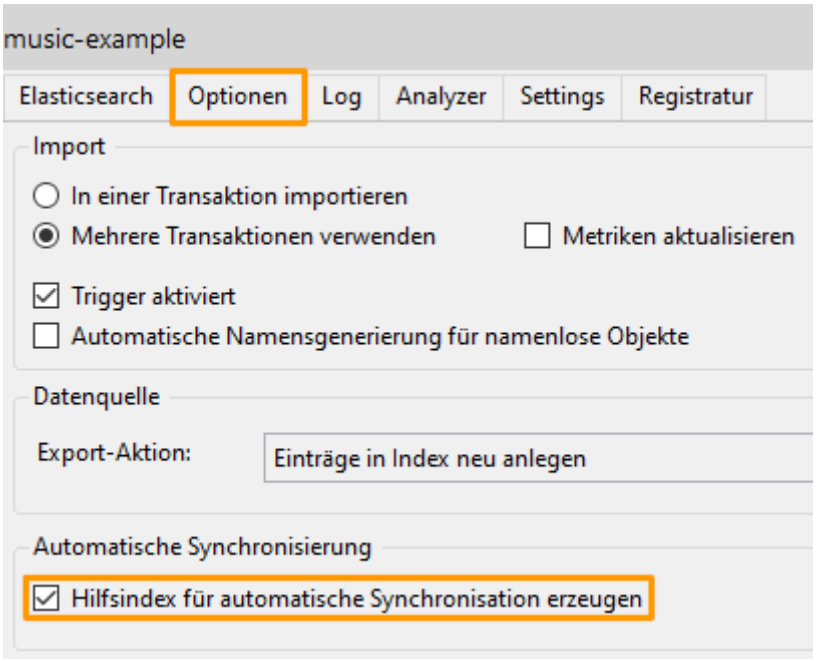
Die IDs der zu exportierenden Objekte aus dem Wissensnetz wurden nun dem Mapping hinzugefügt und dem ID-Feld zugewiesen. Außerdem sollte sichergestellt werden, dass die Root-Objekte auch durch diese IDs identifiziert werden. Bei Auswahl des Root-Objekts sollte dafür im Reiter **Identifizieren** unter **Objekt identifizieren durch folgende Abbildungen** nur die zuvor zum Mapping hinzugefügte ID-Abbildung definiert sein. Wird das Root-Objekt durch ein/mehrere andere Attribute identifiziert, sollten diese gelöscht und nur die ID-Abbildung hinzugefügt werden. Dies sieht wie folgt aus:



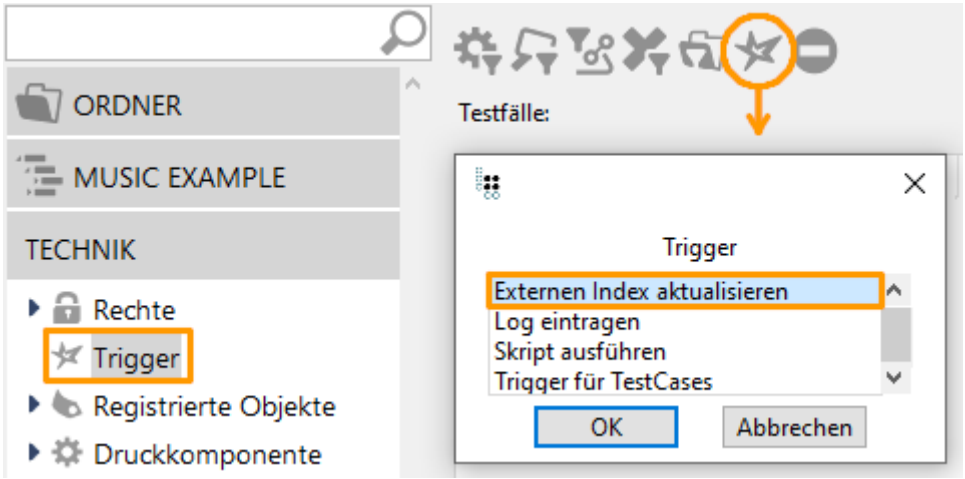
Nun werden bei einem Export die Einträge im externen Index über die IDs der Root-Objekte aus dem Mapping identifiziert. Dies ist wichtig für die Zuordnung von Objekten im Wissensnetz zu den korrespondierenden Einträgen im externen Index und ermöglicht weitere Funktionen, wie z.B. eine automatische Aktualisierung des externen Index bei internen Änderungen im Wissensnetz (Synchronisierung).

1.5.7.5. Synchronisation

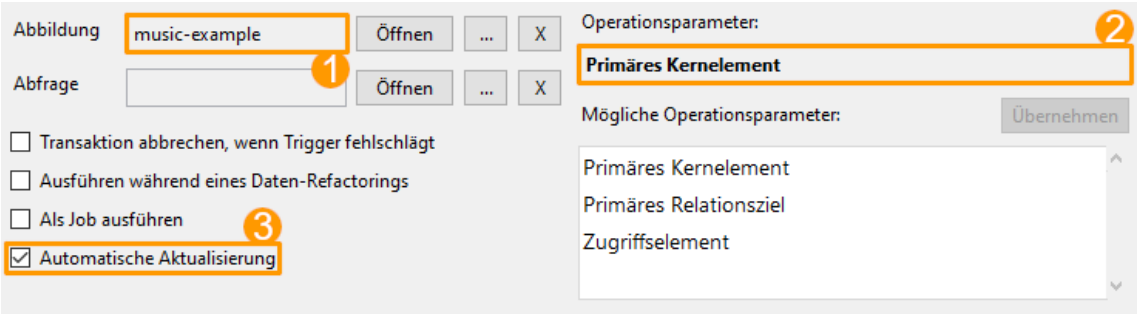
Damit die Daten im externen Index aktuell und somit konsistent zum Wissensnetz bleiben, sollte vor dem initialen Export unter dem Reiter **Optionen** die automatische Synchronisierung aktiviert werden:



Dadurch entsteht eine neue Spalte mit dem Bezeichner "_dependantIDs" im externen Index, welche als Hilfsindex für die Instandhaltung der Konsistenz verwendet wird. Jedoch reicht es noch nicht aus, allein diesen Hilfsindex zu erzeugen. Zusätzlich muss ein Trigger konfiguriert werden, der auf interne Datenänderungen anspringt und die eigentliche Akutalisierung einleitet:



Anschließend öffnet sich die Konfigurationsansicht für den Trigger, wo folgende Einstellungen vorgenommen werden müssen:



1. Auswahl des Mappings, für das der Trigger gelten soll.
2. Auswahl des Parameters **Primäres Kernelement**.
3. **Automatische Aktualisierung** aktivieren.

Nachdem der Trigger konfiguriert wurde, wird der externe Index bei einer internen Datenänderung im Wissensnetz automatisch aktualisiert.

