Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Datenbanken und Informationssysteme

Matching von Musikdatenbanken

Bachelorarbeit
im Studiengang Informatik

Philipp Daniel Rohde
Matrikelnummer: 2886190

Prüfer: Prof. Dr. Udo Lipeck
Zweitprüfer: Dr. Hans Hermann Brüggemann
Betreuer: M.Sc. Oliver Pabst

18. August 2016

Zusammenfassung

In der heutigen Zeit entsteht in vielen Bereichen des täglichen Lebens, wie bspw. im Onlinehandel und bei Multimediadiensten, eine große Menge an Daten. Diese Daten werden in Datenbanken gespeichert. Jeder Betreiber einer Datenbank hat sein eigenes Datenbankschema, welches genau auf seine Anforderungen zugeschnitten ist. Bei Datenmengen von mehreren Gigabytes bis hin zu Petabytes werden aufgrund der schwierigen Handhabung in der Regel nur Informationen gespeichert, die für den Anwendungsfall von Interesse sind. Ein Teil des Alltagsgeschäfts ist das Zusammenlegen von Projekten und die Fusionierung von Unternehmen. In diesen Fällen müssen die Daten beider Beteiligten ebenfalls vereinigt werden. Da z.B. Multimediadienste ein überschneidendes Angebot an Musik und Filmen haben, werden ähnliche Daten gespeichert. Damit im Datenbestand keine Duplikate entstehen, muss beim Zusammenlegen der Datenbanken eine Identifikation der Einträge vorgenommen werden, die das gleiche Objekt in der Realwelt beschreiben. Dieser Prozess wird *Matching* genannt.

So wie es mehrere Ansätze für den Matching-Prozess gibt, können in einer Datenbank die Daten in verschiedenen Datentypen, wie bspw. Geodaten, Zeichenketten und Zahlen, gespeichert sein. Das *SimMatching-Verfahren* z.B. wurde für das Matching von Geodaten entwickelt und weicht u.a. durch einen iterativen Ansatz vom klassischen Grundkonzept ab. In dieser Arbeit wird anhand zweier Musikdatenbanken untersucht, ob sich das SimMatching-Verfahren auch für das Matching von String-basierten Daten eignet. Dazu werden im ersten Teil der Arbeit die Grundlagen erläutert und eine Analyse der beiden Datenbanken vorgenommen, um basierend auf den Analyseergebnissen ein Konzept für das Matching nach dem SimMatching-Verfahren zu entwerfen. Im zweiten Teil der Arbeit wird dann die Implementierung des erarbeiteten Konzepts vorgestellt und die Eignung des Verfahrens für den Anwendungsfall von String-basierten Daten bewertet.

Abstract

Nowadays huge amounts of data are generated in many areas of everyday life, e.g. online business and multimedia services. This data is stored in databases. Each database operator uses his own database schema which fulfills his requirements. Quite often the volume of data reaches from several gigabytes up to petabytes. Working with such data becomes difficult. Therefore in most cases only information of actual interest for the application is stored. As a part of everyday buisness projects are combined or companies get merged. In these cases the data of both parties needs to be merged as well. The data overlaps, because e.g. multimedia services offer the same songs or movies. To avoid duplicate entries it is necessary to identify database entries which represent the same object of the real world. This must be done before the data can be merged. This process is called *matching*.

Just as there are several ways how a matching could be organised, the data in a database could be saved in various types, such as geographical data, strings or numbers. The *SimMatching process* for example was developed for geographical data. Among other things it differs from the classical approach by using an iterative approach. This bachelor thesis is about checking and testing the SimMatching process on string-based data with two music databases. The basics, the analysis of the databases as well as the development of a concept for the matching are desribed in the first part of the thesis. The second part is about the implementation of the developed concept and the conclusion of using the SimMatching algorithm on string-based data.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In Datenbanken kommt es häufig zu Duplikaten, also identischen Einträgen. Diese Einträge repräsentieren folglich das gleiche Objekt. Zu Duplikaten kann es durch Fehler in der Datenerfassung, bspw. Schreibfehlern, oder bei der Datenintegration, dem Zusammenführen von zwei oder mehr Datenquellen, kommen. Da Duplikate die Handhabung der Daten erschwert, sollte eine Datenbank frei von Duplikaten sein. Daher ist eine Identifizierung der Datensätze, die das gleiche Realweltobjekt, Objekt in der realen Welt, beschreiben, nötig.

Zur Identifizierung von Duplikaten muss die Ähnlichkeit zweier Objekte bestimmt werden. Mit Ähnlichkeitsmaßen lässt sich die Ähnlichkeit der Objekte als ein Zahlenwert ausdrücken. Viele Ähnlichkeitsmaße beschäftigen sich mit der Bestimmung der Ähnlichkeit zweier Zeichenketten. Es gibt aber auch Ähnlichkeitsmaße für andere Anwendungsdomänen, so lässt sich bspw. die Ähnlichkeit zweier geometrischer Figuren bestimmen.

Falls die Schemata verschiedener Datenquellen unterschiedlich sind, muss vor dem eigentlichen Matching ein gemeinsames Schema gefunden werden. Außerdem müssen die Daten im Vorfeld behandelt werden, denn mit einer Datenvorbehandlung, auch Preprocessing genannt, können Datenfehler behoben, Inkonsistenzen aufgelöst und gegebenenfalls fehlende Daten ergänzt werden. Diese Bereinigung der Daten führt zu einem besseren Matching-Ergebnis.

Es gibt verschiedene Ansätze für das Matching zweier oder mehrerer Datenquellen. Einer dieser Ansätze ist das an der Leibniz Universität Hannover im Rahmen einer Dissertation entwickelte SimMatching-Verfahren [Sch15]. Schäfers verwendet einen iterativen Matching-Prozess, der für das Matching von räumlichen Daten entwickelt wurde. Das Verfahren berücksichtigt nicht nur die Ähnlichkeit basierend auf den Attributen der Objekte, sondern auch die Beziehungen zu anderen Objekten im Datenbestand. Da andere Matching-Verfahren die Beziehungen innerhalb der Daten nicht berücksichtigen, aber auch in anderen Datenbeständen als räumlichen Datenbanken eine Beziehung zwi-

schen den Datensätzen besteht, wird in dieser Arbeit am Beispiel von Musikdatenbanken geprüft, ob sich das SimMatching-Verfahren auch für String-basierte Daten eignet.

1.2 Überblick

Nach diesem einleitenden Kapitel werden in **Kapitel 2** die Grundlagen dieser Arbeit erläutert. Dazu werden die Begriffe *Matching*, *Preprocessing*, *Indexing* und *Ähnlichkeitsmaß* erklärt. Das Matching wird am Beispiel des allgemeinen Matching-Verfahrens beschrieben bevor das SimMatching erklärt und die Unterschiede zum allgemeinen Vorgehen aufgezeigt werden. Für das Preprocessing und das Indexing werden verbreitete Techniken und Algorithmen erklärt, insbesondere werden die später verwendeten Vorgehensweisen erläutert. Außerdem werden die in dieser Arbeit verwendeten Ähnlichkeitsmaße vorgestellt.

In **Kapitel 3** werden die beiden verwendeten Datenbanken analysiert. Dazu wird jeweils das Datenbankschema in zueinander gehörende Gruppen eingeteilt und untersucht. Die Untersuchung dient dazu herauszufinden welche Daten wo und in welchem Format abgespeichert sind. Neben dem Datenbankschema wird auch die Datenqualität betrachtet, da diese für den Matching-Prozess ausschlaggebend ist. Am Ende des Kapitels werden die Unterschiede und Gemeinsamkeiten der Datenquellen aufgezeigt.

Das im Rahmen dieser Arbeit entwickelte Konzept wird in **Kapitel 4** vorgestellt. Basierend auf den Ergebnissen der Datenbankanalyse wird ein gemeinsames Datenbankschema für die beiden Datenquellen entworfen. Darauf aufbauend wird erläutert welche Konvertierungen, Bereinigungen usw. im Preprocessing getätigt werden müssen. Es wird ebenfalls aufgezeigt welche Indexing-Verfahren genutzt werden, um für die verschiedenen Entities die Auswahl an möglichen Kandidaten zu reduzieren. Außerdem wird konkret erläutert wie das SimMatching-Verfahren auf die neue Anwendungsdomäne angepasst werden kann.

Die Implementierung des erarbeiteten Konzepts wird in **Kapitel 5** beschrieben. Hier wird vor allem auf die Besonderheiten der Implementierung des SimMatching-Verfahrens für Musikdatenbanken eingegangen. Dies schließt neben Abweichungen von der ursprünglichen Implementierung in [Sch15] auch Optimierungen und Schwierigkeiten während der Implementierung ein.

Im abschließenden **Kapitel 6** wird ein Fazit gezogen und beurteilt, ob sich das Verfahren SimMatching auch für String-basierte Datenbestände eignet.

Kapitel 2

Grundlagen

In diesem Kapitel werden die einzelnen Schritte des Gesamtprozesses *Datenbankmatching* vorgestellt. Dies beinhaltet neben der Erläuterung des Matching-Prozesses am Beispiel des klassischen Matching-Verfahrens auch die Beschreibung des in dieser Arbeit verwendeten SimMatching-Verfahrens. Ebenso werden die für diese Arbeit wichtigen Schritte Preprocessing und Indexing genauer erläutert, sowie die genutzten Ähnlichkeitsmaße beschrieben.

2.1 Matching

Bei der Integration von Daten einer Datenbank in eine andere ist es wichtig, dass die Objekte einer Datenbank den entsprechenden Objekten der anderen zugeordnet werden können. Christen definiert das Matching als *„task of identifying and matching individual records from disparate databases that refer to the same real-world entities or objects“* [Chr12]. Das Matching ist demnach die Identifizierung und Vereinigung von Einträgen unterschiedlicher Datenbanken, die dasselbe Realweltobjekt repräsentieren. Im Folgenden werden die drei Grundbestandteile Kandidatenerzeugung, Ähnlichkeitsbestimmung und Anwendung von Matching-Regeln erläutert.

Schritt I: Finden von Kandidaten

Zunächst müssen aus den zu matchenden Datenbeständen mögliche Kandidaten herausgesucht werden. Der einfachste Ansatz ist die Bildung des Kreuzproduktes, doch liefert dies sehr viele mögliche Kandidaten, von denen nur wenige potenzielle Matching-Paare bilden, da auf diese Weise jeder Eintrag der einen Datenquelle mit allen Einträgen der anderen ein Matching-Paar bildet. Daher muss eine Vorauswahl getroffen werden, um die Anzahl der Kandidaten zu reduzieren. Dieser Vorgang wird Indexing genannt und in Abschnitt 2.4 genauer beschrieben.

Schritt II: Bestimmen der Ähnlichkeit

Nachdem die möglichen Kandidaten bestimmt wurden, müssen diese miteinander verglichen werden. Eine vollständige Gleichheit wird es nur in wenigen Fällen geben, da die Daten z.B. Tippfehler enthalten, daher muss die Ähnlichkeit zweier Datenbankobjekte mittels Ähnlichkeitsmaßen bestimmt werden.

Für Zeichenketten existieren bereits diverse konkrete Ähnlichkeits- und Distanzmaße. Eine genauere Untersuchung der verschiedenen Ähnlichkeitsmaße für Strings wurde von Prante [Pra12] im Rahmen einer Masterarbeit an der Leibniz Universität Hannover durchgeführt. Die in dieser Arbeit verwendeten Ähnlichkeitsmaße werden in Abschnitt 2.5 erläutert.

Schritt III: Anwendung von Matching-Regeln

Matching-Regeln legen fest unter welchen Bedingungen die Zuordnung der Einträge eines Kandidatenpaares, für das im vorherigen Schritt die Ähnlichkeit berechnet wurde, bestätigt wird. In den meisten Ansätzen wird die Zuordnung lediglich bestätigt oder verworfen. Die untersuchten Paare werden also in die Gruppen *matched* und *unmatched* eingeteilt.

Für diese Zuordnung können die Ähnlichkeitswerte mehrerer Ähnlichkeitsmaße oder Attribute verwendet werden. Als Grundlage kann man dabei bspw. die normierte Summe oder einen gewichteten Ansatz verwenden. Die einfachste Art von Matching-Regeln sind die Schwellwertregeln, die alle Kandidaten, die einen festen vordefinierten Schwellwert überschreiten, bestätigen und alle anderen ablehnen. Da solche Regeln sehr von der Qualität der berechneten Ähnlichkeit abhängig sind, reicht es oft nicht aus nur den Schwellwert zu betrachten.

Eine Verbesserung dieses Ansatzes kann man erreichen, indem man Regeln der Art *„Wenn Attribut a eine Ähnlichkeit von mindestens x hat, so muss Attribut b eine Ähnlichkeit von nur noch mindestens y besitzen“* aufstellt [Sch15].

Die Matching-Regeln müssen jedoch nicht zwangsläufig manuell erstellt werden. Es ist denkbar diese Regeln in Systemen, die maschinelles Lernen verwenden, zu erlernen.

2.2 SimMatching

Als Matching-Verfahren wird in dieser Arbeit das im Rahmen einer Dissertation an der Leibniz Universität Hannover von Schäfers entwickelte SimMatching-Verfahren [Sch15] verwendet werden. Dieses Verfahren setzt auf Anpassbarkeit und Effizienz und wurde ursprünglich für das Matching von räumlichen Datenbanken entwickelt. SimMatching verwendet das Baukastenprinzip, so können bspw. einzelne Teile des Verfahrens weggelassen oder ergänzt werden, wodurch eine flexible Anpassung an verschiedene Eingabe-

daten ermöglicht wird. Das Verfahren folgt außerdem dem Greedy-Paradigma, d.h. es wird stets die Zuordnung mit dem aktuell höchsten Ähnlichkeitswert bevorzugt.

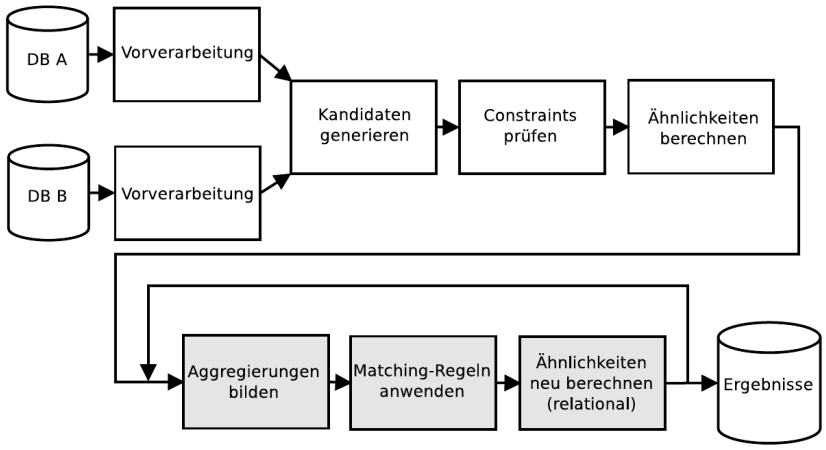

Abbildung 2.1: Ablauf des SimMatching-Verfahrens (entnommen aus [Sch15])

Wie in Abbildung 2.1 zu erkennen ist, handelt es sich um ein iteratives Verfahren. Für Zuordnungen gibt es nicht nur die Zustände *matched* und *unmatched* wie in Abschnitt 2.1 beschrieben, sondern auch *possible*, da in einer Iteration eine Zuordnung möglicherweise weder bestätigt noch verworfen werden kann. Zuordnungen im Zustand *possible* werden in der nächsten Iteration des Verfahrens erneut untersucht. Dieses Verfahren setzt auf zwei verschiedene Typen von Ähnlichkeitsmaßen, *attributbasierte Ähnlichkeitsmaße* und *relationale Ähnlichkeitsmaße*. Erstere sind die auch im klassischen Verfahren genutzten Ähnlichkeiten der Attribute zweier Objekte. Die attributbasierte Ähnlichkeit ergibt sich aus den Ähnlichkeiten der Attribute der Objekte. Bspw. ergibt sich die Ähnlichkeit zweier Musiker durch die Ähnlichkeiten ihrer Vor- und Nachnamen. Unter relationaler Ähnlichkeit wird im Anwendungsfall von Musikdatenbanken eine kontextabhängige Ähnlichkeit verstanden. Bspw. kann für ein mögliches Matching-Paar der Künstler eine höhere Ähnlichkeit angenommen werden, wenn ihnen die gleichen bereits gematchten Musikstücke zugeordnet sind. Nachfolgend werden die einzelnen Schritte des Verfahrens genauer erläutert und so die Unterschiede zum klassischen Verfahren (vgl. Abschnitt 2.1) aufgezeigt.

Schritt I: Vorverarbeitung

Mit Vorverarbeitung ist der in dieser Arbeit als Preprocessing bezeichnete Schritt gemeint, also das Vorbereiten der Daten auf den Matching-Prozess. Wie genau diese Vorverarbeitung aussieht ist abhängig von den verwendeten Daten. In dieser Arbeit zählt dazu neben dem Bereinigen der Daten, bspw. Formatumwandlung von Daten (siehe Abschnitt 2.3), auch das Anpassen der Datenbankschemata (siehe Abschnitt 4.1).

Schritt II: Kandidaten generieren

Dieser Schritt ist allen Matching-Verfahren gemein und wurde bereits bei der Beschreibung des klassischen Verfahrens (siehe Abschnitt 2.1) behandelt.

Schritt III: Constraints prüfen

Constraints sind Einschränkungen, die an die Matching-Kandidaten gestellt werden. Es wird zwischen Must-Match-Constraints und Cannot-Match-Constraints unterschieden. Ist ein Must-Match-Constraint erfüllt, so führt dies direkt zu einem Match. Analog dazu können mit Cannot-Match-Constraints Matching-Paare direkt verworfen werden. Im Rahmen dieser Arbeit wäre es denkbar Constraints z.B. für den Typ einer Veröffentlichung (Album, Single usw.) zu verwenden. Beide Typen von Einschränkungen können weiter in regelbasierte und instanzbasierte Constraints eingeteilt werden. Einer regelbasierten Einschränkung liegt eine bestimmte Regel zugrunde, wie bspw. die maximale Ähnlichkeit in Attribut X muss über einem bestimmten Wert Y liegen. Instanzbasierte Einschränkungen hingegen basieren auf Objekten und bestätigen oder verwerfen somit bestimmte Kandidatenpaare direkt. Es können mit einem instanzbasierten Must-Match-Constraint also direkt bestimmte Datenbankobjekte einander zugeordnet werden. Es ergibt sich die in Abbildung 2.2 aufgezeigte Hierarchie für Constraints.

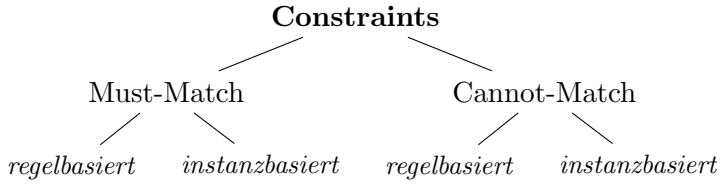

Abbildung 2.2: Hierarchie der Constraints (nach [Sch15])

Schritt IV: Ähnlichkeiten berechnen

Für jedes Matching-Paar wird eine gewichtete Gesamtähnlichkeit aus den verschiedenen Ähnlichkeitsmaßen berechnet:

$$sim(a,b) = (1-\alpha) * sim_{attr}(a,b) + \alpha * sim_{rel}(a,b) \text{ mit } 0 \leq \alpha \leq 1 \tag{2.1}$$

Dabei bezeichnet sim_{attr} die gewichtete Summe der attributbasierten Ähnlichkeitswerte und sim_{rel} die gewichtete Summe der relationalen Ähnlichkeitswerte.

In der Literatur wird zur Speicherung der Gesamtähnlichkeitswerte für die verschiedenen möglichen Zuordnungen eine Ähnlichkeitsmatrix vorgeschlagen, doch bedingt durch das Anwenden eines Indexing-Verfahrens würde hieraus eine dünnbesetzte Matrix der Größe $m \times n$ entstehen, wenn Datenbank A m und Datenbank B n Einträge hat. Diese

Matrix würde eine effiziente Verwaltung erschweren, daher wird in diesem Matching-Verfahren zur Speicherung eine Prioritätswarteschlange verwendet. Die Prioritätswarteschlange wird so gebildet, dass der Kopf das Paar mit der größten Gesamtähnlichkeit ist.

Schritte V - VII: Iterativer Teil

Die nachfolgenden Schritte bilden den iterativen Teil des SimMatching-Verfahrens. Es wird in jedem Iterationsschritt nur die Zuordnung mit dem höchsten Ähnlichkeitswert betrachtet, was Teil der Greedy-Strategie ist.

Schritt V: Aggregierungen bilden

Dieser Schritt beschreibt das Zusammenführen von zusammengehörigen Objekten zu einem Objekt. Dieser Schritt ist nötig, da sich gezeigt hat, dass bei räumlichen Daten oft die standardmäßige 1:1-Zuordnung nicht ausreichend ist. Objekte werden dann zusammengeführt, wenn die Ähnlichkeit dadurch gegenüber der Ähnlichkeit der einzelnen Objekte steigt.

Schritt VI: Matching-Regeln anwenden

Das Anwenden von Matching-Regeln wurde bereits in Abschnitt 2.1 erläutert.

Schritt VII: Relationale Ähnlichkeiten neu berechnen

Wenn eine Zuordnung bestätigt wurde, dann ändern sich auch die relationalen Ähnlichkeiten und daher müssen diese am Ende eines Iterationsschrittes erneut berechnet werden, sodass ein Neueinstieg bei Schritt V mit neuen Werten neue Ergebnisse liefert.

2.3 Preprocessing

Die meisten Datenbanken, die Objekte aus der realen Welt abbilden, enthalten falsche, inkonsistente und fehlende Daten [Chr12]. Im Preprocessing, der Datenvorbehandlung, werden die Daten soweit wie möglich bereinigt, ergänzt und konvertiert, um ein besseres Ergebnis des Matchings zu erhalten.

Nach [Chr12] sind die relevanten, die Datenqualität beeinflussenden, Faktoren für das Matching Genauigkeit, Vollständigkeit, Konsistenz, Aktualität, Verfügbarkeit und Glaubwürdigkeit. Der Faktor Genauigkeit zielt dabei auf die Dateneingabe ab und ob die Daten auf ihre Richtigkeit geprüft wurden. Die Vollständigkeit stellt die Frage nach der Anzahl der fehlenden Attribute, warum diese fehlen und ob für das Matching wichtige Attribute fehlen. Für die Konsistenz der Daten ist ein einheitliches Format wichtig.

Im Laufe der Zeit können sich die Formate für einzelne Attribute wie bspw. das Datum ändern. Dies führt dann zur Inkonsistenz des Attributs. Diese Inkonsistenz muss im Preprocessing behoben werden. Auch die Aktualität der Daten ist für das Matching relevant. Wenn die Daten zweier Datenbanken zu unterschiedlichen Zeitpunkten aufgenommen wurden, so können sich die Daten geändert haben, wie bspw. die Adresse einer Person oder im Bezug auf den Kontext dieser Arbeit die Mitglieder einer Band. Mit Verfügbarkeit ist gemeint, ob alle für das Matching benötigten Daten in der Datenbank verfügbar sind. Je mehr Attribute zu einem Realweltobjekt gespeichert sind, desto präziser wird die Zuordnung von zwei Datenbankobjekten, die dasselbe Objekt in der realen Welt beschreiben. Der Punkt Glaubwürdigkeit zielt auf die Frage ab ob sich falsche Daten in der Datenbank befinden oder ob die Datensätze als zuverlässig angesehen werden können.

Der erste Schritt in der Datenvorbehandlung von Zeichenketten ist das Entfernen von unerwünschten Tokens, bspw. Schlüsselwörter und Whitespace-Zeichen. Mit *unerwünscht* ist hier *nicht nützlich für das Matching* gemeint. Im zweiten Schritt werden die Token standardisiert. Die Standardisierung sieht dabei u.a. die Ersetzung von Abkürzungen und Spitznamen so wie das Beheben bekannter Schreibfehler vor, aber auch den Ausgleich von unterschiedlichen Schreibweisen wie bspw. Namensvariationen. Anschließend werden die Daten in eine Datenbank zurückgeschrieben, die sich für das Matching eignet. Im Falle des Matchings zweier oder mehrerer Datenbanken eignet sich ein im Vorfeld erstelltes Datenbankschema, das möglichst viele Aspekte der ursprünglichen Schemata beinhaltet. Als möglichen vierten Schritt führt Christen die Überprüfung der Korrektheit der im vorherigen Schritt generierten Datensätze an.

2.4 Indexing

Wenn Datenbank A m und Datenbank B n Einträge hat, dann müssten beim Matchen dieser Datenbanken $m \times n$ Eintragspaare verglichen werden. Es ist offensichtlich, dass dies für große Datenbanken nicht praktikabel ist. Daher muss die Zahl der zu vergleichenden Paare, den *Matching-Kandidaten* oder kurz Kandidaten, reduziert werden. Das Ziel des Indexings ist es, die Anzahl der Kandidaten so weit wie möglich zu reduzieren. Dazu werden Paare entfernt, die nur sehr unwahrscheinlich gematcht werden würden. Gleichzeitig müssen wahrscheinliche Treffer behalten werden.

Im Allgemeinen teilen die verschiedenen Indexing-Verfahren die Datensätze in verschiedene Blöcke auf. Je nach Verfahren kann ein Datensatz in einem oder mehreren dieser Blöcke enthalten sein. Abhängig vom gewählten Verfahren wird das Kriterium, nach dem die Blöcke erstellt werden, *blocking key* oder *sorting key* genannt.

Der wichtigste Schritt im Indexing ist die Auswahl der Kriterien. Die Schlüssel müssen so gewählt werden, dass mögliche Kandidaten der selben Gruppe zugeordnet werden, also den gleichen Wert bezüglich des Kriteriums haben. Diese Werte werden blocking key values, kurz BKV, genannt. Abhängig von den Daten der Datenbanken kann der Vergleich lexikographisch, numerisch oder bei Strings auch phonetisch sein. Für einen

guten blocking key sollte die Qualität der Werte der verwendeten Attribute hoch sein, um eine Fehleinordnung zu vermeiden. Ebenso wichtig ist die Häufigkeit von Werten des Attributs, da diese die Anzahl der möglichen Kandidaten beeinflusst. Es ist üblich für Namen phonetische Codes als (Teil des) BKV zu verwenden.

Das älteste Indexing-Verfahren ist *standard blocking*. Jeder Datensatz wird bei diesem Verfahren genau einem Block zugeordnet. Es werden später also nur noch Einträge miteinander verglichen, die den gleichen BKV haben.

Ein weiteres Verfahren ist *sorted neighbourhood*, welches die Daten nach einem Schlüssel sortiert. Es wird dann ein Fenster fester Größe über die sortierten Datenbanken gelegt und nach und nach weitergeschoben. Als mögliche Kandidaten zu einem Datensatz werden dann die sich im gleichen Fenster befindenden Datensätze betrachtet.