1.5.7.6. Import und Export

An dieser Stelle kann nun ein Import/Export erfolgen. Die Buttons dafür befinden sich über dem Mapping:

1. **Import** Die Daten werden aus dem externen Index ins Wissensnetz importiert
2. **Interaktiver Import**
3. **Export** Die Daten werden aus dem Wissensnetz zum externen Index exportiert
4. **Nach oben** Bewegt die ausgewählte Abbildung nach oben
5. **Nach unten** Bewegt die ausgewählte Abbildung nach unten

Neben dem Import, mit dem Daten aus dem externen Index gemäß dem Schema in das Wissensnetz geladen werden, bietet der Export eine Vielzahl an Möglichkeiten.

1.5.7.7. Browser-Tool Elasticvue

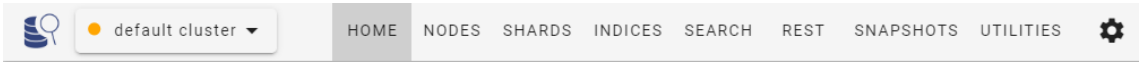
Nachdem die Daten zum externen Index exportiert wurden, können diese mithilfe der Browser-Erweiterung Elasticvue eingesehen werden. Die Erweiterung unterstützt die verbreitetsten Browser

und lässt sich unter https://elasticvue.com/ herunterladen.

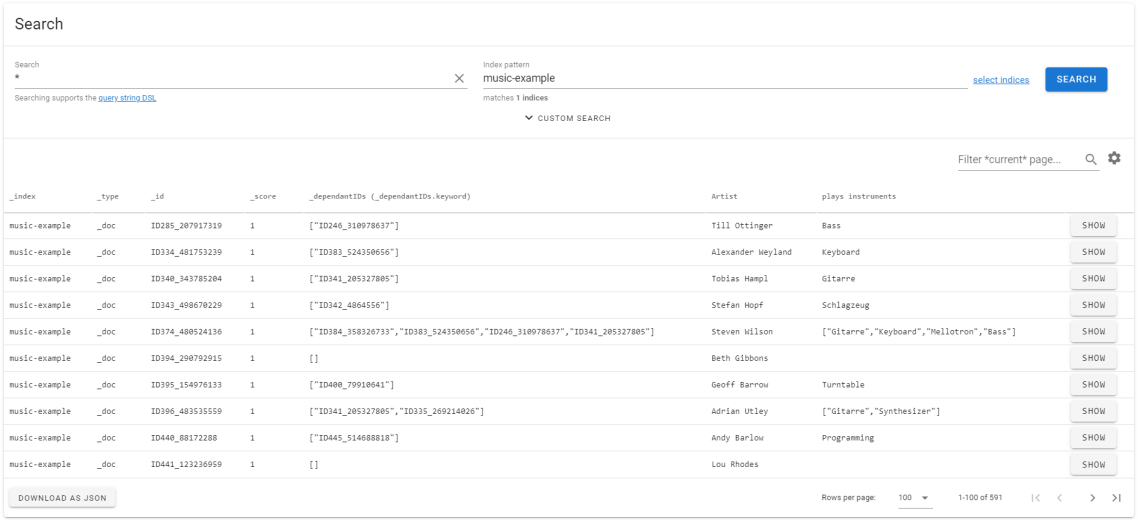
Nach erfolgreicher Installation kann Elasticvue unter den installierten Erweiterungen geöffnet werden. Bei erstmaliger Öffnung wird ein Setup benötigt, wo die Adresse, über die der Elasticsearch-Cluster laufen soll, unter URI eingegeben werden muss. Optional kann noch ein Benutzername und ein Passwort angegeben werden.

HINWEIS	Die URI muss identisch mit der URL sein, die bei der Abbildung eingetragen wurde.

Nach dem initialen Setup wird man auf das Dashboard weitergeleitet, was beim erneuten Öffnen der Erweiterung direkt erscheint. Die Navigationsleiste des Dashboards sieht wie folgt aus:



Unter **default cluster** können alle bestehenden Cluster ausgewählt, verwaltet, und neue Cluster hinzugefügt werden. Außerdem kann hier die Adresse eines Clusters eingesehen werden, die in den Metadaten der Abbildung in i-views eintragen werden muss. Unter **Home** werden viele weitere Metadaten zu einem Cluster angezeigt. Jeder Cluster besteht aus mehreren Nodes, für die man Informationen und eine Ressourcenübersicht unter **Nodes** finden kann. Jeder externe Index kann verteilt über mehrere sogenannte Shards verteilt werden, was unter dem Reiter **Shards** angezeigt wird. Der wichtigste Reiter zum Einsehen der Exportierten Daten ist **Indices**, wo alle externen Indexe aufgelistet werden. Wählt man einen Index aus, wird eine detaillierte Übersicht über die exportierten Daten angezeigt:

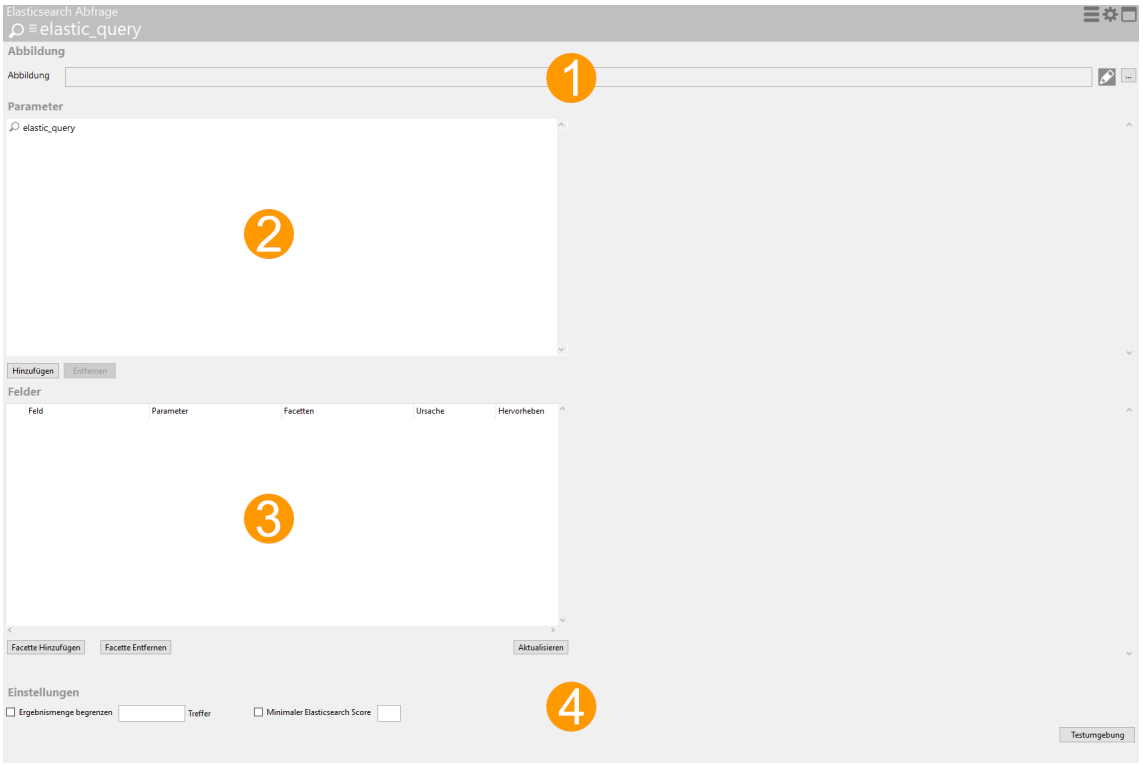


Jede Zeile im externen Index stellt ein exportiertes Objekt aus dem Wissensnetz dar. Da Elasticsearch alle Daten in JSON speichert, können unter **Show** einzelne Datensätze als JSON dargestellt oder mit **Download as JSON** alle Daten in dieser Form heruntergeladen werden. Die Übersicht bietet zusätzlich die Möglichkeit, die Daten mithilfe des Suchfelds **Search** zu durchsuchen, was auf einer vereinfachten Suchsyntax basiert. Für komplexere Suchen kann unter **Custom Search** mithilfe von JSON eine komplexere Query definiert werden.

Für alles rund um Elasticsearch und vor allem hilfreich für benutzerdefinierte Suchen gibt es eine umfassende Dokumentation von Elasticsearch unter https://elastic.co/guide/index.html

1.5.7.8. Abfragen und Facetten

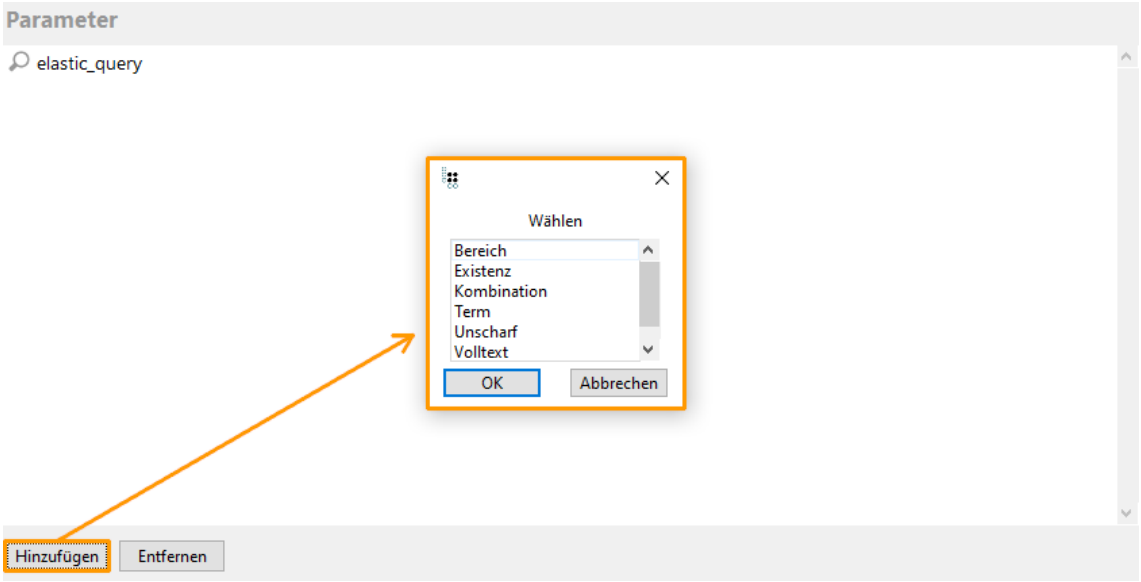
Damit der externe Index sinnvoll genutzt werden kann, sollten Elasticsearch-Abfragen konfiguriert werden. Mithilfe dieser Abfragen können Suchfunktionen und -optionen von Elasticsearch genutzt werden, um die Daten zu durchsuchen und Treffer zurückzugeben. Um eine solche Abfrage zu konfigurieren, muss diese im Arbeitsordner zuerst erstellt werden. Danach öffnet sich die folgende Konfigurationsansicht:



Die Konfigurationsansicht ist in vier Bereiche aufgeteilt:

1. Abbildung Hier wird definiert, für welche Abbildung die Abfrage bereitgestellt werden soll. Auch hier muss die Abbildung erst einmal registriert werden, damit sie ausgewählt werden kann.

2. Parameter In diesem Bereich werden die Parameter definiert, die für die Abfrage genutzt werden sollen. Außerdem wird hier die Suchlogik festgelegt, die für die Abfrage eine entscheidende Rolle spielt. Um eine Logik für die Abfrage zu definieren, müssen Suchkriterien hinzugefügt werden:



Die Auswahl bietet folgende Möglichkeiten:

Allgemeine Parameter

Kombination

Kombiniert einzelne Suchkriterien zu einer zusammengefassten Logik.

Feldparameter

Bereich

Ein Treffer liegt vor, wenn der Wert eines Felds im angegebenen Bereich liegt.

Existenz

Ein Treffer liegt vor, wenn der Wert eines Felds existiert.

Term

Ein Treffer liegt vor, wenn der Wert eines Felds identisch mit dem angegebenen Wert ist.

Unscharf

Ein Treffer liegt vor, wenn ein indexierter Ausdruck ähnlich zum angegebenen Ausdruck ist (Toleranz gegenüber vertauschten, geänderten, fehlenden und überflüssigen Buchstaben).

Volltext

Ein Treffer liegt vor, wenn mithilfe einer Volltextsuche der angegebene Ausdruck in einem anderen gefunden wird.

Geometrie-Parameter

Diese Parameter sind nur in Kombination mit Feldern des Typs *shape* oder *geo_shape* nutzbar.

Abstand

Ein Treffer liegt vor, wenn sich der parametrisierte Referenzpunkt innerhalb einer festgelegten Distanz zur Geometrie des Felds befindet. Die Distanz kann mit einer Einheit

(z.B. `km`, `yards`, `nmi`, …) angegeben werden. Ohne Einheit werden `meters` angenommen.

HINWEIS	Abstandsabfragen sind nur auf Feldern des Typs *geo_shape* möglich.

Bounding Box

Ein Treffer liegt vor, wenn sich die Geometrie des Felds innerhalb der angegebenen Bounding Box befindet.

HINWEIS	Elasticsearch nutzt für den Bounding-Box-Parameter die Notation `BBOX(<minLon>, <maxLon>, <maxLat>, <minLat>)`. Für eine Box mit den Eckpunkten 48°N 8°E und 50°N 10°E ergibt sich also z.B. der Parameterwert `BBOX(8,10,50,48)`. Bounding-Box-Abfragen sind nur auf Feldern des Typs *geo_shape* möglich.

Enthält

Ein Treffer liegt vor, wenn die Geometrie des Parameters vollständig in der Geometrie des Felds enthalten ist.

Getrennt

Ein Treffer liegt vor, wenn die Geometrie des Parameters keine Überschneidung oder Berührpunkte mit der Geometries des Felds hat.