Bei Datenbeständen mit vielen fehlerhaften Einträgen, wie bspw. Namensvariationen, kann es sein, dass standard blocking und sorted neighbourhood Paare, die eigentlich in Betracht gezogen werden müssten, nicht der selben Gruppe zuordnen können. Beim *q-gram based indexing* werden für die Schlüsselwerte Listen mit q-grams erstellt. Ein q-gram ist ein Teilstring der Länge q. Für $q = 2$ wird der Begriff Bigram verwendet. Aus der Zeichenkette 'rohde' entsteht so die Bigram-Liste ['ro', 'oh', 'hd', 'de']. Um Variationen des Schlüsselwerts zu erzeugen werden in einem rekursiven Vorgehen Teillisten der q-gram-Liste erzeugt. Wenn die ursprüngliche Liste k q-grams hatte, dann entstehen in der ersten Stufe k Teillisten mit $k-1$ q-grams. In jeder Teilliste wird ein q-gram der Ausgangsliste entfernt. Dieses Vorgehen wird rekursiv fortgeführt bis eine vorher festgelegte Länge der Teillisten erreicht wurde. Die erzeugten Teillisten werden wieder in Strings umgewandelt und als Indexwert genutzt. Ein Eintrag kann also mehrere Indexwerte haben. Wenn zwei Datensätze einen gleichen Indexwert haben, werden sie als potenzielle Matching-Kandidaten behandelt. Diese Vorgehensweise ist für große Datenbanken allerdings nicht geeignet, da auch bei kurzen BKVs wie *miller* und einer Mindestlänge von drei Bigrams bereits 26 Listen erzeugt werden (vgl. Tabelle 2.1).

Ursprungsliste	Stufe 1	Stufe 2
['mi', 'il', 'll', 'le', 'er']	['il', 'll', 'le', 'er'],	['ll', 'le', 'er'], ['il', 'le', 'er'], ['il', 'll', 'er'], ['il', 'll', 'le'],
	['mi', 'll', 'le', 'er'],	['ll', 'le', 'er'], ['mi', 'le', 'er'], ['mi', 'll', 'er'], ['mi', 'll', 'le'],
	['mi', 'il', 'le', 'er'],	['il', 'le', 'er'], ['mi', 'le', 'er'], ['mi', 'il', 'er'], ['mi', 'il', 'le'],
	['mi', 'il', 'll', 'er'],	['il', 'll', 'er'], ['mi', 'll', 'er'], ['mi', 'il', 'er'], ['mi', 'il', 'll'],
	['mi', 'il', 'll', 'le']	['il', 'll', 'le'], ['mi', 'll', 'le'], ['mi', 'il', 'le'], ['mi', 'il', 'll']
		$1 + 5 + 20 = 26$ Listen

Tabelle 2.1: Bigram-Listen am Beispiel des Nachnamens 'miller'[1]

[1]entnommen aus dem Beispiel zu *q-gram based indexing* in [Chr12]

2.5 Ähnlichkeitsmaße

Um die Ähnlichkeit zweier Objekte zu messen werden sogenannte Ähnlichkeitsmaße verwendet.

Definition: Ähnlichkeitsmaß (nach [CMZ09])
Sei M eine Menge, dann heißt die Funktion $s: M \times M \to \mathbb{R}$ ein *Ähnlichkeitsmaß* falls für alle $a, b, c \in M$ gilt:

i) $s(a,b) = s(b,a)$ (Symmetrie)
ii) $s(a,a) \geq 0$ (Positivität der Selbstähnlichkeit)
iii) $s(a,a) \geq s(a,b)$ (Selbstähnlichkeit größer als beliebige Ähnlichkeiten)
iv) $s(a,b) + s(b,c) \leq s(a,c) + s(b,b)$ (Dreiecksungleichung)
v) $s(a,a) = s(b,b) = s(a,b) \Leftrightarrow a = b$ (Identität)
vi) $0 \leq s(a,b) \leq 1$ (Normierung; optional)

Der letzte Punkt ist besonders entscheidend bei der Verwendung mehrerer Ähnlichkeitsmaße.

Im Folgenden werden die in dieser Arbeit verwendeten Ähnlichkeitsmaße vorgestellt. Die *Jaro-Distance* und die *numerische Ähnlichkeit* dienen zur Berechnung der attributbasierten Ähnlichkeit. Die relationale Ähnlichkeit wird mit der *DirectNeighbourhoodSimilarity* bestimmt.

2.5.1 Jaro-Distance

Ursprünglich diente die Jaro-Distance [EIV07] zum Vergleich von Namen. Seien a und b zwei Strings sowie $|a|$ und $|b|$ die Länge des jeweiligen Strings, so bestimmt man die Anzahl c gemeinsamer Zeichen mit:

$$c = \left| \left\{ a[i] : a[i] = b[j] \text{ und } |i-j| \leq \left\lfloor \frac{max(|a|,|b|)}{2} \right\rfloor - 1 \right\} \right| \tag{2.2}$$

Zwei Zeichen gelten demnach als gemeinsames Zeichen, wenn sie höchstens um die halbe Länge des längeren Strings voneinander entfernt sind.

Zusätzlich muss die Anzahl an Transpositionen t der gemeinsamen Zeichen berechnet werden. Diese Zahl beschreibt die Anzahl an Positionen, auf denen sich die Zeichen der beiden Strings unterscheiden. Seien m_a und m_b die Strings bestehend aus den gemeinsamen Zeichen in der Reihenfolge des jeweiligen Strings, dann gilt:

$$t = \sum_{i=0}^{c} m_a[i] \neq m_b[i] \tag{2.3}$$

Es ergibt sich dann das Ähnlichkeitsmaß mit:

$$sim_{jaro}(a,b) = \frac{1}{3}\left(\frac{c}{|a|} + \frac{c}{|b|} + \frac{c - \frac{t}{2}}{c}\right) \tag{2.4}$$

Das nachfolgende Beispiel zeigt die Ähnlichkeitsbestimmung der Strings „Rohde“ und „Rhode“ mittels des Jaro-Distance-Verfahrens.

$$
\begin{aligned}
a &= Rohde \\
b &= Rhode \\
c &= 5 \\
t &= 2 \\
sim_{jaro}(a, b) &= 0.9\overline{33}
\end{aligned}
$$

Da Buchstabendreher einer der häufigsten Fehler in manuell eingegeben Daten ist, soll für zwei Strings mit einem Buchstabendreher eine hohe Ähnlichkeit ermittelt werden. Nach Prante [Pra12] ist der Vorteil der Jaro-Distance, dass neben der Reihenfolge der Zeichen auch der Zeichenvorrat bei der Berechnung berücksichtigt wird. Außerdem ist das Verfahren unabhängig von der verwendeten Sprache, da nur die einzelnen Zeichen betrachtet werden. In Prantes Untersuchung lieferte die Jaro-Distance, abgesehen von der *field similarity*, den höchsten Ähnlichkeitswert für zwei Strings mit einem Buchstabendreher. Der Nachteil des Verfahrens Field Similarity ist, dass nur der Zeichenvorrat betrachtet wird, nicht aber die Reihenfolge. So liefert das Verfahren auch dann eine Übereinstimmung, wenn zwei Wörter denselben Zeichenvorrat haben, wie z.B. „rosenbusch“ und „buschrosen“ (vgl. [Pra12]). Aus diesem Grund und den oben genannten Vorteilen wurde in dieser Arbeit die Jaro-Distance und nicht die Field Similarity verwendet.

2.5.2 DirectNeighbourhoodSimilarity

Nicht immer lassen sich zwei Objekte über die Ähnlichkeit der Attribute eindeutig zuordnen. Schäfers [Sch15] nutzt daher zur Erhöhung der Wahrscheinlichkeit, dass zwei Objekte ein gültiges Matching bilden, auch die Objekte in der unmittelbaren Nachbarschaft. Wenn die Zuordnung der benachbarten Objekte bereits bestätigt ist, so ist es wahrscheinlich, dass auch die beiden betrachteten Objekte eine gültige Zuordnung beschreiben. Schäfers definiert das Ähnlichkeitsmaß wie folgt:

$$
sim_{Neighbourhood}(a, b) = \frac{|\text{bestätigte benachbarte Zuordnungen}|}{|\text{noch mögliche + bestätigte benachbarte Zuordnungen}|} \tag{2.5}
$$

Es wird also das Verhältnis zwischen den bereits bestätigten Zuordnungen in der Nachbarschaft und allen Zuordnungen in der Nachbarschaft gebildet. Alle Zuordnungen meint die Summe der möglichen noch nicht bestätigten und der bereits bestätigten Zuordnungen.

Für diese Arbeit muss der Begriff der Nachbarschaft definiert werden, da hier keine intuitive Nachbarschaft der Objekte existiert. Als Nachbarschaft wird die Beziehung zwischen den Datenbankobjekten verwendet, bspw. die Beziehung zwischen Künstlern und

Musikstücken. Die Nachbarschaft ergibt sich demnach aus den Relationships. Außerdem wird ausgenutzt, dass es im Rahmen dieser Arbeit nur 1:1-Zuordnungen gibt.

In Abbildung 2.3 ist eine Beispielsituation aus dem Anwendungsbereich dieser Arbeit dargestellt. Es gibt zwei Künstler mit jeweils drei Musikstücken, wobei für zwei der Musikstücke die Zuordnung bereits bestätigt wurde. Musikstück 1 wurde Musikstück B zugeordnet und Musikstück 2 wurde Musikstück C zugeordnet. Die relationale Ähnlichkeit berechnet sich dann wie folgt:

$$sim_{Neighbourhood}(\text{artist 1}, \text{artist A}) = \frac{2}{1+2} = \frac{2}{3} = 0,\overline{66} \tag{2.6}$$

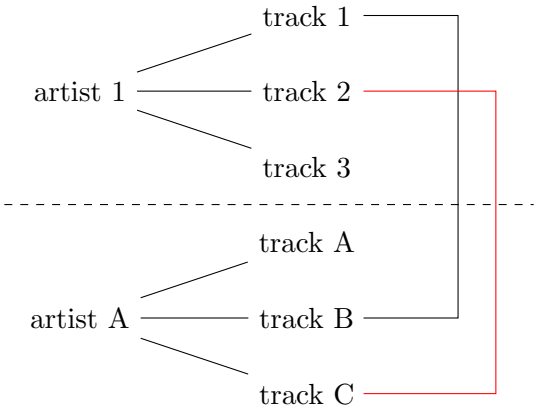

Abbildung 2.3: Beispiel: DirectNeighbourhoodSimilarity

Betrachtet man in Abbildung 2.3 nur die schwarzen Linien, so ergibt sich eine neue Situation. Die Zuordnung der Musikstücke 2 und C wurde noch nicht bestätigt und somit ist in diesem Fall nur die Zuordnung von 1 und B bestätigt. Die Ähnlichkeit beträgt dann:

$$sim_{Neighbourhood}(\text{artist 1}, \text{artist A}) = \frac{1}{4+1} = \frac{1}{5} = 0,2 \tag{2.7}$$

Die 4 ergibt sich aus den noch möglichen Zuordnungen (2, A), (2, C), (3, A) und (3, C).

2.5.3 Numerische Ähnlichkeit

Die Ähnlichkeit von numerischen Attributen wird wie folgt berechnet:

$$sim_{num}(a, b) = 1 - \frac{|a-b|}{max_{attr} - min_{attr}} \tag{2.8}$$

Die Differenz der zu betrachtenden Werte wird also ins Verhältnis zur Größe des Wertebereichs gesetzt.

Dieses Vorgehen ermöglicht es den Abstand zweier Werte relativ zur Größe des Wertebereichs zu bewerten. So wirkt sich eine kleine Abweichung in einem kleinen Wertebereich stärker aus als in einem größeren.

Ein Beispiel für einen kleinen Wertebereich, wie bspw. das Veröffentlichungsjahr:

$$\begin{aligned} min_{year} &= 1889 \\ max_{year} &= 2016 \\ sim_{num}(2000, 2010) &= 1 - \frac{|2000 - 2010|}{2016 - 1889} \\ &= 1 - \frac{10}{127} \\ &\approx 0,92 \end{aligned}$$

Ein Beispiel für einen großen Wertebereich, wie bspw. die Spieldauer eines Musikstücks in Millisekunden:

$$\begin{aligned} min_{duration} &= 10000 \ = 10 \ s \\ max_{duration} &= 2100000 \ = 35 \ min \\ sim_{num}(56000, 66000) &= 1 - \frac{|56000 - 66000|}{2100000 - 10000} \\ &= 1 - \frac{10000}{2090000} \\ &\approx 0,995 \end{aligned}$$

Kapitel 3

Datenbankanalyse

In dieser Arbeit werden beispielhaft die öffentlich zur Verfügung stehenden Dumps der Datenbanken von *Discogs*[2] und *MusicBrainz*[3] verwendet. Dieses Kapitel beschäftigt sich mit der Analyse des Aufbaus und Inhalts der Musikdatenbanken. Dazu werden die beiden Datenbanken zunächst separat in den Abschnitten 3.1 (Discogs) und 3.2 (MusicBrainz) untersucht. Ziel der Analyse ist es herauszufinden, welche Daten in beiden Datenbanken gespeichert sind, um ein geeignetes Schema für das Preprocessing zu finden. In Abschnitt 3.3 werden die vorher gewonnenen Erkenntnisse genutzt, um Unterschiede und Gemeinsamkeiten aufzuzeigen, ehe dann in Abschnitt 4.1 das gemeinsame Schema entwickelt wird.

3.1 Discogs

Discogs ist eine Online-Datenbank für Diskografien von Musikern und Plattenlabeln. Das Ziel von Discogs ist es eine möglichst umfassende Musikdatenbank aufzubauen. Die Benutzer der Seite können selbst zum Datenbestand beitragen. Nach eigenen Angaben haben zum Zeitpunkt dieser Arbeit mehr als 283000 Benutzer Daten eingetragen und es sind mehr als 7,1 Millionen Aufnahmen und 4,4 Millionen Künstler in der Datenbank gespeichert.

[2] http://data.discogs.com/
[3] https://musicbrainz.org/doc/MusicBrainz_Database/Download

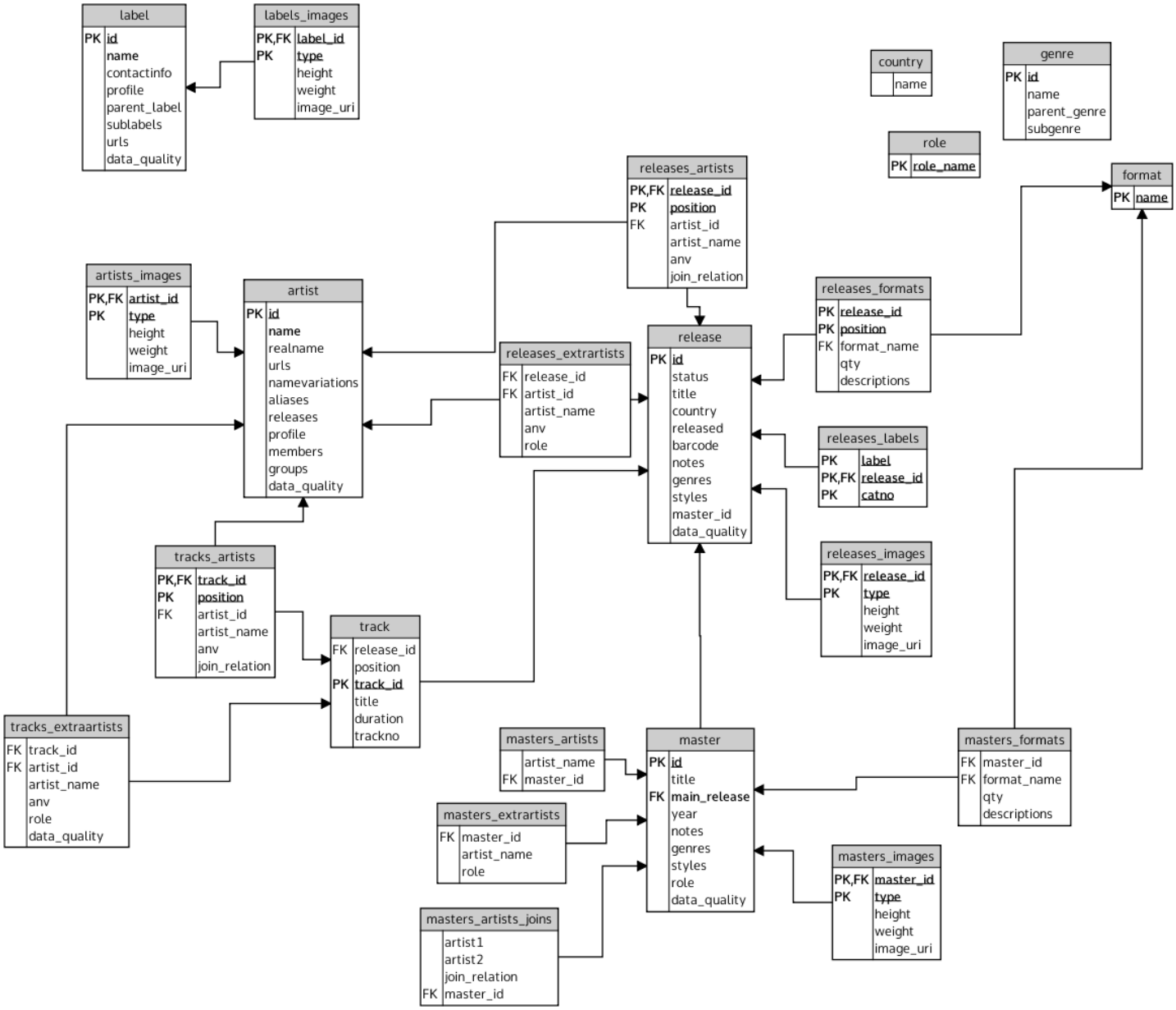

Abbildung 3.1: Datenbankschema Discogs[4]

Wie Abbildung 3.1 zeigt, ist das Schema der Datenbank sehr einfach gehalten. Das Schema stützt sich auf die fünf Hauptentities `artist`, `track`, `label`, `release` und `master`. Diese Entities bilden die Künstler, Musikstücke, Plattenfirmen, Veröffentlichungen und Mastereinträge ab und bilden den Kern des Schemas. Über Fremdschlüsselbeziehungen werden die Entities miteinander verknüpft. Weitere Informationen sind in kleineren Relationen abgelegt, die jeweils nur zu einem der genannten Hauptentities gehören, wie bspw. die Tabellen `*_images`, die verwendet werden, um den Entities Bilder zuzuord-

[4]basierend auf create_indexes.sql und create_tables.sql auf https://github.com/philipmat/discogs-xml2db; ergänzt um vermutete Fremdschlüsselbeziehungen

nen. Über eine Fremdschlüsselbeziehung wird eine Verknüpfung mit der Tabelle des Hauptentities vorgenommen. Im Attribut `type` wird die Art des Bildes gespeichert. Das Hauptbild erhält den Eintrag *primary* und alle weiteren Bilder den Eintrag *secondary*. Die Größe des Bildes in Pixeln ist in den Attributen `height` und `width` hinterlegt. Das Attribut `image_uri` sollte den Pfad des entsprechenden Bildes enthalten, eine Analyse des verwendeten Dumps zeigt jedoch, dass dieses Attribut in allen Tabellen für jeden Eintrag leer ist.

Es bietet sich an für die genauere Untersuchung des Datenbankschemas die Relationen in Gruppen um die Hauptentities einzuordnen. So entsteht jeweils eine Gruppe mit den Informationen über Hauptentities und zusätzlich eine kleine Gruppe mit Relationen, die sich nicht eindeutig einer solchen Gruppe zuordnen lassen.

3.1.1 Artist

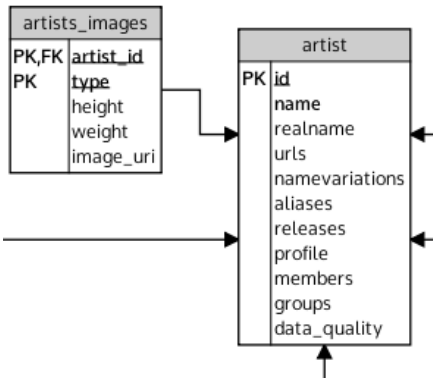

Abbildung 3.2: Ausschnitt „Artist“ des Datenbankschemas von Discogs

In der Tabelle `artist` werden alle Künstler der erfassten Musikstücke gespeichert. Als Künstler gelten hierbei sowohl die Bands als auch ihre Mitglieder. Die Tabelle enthält eine eindeutige ID im Feld `id`. Diese ID ist der Primärschlüssel der Relation. Außerdem wird der Name des Künstlers gespeichert. Alle weiteren Attribute sind optional. Liegt zu einem Künstler auch sein bürgerlicher Name vor, so ist dieser im Attribut `realname` gespeichert. `urls` kann mehrere Internetauftritte eines Künstlers enthalten. Variationen des Namens, also andere Schreibweisen und Abkürzungen, können im Attribut `namevariations` hinterlegt werden. Weitere Namen eines Künstlers sind unter `aliases` gespeichert. Eine Beschreibung des Interpreten kann im Feld `profile` gespeichert werden. Diese Beschreibung ist im weiteren Verlauf der Arbeit jedoch nicht relevant und wird daher nicht berücksichtigt. Alle ehemaligen und aktuellen Mitglieder einer Band können im Attribut `members` hinterlegt werden. Analog dazu können zu einer Einzelperson alle Bands, in denen diese Person Mitglied war, im Attribut `groups` gespeichert werden. Eine Betrachtung der Daten hat ergeben, dass die Attribute `releases` und `data_quality` für jeden Eintrag in `artist` leer sind, daher wird ihnen in dieser Arbeit keine Bedeutung beigemessen.

3.1.2 Track

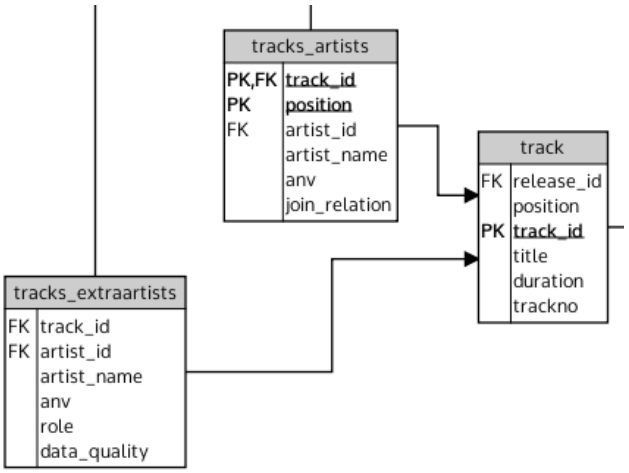

Abbildung 3.3: Ausschnitt „Track“ des Datenbankschemas von Discogs

Ein Track ist hier ein Musikstück einer Veröffentlichung, d.h. wenn ein Album in zwei Versionen erschienen ist, dann gibt es zu jedem Lied, das auf beiden Versionen zu finden ist, zwei (leicht) unterschiedliche Einträge in der Datenbank. Der Primärschlüssel der Relation `track` ist das Attribut `track_id` welches ein eindeutiger Hash ist. Weiterhin wird zu einem Lied die ID der Veröffentlichung unter `release_id` gespeichert. Der Inhalt des Attributs `position` ist abhängig von der Art des Releases. Bei einem Album auf CD entspricht die Position der Tracknummer. Bei Schallplattenveröffentlichungen ist die Position bspw. `B.2`, was angibt das es sich bei dem Track um den zweiten Track auf Seite B handelt. Der Name des Liedes wird im Attribut `title` festgehalten. Für einige Lieder ist die Spieldauer bekannt und im Format `mm:ss` im Feld `duration` hinterlegt. Die Tracknummer wird in `trackno` gespeichert.

Die Künstler eines Tracks werden in der Tabelle `tracks_artists` gespeichert. Der Primärschlüssel besteht aus den Attributen `track_id` und `position`. Das Attribut `track_id` ist die ID des Tracks, es gibt allerdings nur eine vermutete Fremdschlüsselbeziehung zu `id` von `track`. Das Attribut `position` gibt an, an welcher Stelle der Künstler genannt wird, wenn mehr als ein Künstler an diesem Track mitgewirkt haben. Die `artist_id` ist die ID des Künstlers, der an diesem Stück mitgearbeitet hat und an der gegebenen Position genannt wird. Ebenso wie bei der ID des Tracks gibt es auch hier nur eine vermutete Fremdschlüsselbeziehung. Zusätzlich zur ID des Interpreten wird im Feld `artist_name` auch der Name in dieser Relation gespeichert. Bei Discogs wird ein System mit dem Namen *Artist Name Variation* verwendet, um Variationen des Künstlernamens mit einem Profil des Künstlers zu verlinken. Im Attribut `anv` kann eine solche Variation eingetragen werden. Das Attribut `join_relation` gibt an, wie der Künstlername mit den folgenden Künstlern verbunden wird, bpsw. *feat.* oder Komma.

Künstler, die nicht direkt an der Interpretation eines Stücks mitgewirkt haben, werden in der Relation `tracks_extraartists` aufgeführt. Die Attribute `track_id`, `ar-`

`tist_id`, `artist_name` und `anv` haben hier die gleiche Bedeutung wie in der Tabelle `tracks_artists`. Das Attribut `role` gibt die Art der Mitarbeit an, wie bspw. *written by*. Wie in allen anderen Relationen des Dumps ist auch hier das Attribut `data_quality` immer leer.

3.1.3 Label

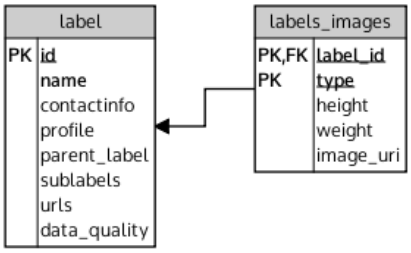

Abbildung 3.4: Ausschnitt „Label“ des Datenbankschemas von Discogs

In der Relation `label` werden Informationen zu den Plattenfirmen der erfassten Musikstücke gespeichert. Jedes Label erhält dabei eine eindeutige ID im Attribut `id` welche auch der Primärschlüssel ist. Der Name wird im Feld `name` gespeichert. Die weiteren Attribute sind optional. Die Adresse, Telefon-, Faxnummer und E-Mail-Adresse können unter `contactinfo` hinterlegt werden. Wie auch bei den Künstlern enthält das Attribut `profile` eine Beschreibung. Da diese Profilbeschreibung im Verlauf dieser Arbeit keine Relevanz besitzt, wird dieses Attribut nicht weiter betrachtet. Über die Felder `parent_label` und `sublabels` können die Beziehungen zwischen den Plattenfirmen eingetragen werden. Das Attribut `urls` dient zum Ablegen von Internetpräsenzen des Labels. Für jeden Eintrag in `label` ist das Attribut `data_quality` im zur Verfügung stehendem Dump leer.