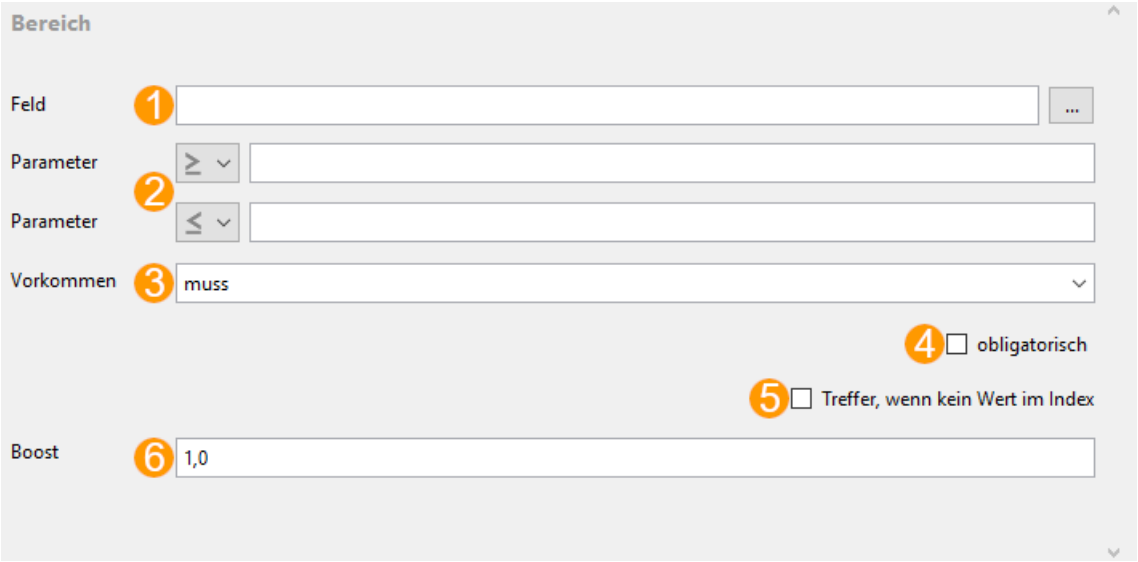
Innerhalb

Ein Treffer liegt vor, wenn die Geometrie des Parameters die Geometrie des Felds vollständig enthält.

Schneidung

Ein Treffer liegt vor, wenn es zwischen der Geometrie des Parameters und der Geometrie des Felds eine beliebige Überschneidung gibt.

Je nachdem, welches Suchkriterium ausgewählt wurde, gibt es verschiedene Einstellungen in der Konfigurationsansicht. Mit einigen Ausnahmen ähneln sich diese, sodass am Beispiel des Bereich-Kriteriums die wichtigsten Einstellungen demonstriert werden können:



1. Hier wird das Feld des externen Index angegeben, worauf das Suchkriterium angewendet wird
2. Generell können hier die Parameter für eine Abfrage angegeben werden. Im Fall des Bereich-Kriteriums gibt es beispielsweise zwei Parameter, welche die Ober- und Untergrenze bilden, und die Möglichkeit, die Vergleichsoperatoren zu ändern.
3. Hier kann die Logik ausgewählt werden, wonach sich die Abfrage richtet. Folgende Auswahlmöglichkeiten werden geboten:

   muss

   Equivalent zu logischem UND. Für einen Treffer müssen alle Suchkriterien erfüllt sein. Erhöht den Elasticsearch-Score für einen Treffer.

   soll

   Equivalent zu logischem ODER. Für einen Treffer muss mindestens ein Suchkriterium erfüllt sein. Erhöht den Elasticsearch-Score für einen Treffer.

   darf nicht

   Gegenteil von muss: Ein Treffer liegt vor, wenn der angegebene Wert das Suchkriterium nicht erfüllt.

   Filter

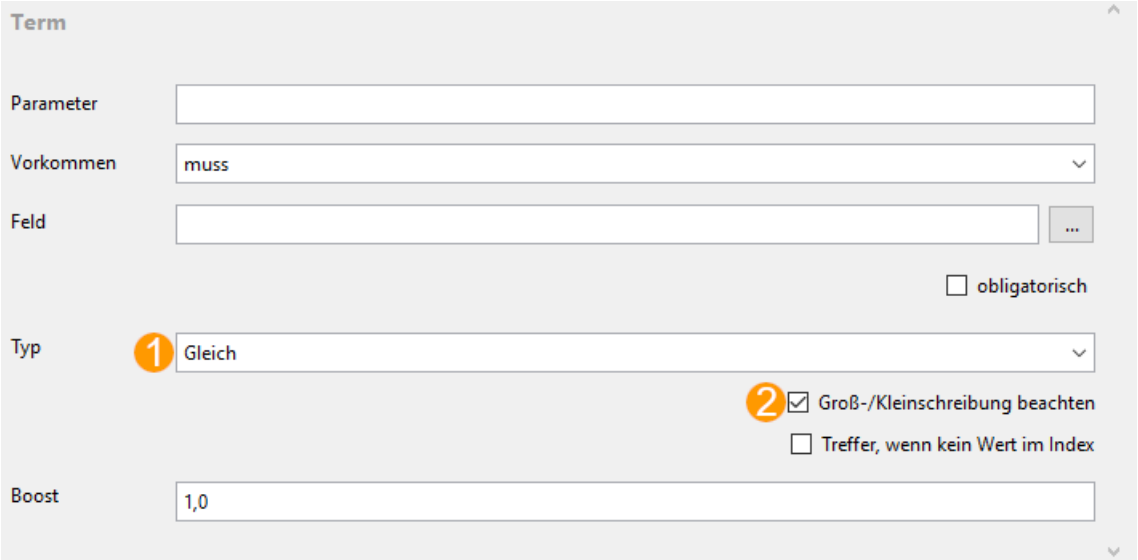
   Äquivalent zu muss, jedoch bleibt der Elasticsearch-Score unberührt.

4. Wenn aktiviert: Es wird geprüft, ob die angegebenen Parameter schon als Parameter im System definiert wurden. Falls die Parameter noch nicht existieren, wird die Abfrage nicht durchgeführt.
5. Gibt eine leere Treffermenge zurück, wenn kein Wert im Index existiert.
6. Kann für eine Gewichtung einzelner Suchkriterien verwendet werden. Der eingegebene Wert (Fließkommazahl) gilt als Faktor, der bei Treffern mit dem Elasticsearch-Score verrechnet wird. Der Standardwert liegt bei 1,0. Je höher der Wert, desto mehr Gewicht hat das Suchkriterium

bei der Berechnung des Elasticsearch-Scores.

Zu den Ausnahmen mit zusätzlichen spezifischen Einstellungsmöglichkeiten zählen:

Term:



1. Hier kann ein Typ für das Suchkriterium *Term* ausgewählt werden. Mögliche Typen sind:

   Präfix

   Es liegt ein Treffer vor, wenn der angegebene Suchterm mit dem Anfang eines indexierten Ausdrucks übereinstimmt (Beispiel: Suchterm "Te" → Treffer bei "Term").

   Platzhalter (Wildcards)

   Es können Wildcards bei der Suche verwendet werden, die beliebige Teile eines Terms ersetzen (Beispiel: Suchterm "Wild*" → Treffer bei "Wildcards" | Der Stern steht als Platzhalter für eine beliebige Anzahl an Buchstaben).

   Regulärer Ausdruck

   Sucht mithilfe von regulären Ausdrücken.

   Gleich

   Ein Treffer besteht nur dann, wenn der angegebene Suchterm identisch mit dem indexierten Ausdruck ist.

2. Wenn deaktiviert: Ein Suchterm ist mit einem indexierten Ausdruck auch dann identisch, wenn die Groß- und Kleinschreibung beliebiger Buchstaben verschieden ist.

Volltext:

1. Ähnlich zum Suchkriterium Term kann auch hier ein Typ ausgewählt werden:

Matching

Es liegt ein Treffer vor, wenn der angegebene Ausdruck mit dem indexierten Volltext identisch ist.

Enthält Phrase

Es liegt ein Treffer vor, wenn der angegebene Ausdruck mit einem Teil des indexierten Volltextes übereinstimmt.

Beginnt mit Phrase

Es liegt ein Treffer vor, wenn der indexierte Volltext mit dem angegebenen Ausdruck beginnt.

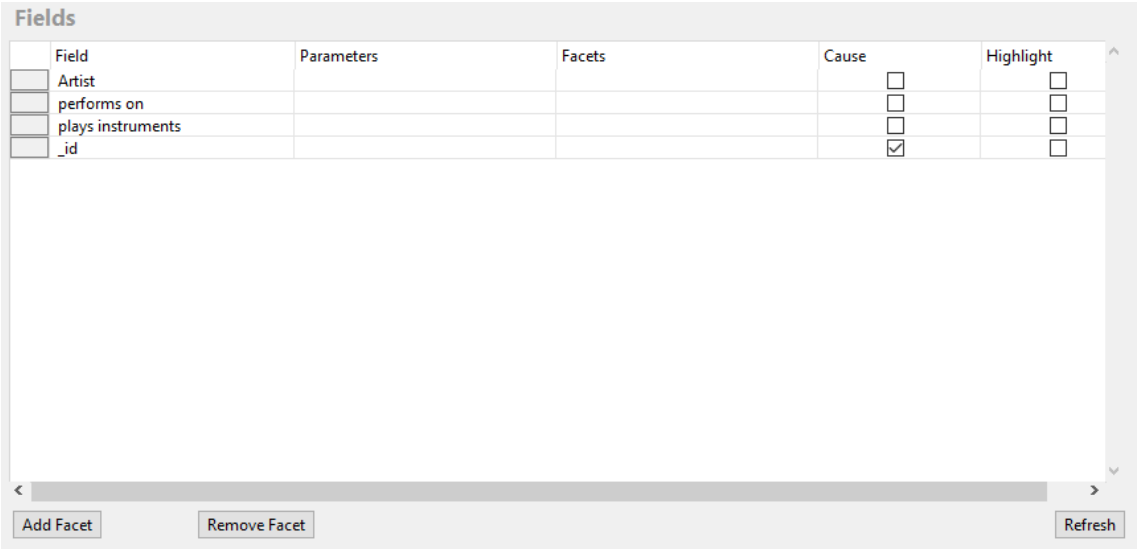
Suchsyntax anwenden

Es wird eine von Elasticsearch vordefinierte Suchsyntax angewendet.

Vereinfachte Suchsyntax

Es wird eine von Elasticsearch vordefinierte vereinfachte Suchsyntax angewendet. Bietet weniger Optionen als die normale Suchsyntax, ist dafür aber fehlertoleranter.

3. Felder Nachdem eine Abbildung ausgewählt wurde, werden hier die Felder des externen Index angezeigt. Hier können jedem Feld separat weitere Funktionen und Optionen zugeordnet werden. Dafür gibt es die folgende Konfigurationsansicht:



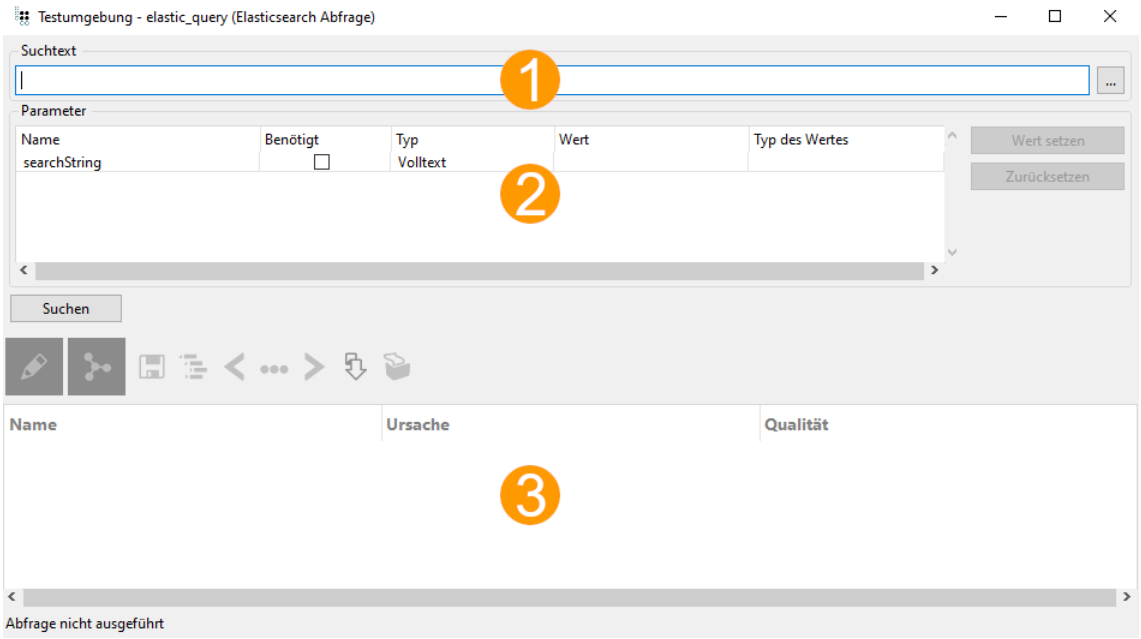
Zu sehen sind die Felder des lokal erstellten Schemas. Wird bei einer Abfrage ein bestimmtes Feld angegeben, welches mit einem benutzerdefinierten Parameter duchsucht werden soll, wird der Name des Parameters in der entsprechenden Zeile unter **Parameter** angezeigt. Für jedes Feld kann die Spalte **Ursache** aktiviert werden. Ist dies der Fall, werden die Werte dieses Felds in der Ergebnistabelle angezeigt. Unter **Hervorheben** kann für jedes Feld aktiviert werden, ob bei Treffern in der Ergebnistabelle Übereinstimmungen auf dieses Feld hervorgehoben werden sollen. Zusätzlich kann jedem Feld eine Facette hinzugefügt werden, was durch Auswählen des Feldes und mit dem Button **Facette Hinzufügen** erfolgt. Mit **Facette Entfernen** können diese auch wieder gelöscht werden. Facetten bieten die Möglichkeit, Treffer mithilfe von Termen oder Werten gruppieren und kategorisieren zu können. Mit **Aktualisieren** wird die Ansicht neu geladen.