3.1.4 Release

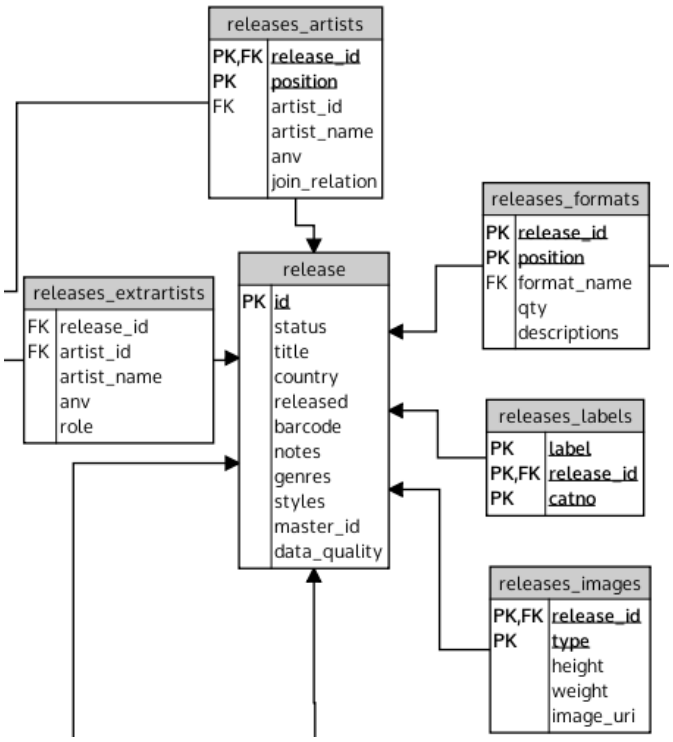

Abbildung 3.5: Ausschnitt „Release“ des Datenbankschemas von Discogs

Veröffentlichungen werden in der Tabelle `release` gespeichert. Wenn ein Album in mehreren Versionen erschienen ist, dann gibt es für jede dieser Versionen einen eigenen Eintrag in dieser Relation. Der Primärschlüssel ist eine eindeutige ID, die im Feld `id` gespeichert wird. Das Attribut `status` beinhaltet die Information darüber, ob ein Release von Discogs akzeptiert, abgelehnt, gelöscht wurde oder ob es ein Entwurf ist. Der Name der Veröffentlichung findet sich im Feld `title`. Das Veröffentlichungsland wird im Attribut `country` gespeichert. Das Veröffentlichungsdatum wird in `released` hinterlegt. Die Werte in diesem Attribut sind inkonsistent. Auch wenn das Format `yyyy-mm-dd` überwiegt, gibt es Einträge wie *21/08/1999* und *Aug 1967*. Dieser Umstand kann beim Preprocessing Schwierigkeiten bereiten, da die Daten auf das selbe Format gebracht werden müssen. Zu einigen Veröffentlichungen ist der Barcode bekannt und im Feld `barcode` hinterlegt. Anmerkungen zu einer Veröffentlichung können im Feld `notes` gespeichert werden. Die Musikrichtungen, denen die Veröffentlichung zugeordnet ist, werden im Attribut `genres` gespeichert. Stilrichtungen des zugeordneten Genres können im Feld `styles` angegeben werden. Ein Album, das bspw. ins Genre *Rock* eingetragen wurde, könnte z.B. den Stilrichtungen *Alternative Rock* und *Symphonic Rock* zugeordnet sein. Im Attribut `master_id` ist die ID des zur Veröffentlichung gehörenden Eintrags in der Tabelle `master` (siehe Unterabschnitt 3.1.5) eingetragen. Wie bereits bei allen

vorherigen Relationen ist auch hier im vorhandenen Dump das Attribut `data_quality` immer leer.

Die Künstler, die an der Veröffentlichung mitgewirkt haben, werden in der Tabelle `releases_artists` gelistet. Der Primärschlüssel der Relation ist die Kombination der Attribute `release_id`, welches auch Fremdschlüsselbeziehung zum Attribut `id` der Relation `release` ist, und `position`. In `position` wird die Position des Künstlers gespeichert, an der er genannt wird. Die Attribute `artist_id` und `artist_name` beinhalten die ID und den Namen des beteiligten Künstlers. Die ID bildet eine vermutete Fremdschlüsselbeziehung zum Attribut `id` der Relation `artist`. Auch in dieser Relation werden Variationen des Künstlernamens im Feld `anv` gespeichert. Um anzugeben wie der Künstler mit möglichen folgenden Künstlern für das Release verbunden werden soll ist das Attribut `join_relation` gedacht.

Geht die Mitarbeit eines Künstlers darüber hinaus, dann wird seine Mitarbeit in der Relation `releases_extraartists` festgehalten. Die Attribute `release_id`, `artist_id`, `artist_name` und `anv` sind analog zu denen in `releases_artists`, abgesehen davon, dass die Relation `releases_extraartists` keine Schlüssel enthält. Die Fremdschlüsselbeziehungen sind alle vermutet und werden in Abschnitt 3.1.7 geprüft. Die Art der Mitwirkung wird im Attribut `role` festgehalten. Mögliche Werte sind z.B. *Producer* und *Recorded by*.

Informationen über die Plattenfirma, die an der Veröffentlichung beteiligt ist, sind in der Tabelle `releases_labels` gespeichert. Alle Attribute der Relation bilden zusammen den Primärschlüssel. Das Attribut `label` beinhaltet den Namen der Plattenfirma. Die ID der Veröffentlichung ist im Feld `release_id` hinterlegt und bildet eine Fremdschlüsselbeziehung zum Attribut `id` der Tabelle `release`. Die Katalognummer der Veröffentlichung ist unter `catno` gespeichert.

Informationen darüber auf welchen Medien eine Veröffentlichung erschienen ist, sind unter `releases_formats` zu finden. Der Primärschlüssel der Relation bildet sich aus den Attributen `release_id`, `position`. Das Attribut `release_id` bezeichnet dabei per Fremdschlüsselbeziehung die ID der Veröffentlichung. Die Position, an der das Veröffentlichungsmedium genannt wird, ist im Feld `position` gespeichert. Über eine Fremdschlüsselbeziehung auf das Attribut `name` der Relation `format` gibt das Attribut `format_name` das Veröffentlichungsmedium, bspw. *CD* oder *Vinyl*, an. Das Feld `qty` gibt an, wie viele physische Medien zur Veröffentlichung gehören. Bei einem 2-CD-Release ist `qty` also zwei. Eine Beschreibung kann unter `descriptions` angegeben werden. Gängige Anmerkungen sind bspw. *Album*, *Compilation* und *12¨*.

3.1.5 Master

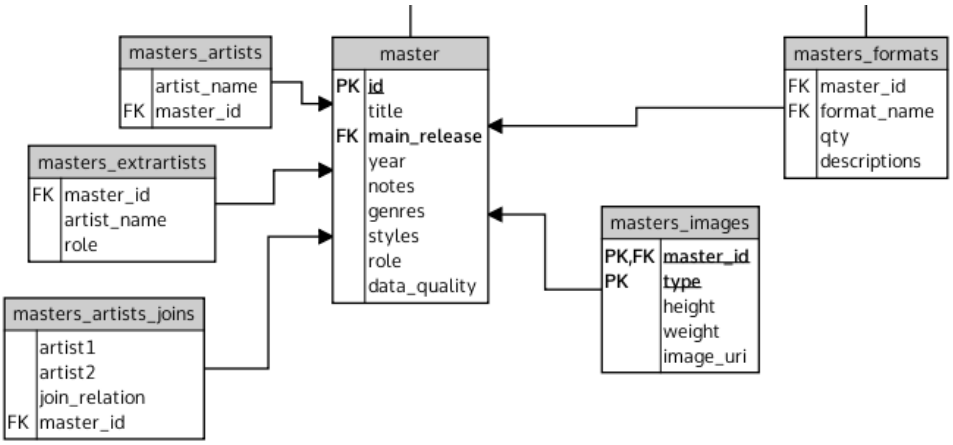

Abbildung 3.6: Ausschnitt „Master“ des Datenbankschemas von Discogs

Die Tabelle `master` dient dazu die verschiedenen Versionen einer Veröffentlichung zu organisieren. Der Primärschlüssel der Relation setzt sich zusammen aus der Kombination der eindeutigen ID im Attribut `id` und dem Attribut `main_release`. Das Attribut `main_release` beschreibt dabei die ID der Hauptveröffentlichung. Der Name der Veröffentlichung, nicht der der Hauptveröffentlichung, sondern der gemeinsame Name aller Veröffentlichungen zu diesem Mastereintrag, befindet sich im Feld `title`. Das Veröffentlichungsjahr ist in `year` gespeichert. Anmerkungen können unter `notes` hinterlegt werden, wie bspw. ein Hinweis darauf, dass ein Track nur auf einem bestimmten Release zu finden ist. Auch hier wird das Genre der Veröffentlichung unter `genres` und die Stilrichtung unter `styles` abgespeichert. Die Attribute `role` und `data_quality` sind im vorhandenen Dump stets leer.

In der Tabelle `masters_artists` werden Informationen über die mit dem Mastereintrag verbundenen Künstler hinterlegt. Dazu wird die ID des Eintrags in der Relation `master` im Attribut `master_id` gespeichert und der Name des Künstlers unter `artist_name`. Die Fremdschlüsselbeziehung auf `id` von `master` ist lediglich vermutet und wird später geprüft (siehe Abschnitt 3.1.7). In dieser Tabelle sind jedoch nur die Künstler zu Mastereinträgen mit genau einem Künstler gespeichert.

In der Relation `masters_artists_joins` wird für Mastereinträge mit mehr als einem Künstler festgehalten wie diese miteinander verbunden sind. Die ID des entsprechenden Eintrags wird im Attribut `master_id` hinterlegt. Es besteht eine vermutete Verbindung zum Mastereintrag über eine Fremdschlüsselbeziehung auf `id`. Die Namen zweier Künstler werden in den Feldern `artist1` und `artist2` gespeichert. Wie die beiden Künstler miteinander verbunden werden, befindet sich in `join_relation`. Gängige Werte sind bspw. *&* und *,*.

Die Relationen `masters_extraartists` und `masters_formats` enthalten im zur Verfügung stehendem Dump keine Daten. Aus diesem Grund werden diese Relationen

in weiteren Betrachtungen nicht berücksichtigt. Die Fremdschlüsselbeziehungen zu *id* von `master` und `name` von `format` sind nur vermutet.

3.1.6 Weitere Relationen

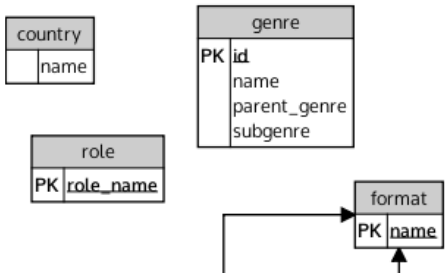

Abbildung 3.7: Ausschnitt weiterer Relationen des Datenbankschemas von Discogs

In der Relation `format` werden alle erlaubten Medien für Veröffentlichungen gelistet. Das einzige Attribut `name` ist der Primärschlüssel und gibt den Namen des Mediums an, bspw. *CD*. Die Tabellen `genre`, `role` und `country` sind im vorhandenen Dump leer und werden daher in weiteren Betrachtungen nicht berücksichtigt.

Die Relation `country` sollte eigentlich eine Liste aller Länder sein. Die Tabelle `role` ist gedacht, um eine Liste aller möglichen Rollen zu pflegen. Die Relation `genre` sollte nicht nur als Liste aller Genres dienen, sondern auch um die Beziehungen zwischen Genres zu speichern. Im Dump enthalten die eben genannten Relationen allerdings keine Daten.

3.1.7 Datenqualität

In diesem Unterabschnitt wird die Qualität der Daten von Discogs untersucht. Ziel dieser Untersuchung ist die Bestätigung der in den vorangegangenen Unterabschnitten dieses Kapitels angenommenen Fremdschlüsselbeziehungen sowie das Finden von zu behebenden Inkonsistenzen und Datenfehlern.

Fremdschlüsselbeziehungen

Bei den angenommenen Fremdschlüsselbeziehungen handelt es sich um Fremdschlüsselbeziehungen, die möglicherweise von den Daten impliziert werden, die aber nicht explizit gelten, wie bspw. die mögliche Referenz des Attributs `track.release_id` auf `release.id`. Die Fremdschlüsselbeziehungen werden mit Anfragen wie in Quelltext 3.1 geprüft.

```
1 SELECT release_id FROM track
2 EXCEPT SELECT id FROM release;
```

Quelltext 3.1: Prüfen einer Fremdschlüsselbeziehung

Da die Anfrage ein leeres Ergebnis liefert, ist die vermutete Fremdschlüsselbeziehung bestätigt, denn in der Relation `track` kommen keine Werte für Veröffentlichungs-IDs vor, die nicht in der Tabelle `release` vorkommen. Das Attribut `release_id` der Relation `track` referenziert also nur vorkommende Veröffentlichungs-IDs. Auf diese Weise konnten die meisten vermuteten Fremdschlüsselbeziehungen bestätigt werden.

Es gibt jedoch in der Tabelle `master` keine Fremdschlüsselbeziehung vom Attribut `main_release` auf die Veröffentlichungs-ID. Die Überprüfung hat ergeben, dass 1.680 Einträge in der Relation `master` keine gültige Veröffentlichungs-ID referenzieren. Das sind etwa 0,17% der 966.036 Einträge. Von diesen 1.680 Mastereinträgen werden jedoch nur 364 als Mastereintrag einer Veröffentlichung referenziert. Davon werden 135 mindestens zwei Mal referenziert (Ergebnis der Anfrage aus Quelltext 3.2). Um dieses Fremdschlüsselproblem zu lösen, könnte eine zufällig gewählte ID einer Veröffentlichung, die den betroffenen Mastereintrag referenziert, anstelle der ungültigen ID verwendet werden. Dieses Vorgehen wird gewählt, da im späteren Verlauf der Arbeit die Information über die Hauptveröffentlichung nicht mehr von großer Bedeutung ist.

```
1  SELECT count(*)
2  FROM (
3      SELECT master_id, count(master_id)
4      FROM discogs.release
5      WHERE master_id IN (
6          SELECT id FROM discogs.master
7          WHERE main_release NOT IN (
8              SELECT id FROM discogs.release
9          )
10         AND id IN (
11             SELECT DISTINCT master_id FROM discogs.release
12         )
13     )
14     GROUP BY master_id
15     HAVING count(master_id) > 1
16 ) AS result;
```

Quelltext 3.2: Anzahl mehrfach referenzierter Master mit ungültigem `main_release`

Außerdem gelten die in Abschnitt 3.1 angenommenen Fremdschlüsselbeziehungen der Relationen `tracks_artists`, `tracks_extraartists`, `releases_artists` und `releases_extraartists` auf die Künstler-ID nicht. Es sind 130.579 der 12.834.514 Einträge der Relation `tracks_artists` betroffen, wobei 57 unterschiedliche Künstler-IDs auftreten, die ungültig sind. Viele dieser Einträge haben im Attribut `artist_name` einen Eintrag wie *Unknown Artist*, *No Artist* oder *Various*. Diese Einträge werden fortan als *unbekannte Künstler* bezeichnet. Nach dem Filtern dieser Einträge sind lediglich noch 293 Einträge betroffen. Diese machen einen Anteil von ca. 0,0023% der Einträge der Relation aus. Es handelt sich dabei um ungültige Referenzen von 177 verschiedenen Künstlern (unterschiedliche Werte im Attribut `artist_name`). Die ID für unbekannte Künstler kann auf *NULL* gesetzt werden, da für unbekannte Künstler keine ID vorgese-

hen ist. Eine Rekonstruktion der ID über den Namen ist möglich, da die Eindeutigkeit der Künstlernamen gegeben ist. Es existieren allerdings Duplikate von 29 Künstlereinträgen. Bei insgesamt 4.308.702 Künstlereinträgen eine verschwindend geringe Menge. Im nächsten Punkt dieses Abschnitts wird die Duplikateliminierung behandelt.

Für die Relation `tracks_extraartists` ergibt sich das gleiche Problem, doch sind hier nach dem Filtern der Einträge mit unbekanntem Künstler noch 123.531 unterschiedliche Künstlernamen zu finden. Ein großer Teil dieser noch verbleibenden Einträge hat im Attribut `role` Werte wie *Written By*, *Executive Producer*, Danksagungen usw. Wenn man auch diese Einträge filtert, da sie nicht zwingend einen Eintrag in `artist` benötigen, weil sie möglicherweise keine Künstler sind, bleiben noch <100 Einträge, die eventuell Künstler sein könnten. Doch handelt es sich dabei um einen Anteil von weniger als 0,0003% der 31.317.709 Einträge in der Relation. Die Informationen über die erweiterten Mitwirkungen, wie sie in den Relationen `*_extraartists` gespeichert sind, sind für das Matching nicht von großer Bedeutung. Daher übersteigt der Aufwand für das Filtern aller Einträge, die höchstwahrscheinlich auf keinen Künstler verweisen, den möglichen Nutzen der Rekonstruktion von Künstler-IDs. Alle Fremdschlüsselbeziehung verletzenden IDs werden also mit *NULL* ersetzt.

Da die Relation `releases_artists` Künstler ihren Veröffentlichungen zuordnen soll, wäre eine Fremdschlüsselbeziehung zu beiden Relationen sinnvoll. Die Fremdschlüsselbeziehung zu den Künstlern wurde in Unterabschnitt 3.1.4 nur vermutet. Die Überprüfung hat ergeben, dass diese Fremdschlüsselbeziehung nicht ohne Einschränkungen gilt. Nach dem Filtern von unbekannten Künstlern sind 73 der 7.965.778 Einträge (ca. 0,0009%) betroffen, in denen 61 unterschiedliche Künstlernamen vorkommen. Sollten die IDs nicht über den Namen rekonstruiert werden können, so wird die ID auf *NULL* gesetzt, um die Fremdschlüsselbeziehung nicht zu verletzen. Dabei geht die Information über den Künstler allerdings verloren.

Bei der Relation `releases_extraartists` verhält es sich ähnlich zu der bereits betrachteten Relation `tracks_extraartists`. Nach dem Filtern von unbekannten Künstlern verbleiben noch Personen mit Rollen wie *Written By* oder *Photography*, die keinen Eintrag als Künstler haben, da sie keine sind. Nach dem auch diese gefiltert wurden, finden sich noch <300 Personen, die möglicherweise doch ein Künstler sind. Damit sind weniger als 0,001% der 29.190.354 Einträge betroffen. Da auch hier, wie bereits bei den Musikstücken, der Nutzen für das Matching geringer als der Aufwand für das Filtern und Rekonstruieren der IDs ist, werden alle IDs, die die Fremdschlüsselbeziehung verletzen, auf *NULL* gesetzt.

Wie Tabelle 3.1 zeigt, konnten die meisten der vermuteten Fremdschlüsselbeziehungen direkt bestätigt werden. Die verbleibenden Fremdschlüsselbeziehungen können wie oben beschrieben hergestellt werden, sodass im Folgenden alle Fremdschlüsselbeziehungen genutzt werden können. Außerdem konnte bestätigt werden, dass alle in der Tabelle `releases_labels` vorkommenden Namen von Plattenfirmen auch in der Relation `label` enthalten sind.

Attribut	vermuteter Fremdschlüssel	gilt
track.release_id	release.id	ja
releases_artists.release_id	release.id	ja
releases_extraartists.release_id	release.id	ja
master.main_release	release.id	nein
tracks_artists.track_id	track.track_id	ja
tracks_extraartists.track_id	track.track_id	ja
tracks_artists.artist_id	artist.id	nein
tracks_extraartists.artist_id	artist.id	nein
releases_artists.artist_id	artist.id	nein
releases_extraartists.artist_id	artist.id	nein
masters_artists.master_id	master.id	ja
masters_extraartists.master_id	master.id	ja
masters_artists_joins.master_id	master.id	ja
masters_formats.master_id	master.id	ja
masters_formats.format_name	format.name	ja

Tabelle 3.1: Gültigkeit der vermuteten Fremdschlüsselbeziehungen bei Discogs

Duplikate

Es existieren 58 Einträge in der Tabelle `artist`, mit mehrfach vorkommenden Namen. Jeder dieser Namen kommt genau zwei Mal vor. Dies legt den Verdacht nahe, dass es sich dabei um 29 Duplikate handelt. Die Duplikate wurden von Hand gelöscht und die Referenzen auf die andere Instanz des gleichen Künstlers geändert.

Veröffentlichungsdatum

Die Daten von Discogs weisen eine Inkonsistenz bei den Formaten für das Veröffentlichungsdatum auf. Zusätzlich, zu den in Tabelle 3.2 genannten Formaten, kommen noch Platzhalter für unbekannte Veröffentlichungsdaten wie bspw. *?*, *???*, *none*, *not known* und *unknown* vor, sowie nur teilweise vollständige Angaben wie z.B. *1997-0-15* oder Kombinationen aus den oben genannten Formaten und Platzhaltern wie bspw. *2001-12-??*. Doch sind damit nicht alle Fälle abgedeckt, denn es gibt auch Einträge wie bspw. *Oct-1971*. Doch da die nicht in Tabelle 3.2 aufgeführten Formate weniger als $0,001\%$ ausmachen, werden diese im Preprocessing außer Acht gelassen und nur die genannten Formate berichtigt. Für die anderen Einträge wird der Verlust der Information über das Veröffentlichungsdatum in Kauf genommen.

Format	Beispiel	RegExp
yyyy	2016	[0-9]{4}
yyyy-mm	2016-08	[0-9]{4}-[0-9]{2}
yyyy-mm-dd	2016-08-22	[0-9]{4}-[0-9]{2}-[0-9]{2}
yyyy mm dd	2016 08 22	[0-9]{4} [0-9]{2} [0-9]{2}
yyyymmdd	20160822	[0-9]{8}
yyyy/mm/dd	2016/08/22	[0-9]{4}/[0-9]{2}/[0-9]{2}
(m)m/dd/(yy)yy	8/22/16	[0-9]{1,2}/[0-9]{2}/([0-9]{2}){1,2}
dd.mm.yyyy	22.08.2016	[0-9]{2}.[0-9]{2}.[0-9]{4}
mm-dd-yyyy	08-22-2016	[0-9]{2}-[0-9]{2}-[0-9]{4}
Month dd, yyyy	August 22, 2016	[a-zA-z]{3,9} [0-9]{1,2}, [0-9]{4}

Tabelle 3.2: Unterschiedliche Datumsformate bei Discogs

Spieldauer

Die Daten von Discogs weisen ebenfalls eine Inkonsistenz bei den Formaten für die Spieldauer eines Musikstücks auf. Zusätzlich zu den in Tabelle 3.3 genannten Formaten gibt es noch Einträge, die mit Doppelpunkten beginnen oder enden. Diese Einträge können als Einträge des Formats *mm:ss* interpretiert werden. Es gibt auch Einträge mit Spielzeiten wie bspw. *100000:00:00*. Die Spieldauer dieser Stücke kann durch die Beschränkung des Datentyps nicht in Millisekunden dargestellt werden und ist mit 100.000 Stunden unsinnig. Als Konsequenz werden die Spieldauern für Stücke mit solchen Spielzeiten im Preprocessing gelöscht. Davon sind 23 der 59.128.543 Einträge für Musikstücke betroffen.

Format	Beispiel
mm:ss	03:28
mm.ss	03.28
mmss	328

Tabelle 3.3: Unterschiedliche Zeitformate bei Discogs

Redundanzen

Das Schema von Discogs sieht das Speichern von redundanten Informationen vor. Zur Referenzierung eines Künstlers muss nur die ID gespeichert werden, doch wird in vielen Relationen neben der ID auch der Name des Künstlers gespeichert. Die gespeicherten Namen passen zu den gespeicherten IDs. Dies wurde mit Anfragen wie der in Quelltext 3.3 gezeigt. Die Information über den Namen kann also ohne Bedenken weggelassen werden.

```sql
SELECT t.artist_id, t.artist_name, a.id, a.name
FROM discogs.tracks_artists AS t, discogs.artist AS a
WHERE t.artist_id IN (SELECT id FROM discogs.artist)
  AND t.artist_id = a.id
  AND lower(t.artist_name) != lower(a.name);
```

Quelltext 3.3: Überprüfung der Redundanzen

Außerdem werden für Einzelpersonen alle Bands gespeichert, in denen sie Mitglied sind, und für Bands die Mitglieder. Um eine Zuordnung vornehmen zu können, reicht eine der beiden Richtungen aus. Hier gibt es in den Daten Unterschiede. Für acht Künstler fehlt der Mitgliedseintrag in der Band und für zehn Bands fehlt der Bandeintrag beim Musiker.

Künstlerzuordnung zu Mastereinträgen

Die Zuordnung von Künstlern zu Mastereinträgen ist von der Datenstruktur her komplizierter als die für Musikstücke oder Veröffentlichungen. Es gibt zwei Tabellen für die Zuordnung. Eine Tabelle für Mastereinträge mit genau einem Künstler und eine Tabelle für Mastereinträge mit mindestens zwei Künstlern (siehe Tabelle 3.4). In diesen Tabellen wird die Position, an der der Künstler genannt werden soll, jedoch nicht gespeichert.

master_id	artist_name
533373	Wardruna
888026	Omnia

(a) ein Künstler (`masters_artists`)

master_id	artist1	artist2	join_relation
47318	Jay-Z	Linkin Park	/
754823	Vesa-Matti Loiri	Khalil Gibran	,
754823	Khalil Gibra	Olli Ahvenlahti	,

(b) mehrere Künstler (`masters_artists_joins`)

Tabelle 3.4: Tabellen für die Künstler-Master-Zuordnung

Eine erste Analyse lieferte den Ansatz die Reihenfolge der Künstler über die Einträge in den Attributen `artist1` und `artist2` der Relation `masters_artists_joins` zu rekonstruieren. Wenn ein Künstler nur in `artist1` vorkommt, so müsste es sich dabei um den erstgenannten Künstler handeln. Anschließend würde der Eintrag gesucht, bei dem der zweite Künstler des Datensatzes auf dem ersten Platz ist. Sollte es keine weiteren Datensätze zum Mastereintrag mehr geben, so wird der Künstler in Attribut `artist2` auf die nächste zu vergebende Position gesetzt.

Eine genauere Betrachtung hat gezeigt, dass sich die Daten für dieses Vorgehen nicht eignen, da es in der Relation Datensätze gibt, die keine eindeutige Zuordnung erlauben. Bspw. gibt es für einige Mastereinträge einen weiteren Datensatz, in dem beide

Künstler *NULL* sind. Des Weiteren gibt es Einträge, die im angedachten Verfahren zur Rekonstruktion eine Endlosschleife bewirken würden. In Tabelle 3.5 sind beispielhaft Datensätze aufgeführt, die die eindeutige Zuordnung nicht erlauben.

master_id	artist1	artist2	join_relation
123	Mixrace	Pro-Ton-Isospace	&
123	*NULL*	*NULL*	,

(a) *NULL*-Werte

master_id	artist1	artist2	join_relation
181	Christiaan Kouijzer	Zeusz	+
181	Christiaan Kouijzer	Zeusz	+
181	Zeusz	Christiaan Kouijzer	/

(b) Endlosschleife

Tabelle 3.5: Uneindeutige Zuordnungen in `masters_artists_joins`

Aufgrund der Ergebnisse wird im weiteren Verlauf der Arbeit ein anderer Ansatz verfolgt. Die Künstlerzuordnung zu einem Mastereintrag erfolgt nun über die Künstlerzuordnung der Hauptveröffentlichung.

3.1.8 Zusammenfassung

Abbildung 3.8 zeigt das Datenbankschema von Discogs ohne die Relationen und Attribute, die aufgrund der Ergebnisse der vorangegangenen Analyse des Schemas und der Daten für die weitere Betrachtung im Rahmen dieser Arbeit nicht relevant sind. Es enthält außerdem auch die vermuteten Fremdschlüsselbeziehungen.

Es handelt sich dabei vor allem um die leeren Relationen wie bspw. `genre` und das stets leere Attribut `data_quality`. Doch auch die Relationen für die Bilder sind in der weiteren Untersuchung nicht relevant, denn sie enthalten keine Daten, die beim Matching von Nutzen sind. Außerdem war das Feld `image_uri` immer leer und somit wären die Tabellen lediglich eine Auflistung von Bilddimensionen.

In Abschnitt 3.3 wird untersucht wie `release` und `master` weiter transformiert werden müssen, um ein gemeinsames Schema mit den Daten von *MusicBrainz* erstellen zu können. Dies gilt auch für die Tabellen zur Künstlerzuordnung, wie bspw. `tracks_artists` und `tracks_extraartists`. Die Untersuchung der Datenqualität hat gezeigt, dass diese hoch genug ist, um in der Datenvorverarbeitung die implizierten Fremdschlüsselbeziehungen herzustellen.