HINWEIS	Facetten sind nicht mit jedem Datentyp kompatibel, wie beispielsweise Text oder Search-as-you-type, sondern nur für gruppierbare Werte, wie z.B. Ganz- und Fließkommazahlen, Keywords oder auch Zeit- und Datumswerte.

4. Einstellungen Zusätzlich lassen sich weitere Einstellungen tätigen, die sich auf die Ergebnismenge auswirken:

1. **Ergebnismenge begrenzen** Wenn aktiviert: Die Treffer einer Abfrage auf den externen Index lassen sich begrenzen, indem ein Grenzwert (Ganzzahl) in das Input-Feld eingefügt wird. Die Treffer basieren alle auf einem Score, der von Elasticsearch vergeben wird. Dabei gilt: Je höher der Score, desto relevanter ist der Treffer. Eine Begrenzung umfasst absteigend die relevantesten Treffer.
2. **Minimaler Elasticsearch Score** Wenn aktiviert: Die Treffer einer Abfrage werden anhand des von Elasticsearch vergebenen Scores begrenzt. Der in das Input-Feld eingegebene Wert (Fließkommazahl) dient als Grenzwert. Es werden nur Treffer in Betracht gezogen, deren Score über diesem Wert liegt.

Damit eine Abfrage auf ihr Suchverhalten getestet werden kann, kann rechts neben den Einstellungen die Testumgebung geöffnet werden:

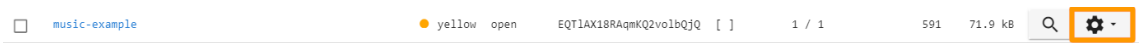


1. Hier kann ein Suchtext eingegeben werden.
2. Werden bei der Konfiguration der Abfragen Parameter definiert, so sind diese hier aufgelistet. Es ist für jeden Parameter einzeln möglich, mit **Wert setzen** einen benutzerdefinierten Wert für die Testsuche zu setzen. Der Wert kann auch wieder mit **Zurücksetzen** gelöscht werden. Wenn alle Werte gesetzt sind oder ein Suchtext eingegeben wurde, kann mit **Suchen** die Testsuche gestartet werden.
3. Nach Ausführen der Testsuche werden hier alle Treffer der Abfrage aufgelistet. Unter **Ursache** wird für jeden Treffer angezeigt, warum das Objekt als Treffer hinsichtlich der Abfrage gilt. In der Spalte **$1** wird jedem Treffer ein Score zugeordnet. Dieser wird als Fließkommazahl angezeigt, wobei der Wert 1,0 den höchsten Score darstellt. Der Score gibt an, wie genau das Suchkriterium im Vergleich zu allen anderen Treffern mit dem Ergebnis übereinstimmt. Je höher der Score, desto mehr Übereinstimmung und Relevanz.

HINWEIS	Es ist davon abzuraten, Werte für die Parameter zu setzen und gleichzeitig den Suchtext zu verwenden, da sonst unerwartete Effekte auftreten können.

1.5.7.9. Settings

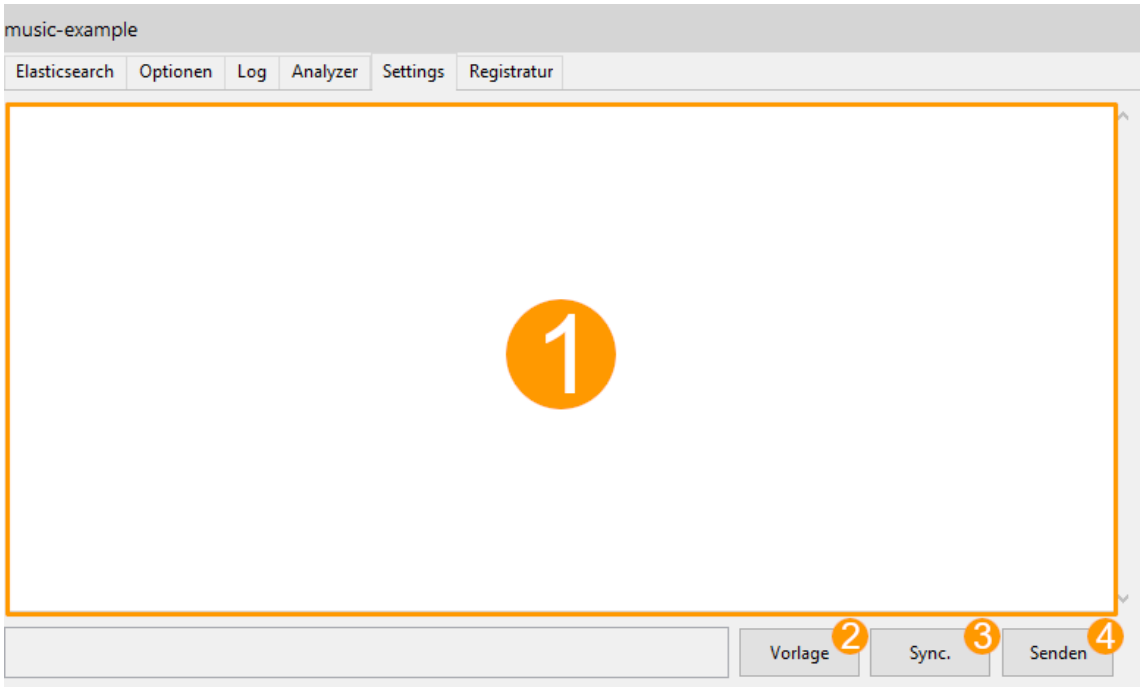
Neben den exportierten Daten können auch Metadaten zum externen Index exportiert werden. Diese können mithilfe von Elasticvue angezeigt werden:



Nach Aufklappen des Zahnrads können die Metadaten unter **Show info** eingesehen werden. Unter den Metadaten befinden sich Informationen wie Name des externen Index, mögliche Aliasse, das

exportierte Schema mit allen definierten Feldern, aber auch Settings. Diese werden zum Teil automatisch beim Erstellen des externen Index erstellt. Dennoch gibt es auch Settings, die sich manuell konfigurieren lassen. Dazu zählen beispielsweise die Analyzer. Mehr dazu gibt es hier.

Im Knowledge-Builder lassen sich die Settings in der Elasticsearch-Abbildung unter dem Reiter **Settings** konfigurieren. Das Eingabefeld kann genutzt werden, um diese in JSON zu definieren:



1. Das Eingabefeld für die Settings. Die Definition der Settings erfolgt in JSON.
2. Hiermit lässt sich eine Vorlage in das Eingabefeld einfügen, die als Basis dient.
3. Aus Elasticsearch werden die Settings in das lokale Eingabefeld geladen.
4. Die definierten Settings werden aus dem Eingabefeld an den externen Index gesendet.