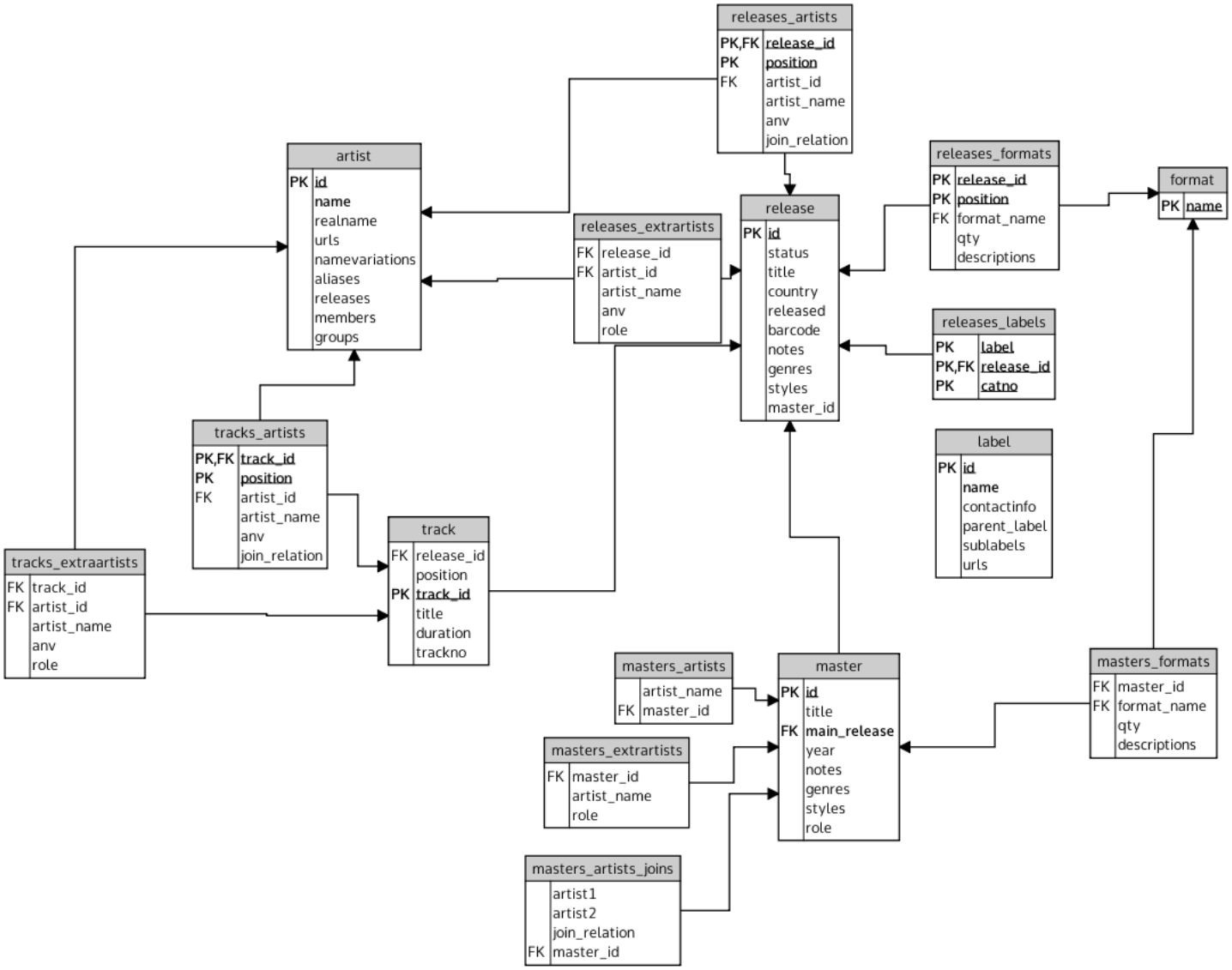

Abbildung 3.8: Bereinigtes Datenbankschema Discogs

3.2 MusicBrainz

Das Internetprojekt MusicBrainz ist ein Projekt zur Erstellung einer freien und offenen Musikdatenbank. Das Projekt wurde als Reaktion auf die Umstellung auf ein proprietäres Datenformat der *CDDB* im Jahr 1999 gegründet.

Die Einträge in der Datenbank sind besser gepflegt und umfangreicher als bei vielen ähnlichen Projekten. Dies liegt zum Teil auch daran, dass Angaben durch die Benutzer einen Kontrollprozess durchlaufen, der sich vom *Peer Review* ableitet. [Met15] Aufgrund der hohen Datenqualität wird in diesem Abschnitt nicht näher auf die Qualität der Daten eingegangen.

Abbildung 3.9 zeigt, dass das Datenbankschema von MusicBrainz komplexer als das von Discogs ist. Es sind nur die wichtigsten Tabellen im Schema aufgeführt. Ähnlich wie bei Discogs können auch in diesem Datenbankschema Entities ausgemacht werden, die den Kern des Schemas bilden. Die Hauptentities sind die Künstler (`artist`), Musikstücke (`track`), Plattenfirmen (`label`), Veröffentlichungen (`release`), Arbeiten (`work`) und Orte (`area`). Zur weiteren Untersuchung des Schemas wird dieses in Gruppen um die Hauptentities eingeteilt, welche einzeln für sich in den Unterabschnitten 3.2.1 bis 3.2.7 untersucht werden. In Unterabschnitt 3.2.8 werden die Ergebnisse der Analyse zusammengefasst.

Da einige Strukturen und Attribute mehrfach genutzt werden, werden diese bereits an dieser Stelle erläutert. Jedem Objekt in der Datenbank wird eine eindeutige 32-stellige MBID (*MusicBrainz ID*) zugeordnet. Das Attribut `gid` dient dazu MBIDs nach der Verschmelzung zweier Objekte von der einen auf die andere weiterzuleiten.

Die Hauptentities haben eine Relation `*_alias`, die alle die gleiche Struktur haben und alternative Namen beinhalten. Die Relationen `*_type` dienen einem einfachen Mapping von IDs und Strings für das häufig auftretende Attribut `type`, welches den Typ des Eintrags, bspw. *Group* oder *Person* für einen Künstler, beschreibt. Die Tabellen `*_ipi` und `*_isni` beinhalten für Künstler und Plattenfirmen die *Interested Parties Information* (IPI) und *International Standard Name Identifier* (ISNI), da ihnen mehr als einer zugeordnet sein kann.

Das Attribut `edits_pending` ist nicht von Bedeutung für diese Arbeit, da es angibt wie viele Änderungen des Datensatzes noch auf eine Bestätigung bzw. Ablehnung warten. Das Feld `last_updated` enthält die Information, wann der Datensatz das letzte Mal aktualisiert wurde und ist daher für das Matching irrelevant, für die Bewertung der Aktualität jedoch von Bedeutung.

Die Felder `{begin|end}_date_*` geben, abhängig vom Kontext, das Start- und Enddatum an. Für einen Musiker wären dies das Geburts- und Sterbedatum und für eine Band deren Gründungs- und Auflösungsdatum. Das Attribut `ended` ist vom Typ boolean und *true* bedeutet, dass etwas zu Ende ist, *false* das Gegenteil. Der Vorgabewert ist *false*. Für eine Person heißt `ended = false`, dass sie noch lebt. Das Attribut `comment` kann eine Bemerkung zum jeweiligen Datensatz enthalten.

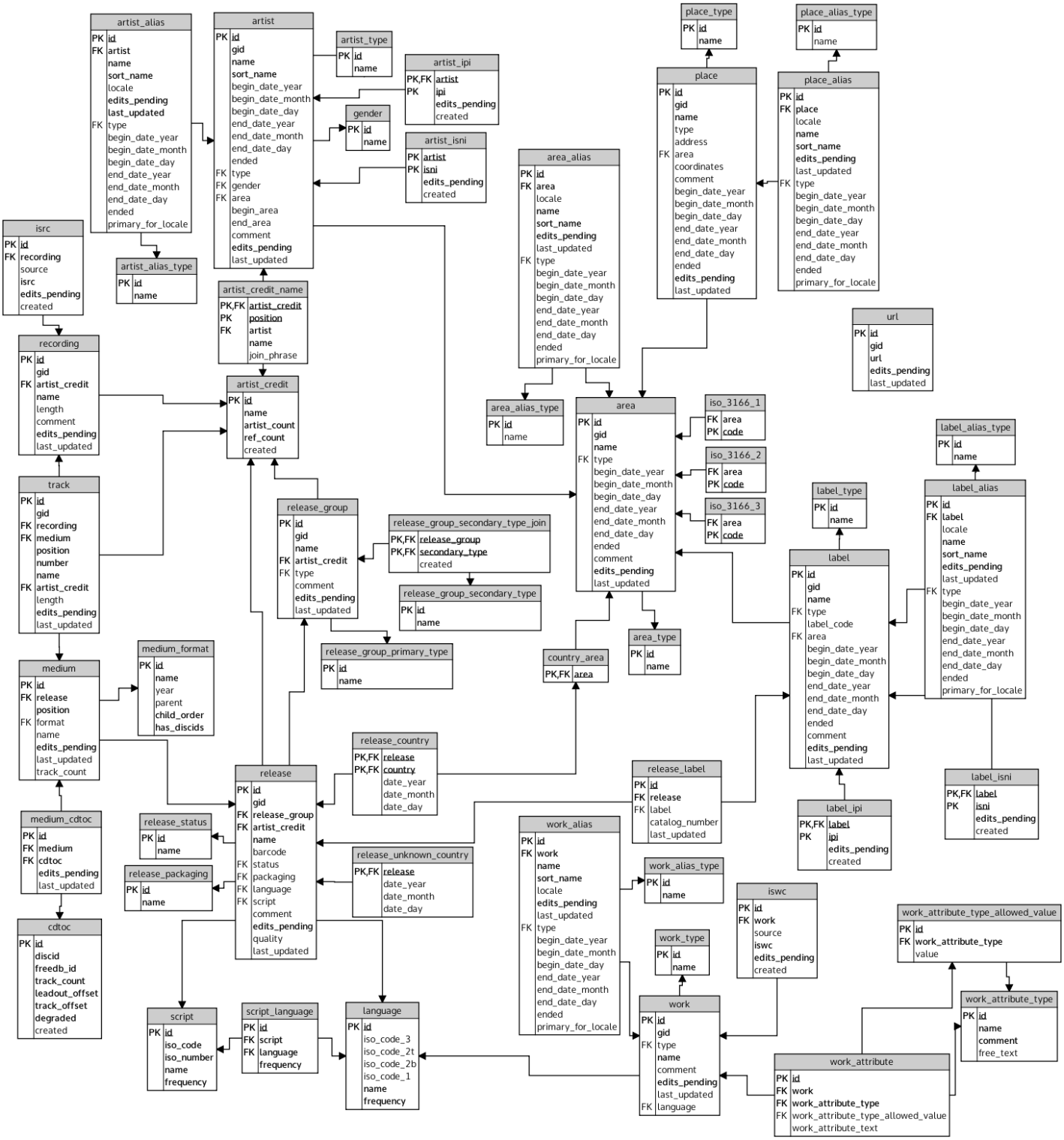

Abbildung 3.9: Datenbankschema MusicBrainz[5]

[5]basierend auf https://musicbrainz.org/doc/MusicBrainz_Database/Schema

3.2.1 Artist

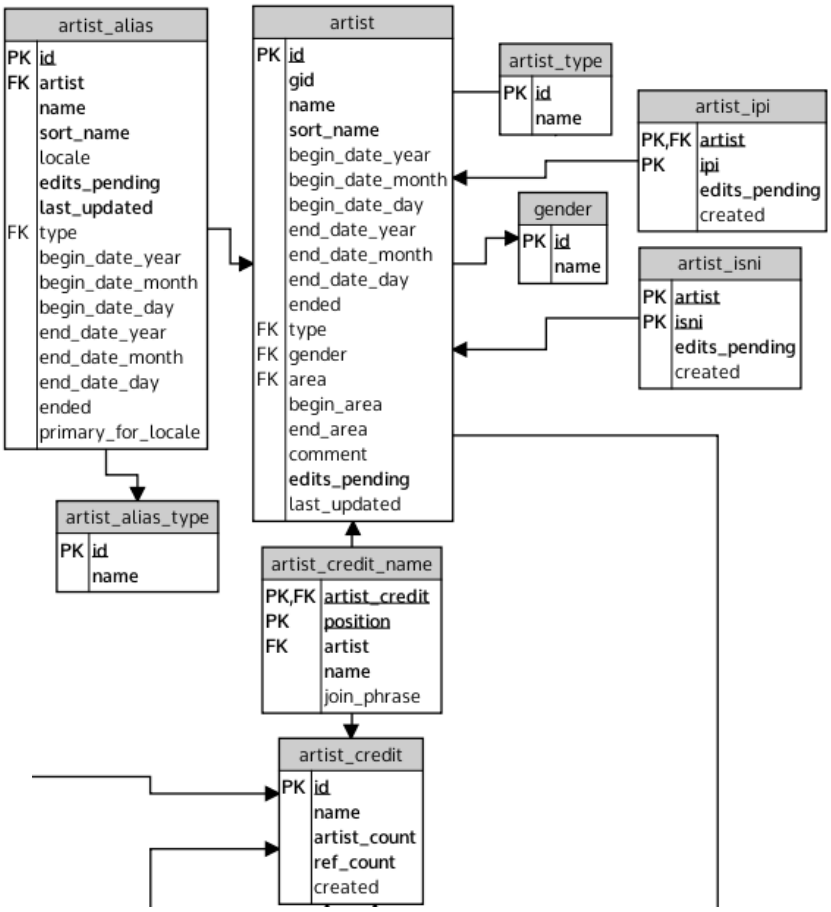

Abbildung 3.10: Ausschnitt „Artist“ des Datenbankschemas von MusicBrainz

In der Tabelle `artist` werden die erfassten Künstler gespeichert. Dabei kann es sich um einzelne Personen, Bands, Orchester usw. handeln. Der Primärschlüssel der Relation ist die eindeutige ID im Feld `id`. Der Name des Künstlers ist im Attribut `name` hinterlegt. Im Feld `sort_name` wird der Name des Künstlers für die Suche gespeichert. Bei Personen könnte dies bspw. *Nachname, Vorname* sein. Um welchen Typ es sich handelt, wird im Feld `type` gespeichert. Das Geschlecht eines Künstlers kann in das Attribut `gender` eingetragen werden, wobei hier die ID des Geschlechts als Fremdschlüsselbeziehung zur Tabelle `gender` eingetragen wird. Das Attribut `area` kann als Nationalität interpretiert werden, die Attribute `begin_area` und `end_area` folglich als Anfangs- und Endort. Dies würde für eine Einzelperson den Begriffen Geburts- und Sterbeort entsprechen. Für eine Musikgruppe wäre `begin_area` dann der Gründungsort.

3.2.2 Track

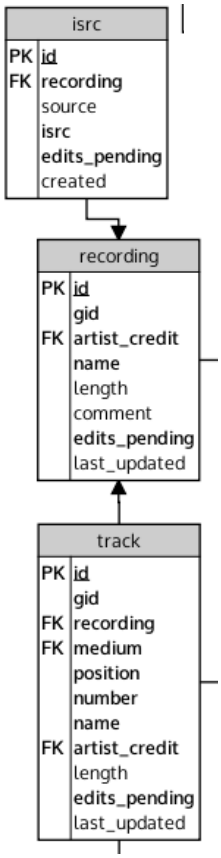

Abbildung 3.11: Ausschnitt „Track“ des Datenbankschemas von MusicBrainz

Musikstücke werden auf MusicBrainz in den Relationen `recording` und `track` gespeichert. Als Track wird hierbei ein Musikstück einer Veröffentlichung angesehen. Ein Track enthält über Fremdschlüsselbeziehungen eine Verlinkung zur Aufnahme (`recording`) und den Künstlern (`artist_credit`) des entsprechenden Mediums (`medium`). Der Name des Musikstücks wird unter `name` gespeichert. Die Titelnummer ist im Feld `position` gespeichert. Für Veröffentlichungen mit nur einem physischen Datenträger ist dieser Wert gleich dem des Attributs `number`. Bei einem Release auf einer doppelseitigen Schallplatte könnte `position` jedoch den Wert *8* haben, während der Wert von `number` *B3* ist. Es handelt sich also um das achte Lied des Albums und um das dritte auf Seite B. Die Spieldauer in Millisekunden wird im Attribut `length` festgehalten.

In der Tabelle `recording` werden die Informationen zum Musikstück gespeichert, die für den Benutzer sichtbar sind, wenn er den Song nicht im Zusammenhang mit einer Veröffentlichung betrachtet. *International Standard Recording Codes* (ISRC) werden den Aufnahmen über die Tabelle `isrc` zugeordnet. Eine Betrachtung der Relation hat ergeben, dass das Attribut `source` immer leer ist oder den Wert *0* hat. Im Folgenden wird dieses Attribut daher nicht weiter betrachtet.

3.2.3 Label

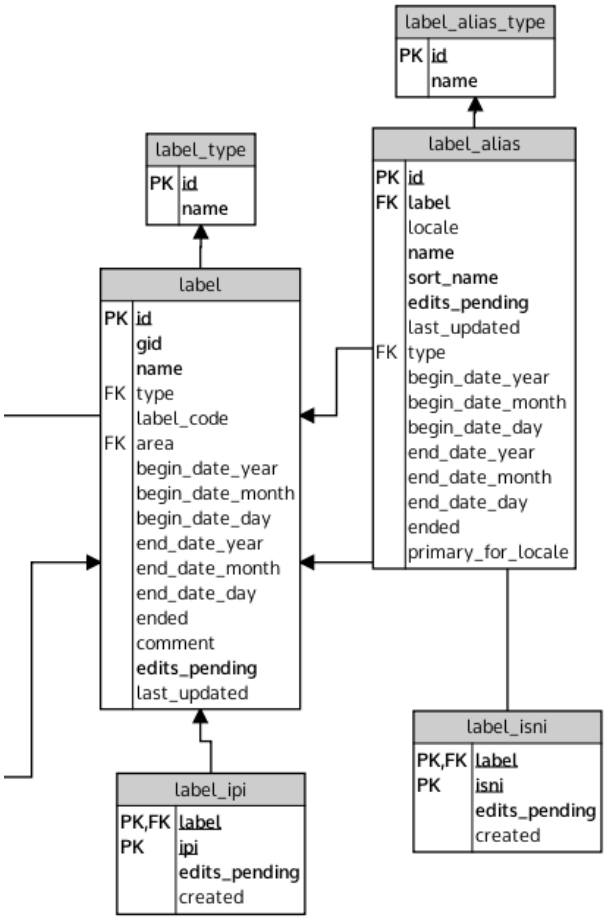

Abbildung 3.12: Ausschnitt „Label“ des Datenbankschemas von MusicBrainz

Informationen zu Plattenfirmen, Verlegern usw. sind in der Relation `label` abgespeichert. Der Primärschlüssel ist eine eindeutige ID im Attribut `id`. Der Name des Labels befindet sich im Attribut `name`. Mögliche Werte für die Art eines Labels sind bspw. *Distributor*, *Publisher* und *Production*. Der *Label Code* einer Plattenfirma kann im Feld `label_code` gespeichert werden.

3.2.4 Release

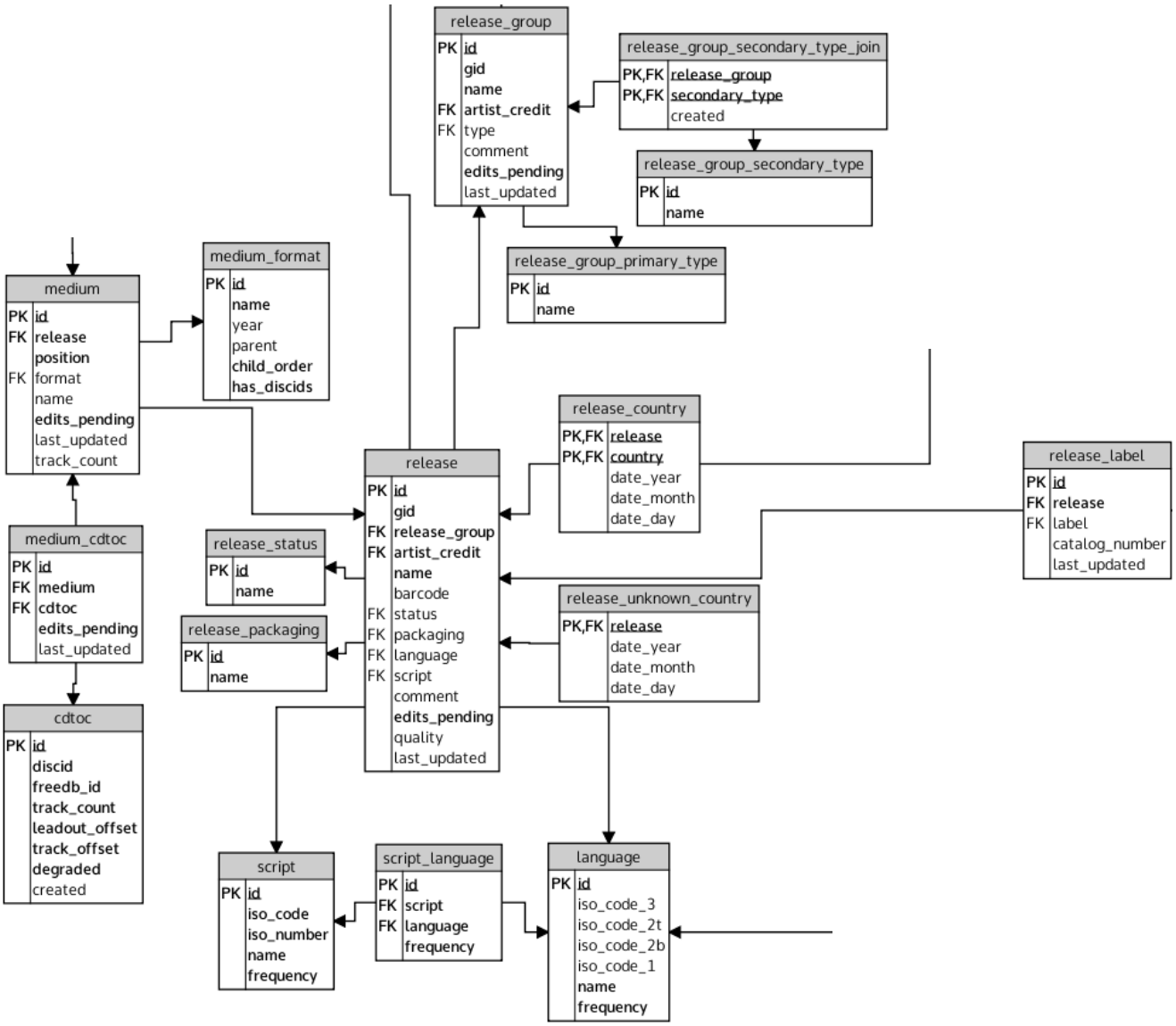

Abbildung 3.13: Ausschnitt „Release“ des Datenbankschemas von MusicBrainz

In der Tabelle `release` werden sämtliche Veröffentlichungen gespeichert. Eine Veröffentlichung ist hierbei eine Version eines Albums. Ist ein Album in mehr als einer Version erschienen, so gibt es in dieser Relation entsprechend viele sehr ähnliche Einträge. Der Barcode der Veröffentlichung kann im Attribut `barcode` hinterlegt werden. Eine Information über die Güte des Datensatzes befindet sich im Feld `quality`.

Weitere Informationen zu einem Release werden über Fremdschlüsselbeziehungen aus anderen Tabellen gelesen. Das Attribut `status` gibt die Art der Veröffentlichung, z.B. *official*, *Promotion*, *Bootleg* usw., über eine Referenz auf `release_status` an. Die Art der Verpackung wird über das Feld `packaging` der Tabelle `release_packaging` ent-

nommen. Die Informationen über die Sprache können aus den Tabellen `script` und `language` abgelesen werden. Dabei enthält `script` die Information über die Schrift, bspw. *Latin* oder *Greek*. Die Tabelle `language` liefert die Information über die Sprache, also z.B. *English* oder *Swedish*. Um eine Verbindung mit den Künstlern herzustellen, wird auf einen Eintrag in `artist_credit` verwiesen.

Um die verschiedenen Versionen einer Veröffentlichung verwalten zu können, gibt es eine Verlinkung auf einen Eintrag in `release_group`. Auch dieses übergeordnete Objekt hat einen Verweis auf `artist_credit`. Zu einer Veröffentlichungsgruppe gehört zudem ein Typ wie *Album* oder *Single*. Veröffentlichungstypen wie *Compilation* und *Soundtrack* werden über `release_group_secondary_type_join` mit der Veröffentlichung und dem Primärtyp verknüpft.

Die restlichen Daten zu einem Release werden über Fremdschlüsselbeziehungen auf die ID der Veröffentlichung gespeichert. Die Plattenfirmen werden über die Relation `release_label` über Fremdschlüsselbeziehungen mit den Veröffentlichungen verknüpft. In dieser Tabelle kann auch die Katalognummer eingetragen werden. Das Erscheinungsland wird über `release_country` zugeordnet. In dieser Relation wird auch das jeweilige Veröffentlichungsdatum hinterlegt. Das Veröffentlichungsmedium wird in der Tabelle `medium` gespeichert. In dieser Tabelle wird das Format, z.B. *CD* oder *Vinyl*, und die Titelanzahl festgehalten. Die Musikstücke der Veröffentlichung haben eine Fremdschlüsselbeziehung auf das Medium.

3.2.5 Work

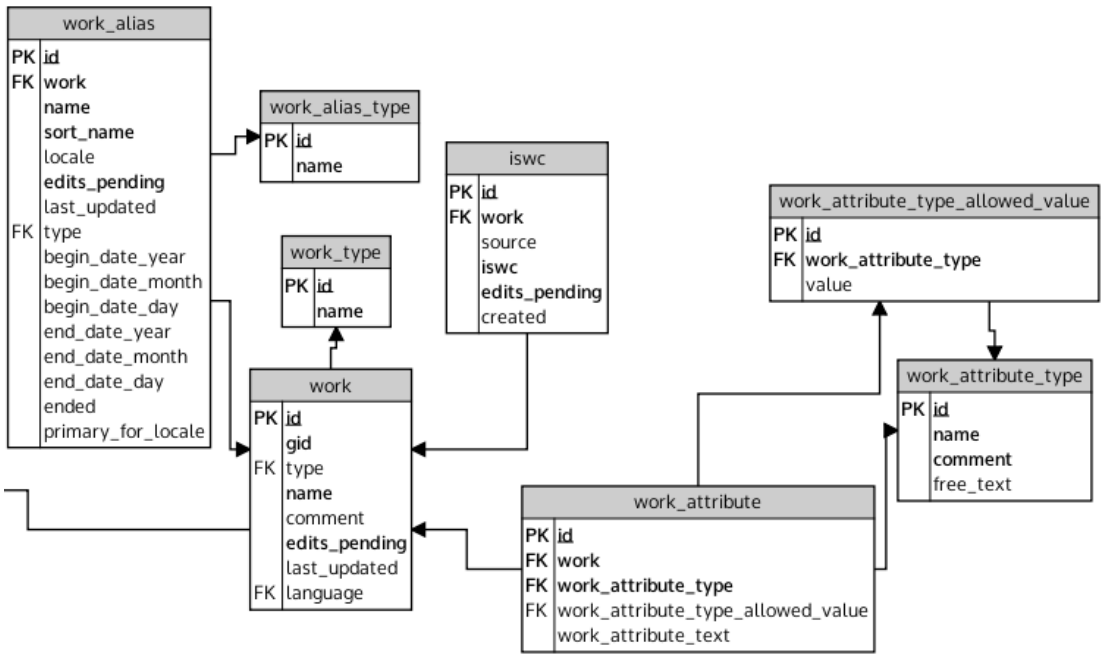

Abbildung 3.14: Ausschnitt „Work“ des Datenbankschemas von MusicBrainz

Bei MusicBrainz werden Lieder, Kompositionen usw. unter `work` gespeichert. Die Relation `recording` ist eine Repräsentation der Aufnahme dieser Arbeit. Der Name des Musikstücks ist im Attribut `name` hinterlegt. Über die Tabelle `iswc` wird den Arbeiten ihr *International Standard Musical Work Code* (ISWS) zugeordnet. Über eine Fremdschlüsselbeziehung zur Tabelle `work_type` wird dem Stück seine Art, z.B. *Musical* oder *Song*, zugewiesen. Es gibt ebenfalls eine Fremdschlüsselbeziehung zur Sprache des Stücks. Weitere Attribute wie bspw. die *GEMA ID* können dem Musikstück über die Relationen `work_attribute_type` und `work_attribute_type_allowed_value` zusammen mit dem Attribut `work_attribute_text` zugewiesen werden. Diese ergänzenden Informationen sind in der Tabelle `work_attribute` gespeichert.