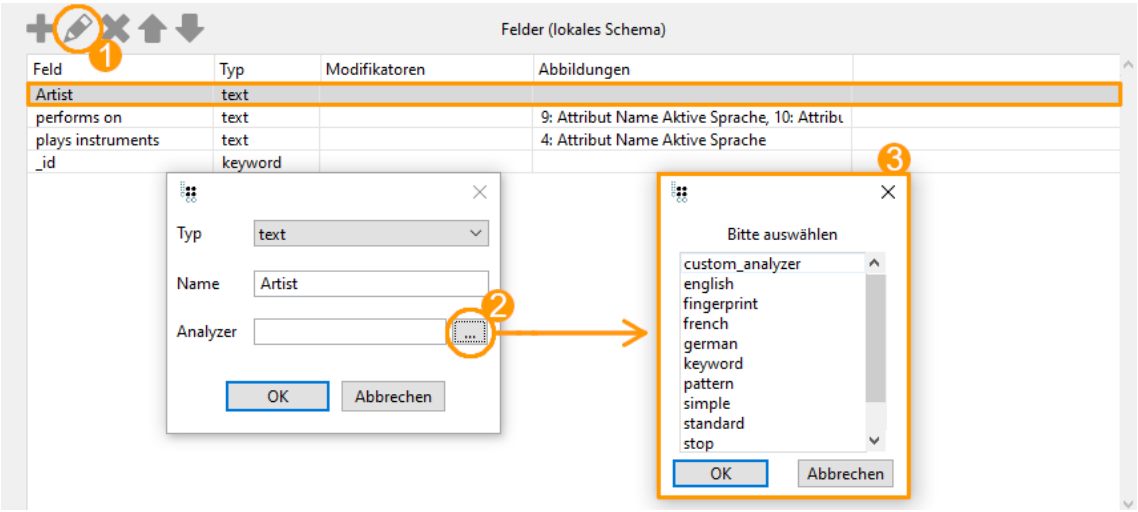
HINWEIS	Sowohl bei erfolgreichem Einrichten der Settings, als auch bei einem Fehler wird unter dem Eingabefeld ein Status angezeigt. Bei einem Fehler wird zudem angezeigt, um welche Art Fehler es sich handelt. Dies kann beispielsweise ein ungültiges Format oder ein Verbindungsfehler zum externen Index sein.

1.5.7.9.1. Analyzer

Eine weitere Funktion von Elasticsearch sind die sogenannten Analyzer. Elasticsearch bietet eine Vielzahl an Textanalysefunktionen an, die in JSON formuliert und als Analyzer an ein Feld des externen Index gebunden werden können. Damit können für einzelne Felder Daten bei einem Export angepasst und die Suchmöglichkeiten erweitert werden. Eine ausführliche Dokumentation von Elasticsearch findet sich hier.

Im Knowledge-Builder gibt es dafür, ähnlich wie für die Settings, eine Konfigurationsansicht unter dem Reiter **Analyzer**. Nachdem ein Analyzer definiert und zum externen Index exportiert wurde,

kann dieser einem Feld hinzugefügt werden:



Neben einigen vordefinierten Analyzern wird der benutzerdefinierte Analyzer ganz oben in der Liste angezeigt. Falls mehr als ein Analyzer definiert wurde, werden diese auch ganz oben angezeigt. Nachdem ein Analyzer ausgewählt wurde, wird dieser nun unter **Modifiers** im lokalen Schema angezeigt.

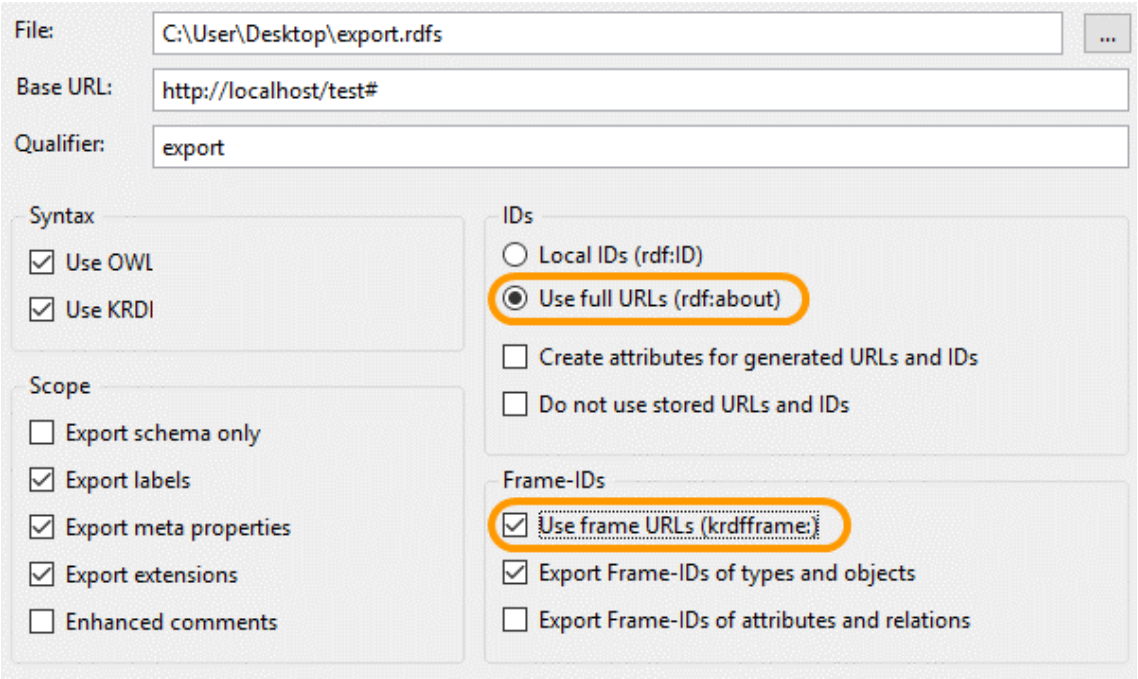
HINWEIS Analyzer können nur Textfeldern zugeordnet werden!

Nachdem ein Analyzer für ein Feld ausgewählt wurde, wird dieser für das zugeordnete Feld aktiv und erweitert somit die Suchmöglichkeiten für dieses.

1.5.8. Gelöschte Individuen aus einem Backup wiederherstellen

Der RDF-Export und -Import eignet sich dazu, gelöschte Individuen aus einem Backup-Netz wieder herzustellen. Dazu wie folgt vorgehen:

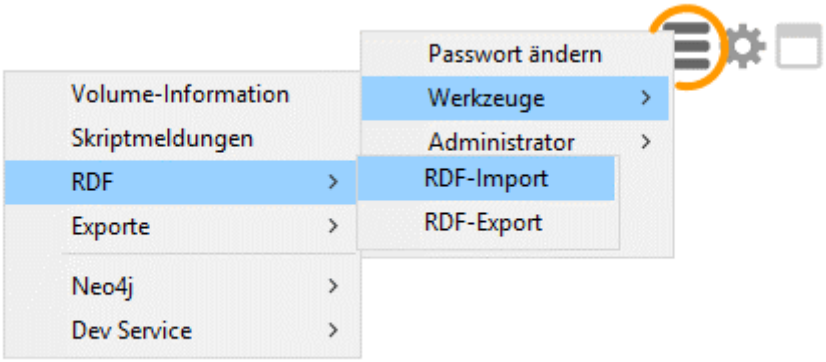
1. Das Backup-Netz im Knowledge-Builder öffnen
2. Einen neuen Ordner anlegen und die wiederherzustellenden Individuen dort ablegen. Hierzu in der Listenansicht der zu übernehmenden Individuen mit Rechtsklick das Kontextmenü öffnen und "Inhalt in neuen Ordner kopieren" wählen, dabei neuen Ordner als Ablageziel wählen
3. Per Kontextmenü auf dem neu erstellten Ordner den RDF-Export öffnen
4. Im Exportdialog einen Dateinamen angeben, die Optionen "URLs verwenden (rdf:about)" und "Frame-URLs verwenden (krdfframe:)" auswählen und den Export ausführen:



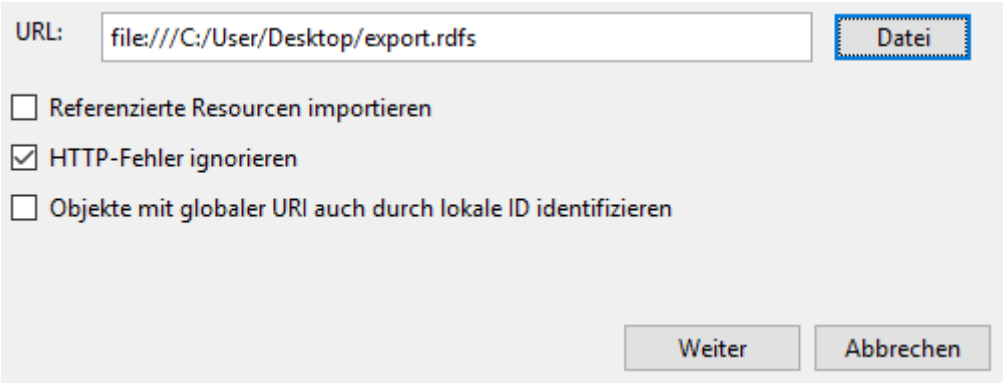
HINWEIS

Die Option "KRDF verwenden" bewirkt, dass i-views spezifische Inhalte zusätzlich mit übernommen werden, welche mittels RDF-Syntax nicht vollständig abgebildet werden können.

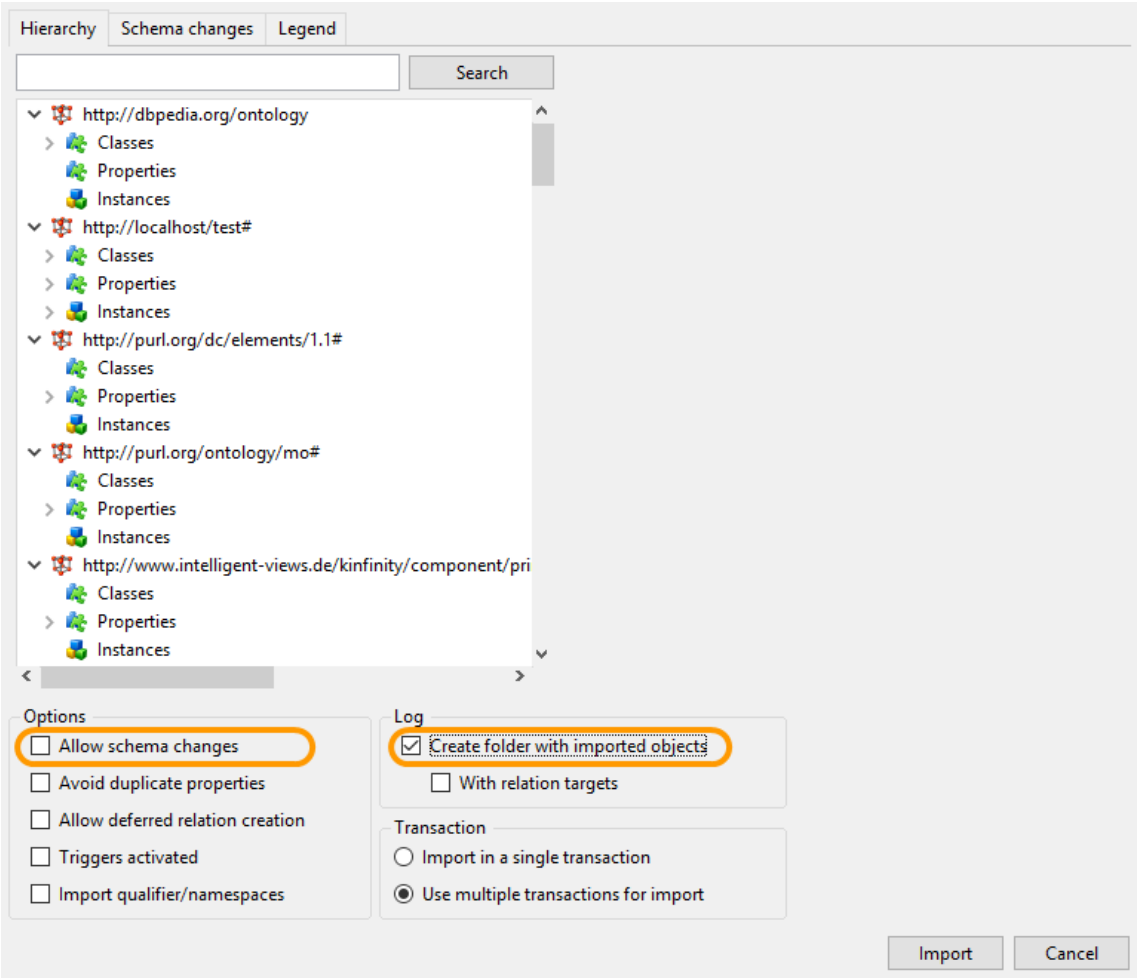
1. Knowledge Builder schließen und das Zielgraph im Knowledge Builder öffnen
2. Im Hauptmenü unter Werkzeuge > RDF > RDF Import den RDF-Import-Dialog öffnen:



1. Datei auswählen und "Weiter" drücken:



1. Im Auswahl-Dialog die Option "Änderungen am Schema erlauben" deaktivieren, "Ordner mit importierten Objekten anlegen" aktivieren:



1. Import ausführen
2. Die wiederhergestellten Individuen überprüfen

1.5.9. Ausgewähltes Schema transportieren

Über das Admin-Tool lässt sich das gesamte Schema mittels RDF-Ex- und Import von einem Knowledge Graph in ein anderes übertragen. Möchte man aber nur ausgewählte Typen übertragen, bietet sich die Funktion "Schema in Ordner kopieren" an, welche für alle Typen per Kontextmenü verfügbar ist. Diese Funktion erstellt eine Referenz auf den gewählten Typ zusammen mit allen anderen (Eigenschafts-)Typen, die erforderlich sind, um im Ziel-Knowledgegraph den gewählten Typ oder Objekte dieses Typs anlegen zu können.

Hat man alle erforderlichen Informationen in einem Ordner gesammelt kann man diesen analog zur Vorgehensweise im vorangegangen Kapitel exportieren und im Zielgraph importieren. Die Option "Änderungen am Schema erlauben" sollte dann jedoch aktiviert werden.