3.2.6 Area

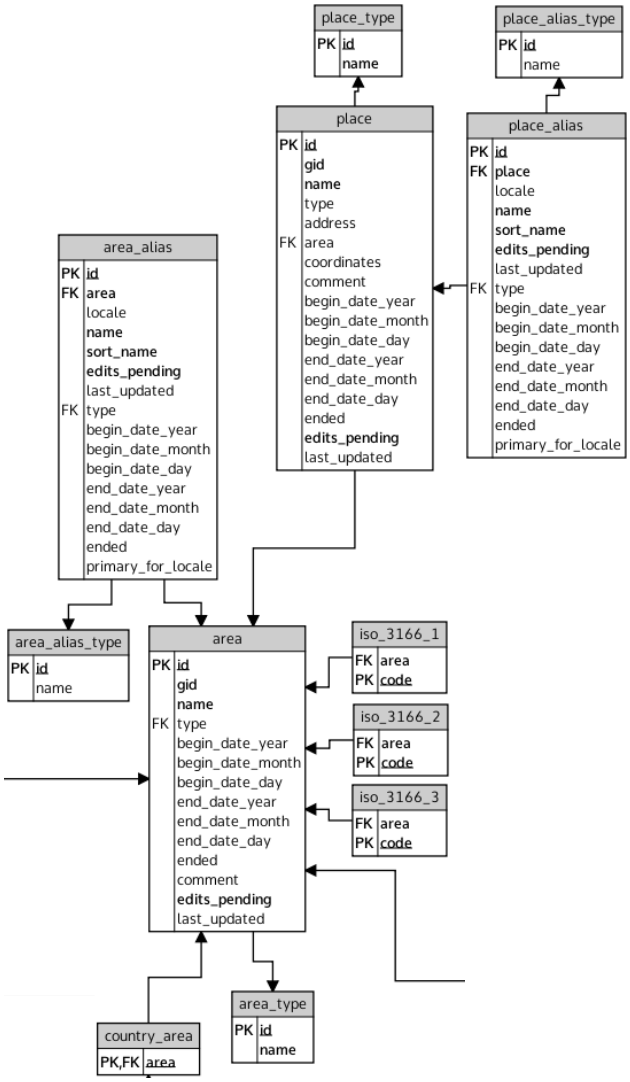

Abbildung 3.15: Ausschnitt „Location“ des Datenbankschemas von MusicBrainz

MusicBrainz hält viele Informationen über Orte, wie Abbildung 3.15 zeigt. In der Relation `area` werden Länder, Verwaltungsbezirke, Städte u.ä. gespeichert. Der Name des Ortes wird im Attribut `name` hinterlegt. Über eine Fremdschlüsselbeziehung auf die Tabelle `area_type` wird angegeben, welcher Art der gespeicherte Ort ist. Die Länder, die für Veröffentlichungen als Veröffentlichungsland eingetragen werden können, sind in der Tabelle `country_area` festgehalten, welche den entsprechenden Eintrag in `area` referenziert. Aufnahmestudios, Sportarenen usw. können in der Relation `place` eingetragen werden. Diese Orte können mittels Fremdschlüsselbeziehung einer Stadt, einem Land usw. zugeordnet werden. Des Weiteren können die genaue Adresse im Feld `address` und die Koordinaten in `coordinates` gespeichert werden.

3.2.7 Weitere Relationen

Abbildung 3.16: Ausschnitt weiterer Relationen des Datenbankschemas von MusicBrainz

Die Relation `url` wird verwendet, um auf externe Ressourcen wie bspw. eine offizielle Homepage zu verweisen. Jeder Eintrag in dieser Tabelle erhält eine eindeutige ID im Attribut `id`, welches auch der Primärschlüssel ist. Die URL der externen Ressource wird im Feld `url` gespeichert. Da die Relation Einträge enthält, mit denen eine Verbindung zu Discogs hergestellt werden kann, wie bspw. *http://www.discogs.com/release/444000*, könnte sie eventuell für das Matching relevant sein.

3.2.8 Zusammenfassung

Das Datenbankschema von MusicBrainz ist deutlich komplexer als das von Discogs. Einige Informationen sind für diese Arbeit im Folgenden nicht weiter von Interesse. Dazu zählen vor allem die nicht erwähnten Relationen, die verschiedene Verlinkungen vornehmen. Im Preprocessing ergeben sich möglicherweise ein paar Ausnahmen. Ebenfalls wurden die Attribute `edits_pending`, `created`, `last_updated` und `quality` entfernt, da sie lediglich intern von Bedeutung sind. Die Relation `script_language` wurde gelöscht, da sie zumindest im Dump nicht vorhanden ist.

In Abschnitt 3.3 wird untersucht, wie mit Relationen wie `release` und `release_group` weiter verfahren werden muss, um ein gemeinsames Datenbankschema mit den Daten von Discogs erstellen zu können.

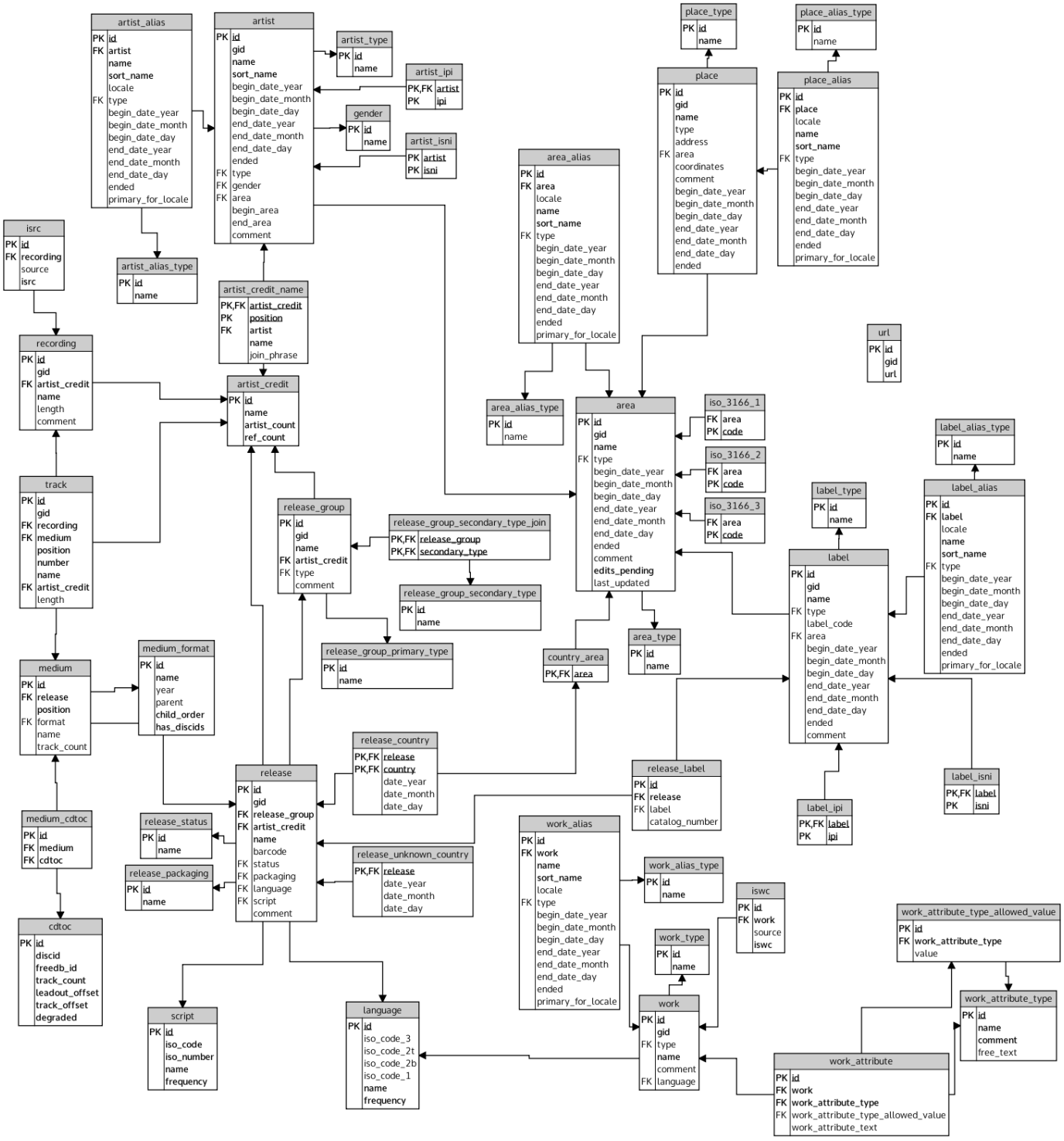

Abbildung 3.17: Bereinigtes Datenbankschema MusicBrainz

3.3 Vergleich

In diesem Abschnitt werden die analysierten Datenbankschemata auf Gemeinsamkeiten und Unterschiede untersucht. Es wird hierbei der Fokus auf die Struktur gelegt. Die Daten der Datenbanken werden in Abschnitt 4.1 bei der Entwicklung eines gemeinsamen Schemas genauer betrachtet.

Die Datenbankschemata von Discogs und MusicBrainz weisen eine ähnliche Struktur auf. So werden unter dem Begriff des Künstlers Einzelpersonen und Bands verstanden. Es gibt jedoch Unterschiede in der Verlinkung von Einzelpersonen zu den Bands. Discogs hat hierfür ein Attribut und MusicBrainz eine Relation. Ebenso verhält es sich mit der Information über andere Namen (Aliase) und Webseiten. MusicBrainz bietet Attribute zum Speichern von Geburts- und Sterbedaten, Discogs hingegen bietet dafür keine Speichermöglichkeit. Es werden nur Informationen verwendet, die in beiden Datenbanken vorhanden sind und die Beziehung zwischen Einzelpersonen und Gruppen wird vereinheitlicht.

Beide Schemata sehen vor die Künstler mit Musikstücken und Veröffentlichungen zu verknüpfen. Dies geschieht jedoch auf unterschiedliche Art und Weise. Discogs nutzt für Musikstücke und Veröffentlichungen eine separate Tabelle, wie bspw. `tracks_artists`, die eine Verknüpfung mit einem Künstler darstellt. MusicBrainz hingegen benutzt die Tabellen `artist_credit_name` und `artist_credit` um eine Verlinkung zu ermöglichen. Dies muss für ein gemeinsames Schema vereinheitlicht werden.

Musikstücke werden bei Discogs als die Tracks einer Veröffentlichung verstanden. MusicBrainz hingegen hat zusätzlich noch Informationen über die Aufnahmen eines Stückes unabhängig von einer Veröffentlichung (Relation `recording`), sowie über das Musikstück unabhängig von einer Interpretation (Relation `work`). Da die Informationen über Aufnahmen und Musikstücke unabhängig von einer Interpretation in den Daten von Discogs nicht vorhanden sind, werden sie beim Matching nicht berücksichtigt.

Plattenfirmen haben bei Discogs anders als bei MusicBrainz keine Aliase. Die Beziehungen zwischen Labels sind bei Discogs in Attributen festgehalten, bei MusicBrainz jedoch über eine Relation. Auch hier werden wie bei den Künstlern nur die Informationen verwendet, die in beiden Datenbeständen vorhanden sind und die Beziehung zwischen den Plattenfirmen vereinheitlicht.

Veröffentlichungen werden bei beiden Datenbanken ähnlich abgespeichert. Es wird zwischen den verschiedenen Versionen einer Veröffentlichung unterschieden und es existiert ein übergeordnetes Objekt zur Verwaltung der unterschiedlichen Varianten. Dieses Objekt heißt bei Discogs `master` und bei MusicBrainz `release_group`. Die Veröffentlichungsmedien jeder Veröffentlichung werden in einer Relation gehalten.

Abbildung 3.18 zeigt das erneut bereinigte Datenbankschema für Discogs unter Berücksichtigung der für ein Matching relevanten Relationen und Attribute. Im Unterschied zum Schema aus Abbildung 3.8 fehlen die Relationen `*_extraartists`, da diese Informationen über ein erweitertes Mitwirken an einem Musikstück oder einer Veröffentlichung beinhalten, die in dieser Weise auf MusicBrainz nicht gespeichert werden.

Abbildung 3.19 zeigt das ebenfalls unter Berücksichtigung der für ein Matching relevanten Daten erneut bereinigte Datenbankschema für MusicBrainz. Im Gegensatz zum Schema aus Abbildung 3.17 fehlen die meisten Relationen, die zum Speichern von Orten wie bspw. Konzerthallen dienen. Außerdem fehlen alle Relationen um das Entity `work`, da die damit verbundenen Informationen auf Discogs nicht enthalten sind. Des Weiteren wurde die Relation `recordings` entfernt, da es auf Discogs kein Gegenstück gibt. Da bei Discogs keine Informationen über Geburts-, Sterbe-, Gründungsdaten u.ä. hinterlegt sind, wurden auch alle damit verbundenen Attribute entfernt.

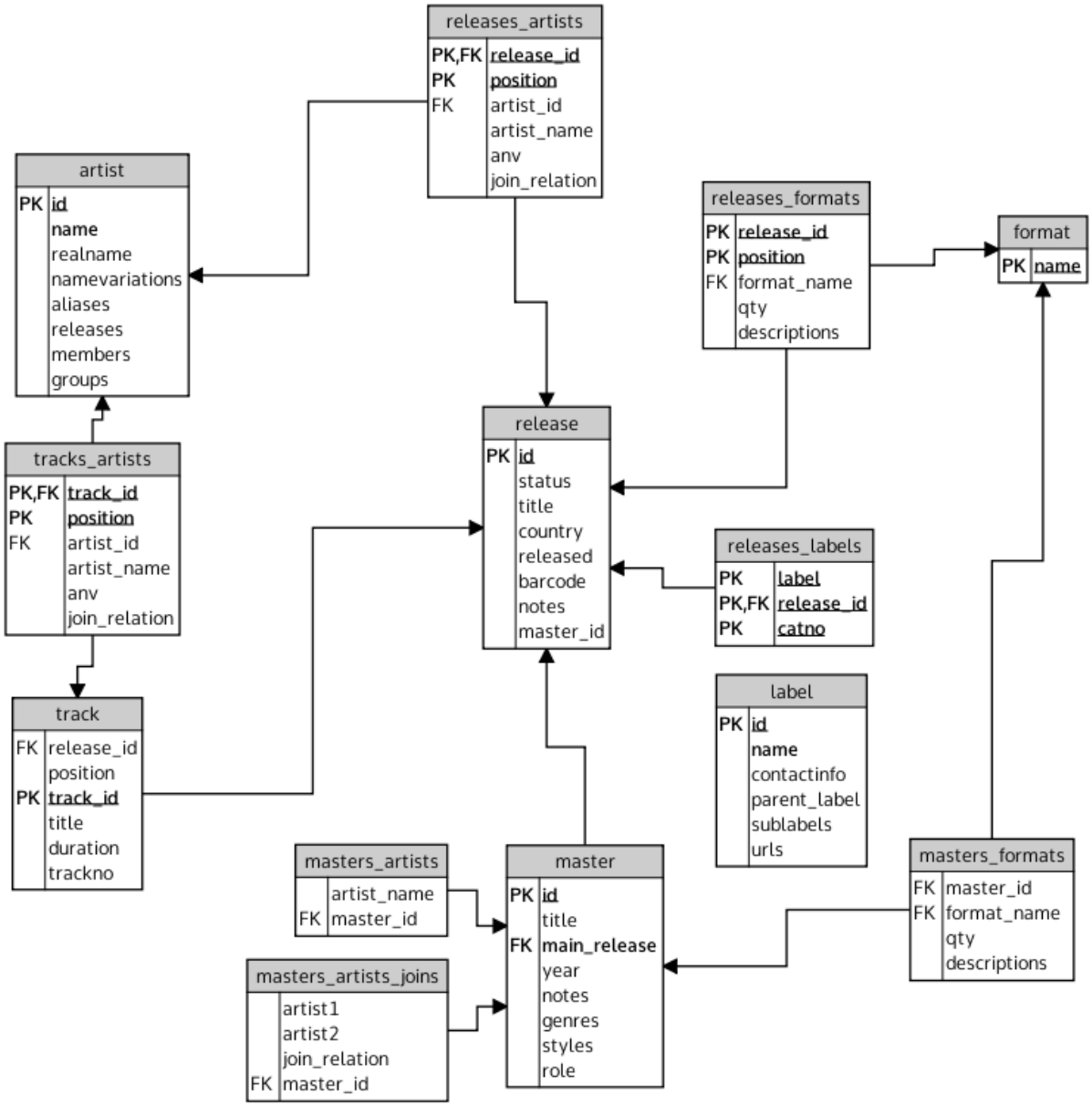

Abbildung 3.18: Erneut bereinigtes Datenbankschema Discogs

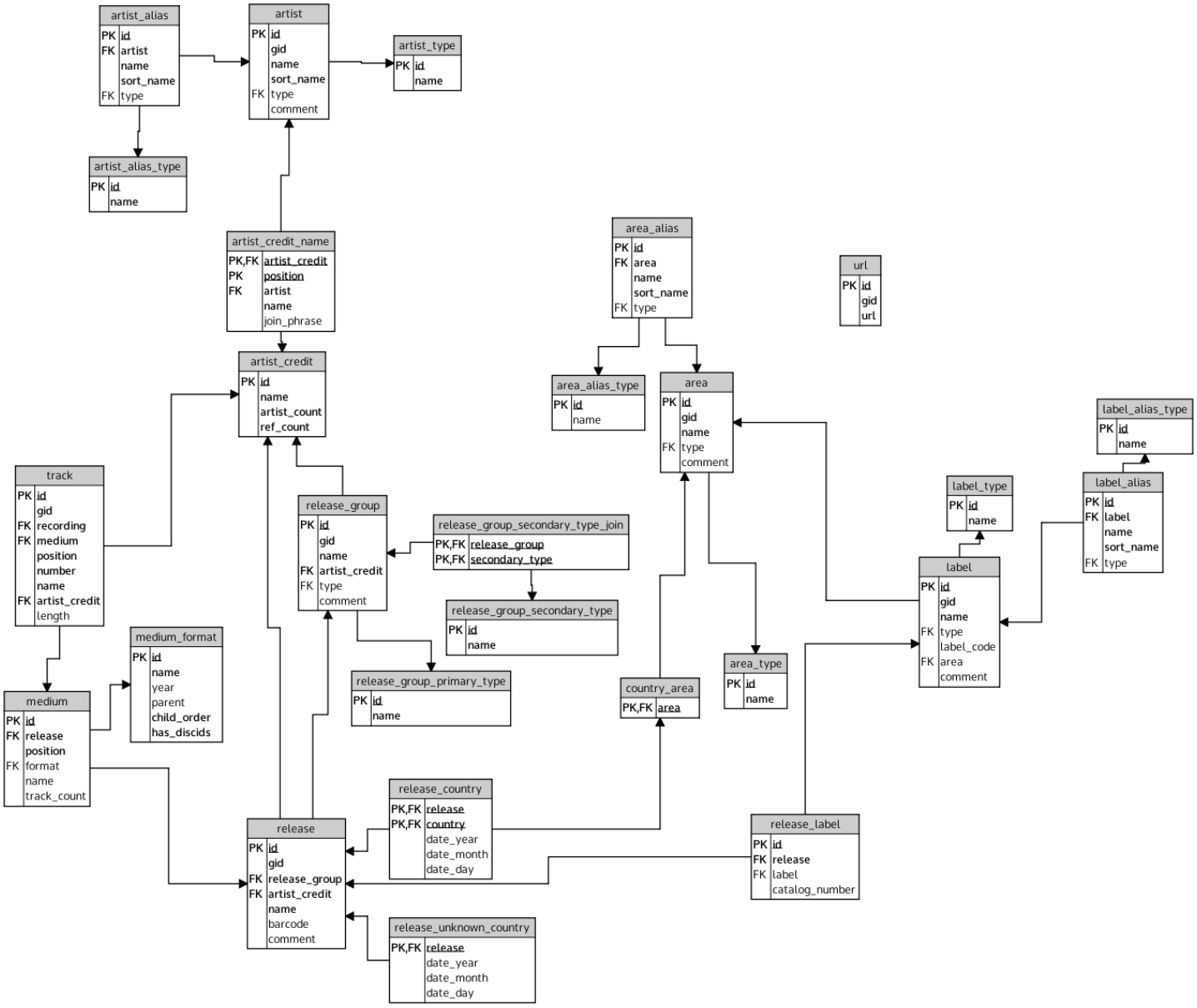

Abbildung 3.19: Erneut bereinigtes Datenbankschema MusicBrainz

Kapitel 4

Konzept

In diesem Kapitel soll das Konzept für das Matching der Musikdatenbanken von Discogs und MusicBrainz erarbeitet werden. In Abschnitt 4.1 wird dazu auf Basis der Untersuchungen des vorangegangenen Kapitels ein gemeinsames Datenbankschema entwickelt. Das gemeinsame Datenbankschema ermöglicht es die Daten der beiden Datenquellen gleich ansprechen zu können. Da die Datenqualität eine Vorbehandlung der Daten vor dem Matching erforderlich macht, wird in Abschnitt 4.2 ein Konzept für das Preprocessing erarbeitet. Das betrifft vor allem das Veröffentlichungsdatum und die Spieldauer in den Daten von Discogs, da diese nicht in einem einheitlichen Format vorliegen. In Abschnitt 4.3 wird dann ein Konzept für das verwendete Indexing-Verfahren vorgestellt. Das Indexing dient dazu die Anzahl der Kandidaten gegenüber dem Kreuzprodukt aus den beiden Datenquellen zu reduzieren. Veränderungen und Anpassungen des SimMatching-Verfahrens an die Gegebenheiten dieser Arbeit werden in Abschnitt 4.4 erläutert, ebenso werden Konkretisierungen des Verfahrens beschrieben, wie bspw. die verwendeten Constraints.

4.1 Datenbanktransformation

In diesem Abschnitt wird ein gemeinsames Datenbankschema für die Datenbanken von Discogs und MusicBrainz entworfen. Dazu werden die in Abschnitt 3.3 gefundenen Gemeinsamkeiten und Unterschiede verwendet und außerdem die Daten genauer betrachtet. Zum Entwerfen des gemeinsamen Schemas werden die Teilbereiche Künstler, Musikstücke, Plattenfirmen und Veröffentlichungen getrennt voneinander betrachtet. Für beide Datenquellen wird angegeben, wie die vorhandenen Daten in das neue Datenbankschema transformiert werden können. Es wird, sofern nicht anders angegeben, *NOT NULL* für alle Attribute angenommen. Klein geschriebene Relationen, wie `artist`, beziehen sich auf die Relation von Discogs bzw. MusicBrainz, wohingegen die groß geschriebene Relation `ARTIST` die Relation des gemeinsamen Schemas bezeichnet.

4.1.1 Künstler

id	name	realname	members	groups
488289	Omnia		{“Jennifer Evans van der Harten“, “Joe Hennon“, “Luka Aubri-Krieger“, “Steve Evans-van der Harten“}	
791615	Jennifer Evans van der Harten	Jennifer Evans-van der Harten		Omnia

(a) Relation `artist` von Discogs

id	name	sort_name	type	area
170525	Omnia	Omnia	2	150
322400	Jennifer van der Harten	Harten, Jennifer van der	1	

(b) Relation `artist` von MusicBrainz

id	link	entity0	entity1
130472	116310	322400	170525

(c) Relation `l_artist_artist` von MusicBrainz

Tabelle 4.1: Daten zur Band *Omnia*

Wie Tabelle 4.1 zeigt, sind die Repräsentationen eines Künstlers unterschiedlich. Damit das Matching der Künstler nicht nur über ihren Namen läuft, müssen die anderen Attribute errechnet werden. Um die Beziehungen zwischen Künstlern zu vereinfachen, wird eine Relation nach dem Vorbild von MusicBrainz genommen. Auch die Aliase eines Künstlers werden wie bei MusicBrainz in eine eigene Relation ausgelagert. So ergibt sich das folgende gemeinsame Schema:

```
ARTIST(id, name)
ARTIST_ALIAS(artist → ARTIST, alias)
ARTIST_RELATION(musician → ARTIST, group → ARTIST)
```

Transformationsvorschrift für Discogs

Die Relation `ARTIST` kann einfach mit den entsprechenden Daten von Discogs gefüllt werden. Hier ist keine weitere Transformation nötig.

Für alle Einträge in der Spalte `aliases` in den Daten von Discogs wird ein neues Objekt in der Tabelle `ARTIST_ALIAS` angelegt, das die ID des Künstlers und den Alias enthält.

Für jeden Künstler, der mindestens einen Eintrag in der Spalte `groups` hat, wird für jede Zugehörigkeit zu einer Gruppe ein neuer Datensatz zur Relation `ARTIST_RELATION` hinzugefügt. Die ID des Musikers ist bekannt und kann daher einfach in den Datensatz

übernommen werden. Die ID der Gruppe muss jedoch über eine Suche nach dem Namen ermittelt werden.

Transformationsvorschrift für MusicBrainz

Die Tabelle `ARTIST` lässt sich einfach mit den entsprechenden Daten von MusicBrainz befüllen, da die benötigten Daten bereits vorliegen. Es ist keine Transformation nötig.

Die Relation `ARTIST_ALIAS` kann mit den Attributen `id`, `artist` und `name` von MusicBrainzs `artist_alias` gefüllt werden. Eine Transformation der Daten ist für diese Tabelle nicht nötig.

Die Daten für `ARTIST_RELATION` lassen sich nicht so leicht finden. Hier ist die Relation `l_artist_artist` sehr hilfreich, da sie die nötigen Daten bereits in den Attributen `entity0` und `entity1` enthält. Es muss jedoch geprüft werden, ob es sich bei dem im Attribut `link` referenzierten Datenbankobjekt um einen Link vom Typ *103* (member of) handelt, um sicher zu stellen, dass die Reihenfolge der Künstler der gewünschten Reihenfolge entspricht.

4.1.2 Musikstück

Musikstücke sind bei Discogs nur als Tracks der erfassten Veröffentlichungen gespeichert. Die Daten über Aufnahmen und die Stücke als Werk an sich von MusicBrainz sind daher nicht relevant für diese Arbeit.

track_id	title	release_id	position	trackno	duration
199fce54-9...	Good Enough	802389	13	13	5:31

(a) Relation `track` von Discogs

id	name	medium	number	position	length
11094325	Good Enough	601525	13	13	331000

(b) Relation `track` von MusicBrainz

Tabelle 4.2: Daten zum Lied *Good Enough* von *Evanescence*

Tabelle 4.2 zeigt, dass die Schemata zu den Musikstücken im Wesentlichen ähnlich sind. Das Attribut `number` von MusicBrainz entspricht dem Attribut `position` von Discogs und nicht `trackno`. Dies ist allerdings nur für Veröffentlichungen mit bspw. doppelseitigen Schallplatten wichtig. Dabei ist auch zu beachten, dass `trackno` die Position auf dem Release angibt und `position` die Position auf dem Medium. Tabelle 4.3 zeigt dies an einem Beispiel. Die Veröffentlichung des Beispiels besteht aus drei Schallplatten und das Album hat insgesamt 17 Titel. Das Lied *D3* befindet sich insgesamt an Stelle 14 (`trackno` auf Discogs) und an Stelle 6 auf der zweiten Schallplatte (`position` auf MusicBrainz).

Die Werte zur Spieldauer sind in unterschiedlichen Darstellungen. Während Discogs auf das Format `mm:ss` setzt, verwendet MusicBrainz die Spielzeit in Millisekunden. Mit einer simplen Umrechnung kann die Angabe in Millisekunden in das Format `mm:ss` gebracht werden und umgekehrt.

Problematisch ist jedoch, dass die Tracks der einen Quelle einer Veröffentlichung und die der anderen Quelle einem Veröffentlichungsmedium zugeordnet werden. Eine eingehende Untersuchung der Datenbestände hat gezeigt, dass es möglich ist die Verknüpfung auf die Veröffentlichungen zu verwenden. Die Zuordnung von Künstlern lehnt sich an das Schema von MusicBrainz an. Beim Vorbereiten der Daten von Discogs muss also eine weitere Tabelle erzeugt werden. Es ergibt sich dann für die Musikstücke das nachfolgende Schema:

```
TRACK(id, title, duration^NULL, position^NULL, number^NULL, release → RELEASE)
TRACK_ARTIST(track, position, artist → ARTIST, join_phrase^NULL)
```

Hierbei bezeichnet `position` die physische Position wie bspw. *D3*, sprich das dritte Lied auf der zweiten Seite der zweiten Schallplatte, und `number` die in der Datenbank gespeicherte Titelnummer. Da sich die Titelnummer auf Discogs auf die Veröffentlichung bezieht und bei MusicBrainz auf das Medium, also bspw. die zweite Schallplatte, divergieren bei Alben wie dem in Tabelle 4.3 die Titelnummern.

track_id	release_id	position	trackno
d6d66506-1...	965	A2	2
42d96c6b-8...	965	D3	14

(a) Relation `track` von Discogs

id	medium	number	position
12728928	255902	A2	2
12728940	255902	D3	6

(b) Relation `track` von MusicBrainz

Tabelle 4.3: Unterschied zwischen `trackno` und `position`

Transformationsvorschrift für Discogs

Für die Transformation des Schemas von Discogs sind lediglich ein paar Attribute umzubenennen. Für die Tabelle `TRACK` wird aus `track_id` kurz `id`, `release_id` wird zu `release` und `trackno` wird in `number` umbenannt.

Bei `TRACK_ARTIST` werden die Attribute `artist_name` und `anv` aus `tracks_artists` weggelassen. Die Attribute `artist`, `track` und `join_phrase` entstehen durch Umbenennung der entsprechenden Attribute.

Transformationsvorschrift für MusicBrainz

Die Attribute `id`, `title`, `duration`, `position` und `number` können mit den Entsprechungen von MusicBrainz befüllt werden. An dieser Stelle bleiben die Daten zur Spieldauer von Discogs und MusicBrainz in unterschiedlichen Formaten. Dies wird im Verlauf des Preprocessings behoben. Die ID für die Veröffentlichung kann direkt dem referenzierten Medium entnommen werden.

Um die Tabelle `TRACK_ARTIST` zu füllen, müssen alle Einträge aus der Relation `artist_credit_name` eingesammelt werden, die den selben Eintrag in `artist_credit` referenzieren wie das Musikstück. Im Wesentlichen handelt es sich dabei um eine Kopie dieser Datensätze, wobei das Attribut `artist_credit` mit der ID des Musikstücks ersetzt wird.

4.1.3 Label

id	name	contactinfo	parent_label	sublabels
796	Universal Records		Universal Music Group	{Freedream, “New Door Records“, “No Carbon Records“, “Riversal Records“, “Spacy Tracy“}

(a) Discogs

id	name	type	label_code	area	comment
852	Universal Records	3		222	1996-2005 American pop label - “RECORDS“ must be a part of the logo!

(b) MusicBrainz

Tabelle 4.4: Daten zum Label *Universal Records*

Wie die Tabelle 4.4 zeigt, ist das Schema der Relationen für die Labels unterschiedlich. Da bei Discogs oft keine Kontaktdaten eingetragen sind oder mehr als eine Anschrift und bei MusicBrainz meist nur ein Land hinterlegt ist, werden diese Attribute nicht länger berücksichtigt.

```
LABEL(id, name)
LABEL_RELATION(parent → LABEL, child → LABEL)
```

Transformationsvorschrift für Discogs

Die Daten für die Relation `LABEL` können direkt der Relation `label` entnommen werden.

Um die Relation `LABEL_RELATION` zu füllen, muss analog zum Vorgehen bei den Künstlern für jeden Eintrag in `label` mit einem *parent label*-Eintrag die ID des übergeordneten Labels mittels einer Namenssuche ermittelt werden.

Transformationsvorschrift für MusicBrainz

Die Tabelle `LABEL` lässt sich mit den entsprechenden Daten der Relation `label` befüllen. Sofern für ein Label ein Eintrag in `l_label_label.entity1` existiert und der Wert im Attribut `link_type` des referenzierten Datenbankobjekts der Relation `link` den Wert *200* (label ownership) hat, wird ein neuer Datensatz mit den Werten aus `entity0` (parent) und `entity1` (child) erstellt.

4.1.4 Veröffentlichung

id	title	country	released	barcode	master_id
4298819	Runaljod - Yggdrasil	Norway	2013	7 090014 387375	533373

(a) Relation `release` von Discogs

id	name	barcode	release_group
1268917	Runaljod - Yggdrasil	7090014387375	1251984

(b) Relation `release` von MusicBrainz

release	country	date_year	date_month	date_day
1268917	160	2013	3	15

(c) Relation `release_country` von MusicBrainz

Tabelle 4.5: Daten zum Album *Runaljod - Yggdrasil* von *Wardruna*

Da die Schemata für Veröffentlichungen relativ ähnlich sind (vgl. Tabelle 4.5), kann hier direkt ein gemeinsames Schema gefunden werden. Das einziges Problem stellt die Inkonsistenz der Veröffentlichungsdaten auf Discogs dar. Das Schema, das sich ergibt, ist:

```
MASTER(id, title)
MASTER_ARTIST(master → MASTER, position, artist → ARTIST,
    join_phrase^NULL)
RELEASE(id, title, master^NULL → MASTER, barcode^NULL)
RELEASE_ARTIST(release → RELEASE, position, artist → ARTIST,
    join_phrase^NULL)
RELEASE_LABEL(release → RELEASE, label → LABEL, catno^NULL)
RELEASE_COUNTRY(release → RELEASE, country → COUNTRY, date^NULL)
COUNTRY(name)
FORMAT(id, name)
MEDIUM(release → RELEASE, position, format^NULL → FORMAT)
```

Hierbei ist `MASTER` das Elternobjekt verschiedener Versionen einer Veröffentlichung. In der Relation `FORMAT` werden alle möglichen Medientypen gespeichert sowie in `MEDIUM` die Veröffentlichungsmedien.

Transformationsvorschrift für Discogs

Die Relation MASTER lässt sich mit den Daten aus den Attributen id und title füllen. Es fallen also Informationen über die Hauptveröffentlichung und auch über das Veröffentlichungsdatum weg. Die Relation MASTER_ARTIST lässt sich aufgrund der Datenstruktur von Discogs nicht in Analogie zu den Künstlern eines Musikstücks erstellen. Die Untersuchungen in Unterabschnitt 3.1.7 haben ergeben, dass statt einer Rekonstruktion der Struktur aus den vorhandenen Daten die Künstlerzuordnungen der Veröffentlichungen zum Mastereintrag genommen werden.

Für die Tabelle RELEASE müssen die Daten von Discogs nicht transformiert werden. Es werden lediglich die Attribute status, country, date, notes, genres und styles weggelassen. Die Relation RELEASE_ARTIST entsteht durch das Entfernen der Spalten artist_name und anv sowie dem Umbenennen von join_relation in join_phrase aus der Tabelle releases_artists. Bei der Relation RELEASE_LABEL handelt es sich um eine Kopie der Tabelle releases_labels, mit dem Unterschied, dass sich der Primärschlüssel aus dem Label und der Veröffentlichung ergibt. Außerdem wird der Name der Plattenfirma durch eine Namenssuche in label mit der ID ersetzt. Die Tabelle RELEASE_COUNTRY wird mit den Attributen id, country und date der Relation release gefüllt. Die Inkonsistenz der Veröffentlichungsdaten wird im Verlauf des Preprocessings behoben. Die Relation COUNTRY wird aus den unterschiedlichen Werten der Spalte country der Relation release gebildet.

Um die Relation FORMAT zu erhalten, wird eine Spalte mit einer eindeutigen ID zur Tabelle format hinzugefügt. Der Primärschlüssel der neuen Relation ist dann die ID und nicht der Name des Formats. Die Relation MEDIUM ist im Wesentlichen eine Kopie von releases_formats ohne die Attribute qty und description. Der Formatname muss durch die ID ersetzt werden, die dem Format bei der Transformation der Tabelle format zugewiesen wurde.

Transformationsvorschrift für MusicBrainz

Die Relation MASTER wird mit den Daten aus der Tabelle release_group gefüllt. Dabei geht die Information über den Typ verloren. Die Attribute id, title, master und barcode der Relation RELEASE können mit den entsprechenden Daten der Relation release gefüllt werden. Die Tabellen MASTER_ARTIST und RELEASE_ARTIST werden analog zu TRACK_ARTIST gefüllt (siehe Unterabschnitt 4.1.2).

Bei der Relation RELEASE_LABEL handelt es sich im Wesentlichen um eine Kopie der Relation release_label. Lediglich die ID und das Attribut last_updated fallen weg. Der Primärschlüssel wird die Kombination aus Veröffentlichungs-ID und Label-ID.

Da die Relation COUNTRY nur den Namen eines Landes beinhaltet, kann diese Relation erstellt werden, indem für alle Einträge aus country_area das Attribut name des referenzierten Datenbankobjekts der Tabelle area verwendet wird. Die Relation RELEASE_COUNTRY entspricht der Tabelle release_country, nur wird das Veröffentlichungs-

datum im Format *yyyy-mm-dd* zusammengeführt. Wie bereits für `COUNTRY` beschrieben, muss der Name des Landes aus der Relation `area` abgefragt und gespeichert werden. Diese Daten müssen dann mit den Einträgen aus `release_unknown_country` vereinigt werden, da sonst die Information des Veröffentlichungsdatums verloren geht, wenn keine Information über das Land vorliegt. Als Eintrag für die Spalte des Landes wird der *empty string* verwendet.

Die Relation `FORMAT` entsteht als Kopie der Spalten `id` und `name` der Tabelle `medium_format`. Die Tabelle `MEDIUM` erhält man aus der Relation `medium` durch das Entfernen der Spalten `id`, `name`, `edits_pending`, `last_updated` und `track_count`. Zusätzlich werden die Attribute `release` und `position` zum Primärschlüssel.

4.1.5 Zusammenfassung

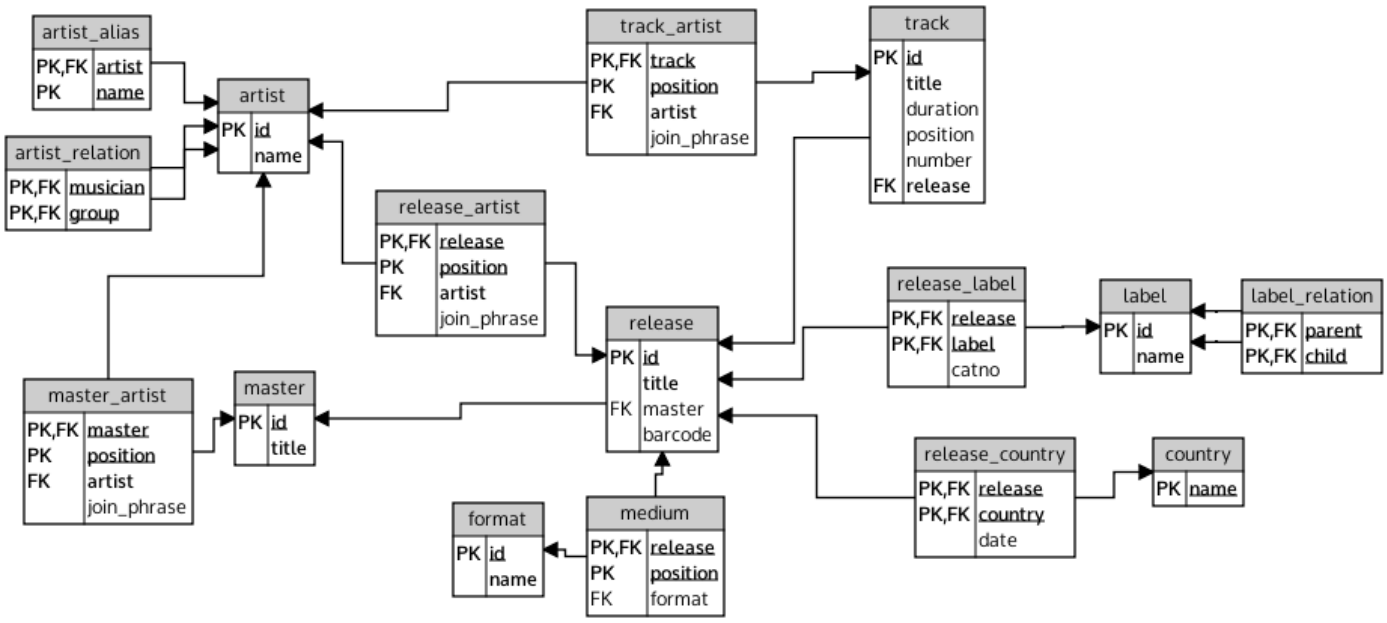

Abbildung 4.1: Gemeinsames Datenbankschema

Die wesentlichen Aspekte der Schemata bleiben bei der Transformation erhalten (vgl. Abbildung 4.1). Es gehen jedoch Informationen wie Geburts-, Sterbe- und Gründungsdaten verloren, da diese nur in den Daten von MusicBrainz enthalten sind. Bei der Transformation fallen ebenfalls die Informationen über Musikrichtungen weg, da diese nur bei Discogs hinterlegt sind.

Entscheidend für die relationalen Ähnlichkeiten ist, dass die Beziehungen zwischen den Entities erhalten bleiben, also bspw. die Zuordnung von Künstlern zu Musikstücken. Denn über diese Beziehungen wird die Nachbarschaft eines Objektes beschrieben. Um die Relationships identifizieren zu können, die für die relationale Ähnlichkeit verwendet werden, wird das relationale Schema (Abbildung 4.1) in ein ER-Diagramm transformiert (Abbildung 4.2).

Die Informationen über erweiterte Mitarbeit (Relationen `*_extraartists`) von Discogs wurden bewusst weggelassen, da sie in den meisten Fällen keinen in der Datenbank gespeicherten Musiker referenzieren. Da solche Personen (oft Produzenten) nicht auf MusicBrainz erfasst sind, gäbe es keine Matching-Partner. Das Berücksichtigen dieser Einträge würde demnach dafür sorgen, dass während des Matching-Prozesses Partner gesucht würden, die nicht existieren.

Es gehen bei der Transformation auch weiterführende Daten verloren, die für den Matching-Prozess jedoch nicht relevant sind. Darunter fallen vor allem gespeicherte Kommentare und Anmerkungen, aber auch die bei Discogs hinterlegten Profiltexte für Musiker und Plattenfirmen und die bei MusicBrainz gespeicherten IDs und Codes wie bspw. der ISRC, sowie die Information über Sprache und Schriftsatz von Musikstücken und Veröffentlichungen. Letztere Information wäre für ein Matching sicherlich interessant, doch liegen diese Daten nur bei MusicBrainz vor und können daher nicht verwendet werden.

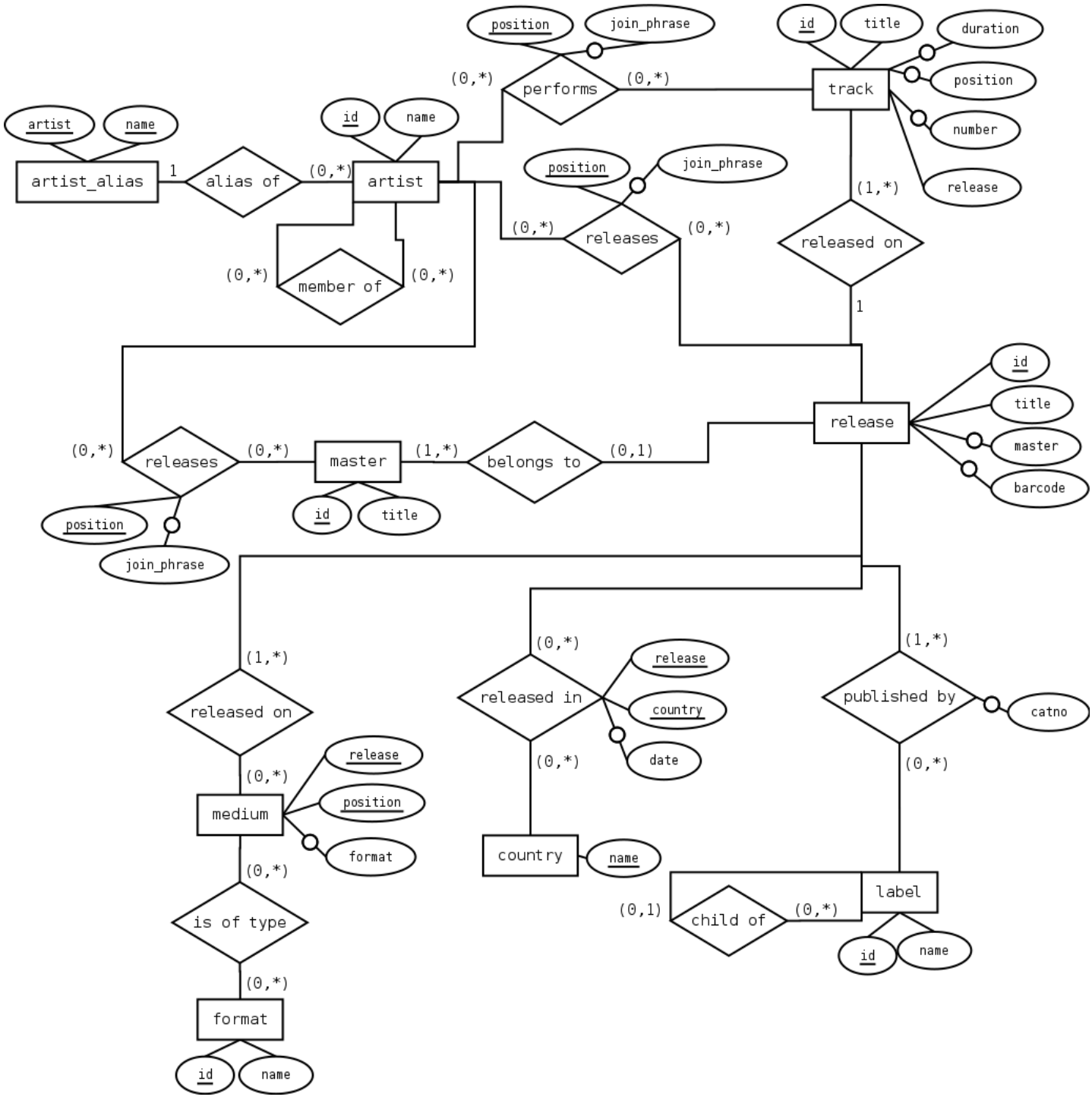

Abbildung 4.2: EER-Diagramm des gemeinsamen Datenbankschemas

4.2 Preprocessing

In diesem Abschnitt wird beschrieben, wie die Daten der beiden Datenbanken vor dem eigentlichen Matching behandelt werden müssen, um die in vorangegangenen Untersuchungen aufgezeigten Probleme wie Inkonsistenzen zu beheben.

Fremdschlüsselbeziehungen

In Unterabschnitt 3.1.7 Punkt 1 wurde festgestellt, dass einige Einträge die vermuteten Fremdschlüsselbeziehungen bei Discogs verletzen. Da nicht alle Relationen, bei denen die vermutete Fremdschlüsselbeziehung verletzt ist, verwendet werden, müssen auch nicht für alle Relationen die Verletzungen behoben werden. Es sind noch die Fremdschlüsselbeziehungen der Relationen `tracks_artists` und `releases_artists` zu korrigieren. Da bereits während der Untersuchung Möglichkeiten zur Korrektur aufgezeigt wurden, werden diese an dieser Stelle nicht erneut erwähnt. Referenzen, die nicht korrigiert werden konnten, werden gelöscht, der Wert also durch *NULL* ersetzt, damit keine Fremdschlüsselbeziehungen verletzt werden.

Duplikate

In Punkt 2 der Untersuchung der Datenqualität von Discogs wurde auf Duplikate bei den Künstlern hingewiesen. Die doppelten Einträge wurden gelöscht und existierende Referenzen auf die ID der anderen Instanz geändert.

Veröffentlichungsdatum

Wie in Punkt 3 von Unterabschnitt 3.1.7 gezeigt wurde, sind die Veröffentlichungsdaten von Discogs nicht in einem einheitlichen Format hinterlegt. Damit die Veröffentlichungsdaten mit denen von MusicBrainz verglichen werden können, müssen die Daten von Discogs auf ein gemeinsames Format gebracht werden. Der Algorithmus 4.1 zur Umwandlung des Datumsformats aus dem Attribut `released` der Relation `release` konvertiert die in Tabelle 3.2 aufgezählten Formate in das Format *yyyy-mm-dd*.

Algorithmus 4.1 Umwandlung der Datumsformate von Discogs

```
1: function ConvertDiscogsDates(String date)
2:     String format = getDateFormat(date)          ▷ aktuelles Format feststellen
3:     String day = getDay(date, format)
4:     String month = getMonth(date, format)
5:     String year = getYear(date, format)
6:     return year + '-' + month + '-' day          ▷ Format der Rückgabe: yyyy-mm-dd
```

Spieldauer

Die Spieldauer der Musikstücke ist auf MusicBrainz in Millisekunden und bei Discogs in den Formaten *mm:ss*, *mm.ss* und *mmss* gespeichert. Um die Laufzeit besser vergleichen zu können, müssen die Formate angepasst werden. Daher werden die Werte des Attributs `duration` der Relation `track` von Discogs wie in Algorithmus 4.2 gezeigt in Millisekunden umgerechnet. Ungültige Werte werden gelöscht, das schließt auch die in der Untersuchung der Datenqualität gefundenen Spieldauern von bspw. 100.000 Stunden ein.

Algorithmus 4.2 Umwandlung der Formate von Discogs in Millisekunden

```
1: function ConvertDiscogsToMillis(String time)
2:     int m = to_int(substring(time, 1, length-2)
3:     int s = to_int(substring(time, length-1, length)
4:     return m * 60000 + s * 1000
```

Redundanzen

Die in Punkt 5 in Unterabschnitt 3.1.7 erwähnten fehlenden Gruppen- bzw. Mitgliedereinträge werden manuell hinzugefügt. Bedingt durch die Repräsentation im gemeinsamen Datenbankschema müssen nur die fehlenden Gruppenzugehörigkeiten eingetragen werden.

Künstlerzuordnung zu Mastereinträgen

In Punkt 6 der Untersuchung der Datenqualität von Discogs wurde aufgezeigt, dass eine Rekonstruktion der Künstler-Master-Zuordnung gemäß des gemeinsamen Datenbankschemas nicht möglich ist. Es wird daher ersatzweise die Künstler-Veröffentlichung-Zuordnung einer zum Mastereintrag gehörenden Veröffentlichung verwendet.

Fusion von Veröffentlichungen

Bedingt durch die Datenstruktur existieren bei Discogs für Veröffentlichungen, die für mehr als ein Land einen Veröffentlichungseintrag erhalten haben, mehrere Einträge, die sich lediglich im Attribut `country`, möglicherweise auch im Attribut `released`, unterscheiden. Da MusicBrainz einer Veröffentlichung mehrere Einträge über das Erscheinen in einem Land zuordnen kann, ist sinnvoll auch die Veröffentlichungen von Discogs so zu gestalten, dass eine Veröffentlichung mehr als einen Eintrag über Veröffentlichungsländer haben kann. Durch die Fusion von Einträgen in der Tabelle `release` entstehen u.a. in den Tabellen `track` und `medium` Einträge, die nicht mehr auf eine gültige Veröffentlichungs-ID zeigen und daher gelöscht werden können. So verringert sich die Anzahl der zu betrachtenden Datenbankobjekte bereits vor dem Matching-Prozess. Da

es in den Daten von Discogs häufig Veröffentlichungen gibt, die eigentlich die gleiche Veröffentlichung in der realen Welt repräsentieren, diese jedoch unterschiedliche Informationen besitzen, werden die Veröffentlichungen, die zusammen gehören, anhand ihrer Katalognummer identifiziert (vgl. Algorithmus 4.3).

Algorithmus 4.3 Fusion von Veröffentlichungen auf Discogs

```
1: function FusionDiscogsReleases( )
2:     for all (master, catno) combinations do
3:         int newId = getMinId()
4:         change ID in release_country to newId                ▷ ID anpassen
5:         remove tracks of the release          ▷ Einträge der alten Veröffentlichung löschen
6:         remove medium of the release
7:         remove artist entry of the release
8:         remove label entry of the release
9:         remove release                          ▷ Fusionierte Veröffentlichung löschen
```

4.3 Indexing-Verfahren

Beim *sorted neighbourhood*-Verfahren werden die zu matchenden Datenbanken nach einem Sortierschlüssel (*sorting key*) geordnet und es wird ein Fenster mit fester Größe ($w > 1$) über die beiden sortierten Datenbanken gelegt. Als potenzielle Matching-Paare werden alle die Einträge betrachtet, die sich im gleichen Fenster befinden. Angenommen beide Datenbanken enthalten n Datensätze, so beträgt die Anzahl der generierten Kandidatenpaare laut [Chr12]:

$$c = w^2 + (n - w)(2w - 1) = 2nw - 2w^2 - n \tag{4.1}$$

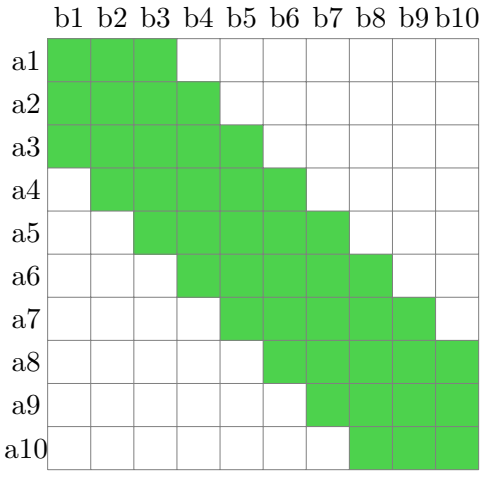

Abbildung 4.3: Beispiel für $w = 3$ und zwei Datenbanken mit je 10 Datensätzen

Abbildung 4.3 zeigt das Verfahren mit einer Fenstergröße $w = 3$. Im ersten Schritt werden die Paare (a1, b1), (a1, b2), (a1, b3), (a2, b1), (a2, b2), (a2, b3), (a3, b1), (a3, b2) und (a3, b3) gebildet. Im nächsten Schritt werden dann die Paare (a2, b2), (a2, b3), (a3, b2) und (a3, b3) erneut gebildet, im Vergleichsschritt werden jedoch alle Kandidatenpaare nur ein Mal verglichen, d.h. das Paar (a2, b2) wird nur ein Mal betrachtet. Neu dazu kommen die Paare (a2, b4), (a3, b4), (a4, b2), (a4, b3), (a4, b4). Nach dem zweiten Schritt gibt es also die in Tabelle 4.6 aufgeführten Kandidatenpaare.

Kandidatenpaare
(a1, b1), (a1, b2), (a1, b3)
(a2, b1), (a2, b2), (a2, b3), (a2, b4)
(a3, b1), (a3, b2), (a3, b3), (a3, b4)
(a4, b2), (a4, b3), (a4, b4)

Tabelle 4.6: Kandidatenpaare nach zwei Schritten

In großen Datenbanken befinden sich bei diesem Vorgehen einige Paare mit dem gleichen Sortierschlüsselwert (SKV) nicht im gleichen Fenster. Daher gibt es eine Variante des Verfahrens, bei der ein invertierter Index über die SKVs gebildet wird. In diesem Index kommt jeder SKV genau ein Mal vor und besitzt eine Referenz auf alle Einträge mit diesem SKV. Das Fenster wird über diesen Index gelegt und nicht direkt über die Daten. Auf diese Weise werden als potenzielle Kandidatenpaare alle Einträge betrachtet, die den gleichen oder einen ähnlichen SKV haben.

Für den Anwendungsfall dieser Arbeit werden die entsprechenden Tabellen also um eine Spalte für den SKV erweitert, wie Tabelle 4.7 zeigt.

id	name	skv
10	Jennifer Evans	evans
13	Amy Lee	lee
25	Steve Evans	evans

Tabelle 4.7: Beispiel: Erweiterung der Tabelle `discogs_artist` um die Spalte `skv`

Anschließend wird für jeden Entitytyp eine Tabelle je Datenquelle angelegt, die jeden SKV nur einmal enthält und zu diesem eine Liste mit den dazugehörigen Einträgen speichert, wie in Tabelle 4.8 gezeigt wird.

skv	ids
evans	{10, 25}
lee	{13}

Tabelle 4.8: Beispiel: Tabelle `discogs_artist_inv_idx` (invertierter Index)

Diese beiden Listen werden mit einem *full outer join* miteinander verschmolzen, um für jeden Entitytyp eine Liste mit den SKVs und den dazugehörigen Objekten der beiden Datenquellen zu haben. Tabelle 4.9 zeigt das Ergebnis der Verschmelzung.

skv	discogs	musicbrainz
evans	{10, 25}	{100}
lee	{13}	{103, 104}

Tabelle 4.9: Beispiel: Tabelle `inv_idx_artist` (verschmolzener invertierter Index)

Auf diese Liste wird ein Fenster mit fester Größe gelegt, um die SKVs zu bestimmen, die die Kandidatenpaare bilden. Hat man die SKV-Paare bestimmt, ergeben sich die Kandidatenpaare aus allen möglichen Kombinationen der Objekte von Discogs und MusicBrainz, wie Tabelle 4.10 zeigt.

discogs	musicbrainz
10	100
10	103
10	104
25	100
25	103
25	104
13	100
13	103
13	104
13	66

Tabelle 4.10: Beispiel: Kandidatenliste `candidates_artist` (Fenstergröße $w = 2$)

In dieser Arbeit wird die sorted neighbourhood Variante mit invertiertem Index genutzt, da im Gegensatz zum *standard blocking*-Verfahren beim sorted neighbourhood Ansatz keine Gruppen von Datensätzen gebildet werden, die miteinander verglichen werden, sondern alle Datensätze verglichen werden, die sich in einer bestimmten Nähe zueinander befinden. Die Variante mit dem invertierten Index wird genutzt, da die verwendeten Datenbestände groß sind und folglich anzunehmen ist, dass es viele Datensätze gibt, die einen gleichen SKV haben, aber weiter auseinander liegen als die Fenstergröße und daher möglicherweise korrekte Zuordnungen nicht betrachtet werden. Laut [Chr12] werden bei diesem Vorgehen zwar mehr Kandidatenpaare gebildet, gleichzeitig sind im Ergebnis aber auch mehr korrekte Zuordnungen enthalten. Die für das Indexing im Rahmen dieser Arbeit verwendeten Sortierschlüssel sind in Tabelle 4.11 aufgeführt.

Entity	Sortierschlüssel
Künstler	Nachname-Vorname
Master	Titel
Musikstück	Titel ohne Suffix
Plattenfirma	Name
Veröffentlichung	Titel ohne Suffix

Tabelle 4.11: Sortierschlüssel

4.4 Matching-Verfahren

In diesem Abschnitt werden die für diese Arbeit benötigten Änderungen und Konkretisierungen des SimMatching-Verfahrens für jeden einzelnen Schritt erläutert. Die Schritte V bis VII bilden weiterhin den iterativen Teil des Verfahrens.

Schritt I: Vorverarbeitung

Die Vorverarbeitung der Daten besteht aus zwei Teilen. Der erste Teil ist die Transformation der Schemata in ein gemeinsames Datenbankschema wie in Abschnitt 4.1 beschrieben. Den zweiten Teil bildet die Datenvorbehandlung aus Abschnitt 4.2.

Schritt II: Kandidaten generieren

Die Kandidatenpaare werden mit dem in Abschnitt 4.3 erläuterten Indexing-Verfahren generiert. Folglich erfolgt die Generierung der Kandidaten mit eben diesem Verfahren.

Schritt III: Constraints prüfen

Noch vor der Berechnung der Ähnlichkeiten können einige Zuordnungen verworfen werden, da sie aufgrund eines unterschiedlichen Attributs nicht das gleiche Realwelt-Objekt abbilden können. In dieser Arbeit können die folgenden Constraints genutzt werden.

- Musikertyp: Bei den Künstlern kann eine Zuordnung verworfen werden, wenn eines der Objekte eine Einzelperson und das andere Objekt eine Gruppe darstellt.
- Veröffentlichungsmedium: Zuordnungen von Veröffentlichungen können verworfen werden, wenn sich das Veröffentlichungsmedium unterscheidet. Da beide Datenbanken unterschiedliche Aufteilungen der Medien haben, werden diese in Gruppen eingeteilt. Dann muss das Veröffentlichungsmedium nicht identisch sein, sondern sich nur in der gleichen Gruppe befinden.
- Position eines Musikstücks: Zwei Datensätze können nicht das gleiche Musikstück beschreiben, wenn die Position auf der Veröffentlichung unterschiedlich ist.

Alle verwendeten Constraints sind regelbasierte Cannot-Match-Constraints.

Schritt IV: Ähnlichkeiten berechnen

Für die Hauptentities `artist`, `track`, `release`, `master` und `label` muss die Ähnlichkeit zweier Objekte berechnet werden. Wie in Abschnitt 2.2 beschrieben, berechnet sich die Ähnlichkeit mit:

$$sim(a,b) = (1-\alpha) * sim_{attr}(a,b) + \alpha * sim_{rel}(a,b) \text{ mit } 0 \leq \alpha \leq 1 \tag{4.2}$$

Für die attributbasierte Ähnlichkeit sim_{attr} gilt dabei:

$$sim_{attr}(a,b) = \sum \omega_i * asm_i(a,b) \text{ mit } \sum \omega_i = 1 \tag{4.3}$$

Dabei bezeichnet $asm_i(a,b)$ die Ähnlichkeit des i-ten Attributs. Zu diesem Zeitpunkt ist für alle Kandidatenpaare die relationale Ähnlichkeit null, da noch keine Zuordnungen bestätigt wurden. Daher wird auf diese Art der Ähnlichkeit erst in Schritt VII eingegangen.

Die Ähnlichkeit von String-Attributen wird mit der in Unterabschnitt 2.5.1 beschriebenen Jaro-Distance berechnet. Die Ähnlichkeit der numerischen Attribute, wie Spieldauer, Position und Veröffentlichungsjahr, wird mit dem in Unterabschnitt 2.5.3 beschriebenen Ähnlichkeitsmaß berechnet.

Für die Identifizierung von Künstlern, die den gleichen realen Künstler repräsentieren, gibt es neben dem Attribut `name` auch noch die Aliase, daher folgt für die Ähnlichkeit nach Gleichung 4.3:

$$\begin{aligned} sim_{attr,artist}(a,b) = \omega_1 * asm_{name}(a,b) + \\ \omega_2 * asm_{alias}(a,b) \text{ mit } \sum \omega_i = 1 \end{aligned} \tag{4.4}$$

Die Schwierigkeit besteht darin die Gewichte ω_i passend zu wählen, d.h. durch die Gewichte werden die Ähnlichkeiten so bestimmt, dass das Matching-Ergebnis optimal wird.

Analog zur Berechnung der Ähnlichkeit von Künstlern werden auch für die anderen Hauptentities alle Attribute verwendet. Sowie bei den Künstlern die Aliase als Attribut aufgefasst werden, werden auch das Veröffentlichungsdatum, -land und -medium für die Veröffentlichung als Attribut betrachtet.

Schritt V: Aggregierungen bilden

Dieser Schritt entfällt in dieser Arbeit, da keine Datenbankobjekte miteinander verschmolzen werden. Das Zusammenlegen der Veröffentlichungen von Discogs ist eine Maßnahme der Datenvorbehandlung.

Schritt VI: Matching-Regeln anwenden

In dieser Arbeit soll nur eine einfache Schwellwert-Regel genutzt werden, d.h. es werden alle Zuordnungen bestätigt, deren Ähnlichkeit einen bestimmten Wert übersteigt.

Schritt VII: relationale Ähnlichkeiten neu berechnen

Die Ähnlichkeit zweier Objekte kann nicht nur über die Attribute berechnet werden, sondern auch über die Beziehungen zu anderen Objekten. Diese Ähnlichkeit nennt man relationale Ähnlichkeit und es gilt:

$$sim_{rel} = \sum \omega_j * rsm_j(a,b) \text{ mit } \sum \omega_j = 1 \tag{4.5}$$

Dabei bezeichnet rsm_j die j-te relationale Ähnlichkeit. Alle relationalen Ähnlichkeiten werden mit dem in Unterabschnitt 2.5.2 beschriebenem Ähnlichkeitsmaß *DirectNeighbourhoodSimilarity* bestimmt. Es muss für jede Verwendung definiert werden wie die Nachbarschaft aussieht. Die verwendeten relationalen Ähnlichkeiten leiten sich aus Abbildung 4.2 ab.

Zur Identifizierung der Künstler können mehrere relationale Ähnlichkeiten verwendet werden. Jede Relationship, an der die Künstler beteiligt sind, wird für eine relationale Ähnlichkeit verwendet. Neben der Beziehung zu anderen Künstlern können daher auch die zu einem Künstler gehörigen Musikstücke, Veröffentlichungen und Mastereinträge verwendet werden. Es ergibt sich somit:

$$\begin{aligned} sim_{rel,artist}(a,b) = \; & \omega_1 * rsm_{relation}(a,b) \\ & \omega_2 * rsm_{track}(a,b) \\ & \omega_3 * rsm_{release}(a,b) \\ & \omega_4 * rsm_{master}(a,b) \text{ mit } \sum \omega_j = 1 \end{aligned} \tag{4.6}$$

Wie auch bei der attributbasierten Ähnlichkeit liegt die Schwierigkeit in einer guten Wahl der Gewichte ω_j.

Die relationale Ähnlichkeit von Musikstücken, Veröffentlichungen, Mastereinträgen und Plattenfirmen berechnet sich analog über die Relationships, an denen das jeweilige Entity beteiligt ist. In Tabelle 4.12 sind alle Beziehungen aus dem ER-Diagramm aufgelistet.

Entitytyp 1	Relationship	Entitytyp 2	Tabelle
Artist	alias of	Artist	artist_alias
Artist	member of	Artist	artist_relation
Artist	performs	Track	track_artist
Artist	releases	Release	release_artist
Artist	releases	Master	master_artist
Label	child of	Label	label_relation
Medium	is of type	Format	*Beziehung über FK*
Release	belongs to	Master	*Beziehung über FK*
Release	published by	Label	release_label
Release	released in	Country	release_country
Release	released on	Medium	*Beziehung über FK*
Track	released on	Release	*Beziehung über FK*

Tabelle 4.12: Übersicht über die relationalen Ähnlichkeiten

Kapitel 5

Implementierung

In diesem Kapitel wird die Implementierung des im vorangegangenen Kapitel erarbeiteten Konzepts für das Matching von Musikdatenbanken mit dem SimMatching-Verfahren beschrieben. In Abschnitt 5.1 wird die Implementierung der Vorbereitung der beiden Datenbanken dargestellt. Darunter ist neben der Transformation in das gemeinsame Datenbankschema auch das Preprocessing zu verstehen. In Abschnitt 5.2 wird die konkrete Implementierung des verwendeten Indexing-Verfahrens erläutert. Die Implementierung des SimMatching-Verfahrens wird in Abschnitt 5.3 beschrieben. Alle Teile der Implementierung werden in *SQL* bzw. *PL/pgSQL* direkt auf dem Datenbankserver implementiert.

5.1 Vorbereitung der Datenbanken

In diesem Abschnitt wird die Implementierung der Datenbanktransformation für die beiden Datenbanken beschrieben. Es wird jedoch nur auf solche Relationen eingegangen, die eine Anfrage an mehr als eine Tabelle stellen oder für die Umwandlungen nötig sind. Die in Abschnitt 4.2 beschriebenen Vorgänge der Datenvorbehandlung werden direkt in die Transformation der Datenbanken integriert und daher ebenfalls in diesem Abschnitt erläutert. Die Tabellen der Quellen werden beim Kopieren in die Zieltabellen transformiert. Die Zieltabellen beschreiben das gemeinsame Schema.

Für komplexere Umformungen und Berechnungen werden Prozeduren in *PL/pgSQL* definiert. Diese Prozeduren können innerhalb einer SQL-Anfrage ebenso genutzt werden wie die eingebauten Funktionen wie bspw. `to_char()` oder `unnest(array)`. Die letztere Funktion ist zum Entschachteln von Arrays. Das Anfrageergebnis einer Anfrage, die `unnest` in der SELECT-Klausel verwendet, liefert für jeden Eintrag im Array eine Zeile.

5.1.1 Discogs

Die Tabellen für Discogs werden gemäß den in Abschnitt 4.1 genannten Transformationsvorschriften erstellt. Für die Erzeugung der Tabellen werden Funktionen benötigt, die Formate eines Attributs angleichen. Die Funktion zum Umrechnen der Spieldauer heißt `to_millis` (siehe Anhang A.1) und wurde nach Algorithmus 4.2 implementiert. Die Funktion für die Bereinigung der Datumsformate heißt `convert_date` (siehe Anhang A.2) und implementiert Algorithmus 4.1 mit den in *PostgreSQL* zur Verfügung stehenden Operationen.

Die in Abschnitt 4.2 erwähnte Rekonstruktion der Künstler-ID über den Künstlernamen für die Relationen `TRACK_ARTIST` und `RELEASE_ARTIST` wird aufgrund von zu hoher Laufzeit nicht implementiert. Ungültige IDs werden daher lediglich mit *NULL* ersetzt. Der Verlust der Information wird zugunsten der Laufzeit akzeptiert.

Künstler

Beim Erstellen der Tabellen für die Aliase muss eine Zeile je Eintrag im entsprechenden Array erzeugt werden. Dies ist möglich mit einer Entschachtelung des Arrays mittels der Funktion `unnest(array)`. Für die Künstlerbeziehungen werden die Namen der Künstler mit den Einträgen im Array `groups` verglichen und bei einer Übereinstimmung der Namen wird ein Datensatz mit der ID des Musikers und der ID der Band angelegt.

Musikstücke

Beim Erstellen der Tabelle für die Musikstücke wird die oben beschriebene Funktion `to_millis(String)` aufgerufen, um die Spieldauer umzurechnen.

Plattenfirmen

Die Beziehung zwischen Plattenfirmen kann abgebildet werden, indem alle Labels mit einem Eintrag im Attribut `parent_label` gesucht werden und anschließend der Name der übergeordneten Plattenfirmen in derselben Relation nachgeschlagen wird. Die IDs der beiden Plattenfirmen bilden dann den Datensatz für die Relation.

Veröffentlichungen

Wie bereits in Unterabschnitt 3.1.7 erwähnt, wird für die Relation `MASTER_ARTIST` die Information über die Künstlerzuordnung zur Hauptveröffentlichung genommen (vgl. Quelltext 5.1). Da für manche Mastereinträge eine ungültige ID im Attribut `main_release` gespeichert ist, muss dieser Fall abgefangen werden. In dieser Situation wird die Zuordnung einer zufälligen Veröffentlichung des Mastereintrags verwendet. Um die Laufzeit zu verbessern, wurde ein Index für das Attribut `master_id` der Relation `release` angelegt. Außerdem wird die bereits bereinigte Relation `RELEASE_ARTIST` verwendet.

```
CREATE discogs_master_artist (master, position, artist, join_phrase) AS
    SELECT m.id, a.position, a.artist, a.join_phrase
    FROM master AS m, discogs_release_artist AS a
    WHERE a.release =
        (SELECT
             CASE WHEN m.main_release IN (SELECT id FROM release)
                 THEN
                     m.main_release
                 ELSE
                     (SELECT id FROM release
                          WHERE master_id = m.id LIMIT 1)
             END);
```

Quelltext 5.1: Erstellen der Tabelle für die Master-Künstler-Beziehung auf Discogs

Beim Erstellen der Tabelle für die Veröffentlichungsländer wird mit dem Aufruf `convert_date(released)` die Inkonsistenz des Datumsformats behoben. Da die Relation `country` im Dump von Discogs leer ist, werden alle unterschiedlichen Einträge im Attribut `country` der Relation `release` für die Tabelle der Veröffentlichungsländer genommen. Wie in Punkt 7 „Fusion von Veröffentlichungen“ in Abschnitt 4.2 erwähnt, werden zusammengehörige Veröffentlichungen nach Algorithmus 4.3 miteinander verschmolzen (siehe Anhang A.4).

5.1.2 MusicBrainz

Auch die Tabellen für MusicBrainz werden gemäß den Transformationsvorschriften erstellt, die in Abschnitt 4.1 genannt sind. Im Folgenden wird auf Besonderheiten und nicht-triviale Situationen eingegangen.

Künstler

Für die Tabelle der Beziehungen zwischen Künstlern dürfen nur solche verwendet werden, die als `link_type` den Wert *103* (*member of*) haben, denn dann ist `entity0` ein Musiker, der ein Mitglied der Gruppe `entity1` ist. Die Implementierung wird in Quelltext 5.2 gezeigt.

```
CREATE TABLE musicbrainz_artist_relation (id, musician, "group") AS
    SELECT a.id, a.entity0, a.entity1
    FROM l_artist_artist AS a JOIN link AS l ON (a.link = l.id)
    WHERE l.link_type = 103;
```

Quelltext 5.2: Erstellen der Tabelle für Künstlerbeziehungen auf MusicBrainz

Musikstücke

Die Informationen über die Künstler eines Stückes müssen indirekt über die Relation `artist_credit` gesammelt werden. Dies lässt sich mit einem Join über das Attribut `artist_credit` der Relationen `track` und `artist_credit_name` bewerkstelligen.

Plattenfirmen

Ähnlich wie schon bei den Künstlern dürfen auch bei den Plattenfirmen nur Beziehungen zwischen diesen berücksichtigt werden, die einen bestimmten Wert im Attribut `link_type` haben. Bei den Plattenfirmen ist dieser Wert *200* (*label ownership*). D.h. `entity0` ist Eigentümer von `entity1`.

Veröffentlichungen

Für das Erstellen der Tabelle über die Künstler eines Mastereintrags bzw. einer Veröffentlichung muss, wie auch schon bei den Musikstücken, der Umweg über die Relation `artist_credit` genommen werden. Da bei MusicBrainz das Veröffentlichungsdatum in drei Spalten als Zahlen gespeichert wird und das gemeinsame Schema das Datum als Zeichenkette im Format *yyyy-mm-dd* erwartet, müssen die einzelnen Werte in Zeichenketten umgewandelt werden. Zu diesem Zweck wurde die Funktion `convert_date(y, m, d)` (siehe Anhang A.3) geschrieben. Für die Tabelle der Veröffentlichungsländer muss der Name des Landes in der Tabelle `area` nachgeschaut werden.

5.2 Indexing

Das in Abschnitt 4.3 vorgestellte *sorted neighbourhood-Verfahren* wird auf dem Datenbankserver mittels PL/pgSQL implementiert. Im ersten Schritt werden die Tabellen der Entitytypen um eine Spalte für den Sortierschlüssel erweitert und der Sortierschlüsselwert in diese eingetragen. Anschließend wird für beide Datenquellen, wie im Beispiel gezeigt, eine Tabelle mit invertiertem Index des Sortierschlüssels erstellt und diese Tabellen dann miteinander verschmolzen. Der Name dieser Tabelle ist `sn_*`, wobei `*` für den Entitytyp steht. Auf diesen Tabellen wird das *sliding window* mittels der in PostgreSQL implementierten *window functions* wie in Quelltext 5.3 realisiert.

```
1 CREATE TABLE wkeys_artist AS
2         SELECT string_agg(key, ',') OVER w AS key,
3                array_agg_multi(discogs) OVER w AS discogs,
4                array_agg_multi(musicbrainz) OVER w AS musicbrainz
5         FROM sn_artist
6         WINDOW w AS (ORDER BY key ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
```

Quelltext 5.3: Sliding Window der Größe 3 für die Künstler

Die Funktion `array_agg_multi` in Quelltext 5.4 ist eine selbst geschriebene Aggregierungsfunktion, die zwei Arrays mittels der bereits vorhandenen Funktion `array_cat` zu einem Array verbindet. Diese Funktion wird benötigt, damit die Arrays mehrerer Schlüsselwerte vereinigt werden können, um alle Einträge zu referenzieren, die zu einer bestimmten Kombination der Schlüsselwerte gehören.

```
1 CREATE AGGREGATE array_agg_multi(int[])
2 (
3   sfunc=array_cat,
4   stype=int[]
5 );
```

Quelltext 5.4: Aggregierungsfunktion `array_agg_multi`

5.3 Matching

Das SimMatching-Verfahren [Sch15] wurde von Schäfers als Hauptspeicher-Algorithmus implementiert. Um mit den großen Datenmengen umgehen zu können, wurden die Daten in Partitionen eingeteilt, die einzeln im Hauptspeicher bearbeitet wurden. Eine Partitionierung lässt sich für den Anwendungsbereich *Musikdaten* nicht so leicht finden, da viele Beziehungen innerhalb der Daten existieren. In dieser Arbeit wird das Matching daher in PL/pgSQL auf dem Datenbankserver implementiert und ausgeführt.

Um eine geringere Laufzeit zu erhalten, wird vorausgesetzt, dass die folgenden Schritte bereits durchgeführt wurden:

- Transformation in ein gemeinsames Datenbankschema
- Preprocessing
- Generierung der Kandidatenpaare

Es wird also mit Schritt III des Verfahrens begonnen (vgl. Abschnitt 4.4).

Constraints

Die Constraints können mit SQL-Anfragen geprüft und daher auch in PL/pgSQL implementiert werden. Unter der Annahme, dass für eine Band immer mindestens ein Bandmitglied eingetragen ist, kann der Künstlertyp (Person / Gruppe) wie in Quelltext 5.5 bestimmt werden.

```
1 SELECT EXISTS(SELECT 1 FROM discogs_artist_relation WHERE "group" = 123);
```

Quelltext 5.5: Künstlertyp

true steht in diesem Fall für eine Gruppe und *false* für eine Person. Wenn die Wahrheitswerte für beide Objekte übereinstimmen, so sind sie vom gleichen Typ und repräsentieren möglicherweise dasselbe Realweltobjekt. Sind die Wahrheitswerte verschieden, so

wird die Zuordnung verworfen, das Paar also aus der Kandidatenliste (Tabelle `candidates_*`) gelöscht. Quelltext 5.6 zeigt wie das Constraint *ArtistCannotMatch* angewendet wird.

```
1 DELETE FROM candidates_artist ca WHERE
2     (SELECT EXISTS (SELECT 1
3                     FROM discogs_artist_relation
4                     WHERE "group" = ca.discogs))
5     <> (SELECT EXISTS (SELECT 1
6                        FROM musicbrainz_artist_relation
7                        WHERE "group" = ca.musicbrainz));
```

Quelltext 5.6: Anwendung des Constraints ArtistCannotMatch

Ähnlichkeitsberechnung

Die Berechnung der Ähnlichkeitswerte wird ebenfalls auf dem Datenbankserver durchgeführt. Für die Jaro-Distance wird die entsprechende Funktion aus der PostgreSQL-Erweiterung *pg_similarity*[6] verwendet.

Die Ähnlichkeitswerte werden in einer Tabelle abgelegt, die als *priority queue* verwendet wird. Dies wird gemacht, damit keine Tabellen in der Größenordnung $10^6 \times 10^6$ mit überwiegend *NULL*-Einträgen angelegt werden müssen. Tabelle 5.1 zeigt eine mögliche Ausprägung der Ähnlichkeitstabelle. Neben dem Gesamtähnlichkeitswert wird auch der Wert für die attributbasierte und die relationale Ähnlichkeit gespeichert, damit bei einer Änderung der relationalen Ähnlichkeit die attributbasierte Ähnlichkeit nicht neu berechnet werden muss.

discogs	musicbrainz	simattr	simrel	sim
13	91	0.89	0.93	0.9033
8	1089	0.77	0.88	0.8067
25	16	0.45	0.05	0.3167

Tabelle 5.1: Beispiel: Ähnlichkeitstabelle `sim_artist`

Da die Ähnlichkeitstabelle die gleichen Daten benötigt wie auch die Kandidatenliste, wird die Kandidatenliste um die Spalten für die verschiedenen Ähnlichkeitswerte erweitert. Auf diese Weise müssen nicht zwei große separate Tabellen verwaltet werden. Die Tabellen `sim_*` heißen also `candidates_*` wobei `*` ein Platzhalter für den Entitytyp ist.

Matching-Regeln

Eine einfache Schwellwertregel, wie sie in dieser Arbeit verwendet wird, lässt sich auf jedem System realisieren. In diesem Fall muss nur der Wert aus der Ähnlichkeitstabelle

[6] http://pgsimilarity.projects.pgfoundry.org/

mit dem Schwellwert verglichen werden. Ist die Ähnlichkeit größer als der Schwellwert, dann wird die Zuordnung bestätigt. Für bestätigte Zuordnungen wird eine Tabelle `matches_*` angelegt, die die IDs von Discogs und MusicBrainz miteinander verknüpft, `*` steht dabei für den Entitytyp.

Wird eine Zuordnung (a, b) bestätigt, dann werden die eventuell vorhandenen Kandidatenpaare (a, x) bzw. (x, b) gelöscht. x steht dabei stellvertretend für alle vorhandenen Werte. Die Kandidatenpaare, die x enthalten, müssen nach einer Bestätigung einer Zuordnung gelöscht werden, da sonst möglicherweise mehr als eine Zuordnung mit dem Objekt bestätigt wird. Dies darf aber nicht passieren, da es nur 1:1-Zuordnungen gibt. Das ergibt sich aus der Tatsache, dass bspw. ein Musiker exakt ein Mal existiert.

Relationale Ähnlichkeit

Die relationale Ähnlichkeit lässt sich mit SQL-Anfragen in PL/pgSQL implementieren. Im Wesentlichen wird die Anzahl der bestätigten benachbarten Zuordnungen und die Anzahl der noch möglichen benachbarten Zuordnungen benötigt. Quelltext 5.7 zeigt wie die Anzahl der bestätigten benachbarten Zuordnungen für die relationale Ähnlichkeit zwischen Künstlern und Musikstücken aus Tabelle 4.12 berechnet werden kann. Die Musikstücke werden hier als die Nachbarschaft betrachtet. Für die Berechnung der noch möglichen benachbarten Zuordnungen muss lediglich die Tabelle `matches_track` durch die Tabelle `candidates_track` ersetzt werden.

```
1  -- 123 / 456 sind lediglich Beispielwerte
2  SELECT COUNT(*)
3  FROM matches_track
4  WHERE EXISTS (SELECT NULL
5                FROM discogs_track_artist
6                WHERE artist = 123
7                AND discogs = track)
8    AND EXISTS (SELECT NULL
9                FROM musicbrainz_track_artist
10               WHERE artist = 456
11               AND musicbrainz = track)
```

Quelltext 5.7: Anzahl bestätigter benachbarter Zuordnungen

Mit diesen beiden Werten kann die relationale Ähnlichkeit gemäß dem in Unterabschnitt 2.5.2 vorgestellten Ähnlichkeitsmaß berechnet werden.

Gesamter Algorithmus

Wie bereits zu Beginn des Abschnitts erwähnt, unterscheidet sich die Implementierung des SimMatching-Verfahrens in dieser Arbeit von der ursprünglichen Implementierung durch Schäfers [Sch15]. Schäfers hat die Daten partitioniert und die Partitionen einzeln

im Hauptspeicher verarbeitet. Die im Rahmen dieser Arbeit entstandene Implementierung sieht die Berechnung direkt auf dem Datenbankserver anstatt im Hauptspeicher vor, da eine Partitionierung der Daten im Anwendungsbereich von *Musikdaten* nicht trivial ist. Die Implementierung orientiert sich also nicht an einem bereits existierenden Quellcode, sondern wurde gänzlich neu geschrieben.

Ein weiterer Unterschied ist, dass bedingt durch die vielen Verknüpfungen der verschiedenen Entitytypen miteinander mehr als nur eine relationale Ähnlichkeit betrachtet wird. Wie unter Schritt VII in Abschnitt 4.4 beschrieben wird daher die relationale Ähnlichkeit als gewichtete Summe der einzelnen relationalen Ähnlichkeiten bestimmt. Außerdem wird beim Anwenden der Matching-Regel für jeden Entitytyp das Kandidatenpaar mit der höchsten Ähnlichkeit betrachtet. Abweichend zu [Sch15] wird nicht nur eine Zuordnung betrachtet, sondern so viele Zuordnungen wie es Entitytypen gibt. Die Änderung ermöglicht es mehr als eine Zuordnung pro Iteration zu bestätigen, dadurch müssen die relationalen Ähnlichkeiten insgesamt seltener aktualisiert werden, was sich wiederum positiv auf die Laufzeit auswirkt.

Die Schritte des Verfahrens wurden jeweils als eine PL/pgSQL-Funktion implementiert. Somit ist es möglich den gesamten Algorithmus übersichtlich in einer weiteren Funktion zu beschreiben (vgl. Anhang B).

5.4 Datenbankoptimierung

Die erste Version der Implementierung nutzte einfache Tabellen und Cursor und diente aufgrund ihrer geringen Performance lediglich zu Testzwecken. Eine Verbesserung der Laufzeit wurde durch die Verwendung von `UNLOGGED TABLES` für Tabellen mit vielen Schreibzugriffen (insert, update, delete) erzielt. Bei `UNLOGGED TABLES` werden Schreibzugriffe, anders als bei einfachen Tabellen, nicht protokolliert.

Außerdem wurden Indizes für sämtliche Spalten angelegt, über die auf die Tabellen zugegriffen wird. Ein Index ermöglicht eine deutlich schnellere Suche nach einem bestimmten Tabelleneintrag. Besonders bei großen Tabellen reduziert die Verwendung eines Indexes bei Suchanfragen die Laufzeit.

In einer weiteren Version der Implementierung wurden dann die ineffizienten Cursor durch große SQL-Anfragen ersetzt. Statt wie vorher über einen Cursor alle Berechnungen und Anfragen für jede einzelne Zeile abzusetzen, wird nun mit einer einzigen Anfrage für alle Zeilen die Berechnung durchgeführt. Quelltext 5.8 zeigt eine beispielhafte Implementierung für das Ändern des Namens in einer Tabelle auf die ersten fünf Zeichen des aktuellen Namens mit beiden Vorgehensweisen.

```
-- Cursor-Implementierung
CREATE FUNCTION cursor_impl() RETURNS VOID AS $$
DECLARE
    curs CURSOR FOR SELECT * FROM test;
    newname text;
BEGIN
    FOR rec IN curs LOOP
        newname = substring(rec.name for 5);
        UPDATE test SET name = newname WHERE id = rec.id;
    END LOOP;
END;
$$ LANGUAGE plpgsql;

-- Implementierung mit einer Anfrage
CREATE FUNCTION query_impl() RETURNS VOID AS $$
BEGIN
    UPDATE test SET name = substring(name for 5);
END;
$$ LANGUAGE plpgsql;
```

Quelltext 5.8: Cursor- und Query-Implementierung

Für eine Tabelle `test` mit 4.308.702 Einträgen und einem Index auf der Spalte `id` dauert die Durchführung der Funktion `cursor_impl` im Schnitt sechs Mal länger als die Durchführung von `query_impl`. Das Ergebnis ist in beiden Fällen allerdings gleich.

Alle Zwischenergebnisse werden im Hauptspeicher gehalten, solange es keinen Vorteil bringt diese in eine Tabelle zu schreiben. Auf Tabellen mit Zwischenergebnissen werden ebenfalls Indizes angelegt, sofern diese die weitere Bearbeitung beschleunigen. Bei der Implementierung wurde darauf geachtet, dass so wenige Lese- und Schreibzugriffe wie möglich durchgeführt werden.

5.5 Evaluation

Aufgrund der hohen Laufzeit einiger Aktionen, wie bspw. dem Erstellen der Kandidatenliste, wurde die Datenmenge beschränkt, auf der die Berechnungen ausgeführt wurden. Da die Laufzeit auch auf einer auf etwa 6,5% beschränkten Datenmenge, ausgehend von den Künstlern auf Discogs, noch zu groß war, wurde eine deutlich kleinere Testmenge erstellt. Auf diese Weise gehen möglicherweise besondere oder interessante Fälle verloren. Ob das Verfahren im Allgemeinen für String-basierte Daten anwendbar ist, hängt von der Güte des Matching-Ergebnisses ab und nicht direkt von der Laufzeit. Eine generelle Aussage über die Eignung des Verfahrens lässt sich demnach immer noch treffen.

Die im Rahmen dieser Arbeit erstellte Implementierung ist nicht effizient genug, um in angemessener Zeit ein Ergebnis für größere Datenmengen wie bspw. bei Discogs und

MusicBrainz zu liefern. Im Folgenden werden die einzelnen Teile der Implementierung bewertet und an kritischen Stellen wird auf die Laufzeit eingegangen.

Die Transformation in ein gemeinsames Datenbankschema hat zum Ziel so viele Informationen wie möglich zu erhalten. Die Schnittmenge an Informationen aus beiden Datenquellen ist relativ klein. Es stehen daher nur wenige Informationen für ein Matching zur Verfügung. Die Qualität des Ergebnisses des Matching-Vorgangs erhöht sich mit zunehmender Menge an Informationen.

Das Preprocessing behandelt nicht alle möglichen Ausnahmefälle, daher gehen einige Informationen verloren. Der Verlust von Informationen ist als negativ für das Matching-Ergebnis zu bewerten, da von den wenigen vorhandenen Informationen für einige Objekte weitere Informationen verloren gehen. Jedoch werden nicht alle Informationen von beiden Datenquellen geliefert und für das Matching sind nur solche Daten nützlich, die von beiden Quellen angeboten werden. Außerdem haben die Funktionen zur Umrechnung der Spieldauer und des Datumsformats eine verhältnismäßig hohe Laufzeit. Die Fusion von Veröffentlichungen auf Discogs ist kritisch und sollte eingehend geprüft werden. Möglicherweise werden Veröffentlichungen, die nicht identisch sind, zusammengelegt oder aber auch identische Veröffentlichungen nicht zusammengelegt. Daher muss geprüft werden, ob die Katalognummer (Attribut `catno`) ein ausreichendes Kriterium ist.

Das in dieser Arbeit verwendete Indexing-Verfahren eignet sich nicht für große Datenmengen. Die Laufzeit war auf einer auf etwa 6,5% eingeschränkten Datenmenge sehr hoch. Besonders viel Zeit nimmt die Erstellung der Kandidatenliste der Musikstücke in Anspruch. Für die 4.026.467 Musikstücke von Discogs und die 1.156.063 von MusicBrainz dauerte das Erstellen der Kandidatenliste knapp sechs Stunden. Diese Laufzeit beinhaltet das Erstellen der Tabelle sowie das Anlegen der Indizes. Die Duplikateliminierung brauchte weitere 8,7 Stunden. Für die Künstler ging dies mit einer Laufzeit von etwa 45 Sekunden inklusive der Duplikateliminierung deutlich schneller. Die Laufzeit des Indexing-Verfahrens ist sehr stark abhängig von der Anzahl der entstehenden Kandidatenpaare. Die Anzahl der entstehenden Kandidatenpaare ist wiederum abhängig vom gewählten Sortierschlüssel. Es konnte kein Sortierschlüssel gefunden werden, der noch genügend Kandidatenpaare erzeugte und gleichzeitig das Indexing in einer angemessenen Zeit durchführbar machte.

Die verwendeten Constraints reduzieren die Anzahl an Kandidatenpaaren für die eine Ähnlichkeitsberechnung durchgeführt werden muss. Dabei ist darauf zu achten, dass Indizes auf den Tabellen, die zur Prüfung des Constraints genutzt werden, verwendet werden, da die Laufzeit sonst sehr hoch ist.

Die Berechnung der Ähnlichkeit hat absolut betrachtet eine sehr hohe Laufzeit, doch liegt das an der Anzahl der Kandidatenpaare, denn die Berechnung der attributbasierten Ähnlichkeit eines Kandidatenpaares dauert im Schnitt 1,2 Millisekunden.

Das Anwenden der Matching-Regeln läuft in einer angemessenen Zeit. Die Matching-Regel wird für das Kandidatenpaar mit der höchsten Ähnlichkeit geprüft. Abweichend von der ursprünglichen Implementierung durch Schäfers [Sch15] wird dies für jeden Entitytyp gemacht, sodass in einer Iteration bspw. eine Künstler- und eine Musikstückzu-

ordnung bestätigt werden kann. Auf diese Weise können in einer Iteration mehr Kandidatenpaare abgearbeitet werden.

Auch bei der Neuberechnung der relationalen Ähnlichkeit lässt sich die hohe Laufzeit mit der Anzahl an Kandidatenpaaren erklären. Im Schnitt werden 2,6 Millisekunden benötigt, um die relationale Ähnlichkeit eines Kandidatenpaares zu bestimmen.

Kapitel 6

Fazit

Die Durchführung dieser Arbeit hat gezeigt, dass das SimMatching-Verfahren ein zufriedenstellendes Matching-Ergebnis liefert. Das SimMatching-Verfahren eignet sich demnach für das Matching von String-basierten Daten. Um dieses Verfahren auf großen Datenmengen - wie sie z.B. Datenbanken von Discogs und MusicBrainz umfassen - anwenden zu können, müssen jedoch Optimierungen vorgenommen werden. Neben der Reduzierung der Laufzeit kann auch das Matching-Ergebnis verbessert werden. Im Folgenden wird auf einige Punkte eingegangen, die in zukünftigen Arbeiten zu diesem Thema berücksichtigt werden sollten.

Verbesserung des Matching-Ergebnisses

Zur Verbesserung des Matching-Ergebnisses könnte mehr Arbeit in die Datenvorbehandlung investiert werden. Dies betrifft insbesondere die Stellen, an denen wissentlich ein Informationsverlust in Kauf genommen wurde (vgl. Abschnitt 4.2).

Außerdem wurden in dieser Arbeit in Analogie zu [Sch15] bei der relationalen Ähnlichkeit nur die bestätigten und die noch möglichen benachbarten Zuordnungen berücksichtigt. In diesem Anwendungsfall hatten die verwendeten Relationships jedoch auch Attribute. Es wäre daher sinnvoll in weiteren Arbeiten zu diesem Thema auch die Relationship-Attribute zu berücksichtigen. Ein Beispiel für einen solchen Fall ist das Attribut`catno` des Relationships `published by` in Abbildung 4.2. Eine Möglichkeit wäre die *DirectNeighbourhoodSimilarity* um einen gewichteten Teil mit den Ähnlichkeiten der Attribute zu erweitern. Die Berechnung der relationalen Ähnlichkeit von Veröffentlichungen könnte in der Nachbarschaft von Plattenfirmen dann wie folgt bestimmt werden:

$$sim_{rel}(a,b) = \omega_1 * sim_{Neighbourhood}(a,b) + \omega_2 * sim_{catno}(a,b) \text{ mit } \omega_1 + \omega_2 = 1 \quad (6.1)$$

Laufzeitverbesserung

Das verwendete Indexing-Verfahren hatte eine hohe Laufzeit. Die Laufzeit des Indexings könnte durch die Verwendung von PostgreSQL 9.5 anstelle von Version 9.4 um die Dauer

der Duplikateliminierung reduziert werden, denn in Version 9.5 können beim Einfügen Tupel ignoriert werden, die einen Konflikt verursachen. In der verwendeten Version 9.4 führte dies zu einem Fehler und damit zum Abbruch der Funktion. Das besagte Feature ist die `ON CONFLICT-Klausel` ([PGDG16b] S. 1587f.). Angenommen der Primärschlüssel der Tabelle `candidates_artists` würde aus der Kombination der Attribute `discogs` und `musicbrainz` bestehen und **`query`** stünde für die SQL-Anfrage, die die Kandidatenpaare erzeugt, dann würde die Anfrage in Quelltext 6.1 eine Kandidatenliste ohne Duplikate erzeugen.

```
1 INSERT INTO candidates_artists query ON CONFLICT DO NOTHING;
```

Quelltext 6.1: ON CONFLICT-Klausel in PostgreSQL 9.5

Allerdings würde nur diese Änderung das verwendete Indexing-Verfahren noch nicht genügend beschleunigen, um auch auf großen Datenmengen in angemessener Zeit zu terminieren. Es wäre daher ratsam ein anderes Indexing-Verfahren zu wählen, dessen Laufzeit möglichst linear zur Eingabegröße ist.

Partitionierung

Wie in Abschnitt 5.3 erwähnt, wurde das SimMatching-Verfahren als Hauptspeicher-Algorithmus mit Partitionierung der Daten entwickelt. Im Bezug auf das in dieser Arbeit entwickelte gemeinsame Datenbankschema könnte eine Partitionierung möglicherweise über den Entitytyp Künstler erstellt werden, denn alle anderen Entitytypen hängen direkt oder indirekt von diesem ab. Eine Partitionierung erfordert ein gutes Indexing und das in dieser Arbeit verwendete Indexing hat sich u.a. durch die hohe Laufzeit als nicht sonderlich praktikabel erwiesen (vgl. Abschnitt 5.5). Dies liegt vermutlich zum Großteil an der Wahl des Sortierschlüssels. Eine Partitionierung der Daten würde es ermöglichen, das Matching wieder als Hauptspeicher-Algorithmus zu implementieren.

Instanzbasierte Must-Match-Constraints

In den Daten von MusicBrainz gibt es eine Tabelle `url`. Diese Tabelle dient dazu externe Quellen zu verlinken. Eine dieser Quellen ist Discogs und die URL auf Discogs enthält die ID des Objekts. Die ID der Band *Evanescence* bspw. ist 163505 und die URL *https://www.discogs.com/artist/163505*. Unter der Voraussetzung, dass alle URLs korrekt sind, könnte die Tabelle für instanzbasierte Must-Match-Constraints genutzt werden. Auf diese Weise würden alle Objekte von MusicBrainz bereits vor der Berechnung von Ähnlichkeiten mit dem entsprechendem Objekt von Discogs gematcht werden, sofern in der Tabelle `url` eine URL zu Discogs gespeichert ist.

Anhang A

Basisfunktionen

In diesem Teil des Anhangs wird die Implementierung der verschiedenen Basisfunktionen aufgezeigt, die bei der Vorbereitung der Datenbanken auf das Matching benötigt werden.

A.1 Konvertierung der Spieldauer bei Discogs

```
1  CREATE FUNCTION to_millis(input text) RETURNS integer AS $$
2  DECLARE
3      format text := '9999';        -- to be sure every value fits in
4      seconds integer := to_number(substring(input from '..$'), format);
5      minutes integer := to_number(
6          substring(input for (position(':' in input)-1)), format);
7  BEGIN
8      RETURN seconds * 1000 + minutes * 60000;
9  END;
10 $$ LANGUAGE plpgsql;
```

Quelltext A.1: Umwandlung des Formats *mm:ss* in Millisekunden

A.2 Konvertierung der Datumsformate bei Discogs

```
CREATE FUNCTION convert_date(text) RETURNS text AS $$
DECLARE
    date ALIAS FOR $1;
    format text := '9909';
    year integer;
    month integer;
    day integer;
    tmp date;
BEGIN
    IF date SIMILAR TO '[0-9]{8}' THEN
        -- yyyymmdd
        RETURN substring(date from 1 for 4) || '-' ||
            substring(date from 5 for 2) || '-' ||
            substring(date from 7 for 2);
    END IF;

    IF date SIMILAR TO '[0-9]{4}%' THEN
        IF char_length(date) = 4 THEN
            -- yyyy
            RETURN date;
        ELSIF date SIMILAR TO '[0-9]{4}-[0-9]{2}' THEN
            -- yyyy-mm
            RETURN date;
        ELSIF date SIMILAR TO '[0-9]{4}%[0-9]{2}%[0-9]{2}' THEN
            -- yyyy-mm-dd
            IF char_length(date) > 10 THEN
                -- replacing _ with % added format yyyy - mm - dd
                date := replace(date, ' ', '');
            END IF;
            RETURN substring(date from 1 for 4) || '-' ||
                substring(date from 6 for 2) || '-' ||
                substring(date from 9 for 2);
        END IF;
    END IF;

    IF date SIMILAR TO '[0-9]{1,2}/[0-9]{2}/([0-9]{2}){1,2}' THEN
        -- (m)m/dd/(yy)yy
        year := to_number(split_part(date, '/', 3), format);
        month := to_number(split_part(date, '/', 1), format);
        day := to_number(split_part(date, '/', 2), format);
        IF year < 100 THEN
            IF year > 17 THEN
                year := year + 1900;
            ELSE
```

```
45                  year := year + 2000;
46              END IF;
47          END IF;
48          RETURN trim(leading ' ' from to_char(year, format)) || '-' ||
49          trim(leading ' ' from to_char(month, format)) || '-' ||
50          trim(leading ' ' from to_char(day, format));
51      END IF;
52
53      IF date SIMILAR TO '[0-9]{2}_[0-9]{2}_[0-9]{4}' THEN
54          IF substring(date from 3 for 1) LIKE '.' THEN
55              -- dd.mm.yyyy
56              RETURN substring(date from 7 for 4) || '-' ||
57                  substring(date from 4 for 2) || '-' ||
58                  substring(date from 1 for 2);
59          ELSIF substring(date from 3 for 1) LIKE '-' THEN
60              -- mm-dd-yyyy
61              RETURN substring(date from 7 for 4) || '-' ||
62                  substring(date from 1 for 2) || '-' ||
63                  substring(date from 4 for 2);
64          END IF;
65      END IF;
66
67      IF date SIMILAR TO '[a-zA-Z]{3,9} [0-9]{1,2}, [0-9]{4}' THEN
68          -- Month dd, yyyy
69          tmp := to_date(date, 'Month DD, YYYY');
70          RETURN to_char(tmp, 'YYYY-MM-DD');
71      END IF;
72
73      RETURN '';          -- unknown date
74  END;
75  $$ LANGUAGE plpgsql;
```

Quelltext A.2: Konvertierung der verschiedenen Datumsformate von Discogs

A.3 Erzeugung des Datumsstrings bei MusicBrainz

```
CREATE FUNCTION convert_date(y integer, m integer, d integer)
    RETURNS text AS $$
DECLARE
    format text := '9909';
    year text;
    month text;
    day text;
BEGIN
    IF y IS NOT NULL THEN
        year := trim(leading ' ' from to_char(y, format));
    END IF;

    IF m IS NOT NULL THEN
        month := trim(leading ' ' from to_char(m, format));
    END IF;

    IF d IS NOT NULL THEN
        day := trim(leading ' ' from to_char(d, format));
    END IF;

    IF year IS NOT NULL AND month IS NOT NULL AND day IS NOT NULL THEN
        RETURN year || '-' || month || '-' || day;        -- yyyy-mm-dd
    ELSIF year IS NOT NULL AND month IS NOT NULL THEN
        RETURN year || '-' || month;        -- yyyy-mm
    ELSIF year IS NOT NULL THEN
        RETURN year;        -- yyyy
    ELSE
        RETURN '';        -- unknown date
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Quelltext A.3: Datumsumwandlung bei MusicBrainz

Der Formatstring `'9909'` ist hierbei universell anwendbar, da er die Zahlen in eine zwei- bis vierstellige Repräsentation umwandelt, wobei die zweistellige Darstellung eine führende Null hat, sollte die Eingabe eine Zahl kleiner als 10 sein. Diese Formatumwandlung produziert leider führende Leerzeichen, welche mittels eines Aufrufs der Funktion `trim(leading [characters] from string)` entfernt werden.

A.4 Fusion von Veröffentlichungen auf Discogs

```
CREATE FUNCTION discogs_fusion_releases() RETURNS VOID AS $$
BEGIN
    -- find the replacement ID for each (master, catno) combination
    CREATE MATERIALIZED VIEW tmp as
        SELECT a.master, a.catno, MIN(a.id) AS newid
        FROM (discogs_release AS a1
                JOIN discogs_release_label AS a2
                ON (a1.id = a2.release)) AS a
             JOIN
             (discogs_release AS b1
                 JOIN discogs_release_label AS b2
                 ON (b1.id = b2.release)) AS b
             ON (a.master = b.master AND a.id < b.id AND a.catno = b.catno)
        GROUP BY a.catno, a.master;
    CREATE INDEX tmp_newid_idx ON tmp(newid);
    CREATE INDEX tmp_mid_idx ON tmp(master);

    -- find all IDs to replace
    CREATE UNLOGGED TABLE tmp2 as
        SELECT r.id AS oldid, tmp.newid
        FROM (discogs_release AS r
                JOIN discogs_release_label AS l
                ON (r.id = l.release))
             JOIN tmp ON (r.master = tmp.master
                          AND l.catno = tmp.catno
                          AND r.id <> tmp.newid);
    CREATE INDEX tmp2_oldid_idx ON tmp2(oldid);
    CREATE INDEX tmp2_newid_idx ON tmp2(newid);

    -- update release country entry and remove possible duplicates
    UPDATE discogs_release_country rc
    SET release = (
        SELECT tmp2.newid FROM tmp2 WHERE rc.release = tmp2.oldid)
    WHERE rc.release IN (SELECT tmp2.oldid FROM tmp2);
    DELETE FROM discogs_release_country rc WHERE EXISTS (
        SELECT 1
        FROM discogs_release_country AS a
        WHERE rc.release = a.release
          AND rc.country = a.country
          AND rc.date = a.date
          AND a.ctid > rc.ctid);

    -- delete entries related to old releases
    DELETE FROM discogs_track_artist ta
    WHERE ta.track IN (
        SELECT id FROM discogs_track
        WHERE release IN (SELECT oldid FROM tmp2));
    DELETE FROM discogs_track tr
    WHERE tr.release IN (SELECT oldid FROM tmp2);
    DELETE FROM discogs_medium me
```

```
51      WHERE me.release IN (SELECT oldid FROM tmp2);
52      DELETE FROM discogs_release_label rl
53      WHERE rl.release IN (SELECT oldid FROM tmp2);
54      DELETE FROM discogs_release_artist ra
55      WHERE ra.release IN (SELECT oldid FROM tmp2);
56      DELETE FROM discogs_release re
57      WHERE re.id IN (SELECT oldid FROM tmp2);
58
59      -- clean up
60      DROP MATERIALIZED VIEW tmp;
61      DROP TABLE tmp2;
62  END;
63  $$ LANGUAGE plpgsql;
```

Quelltext A.4: Fusion von Veröffentlichungen auf Discogs

Anhang B

SimMatching-Verfahren

```
CREATE FUNCTION simmatching() RETURNS VOID AS $$
DECLARE
    cnt int := 1;
    helper int;
    rels text[] := ARRAY['artist', 'artist_alias', 'label', 'track',
        'release', 'master'];
    rel text;
    threshold float := 0.75;
BEGIN
    -- I: Preprocessing
    -- creating the transformed data
    PERFORM transformData();

    -- II: Candidates
    -- creating candidate pairs
    PERFORM createAllCandidates();

    -- III: Constraints
    -- check Constraints
    PERFORM checkConstraints();

    -- prepare the tables for the matches
    FOREACH rel IN ARRAY rels LOOP
        EXECUTE format('CREATE TABLE matches_%s
            (LIKE candidates_%1$s)', rel);
        EXECUTE format('CREATE INDEX matches_%s_discogs_idx
            ON matches_%1$s(discogs)', rel);
        EXECUTE format('CREATE INDEX matches_%s_musicbrainz_idx
            ON matches_%1$s(musicbrainz)', rel);
    END LOOP;

    -- IV: Similarity
```

```
33      -- calculate the similarity of attributes
34      PERFORM calculateSimilarity();
35
36      -- the following steps belong to the iterative part of the algorithm
37      WHILE cnt > 0 LOOP
38          -- V: Aggregations
39          -- this step is left out in this implementation
40          -- because there is nothing to aggregate
41
42          -- VI: Matching Rule
43          -- apply the matching rule(s)
44          PERFORM matching_rule();
45
46          -- VII: Recalculate Relational Similarity
47          PERFORM calculateRelationalSimilarity();
48
49          -- get new counter value
50          cnt := 0;
51          FOREACH rel IN ARRAY rels LOOP
52              EXECUTE format('SELECT count(*) FROM candidates_%s
53                  WHERE sim > %s INTO helper', rel, threshold);
54              cnt := cnt + helper;
55          END LOOP;
56      END LOOP;
57  END;
58  $$ LANGUAGE plpgsql;
```

Quelltext B.1: Implementierung des SimMatching-Verfahrens

Der letzte Teil innerhalb der `WHILE`-Schleife dient dazu die Abbruchbedingung zu prüfen. Der Algorithmus soll terminieren sobald keine Kandidatenpaare mehr vorhanden sind oder keine Zuordnung mehr bestätigt werden kann.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Chr12] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Data-Centric Systems and Applications. Springer-Verlag Berlin Heidelberg, 2012. ISBN 978-3-642-31163-5.

[CMZ09] S. Chen, B. Ma, K. Zhang. On the similarity metric and the distance metric. In *Theoretical Computer Science*, 410. 2009, 2365–2376.

[EIV07] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 2007, 1–16. ISSN 1041-4347.

[Met15] MetaBrainz. How Editing Works — MusicBrainz Wiki, 2015. [Online; abgerufen 25. April 2016], URL `http://wiki.musicbrainz.org/How_Editing_Works`.

[PGDG16a] PostgreSQL Global Development Group. PostgreSQL 9.4.7 Documentation, 2016. [Online; abgerufen 13. Mai 2016], URL `https://www.postgresql.org/files/documentation/pdf/9.4/postgresql-9.4-A4.pdf`.

[PGDG16b] PostgreSQL Global Development Group. PostgreSQL 9.5.2 Documentation, 2016. [Online; abgerufen 02. August 2016], URL `https://www.postgresql.org/files/documentation/pdf/9.5/postgresql-9.5-A4.pdf`.

[Pra12] A. Prante. *String-basiertes Matching von Datenbank-Objekten.* Masterarbeit, Fachgebiet Datenbanken und Informationssysteme, Institut für Praktische Informatik, Leibniz Universität Hannover, 2012.

[Sch15] M. Schäfers. *Ein ähnlichkeitsbasiertes Matching-Verfahren für die Integration räumlicher Datenbanken.* Dissertation, Fachgebiet Datenbanken und Informationssysteme, Institut für Praktische Informatik, Leibniz Universität Hannover, 2015.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit und die zugehörige Implementierung selbstständig verfasst und dabei nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 18. August 2016

Philipp Daniel Rohde